INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
PROGRAMA DE MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL

SIDNEY DA SILVA FERREIRA

A integração da Análise Combinatória e do Pensamento Computacional no Ensino de Matemática por meio da Linguagem Python

Orientador: Jones Colombo

NITERÓI
MAIO/2026

SIDNEY DA SILVA FERREIRA

A integração da Análise Combinatória e do Pensamento Computacional no Ensino de Matemática por meio da Linguagem Python

Dissertação apresentada por **Sidney da Silva Ferreira** ao Programa de Mestrado Profissional em Matemática em Rede Nacional - Universidade Federal Fluminense, como requisito parcial para a obtenção do Grau de Mestre.

Orientador:
Jones Colombo

NITERÓI

2026

Ficha catalográfica automática - SDC/BIME
Gerada com informações fornecidas pelo autor

F383i Ferreira, Sidney da Silva
A integração da Análise Combinatória e do Pensamento Computacional no Ensino de Matemática por meio da Linguagem Python / Sidney da Silva Ferreira. - 2026.
290 p.: il.

Orientador: Jones Colombo.
Dissertação (mestrado profissional)-Universidade Federal Fluminense, Niterói, 2026.

1. Análise combinatória. 2. Ensino de matemática. 3. Informática na educação. 4. Python. 5. Produção intelectual. I. Colombo, Jones, orientador. II. Universidade Federal Fluminense. Instituto de Matemática e Estatística. III. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

SIDNEY DA SILVA FERREIRA

A integração da Análise Combinatória e do Pensamento Computacional no Ensino de Matemática por meio da Linguagem Python

Dissertação apresentada por **Sidney da Silva Ferreira** ao Programa de Mestrado Profissional em Matemática em Rede Nacional - Universidade Federal Fluminense, como requisito parcial para a obtenção do Grau de Mestre. Linha de pesquisa: Matemática na Educação Básica e suas Tecnologias.

Aprovada em: 22/05/2026

Banca Examinadora

Prof. Jones Colombo - Orientador
Doutor - Universidade Federal Fluminense

gov.br Documento assinado digitalmente
JONES COLOMBO
Data: 13/07/2026 12:18:13-0300
Verifique em https://validar.iti.gov.br

Prof. Gladson Octaviano Antunes - Membro
Doutor - Universidade Federal do Estado do Rio de Janeiro

gov.br Documento assinado digitalmente
GLADSON OCTAVIANO ANTUNES
Data: 14/07/2026 11:08:42-0300
Verifique em https://validar.iti.gov.br

Prof. Ana Isabel de Azevedo Spinola Dias - Membro
Doutor - Universidade Federal Fluminense

gov.br Documento assinado digitalmente
ANA ISABEL DE AZEVEDO SPINOLA DIAS
Data: 13/07/2026 15:18:22-0300
Verifique em https://validar.iti.gov.br

Prof. Luiz Manoel Silva de Figueiredo - Membro
Doutor - Universidade Federal Fluminense

gov.br Documento assinado digitalmente
LUIZ MANOEL SILVA DE FIGUEIREDO
Data: 14/07/2026 08:55:08-0300
Verifique em https://validar.iti.gov.br

Niterói
2026

Dedico esta dissertação com profundo carinho e gratidão à minha esposa Jane e aos meus filhos Emmanuel, Gabriel, Guilherme e Ester, meus amores.

Agradecimentos

Em primeiro lugar a Deus, “inteligência suprema, causa primária de todas as coisas”.

Aos meus pais, que me proporcionaram amor, carinho e dedicação para que tivesse uma boa educação.

À minha querida esposa Jane e aos meus filhos Emmanuel, Gabriel, Guilherme e Ester, obrigado pelo apoio, compreensão e incentivo. Sem vocês, nada disso seria possível. O apoio incondicional, paciência e amor de vocês, foram essenciais para superar os desafios desta jornada. Obrigado por estarem ao meu lado em todos os momentos.

À Sociedade Brasileira de Matemática, que luta pela melhoria do ensino da matemática na educação básica, implementou o programa de mestrado profissional em matemática em rede (PROFMAT), o qual desempenha um papel importante na formação de professores qualificados que contribuirão para a promoção da educação matemática.

Sou grato ao meu orientador, Jones Colombo, à coordenadora, Dirce Uesu Pesco, e aos demais docentes do PROFMAT/UFF que acompanharam minha trajetória. Suas lições e “insights” foram inestimáveis em minha jornada de aprendizado.

Em especial agradeço aos professores Prof. Dr. Jones Colombo, Prof. Dr. Gladson Octaviano Antunes, Prof. Dr. Ana Isabel de Azevedo Spinola Dias e Prof. Dr. Luiz Manoel Silva de Figueiredo por fazerem parte da banca avaliadora desse trabalho, por suas contribuições com o mesmo e, consequentemente, com o meu aperfeiçoamento.

Aos colegas de turma pela amizade e cumplicidade em todo decorrer do curso. Em especial, às companheiras de estudos nos feriados e fins de semana, Cleide, Fernanda, Carolina e Milena, pelos incentivos e trocas de ideias.

Este trabalho é resultado do apoio, orientação e inspiração de muitas pessoas. Quero expressar a minha mais profunda gratidão a todos que me acompanharam nesta jornada. Este é um marco que comemoramos juntos.

Resumo

O presente trabalho tem como objetivo investigar e produzir um conjunto de atividades que estimule os estudantes a produzirem seus próprios significados quando estiverem a resolver problemas de contagem utilizando uma perspectiva voltada para a observação do conjunto de resultados de cada problema. Tais conjuntos de resultados são obtidos através de programas na linguagem Python. A pesquisa é fundamentada teoricamente na Didática dos Princípios de Contagem, no Modelo Cognitivo de Lockwood e no Modelo dos Campos Semânticos.

Palavras-chave: Análise combinatória; Ensino de Matemática; Pensamento computacional; Modelo dos Campos Semânticos; Python.

Abstract

The present work aims to investigate and develop a set of activities that encourages students to produce their own meanings when solving counting problems, adopting a perspective focused on observing the set of outcomes of each problem. These sets of outcomes are generated using Python programs. The research is theoretically grounded in the Didactics of Counting Principles, Lockwood's Cognitive Model, and the Model of Semantic Fields.

Keywords: Combinatorics; Teaching Mathematics; Computational thinking; Semantic Fields Model; Python.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Sumário

1 Introdução

Análise Combinatória, na Educação Básica, é a matemática da contagem, como é mencionada em (BRASIL, 2018). Mas, o que significa contar? Significa determinar o número exato de objetos especificados por uma pergunta "Quantos?" ou "De quantas maneiras...?".

Mas, "Contar é dificil" conforme mencionado em (MARTIN, 2001) e, vemos isso na prática, em sala de aula. Professores têm dificuldades em ensinar Análise Combinatória e os alunos têm dificuldades em compreendê-la. Pois, na maioria das vezes, ficam mais preocupados em descobrir qual fórmula irão utilizar na resolução do problema do que entender o que o problema está pedindo de fato.

Segundo (CERIOLI; VIANA, 2012), no ensino do Análise Combinatória adotamos a *Didática da Classificação dos Problemas*, *DCP*. Essa didática baseia-se em dois critérios para resolver um problema de contagem. Primeiro, tentamos aplicar o Princípio Fundamental da Contagem, se não tivermos sucesso, vamos em busca de uma classificação para o problema (É arranjo? É permutação? Ou é combinação?) e utilizamos uma fórmula sem levar em conta o raciocínio combinatório que faz parte do problema.

Consequentemente, a Análise Combinatória e a Probabilidade são frequentemente vistos como temas baseados em regras arbitrárias, e não em princípios lógicos sólidos.

Mesmo tentando abordar o assunto Análise Combinatória em sala de aula utilizando o Princípio Fundamental da Contagem como principal ferramenta pois, temos ciência de que as tais fórmulas são obtidas a partir dele, os alunos ainda ficam com dúvidas e, geralmente, erram a resolução de um problema de Análise Combinatória, a transição desse princípio abstrato para a regra da divisão por um **k**! na Combinação, por exemplo, permanece obscura para muitos alunos. Esses erros vêm sempre acompanhados da mesma questão: "Como são os objetos que estamos contando?". Isso vem de encontro com o que Lockwood e Gibson (2016) consideram em seu artigo, a importância em enumerar o Conjunto de Resultados dos problemas de contagem, ou seja, listar os objetos que estão

sendo contados.

Mas, listar o Conjunto de Resultados pode ser trabalhoso, pode ocorrer que a quantidade de elementos desse conjunto seja muito grande. O que fazer? Uma maneira prática para solucionar esse impasse seria utilizar uma ferramenta computacional. Essa é a nossa proposta, utilizaremos a linguagem de programação em Python para listar os elementos do Conjunto de Resultados dos problemas que iremos resolver, ou seja, trabalharemos com o desenvolvimento de uma habilidade presente em (BRASIL, 2018, p. 539):

> **(EM13MAT405)** Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

Então, no presente trabalho propomos uma abordagem teórico-didática que utiliza a programação em Python como ferramenta de materialização do Conjunto de Resultados, utilizamos a recursão e laços aninhados em Python para modelar o Princípio Fundamental da Contagem, forçando o aluno a empregar restrições de código que explicitam a lógica da correção de significado (as restrições impostas pelo problema). Esta intervenção está fundamentada na triangulação teórica:

- A **Didática dos Princípios de Contagem (DPC)** que é utilizada em (CERIOLI; VIANA, 2012), a fim de apresentar uma abordagem analítica para a resolução de problemas de contagem.
- O **Modelo Cognitivo de Lockwood** desenvolvido na tese de doutorado (LOCKWOOD, 2011), que mostra o papel fundamental desempenhado pelo conjunto de resultados de um problema de contagem, papel esse percebido, principalmente, no trabalho (LOCKWOOD; GIBSON, 2016). A partir disso, utilizamos a enumeração gerada pelo código em Python para conectar o processo de contagem à fórmula.
- O **Modelo dos Campos Semânticos (MCS)** de Lins (2012), que fornece a base para analisar a produção de significado. O Modelo dos Campos Semânticos nos mostra se o aluno está apenas aplicando regras ou se ele realmente está consciente da maneira que está resolvendo o problema.

Assim, o nosso objetivo principal é buscar respostas para a seguinte pergunta: *De que maneira a materialização algorítmica do Princípio Fundamental da Contagem em Python,*

proposta sob o enfoque do Modelo dos Campos Semânticos, influencia a produção de significado e a justificação dos processos de ajuste de contagem na Análise Combinatória por estudantes do Ensino Médio?

Temos como objetivos específicos:

- Mapear o processo de transição dos estudantes entre as estratégias de contagem empírica (árvores de possibilidades) e a generalização matemática exigida pelo Princípio Fundamental da Contagem.
- Categorizar as resoluções e os equívocos dos alunos, identificando como mobilizam e conectam (ou desconectam) as Fórmulas/Expressões, os Processos de Contagem e o Conjunto de Resultados diante de cada tarefa.
- Investigar a produção de significado que emerge quando os estudantes realizam a leitura e a interpretação da sintaxe de programação em Python (especificamente a estrutura de laços de repetição *for* aninhados) como representação algorítmica do Princípio Fundamental da Contagem.
- Analisar o papel da mediação dialógica na negociação de significados, observando como as intervenções tensionam os resíduos do cotidiano dos alunos para guiá-los à estabilização do campo semântico da matemática formal.
- Compreender de que maneira o código computacional, atuando como um novo texto e artefato mediador, auxilia na visualização exaustiva dos agrupamentos e na superação de obstáculos clássicos do raciocínio combinatório.

A perspectiva de pesquisa adotada é de natureza qualitativa, adotando como estratégia metodológica o estudo de caso. O foco metodológico reside na análise interpretativa da enunciação dos estudantes. Através da comparação entre as justificativas iniciais e as justificativas posteriores (justificativas algorítmicas), buscaremos diagnosticar a produção de significado dos conceitos combinatórios, utilizando a linguagem do código Python como principal evidência da conscientização da lógica por trás do Princípio Fundamental da Contagem.

O presente trabalho está estruturado em cinco capítulos principais e dois anexos. Após esta Introdução, o Capítulo 2 revisa o referencial teórico, detalhando a articulação entre a Didática dos Princípio de Contagem da Análise Combinatória, o Modelo Cognitivo de Lockwood e o Modelo dos Campos Semânticos. O Capítulo 3 apresenta a Sequência

Didática proposta para o Ensino Médio, incluindo códigos em Python. O Capítulo 4 analisa a sequência didática e os resultados esperados sob a ótica da enunciação. Por fim, o Capítulo 5 apresenta as conclusões e sugestões para futuras investigações. O Apêndice A faz uma brevíssima introdução ao Python, trazendo o que necessitaremos da linguagem no desenvolvimento do trabalho. O Apêndice B trás as resoluções dos problemas que não puderam ser aplicados em sala de aula devido ao curto espaço de tempo.

2 Referencial Teórico

O presente capítulo estabelece a fundamentação teórica que sustenta a metodologia da pesquisa, justificando a interconexão de diferentes quadros conceituais para analisar a produção de significado na Análise Combinatória. Partindo da complexidade intrínseca do tema no Ensino Médio, este capítulo é organizado em uma triangulação teórica que aborda as dimensões didática, cognitiva e epistemológica do problema. Primeiramente, a Seção 2.1 apresenta a Análise Combinatória e seus princípios didáticos, utilizando o trabalho (CERIOLI; VIANA, 2012), para ancorar a proposta na necessidade de priorizar o Princípio Fundamental da Contagem como o alicerce fundamental do raciocínio. Em seguida, a Seção 2.2 introduz o Modelo Cognitivo elaborado por Lockwood (2011). Este referencial e (LOCKWOOD; CHENNE, 2020) justificam o uso da programação em Python como uma ferramenta essencial para a materialização da contagem, permitindo que o aluno visualize os Conjunto de Resultados e, assim, relacione o processo de contagem à fórmula. Por fim, a Seção 2.4 apresenta o Modelo dos Campos Semânticos (MCS) de Romulo C. Lins. Este será o arcabouço epistemológico primário, fornecendo a lente para analisar a qualidade da justificação (enunciação) do aluno e diagnosticar a produção de significado dos conceitos, como a correção do excesso de contagem. A articulação desses referenciais garante que a proposta didática não seja apenas uma inovação tecnológica, mas uma intervenção rigorosamente fundamentada na teoria da aprendizagem matemática.

2.1 A Análise Combinatória e a Didática dos Princípios de Contagem

Análise Combinatória, em parte, é a matemática da contagem. O que significa contar? Significa determinar o número exato de objetos especificados por uma pergunta "Quantos?" ou "De quantas maneiras...?". Uma definição mais geral para Análise Combinatória é dada por Cerioli e Viana (2012), para eles Análise Combinatória "é o ramo da

matemática que estuda as configurações construídas a partir de estruturas finitas", onde:

- "uma estrutura é um objeto matemático complexo, que consiste essencialmente dos seguintes itens: um ou mais conjuntos de objetos; uma ou mais operações sobre objetos destes conjuntos; uma ou mais relações entre objetos destes conjuntos."
- "uma estrutura é discreta se todos os conjuntos, operações e relações são finitos ou, no pior caso, podem ser dados como uma lista infinita de elementos. Uma estrutura é finita se todos os conjuntos, operações e relações envolvidas são finitos."
- "configurações são objetos finitos, obtidos a partir de estruturas finitas, por meio de um número finito de aplicações de certas regras".

Segundo Morgado *et al.* (2006), há dois tipos de problemas que ocorrem frequentemente em Análise Combinatória:

1. "Demonstrar a existência de subconjuntos de elementos de um conjunto finito dado e que satisfazem certas condições.
2. Contar ou classificar os subconjuntos de um conjunto finito e que satisfazem certas condições dadas." (MORGADO *et al.*, 2006)

O ensino de Análise Combinatória é necessário na Educação Básica porque trata noções matemáticas fundamentais e trabalha processos ou habilidades cognitivas essenciais para o desenvolvimento intelectual: números, conjuntos, contagem, compreensão de texto, organização das ideias, raciocínio lógico. Além disso, a Análise Combinatória tem inúmeras aplicações em teoria da probabilidade, teoria dos jogos, ciência da computação, gestão, genética, estatística. Conforme menciona a pesquisadora Lyn D. English:

> Problemas combinatórios também facilitam o desenvolvimento de processos de enumeração, bem como conjecturas, generalizações e pensamento sistemático. Por exemplo, para determinar todas as roupas possíveis a partir de um conjunto de camisas e calças de cores diferentes, é necessário combinar sistematicamente uma camisa colorida com cada par de calças e, em seguida, repetir o processo com cada uma das camisas coloridas restantes. Este é um procedimento mais eficiente do que combinar camisas com calças aleatoriamente. O desenvolvimento dos importantes conceitos de relações, classes de equivalência, mapeamento e funções também é promovido por meio de atividades combinatórias. Além disso, dada a ampla aplicabilidade do domínio combinatório (por exemplo, química, biologia, física), problemas interdisciplinares podem ser criados em contextos do mundo real para os alunos. (ENGLISH, 2005)

Análise Combinatória, na Educação Básica, é a matemática da contagem, como é mencionada em (BRASIL, 2018). E, como a Análise Combinatória aparece no currículo da Educação Básica?

No Ensino Fundamental I, a Base Nacional Curricular Comum menciona como objeto de conhecimento para o 1º ano do Fundamental os Processos de Contagem e para o 4º ano do Fundamental os Problemas de Contagem, sendo que nessa etapa do ensino já devem desenvolver a seguinte habilidade:

> **(EF04MA08)** Resolver, com o suporte de imagem e/ou material manipulável, problemas simples de contagem, como a determinação do número de agrupamentos possíveis ao se combinar cada elemento de uma coleção com todos os elementos de outra, utilizando estratégias e formas de registro pessoais. (BRASIL, 2018, p. 291)

No Ensino Fundamental II, especificamente no 8º ano, a Base Nacional Curricular Comum menciona o Princípio Multiplicativo ou Princípio Fundamental da Contagem, e nessa etapa do ensino os alunos devem desenvolver a seguinte habilidade:

> **(EF08MA03)** Resolver e elaborar problemas de contagem cuja resolução envolva a aplicação do princípio multiplicativo. (BRASIL, 2018, p. 313)

E, no Ensino Médio, a Análise Combinatória é estudada no 2º Ano ou no 3º Ano, de acordo com o currículo de cada estado. Segundo a Base Nacional Curricular Comum, nessa etapa do Ensino a seguinte habilidade deve ser desenvolvida:

> **(EM13MAT310)** Resolver e elaborar problemas de contagem envolvendo agrupamentos ordenáveis ou não de elementos, por meio dos princípios multiplicativo e aditivo, recorrendo a estratégias diversas, como o diagrama de árvore. (BRASIL, 2018, p. 537)

Mas, "Contar é dificil" conforme mencionado em (MARTIN, 2001) e, vemos isso na prática, em sala de aula. Professores têm dificuldades em ensinar Análise Combinatória e os alunos têm dificuldades em compreendê-la. Isso pode estar relacionado à forma com que tradicionalmente ela é abordada, baseada em definições e na aplicação de fórmulas.

Segundo (CERIOLI; VIANA, 2012), no ensino do Análise Combinatória adotamos a *Didática da Classificação dos Problemas*, *DCP*.

> Esta didática é baseada em uma metodologia que consiste, essencialmente, em resolver os problemas de contagem aplicando o seguinte critério:

1. Tentar aplicar o Princípio Multiplicativo, também chamado de Princípio Fundamental da Contagem;
   Se a tentativa for infrutífera, então:
2. Classificar o problema em problema de arranjo, problema de permutação ou problema de combinação;
3. Em seguida, aplicar a fórmula correspondente para resolver o problema. (CERIOLI; VIANA, 2012)

A Didática da Classificação dos Problemas está presente na maioria dos livros didáticos do Ensino Médio onde a Análise Combinatória é abordada. E, na maioria das vezes, essa metodologia é aplicada de forma indiscriminada, as aplicações das fórmulas são feitas sem levar em conta o raciocínio combinatório que faz parte do problema. Com isso, podemos afirmar que o obstáculo na resolução de um problema de Contagem reside na tendência dos estudantes de aplicar fórmulas de forma mecânica (Permutação, Arranjo, Combinação) sem discernir as características dos objetos que estão sendo contados, sem entender e descrever como eles são formados.

Consequentemente, a Análise Combinatória e a Probabilidade são frequentemente vistos como temas baseados em regras arbitrárias, e não em princípios lógicos sólidos.

Com o propósito de apresentar uma abordagem analítica para a resolução de problemas de contagem, Cerioli e Viana (2012) apresentam o que chamam de Didática dos Princípios de Contagem (DPC).

A DPC está fundamentada no Método dos Princípios de Contagem, que consiste dos seguintes passos, segundo Cerioli e Viana (2012):

1. "Ler e reler atentamente o enunciado do problema.
2. Identificar os objetos básicos e as configurações em questão.
3. Introduzir uma notação matemática adequada para descrever de maneira precisa as configurações.
4. Compreender como tais configurações podem ser formadas a partir dos objetos básicos.
5. Obter uma especificação de como as configurações podem ser formadas a partir dos objetos básicos, de maneira que:
   (a) todas as configurações em questão podem ser formadas da maneira descrita;
   (b) somente as configurações em questão podem ser assim formadas;

(c) cada configuração pode ser formada de uma única maneira, a partir dos objetos básicos, de acordo com esta especificação.

6. Aplicar os princípios básicos de contagem de acordo com a especificação obtida, para obter o número de configurações possíveis."

Os princípios básicos de contagem, mencionados no item de número 6 acima são os seguintes, considerando que $n(A)$ representa o número de elementos de um conjunto A:

- Princípio da Bijeção: permite contar o número de elementos de um conjunto A de maneira indireta, através de um conjunto B do qual sabemos: determinar o número de elementos; e que tem o mesmo número de elementos de A. Logo, $n(A) = n(B)$.

- Princípio Aditivo: permite contar o número de elementos de um conjunto A de maneira indireta, através da "classificação" de seus elementos em vários subconjuntos $A_1, A_2, \cdots A_n$, de modo que: sabemos determinar o número de elementos de A_1 , o número de elementos de A_2 , $\cdots$, o número de elementos de A_n ; quaisquer dois subconjuntos dentre $A_1, A_2, \cdots, A_n$ não possuem elementos em comum; e todos os elementos de A estão distribuídos entre os subconjuntos $A_1, A_2, \cdots, A_n$. Logo, $n(A) = n(A_1) + n(A_2) + \cdots + n(A_n)$.

- Princípio Multiplicativo (Princípio Fundamental da Contagem): permite contar o número de elementos de um conjunto A quando reconhecemos que cada elemento de A, e somente estes, pode ser formado após a tomada em sequência de k decisões, $d_1, d_2, d_3, \cdots, d_k$, de maneira que: a decisão d_1 pode ser tomada de m_1 maneiras; para cada maneira como a decisão d_1 pode ser tomada, a decisão d_2 pode ser tomada de m_2 maneiras; para cada maneira como as decisões d_1 e d_2 podem ser tomadas (nesta ordem), a decisão d_3 pode ser tomada de m_3 maneiras; $\cdots$; para cada maneira como as decisões $d_1, d_2, \cdots, d_{i-1}$ podem ser tomadas (nesta ordem), a decisão d_i pode ser tomada de m_i maneiras; $\cdots$; para cada maneira como as decisões $d_1, d_2, \cdots, d_{i-1}, d_i, \cdots, d_{k-1}$ podem ser tomadas (nesta ordem), a decisão d_k pode ser tomada de m_k maneiras; então, $n(A) = m_1 \times m_2 \times m_3 \cdots \times m_i \times \cdots \times m_{k-1} \times m_k$.

- Princípio k para 1: permite contar o número de elementos de um conjunto A de maneira indireta, através de um conjunto B, do qual sabemos: determinar o número de elementos; que o número de elementos de B é um divisor (ou um múltiplo) do número de elementos de A; determinar o fator (ou divisor) k que relaciona o número

de elementos de A com o número de elementos de B. Assim, $n(A) = k \times n(B)$ ou $n(A) = \frac{n(B)}{k}$

- Princípio da Inclusão-Exclusão: permite contar o número de elementos de um conjunto A de maneira indireta, através da "separação" de seus elementos em vários subconjuntos $A_1, A_2, \cdots A_n$, de modo que: sabemos determinar o número de elementos de A_1 , o número de elementos de A_2 , $\cdots$, o número de elementos de A_n ; para quaisquer k subconjuntos $A_{i1}, A_{i2}, \cdots A_{ik}$ dentre $A_1, A_2, \cdots, A_n$, sabemos determinar quantos elementos $A_{i1}, A_{i2}, \cdots, A_{ik}$ possuem em comum; e $A_1, A_2, \cdots, A_n$ "exaurem" todos os elementos de A. (CERIOLI; VIANA, 2012)

No nosso trabalho, procuraremos resolver os problemas de contagem utilizando o Princípio Fundamental da Contagem e junto deste, quando houver necessidade, o Princípio Aditivo e o Princípio k para 1.

No trabalho de (CERIOLI; VIANA, 2012) é enfatizado que: "Representar as configurações de maneira precisa, a partir da especificação dos elementos que a formam e das operações e relações que podem ser aplicadas sobre estes elementos para formá-las, é um passo extremamente importante no processo de contagem.", ou seja, o quinto passo do Método dos Princípios de Contagem é o mais importante. E, por experiência em sala de aula(nos esforçamos em utilizar a DPC quando estamos tratando do assunto em sala), é nesse passo que está o grande nó, nele que os erros, em grande parte, ocorrem, quando da utilização da DPC. Na maioria esmagadora das vezes, os alunos encontram um número maior de elementos daquilo que deveriam encontrar, ou se equivocam na resposta, pois não sabem o que estão contando. Nesse quinto passo gastamos um pouco mais de tempo, pois, geralmente, temos que listar vários elementos da configuração procurada, para auxiliar no entendimento do que estamos contando de mais ou de menos. No entanto, conseguimos, e muito, minimizar os equívocos na resolução dos problemas de contagem, e, vemos, também, quando necessário, os Princípios Aditivo e k para 1 entrarem em ação.

Devido a essas dificuldades, Batanero, Navarro-Pelayo e Godino (1997) observam a necessidade de aprimoramento nessa área e afirmam o seguinte:

> [...]A Análise Combinatória é uma área que a maioria dos alunos considera muito difícil. Dois passos fundamentais para facilitar a aprendizagem desse assunto são compreender a natureza dos erros dos alunos na resolução de problemas combinatórios e identificar as variáveis que podem influenciar essa dificuldade.

Este apelo de Batanero, Navarro-Pelayo e Godino (1997) reconhece as dificuldades acima e também destaca a necessidade de uma análise mais aprofundada dos erros dos alunos. Mas, além de (e talvez antes de) examinar os erros dos alunos, é necessário que compreendamos melhor as formas de pensar que os alunos trazem para resolver problemas de Contagem em primeiro lugar - entender os mecanismos de raciocínio dos alunos.

Segundo Lockwood (2011), para ajudar os alunos a ter sucesso em Análise Combinatória, precisamos de uma compreensão mais profunda do pensamento dos alunos sobre a atividade matemática de resolver problemas de Contagem. Para isso, realizou uma pesquisa com estudantes do ensino superior e buscou aprender mais sobre suas maneiras de pensar quando ao resolver problemas de Contagem, ou seja, observar como estudantes organizam mentalmente os resultados a serem contados e como isso se relaciona a expressões e processos de contagem.

2.2 O Modelo de Lockwood

A pesquisa de Lockwood é estruturada no que ela chama de *set-oriented thinking* (pensamento orientado em conjunto), isto é, os estudantes constroem e manipulam explicitamente o conjunto de resultados possíveis como base para produzir expressões e fórmulas de contagem. A pesquisadora construiu um modelo, Figura 1, que representa uma análise conceitual do pensamento e da atividade dos alunos quando submetidos a um problema de Contagem.

Em (LOCKWOOD, 2011) é explicado as principais relações entre os componentes desse modelo e o que tais componentes representam, vejamos a seguir:

- "**Fórmulas/Expressões** referem-se a expressões matemáticas que geram um valor numérico. A fórmula pode ter algum significado combinatório, por exemplo $C_{5,3}$(combinação simples de cinco objetos tomados três a três), ou pode ser alguma combinação de operações numéricas, por exemplo $1 \times 9 + 4 \times 8$.

- **Processos de Contagem** referem-se ao processo de enumeração (ou série de processos) no qual o aluno se envolve (mental ou fisicamente) ao resolver um problema de Contagem. Por exemplo, a implementação de uma divisão em casos ou aplicações sucessivas do Princípio Fundamental da Contagem poderiam representar processos de contagem que o aluno pode executar ao resolver um problema.

- **Conjuntos de Resultados** referem-se aos objetos que estão sendo contados - aque-

les conjuntos de elementos que o estudante pode imaginar sendo gerados ou enumerados por um processo de contagem. A quantidade de elementos do Conjunto de Resultados determina a resposta para o problema de Contagem."

Figura 1: Modelo do pensamento combinatório dos alunos.
Fonte: Adaptado de Lockwood (2011).

Com o objetivo de explicar as relações existentes entre os componentes do Modelo de Lockwood, consideremos o seguinte problema: "Uma moeda é lançada três vezes. Qual o número de sequências possíveis *cara* e *coroa*?".

- **Processos de Contagem ⇔ Fórmulas/Expressões.** Por exemplo, um processo de contagem para resolver o problema acima seria considerar iterativamente as opções para cada lançamento da moeda e utilizar a multiplicação. Existem duas opções para o primeiro lançamento; escolhido a face do primeiro lançamento, existem duas opções para o segundo lançamento e, analogamente, escolhidos as faces dos dois primeiros lançamentos, existem duas opções para o terceiro lançamento. Como o número de opções é independente em cada etapa, podemos multiplicar o número de opções de cada etapa. Esse processo de considerar opções para cada posição resulta numa expressão específica: $2 \times 2 \times 2$. Da mesma forma, dada uma expressão como $2 \times 2 \times 2$, podemos interpretá-la como o reflexo de um processo de contagem subjacente.

- **Conjuntos de Resultados ⇔ Fórmulas/Expressões.** Essa relação só começou a ser melhor entendida no trabalho (LOCKWOOD; SWINYARD; CAUGHMAN, 2015).

> (...) os estudantes neste estudo fornecem uma prova de existência de que, dadas as circunstâncias apropriadas, a relação entre conjuntos de resultados e fórmulas/expressões pode desenvolver-se nos estudantes como um aspecto natural do raciocínio sobre contagem. (...)Olhar para e argumentar sobre os resultados foi um aspecto fundamental no processo de decisão sobre quais fórmulas poderiam ser apropriadas, e isso lhes permitiu evitar respostas sem sentido e afastou-os da tentação de simplesmente aplicar as fórmulas cegamente.
>
> (LOCKWOOD; SWINYARD; CAUGHMAN, 2015)

- **Conjuntos de Resultados ⇔ Processos de Contagem.** Mantendo o exemplo do lançamento da moeda três vezes, o processo descrito na primeira relação produz uma listagem específica do conjunto de resultados. Ou seja, ao considerar inicialmente que a face do primeiro lançamento pode ser *cara* ou *cara*, e, em seguida, notar que para cada uma dessas escolhas a face do segundo lançamento pode ser *cara* ou *cara*, e assim por diante, o conjunto de resultados pode ser gerado. O diagrama de árvore na Figura 2 mostra, de forma mais clara, a geração desse conjunto de resultados.

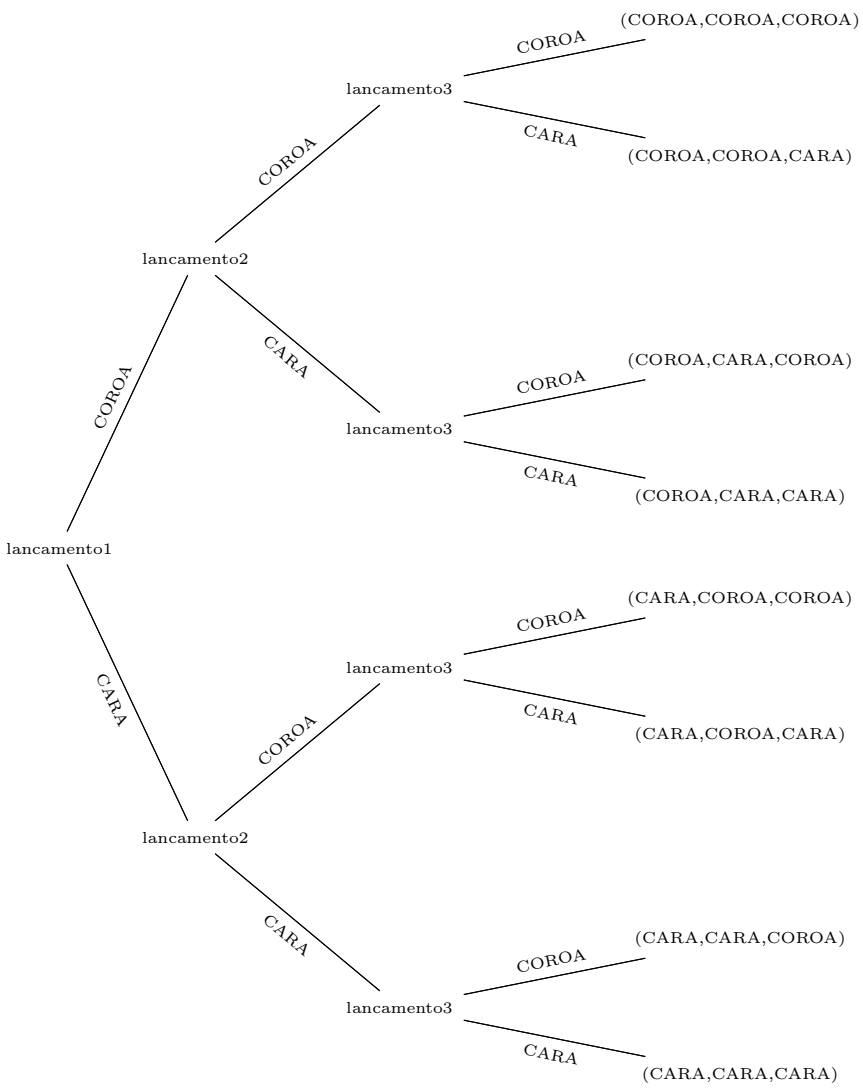


Figura 2: Árvore de Possibilidades para os 3 lançamentos da moeda.
Fonte: Autoria própria(2026).

Agora, dado o conjunto de resultados do problema, como podemos obter um processo de contagem?

Podemos tomar os elementos desse conjunto e organizá-los em um padrão análogo à ordem alfabética de um dicionário. Teremos, então:

$$\{(cara,cara,cara),(cara,cara,coroa),(cara,coroa,cara),\ldots,(coroa,coroa,coroa)\}.$$

Nesse caso, estamos utilizando uma estratégia de fixação de variáveis para garantir que todos os casos sejam cobertos sem repetições, criando, assim, um processo de contagem.

Lockwood (2011) afirma que geralmente, os alunos não utilizam os Conjuntos de Re-

sultados enquanto resolvem um problema de contagem, mas, em vez disso, frequentemente se baseiam em palavras-chave ou situações memorizadas para determinar se a ordem importa ou não, para decidir qual fórmula aplicar. Em (LOCKWOOD; GIBSON, 2016), vemos com clareza o quão importante é o componente Conjunto de Resultados:

> [...]Alunos de todos os níveis têm dificuldade em resolver corretamente problemas de contagem, e descobrimos um fator específico que parece pelo menos estar positivamente correlacionado com o sucesso na resolução de problemas de Contagem. Nosso estudo nos motiva a analisar mais de perto como o listar os resultados pode se relacionar com o pensamento e a aprendizagem dos alunos sobre problemas de Contagem e também a desenvolver e estudar instruções que possam fomentar essa atividade produtiva de listar.

Analisando as relações entre os componentes do Modelo de Lockwood, vemos que um dos princípios básicos de contagem, utilizado na Metodologia da Didática dos Princípios de Contagem (CERIOLI; VIANA, 2012), surge na justicativa dessa relações. O Princípio Fundamental da Contagem(PFC). Por isso, entendemos o quanto é importante o bom entendimento desse Princípio, conforme menciona Duro (2012):

> O PFC é o elemento fundamental do pensamento combinatório, pois é a partir dele que todas as construções cognitivas posteriores (permutações, arranjos e combinações) se constituirão para o sujeito. A não compreensão do PFC poderá gerar grandes dificuldades no reconhecimento de possíveis aplicações no processo de resolução de problemas.

Em seu trabalho de mestrado, Sturm (1999) descreve uma proposta de utilização do Princípio Fundamental da Contagem em problemas que envolvem o ensino e a aprendizagem de análise combinatória, em detrimento do emprego imediato de fórmulas. Para o autor, os resultados apresentados mostram aspectos positivos em relação ao uso do Princípio Fundamental da Contagem na resolução de problemas em detrimento do uso imediato e sem significação das fórmulas.

Dutra (2014), em seu trabalho de mestrado, também menciona a importância do Princípio Fundamental da Contagem na resolução de problemas de Contagem:

> [...] julgamos que utilizar a resolução de problemas como modalidade didática para ensinar Análise Combinatória, é um dos caminhos que apresenta significado e contextualização dos conceitos abordados no campo da Análise Combinatória. Além disso, o uso predominante do Princípio Fundadmental da Contagem nestas resoluções torna o uso das fórmulas de Arranjo, Combinação

> e Permutação uma ferramenta que favorece a mobilização e aquisição de conhecimentos significativos neste domínio. Com efeito, o aluno pode participar da construção das fórmulas e perceber que o uso das mesmas não é um conhecimento mecânico, e sim, requer do sujeito desenvolver competências necessárias para alcançar os resultados esperados a partir dos cálculos, passando pela interpretação dos problemas considerando, se necessário, diversas representações (por tabelas, enumeração, árvore de possibilidades, etc.) favorecendo, por conseguinte, a construção do raciocínio combinatório.

Listar o Conjunto de Resultados fortalece a compreensão da natureza do conjunto formado. Mas, listar tal conjunto pode ser trabalhoso, pode ocorrer que a quantidade de elementos desse conjunto seja muito grande. O que fazer? Resolver uma versão semelhante, mas menor, de um problema é uma importante técnica de resolução de problemas mas, às vezes, pode ser difícil. Então, uma maneira prática para solucionar esse impasse seria utilizar uma ferramenta computacional capaz de fornecer esse conjunto, independente de seu tamanho. Essa é a nossa proposta, utilizaremos a linguagem de programação Python para listar os elementos do Conjunto de Resultados dos problemas que iremos resolver, ou seja, trabalharemos com o desenvolvimento de uma habilidade presente em (BRASIL, 2018, p. 539):

> **(EM13MAT405)** Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

Nessa habilidade encontramos termos(programação, algoritmos) relacionados ao tema *Pensamento Computacional*.

2.3 Pensamento Computacional

Em seu trabalho de mestrado, a pesquisadora Pacheco (2024) comenta em que consiste o termo pensamento computacional e menciona que tal termo surgiu a partir de pesquisas na área educacional:

> Fragmentar um problema difícil, reconhecer padrões, fazer abstrações e criar algoritmos são habilidades utilizadas por cientistas da computação. Com isso, é possível identificar, orientar e contribuir para o desenvolvimento de soluções dos desafios. Juntas, essas habilidades compõem um método para resolver problemas chamado de pensamento computacional. Esse termo ganhou destaque por meio da pesquisadora Jeannette Wing (WING, 2006)[...]

O termo pensamento computacional, mesmo não aparecendo nas competências e nem nas habilidades da Matemática elencadas na Base Nacional Curricular Comum (BRASIL, 2018), é mencionado várias vezes ao longo das discussões propostas no documento sempre nas seções de Matemática desde os Anos Finais do Ensino Fundamental até o Ensino Médio, sendo o seu desenvolvimento uma consequência do ensino de Matemática, comenta o professor Leonardo Barichello em (BARICHELLO, 2023).

Em 2022 foi homologado o texto complementar "BNCC Computação" (BRASIL, 2022) onde são expostos três eixos: pensamento computacional, mundo digital e cultura digital. Com a obrigatoriedade de ser incluído nas escolas a partir de outubro de 2023, nesse texto o pensamento computacional é incluído nas três etapas do ensino básico, descrevendo em cada etapa quais são os objetivos de aprendizagem.

Conforme mencionado em Dias (2024), no inciso I do Art. 3º da Lei nº 14533, de 11 de janeiro de 2023, é apresentada uma descrição do que a legislação entende como pensamento computacional:

> I - pensamento computacional, que se refere à capacidade de compreender, analisar, definir, modelar, resolver, comparar e automatizar problemas e suas soluções de forma metódica e sistemática, por meio do desenvolvimento da capacidade de criar e adaptar algoritmos, com aplicação de fundamentos da computação para alavancar e aprimorar a aprendizagem e o pensamento criativo e crítico nas diversas áreas do conhecimento (BRASIL, 2023).

A primeira aparição do termo "pensamento computacional" surgiu no artigo Computational Thinking publicado na revista Communications of the ACM. Neste artigo, Wing define que o pensamento computacional "envolve a resolução de problemas, projeção de sistemas, e compreensão do comportamento humano, através da extração de conceitos fundamentais da ciência da computação" (WING, 2006). A pesquisadora Pacheco (2024) cita que Wing defende que o pensamento computacional pode ser pensado para além da ciência da computação e que não há uma definição única sobre o que é.

No artigo Computational thinking's influence on research and education for all publicado na revista Italian Journal of Educational Technology, Wing trás a seguinte definição para o termo "pensamento computacional:

> Uso o termo "pensamento computacional" como abreviação de "pensar como um cientista da computação" (WING, 2006). Para ser mais descritiva, no entanto, defino agora o pensamento computacional[...] da seguinte forma:
>
> *O pensamento computacional são os processos de pensamento envolvidos na formulação de um problema e na expressão de sua(s) solução(ões) de forma*

> *que um computador — humano ou máquina — possa executá-la com eficácia.* (WING, 2017).

Segundo Wing (2017), "o pensamento computacional descreve a atividade mental de formular um problema para admitir uma solução computacional. A solução pode ser executada por um humano ou por uma máquina. Este último ponto é importante. Primeiro, os humanos computam. Segundo, as pessoas podem aprender o pensamento computacional sem uma máquina. Terceiro, os "computadores" de hoje combinam a inteligência humana e a inteligência das máquinas. Além disso, o pensamento computacional não se trata apenas de resolução de problemas, mas também de formulação de problemas."

Segundo Brackmann (2017), algumas definições a respeito do pensamento computacional destacam a presença de quatro pilares principais: decomposição, reconhecimento de padrões, abstração e algoritmos.

> O Pensamento Computacional envolve identificar um problema complexo e quebrá-lo em pedaços menores e mais fáceis de gerenciar (DECOMPOSIÇÃO). Cada um desses problemas menores pode ser analisado individualmente com maior profundidade, identificando problemas parecidos que já foram solucionados anteriormente (RECONHECIMENTO DE PADRÕES), focando apenas nos detalhes que são importantes, enquanto informações irrelevantes são ignoradas (ABSTRAÇÃO). Por último, passos ou regras simples podem ser criados para resolver cada um dos subproblemas encontrados (ALGORITMOS). Seguindo os passos ou regras utilizadas para criar um código, é possível também ser compreendido por sistemas computacionais e, consequentemente, utilizado na resolução de problemas complexos eficientemente, independentemente da carreira profissional que o estudante deseja seguir. (BRACKMANN, 2017).

Com a finalidade de conhecer um pouco mais sobre o tema Pensamento Computacional como ferramenta na resolução de problemas matemáticos indicamos os trabalhos desenvolvidos no PROFMAT: (PACHECO, 2024), (LIRA, 2024), (COQUI, 2025), (BLACK, 2024), (CORRÊA, 2022), (RECH, 2024) e (BORGES, 2021). Vale mencionar, também, o volume sobre Pensamento Computacional da Coleção Livro Aberto de Matemática (BARICHELLO, 2023).

No nosso trabalho, a programação em Python estará promovendo a representação concreta por código da lógica de contagem. O código Python atua como o Processo de Contagem em execução. A saída da execução (a lista de agrupamentos) atua como o Conjunto de Resultados (LOCKWOOD; GIBSON, 2016). Essa materialização do conjunto de resultados permite que o aluno não apenas chegue ao número final, mas também

visualize o que o Princípio Fundamental da Contagem está contando em excesso e utilize o comando de código (a restrição) para corrigir esse excesso, unindo Processo, Fórmula e Resultado em uma única atividade. Ou seja, a visualização concreta dos conjuntos gerados pelo Python (em vez de apenas o número final) ajuda o aluno a tematizar (tornar consciente) o significado de cada operação combinatória, facilitando a resolução de problemas de contagem futuros. Com isso, estamos mobilizando os quatro pilares do pensamento computacional:

- **Decomposição:** É a essência do Princípio Fundamental da Contagem. Um problema complexo de contagem é quebrado em uma sequência de n etapas ou decisões mais simples. Se o problema envolve formar senhas, por exemplo, a decomposição exige olhar para a escolha de cada caractere individualmente.
- **Reconhecimento de Padrões:** Por exemplo, o estudante reconhece que a lógica multiplicativa usada para contar as possíveis senhas alfanuméricas é estruturalmente idêntica à lógica usada para mapear todos os resultados do lançamento simultâneo de várias moedas e dados. O padrão não reside no objeto físico (a senha ou o dado), mas na natureza da tomada de decisão sequencial. Sob a ótica do pensamento computacional, esse reconhecimento ocorre quando o aluno percebe que a mesma arquitetura de laços de repetição aninhados (*for* dentro de *for*) pode ser reaproveitada para gerar ambas as coleções de agrupamentos.
- **Abstração:** Envolve ignorar características irrelevantes dos objetos (como a cor de uma camisa, focando apenas em quantas são) para modelar o problema matematicamente. No contexto computacional, isso se traduz na definição de variáveis e tipos de dados.
- **Algoritmos:** É a construção do passo a passo. Na contagem, o algoritmo descreve a ação de listar ou gerar sistematicamente cada elemento do Conjunto de Resultados em vez de simplesmente calcular o total.

É importante ressaltar que no trabalho (LOCKWOOD; CHENNE, 2020), os autores também utilizaram a programação em Python para reforçar a compreensão de quatro tipos de problemas combinatórios, utilizando para isso, o comando for e declarações condicionais (if).

Nesse trabalho, o Modelo dos Campos Semânticos de Romulo C. Lins((LINS, 1994), (LINS, 1999), (LINS, 2012)) será a principal lente para analisar os dados coletados(as

enunciações) na resolução de um problema de contagem, permitindo ir além da simples observação de acerto e erro na resolução de tal.

2.4 O Modelo dos Campos Semânticos(MCS)

> (...)o MCS só existe em ação. Ele não é uma teoria para ser estudada, é uma teorização para ser usada. (LINS, 2012)

O Modelo dos Campos Semânticos foi desenvolvido pelo Professor Dr. Romulo Campos Lins com o objetivo de sanar uma de suas inquietações, em particular, "(...) dar conta de caracterizar o que os alunos estavam pensando quando "erravam", mas sem recorrer a esta ideia do erro." (LINS, 2012).

O Modelo dos Campos Semânticos foi apresentado por Lins, em 1992, a partir de sua pesquisa de doutorado intitulada "A framework for understanding what algebraic thinking is"(Um quadro de referência para entender o que é pensamento algébrico), a qual foi desenvolvida na Universidade de Nottingham na Inglaterra. Em 1993, na Revista de Educação Matemática (SBEM – São Paulo), Lins publicou o artigo "Epistemologia, história e educação matemática: tornando mais sólidas as bases da pesquisa", apresentando alguns elementos do Modelo dos Campos Semânticos, neste que é o primeiro artigo em que o Modelo é apresentado. Nesse artigo, Lins considera o que, para ele, é Epistemologia:

> "Epistemologia é a atividade humana que estuda as seguintes questões (i) o que é conhecimento?; (ii) como é que conhecimento é produzido?; e, (iii) como é que conhecemos o que conhecemos?" (LINS, 1993)

Partindo dessa definição, Lins procurou responder, nos seus estudo iniciais, a essas perguntas durante a investigação sobre o Modelo dos Campos Semânticos: *O que é conhecimento? O que é significado?*.

O primeiro artigo em que o Modelo dos Campos Semânticos é, de fato, enfatizado, foi publicado em 1994, na revista Dynamis (Blumenau, v.1, n.7), "O Modelo Teórico dos Campos Semânticos: uma análise epistemológica da Álgebra e do pensamento algébrico".

> "O MCS é um modelo epistemológico que propõe que *conhecimento* é uma *crença-afirmação* junto com uma *justificação* para a *crença-afirmação*. Indicamos, desta forma, que *conhecimento* é algo do domínio da enunciação - e que, portanto, todo *conhecimento* tem um *sujeito* - e não do domínio do *enunciado*; podemos também expressar este fato dizendo que *conhecimento* é do domínio

> da *fala*, e não do *texto*. Desde este ponto de vista, a Matemática é um *texto*, e não *conhecimento*; tem-se *conhecimento* apenas na medida em que pessoas se dispõem a *enunciar* este *texto*. A um *conhecimento* que fala deste texto - a Matemática - chamaremos, naturalmente, de *conhecimento matemático*." (LINS, 1994)

Para compreender o Modelo dos Campos Semânticos, Lins apresenta, em (LINS, 2012), algumas noções centrais:

- *Conhecimento*: "consiste em uma crença-afirmação (o sujeito enuncia algo em que acredita) junto com uma justificação (aquilo que o sujeito entende como lhe autorizando a dizer o que diz)." (LINS, 2012)
- *Crença*: algo em que o sujeito acredita e age de maneira coerente com o que diz.
- *Justificação*: o sujeito acredita em algo e o expressa em forma de *afirmação*, seguida de uma *justificação*. "Não é justificativa. Não é explicação para o que digo. Não é algum tipo de conexão lógica com coisas sabidas. É apenas o que o sujeito do conhecimento (aquele que o produz, o enuncia) acredita que o autoriza a dizer o que diz." (LINS, 2012)

> "Parte essencial do Modelo dos Campos Semânticos é que conhecimento é entendido como uma *crença* - algo em que o sujeito acredita e expressa, e que caracteriza-se, portanto, como uma *afirmação* - junto com o que o sujeito considera ser uma *justificação* para sua *crença-afirmação*." (LINS, 1993)

- *Autor-texto-leitor*: "quem produz uma enunciação é o *autor*. O autor fala sempre na direção de um *leitor*, que é constituído (produzido, instaurado, instalado, introduzido) pelo o autor. Quem produz significado para um resíduo de enunciação é o leitor. O leitor sempre fala na direção de um autor, que é constituído (produzido, instaurado, instalado, introduzido) pelo o leitor". (LINS, 2012)
- *Interlocutor*: Lins (2012) afirma que o interlocutor não é um ser biológico, e sim um ser cognitivo que não deve ser confundido com um sujeito com quem se dialoga, mas com quem se troca ideias. Com isso, "toda produção de conhecimento é feita na direção de um interlocutor que, acredito, produziria a mesma enunciação com a mesma justificação." (LINS, 1999)
- *Legitimidade/verdade*: o termo legitimidade, refere-se ao que o sujeito julga ser ou não verdadeiro dizer, mesmo quando o que é afirmado não seja uma verdade universal.

> "Como consequência de ser enunciado na direção de um interlocutor, e de ter mesmo sido produzido, todo conhecimento é verdadeiro. Isto não quer dizer que aquilo que é afirmado seja "verdade". " (LINS, 2012)

- *Resíduo de enunciação*: enunciação e resíduo de enunciação são coisas distintas no Modelo dos Campos Semânticos. A enunciação ocorre quando há produção de conhecimento e produção de significado – "[...] a enunciação é feita na direção de um interlocutor" (LINS, 2012). O resíduo de enunciação, por sua vez, para Lins (2012), é algo com o que o aluno se depara, a partir do que foi falado por alguém ou visto num livro. O pesquisador ainda ressalta que um resíduo de enunciação não é nem menos e nem mais relevante que uma enunciação, ele é de outra ordem.

- *Leitura plausível/Leitura positiva*: Lins (2012), menciona que a leitura é "plausível porque faz, sentido", é aceitável neste contexto "parece ser que é assim", e é positiva porque é o oposto de uma "leitura pela falta. E, continua, "(...) a leitura plausível se aplica de modo geral aos processos de produção de conhecimento e significado; ela indica um processo no qual o todo do que eu acredito que foi dito faz sentido.(...) A leitura positiva tem por objetivo, por assim dizer, mapear o terreno ao mesmo tempo que trata de saber onde o outro está."(LINS, 2012)

- *Significado/objeto*: Para Lins (1994), "significado é a relação que se estabelece entre uma crença-afirmação e uma justificação para ela no momento da enunciação."

> "Significado de um objeto é aquilo que efetivamente se diz a respeito de um objeto, no interior de uma atividade. Objeto é aquilo para que se produz significado. Sempre que há produção de significado há produção de conhecimento e vice-versa, mas conhecimento e significado são coisas de naturezas distintas." (LINS, 2012)

- *Sujeito biológico/Sujeito cognitivo*: Para Lins (2012), "O sujeito biológico é o outro. Não é na direção de um outro que o sujeito cognitivo fala, mesmo que um sujeito biológico esteja a sua frente. Falamos sempre na direção de um sujeito cognitivo, um interlocutor."

- *Campo Semântico*: Segundo Lins, em (LINS, 1993), "Campo Semântico é uma coleção de conhecimentos cujas justificações estão todas relacionadas a um mesmo *núcleo* ou todas são produzidas a partir de um mesmo conjunto de princípios." Ou melhor, "é um modo de criar significados.", como vemos em (LINS, 1994).

- *Núcleo*: O núcleo é constituído de estipulações locais, que, conforme (LINS, 2012), "[...]são, localmente, verdades absolutas, que não requerem, localmente, justificação". O mesmo autor, em (LINS, 1999), considera que, "Dentro de uma atividade, núcleos podem ser mais, ou menos, estáveis (permanentes/mutáveis), e mais, ou menos, consistentes. O que certamente eles não são: dados a priori. Não concebo um núcleo como algo que fica guardado em algum canto de minha cabeça, um pacote que utilizo quando preciso."

Além dos principais elementos do Modelo dos Campos Semânticos – apontados em (LINS, 2012) e listados acima, apresentamos alguns processos que podem ocorrer durante a produção de significados.

Quando um sujeito diz alguma coisa e um segundo sujeito produz o seguinte significado para aquilo que foi dito pelo primeiro: "isso não pode ser dito", nos encontramos diante do processo de *estranhamento*.

> "Na Matemática do matemático $(-1) \times (-1) = 1$, e assim também na da escola, mas na rua isto não é nada, a não ser um rabisco num papel ou numa lousa, um vestígio, a pegada de um monstro que se deixou escapar." (LINS, 2004)

Lins (2004) afirma, assim, sobre o processo de estranhamento: "(...) de um lado aquele para quem uma coisa é natural - ainda que estranha - e de outro aquele para quem aquilo não pode ser dito. Esta é a característica fundamental deste processo de estranhamento (...)".

Para Lins (2008), o Modelo dos Campos Semânticos começa a ser útil quando ele começa "a oferecer elementos para que se produza um melhor entendimento das interações e, é evidente, na sala de aula em particular, permita interações produtivas, interações que eventualmente levem ao compartilhamento de algo, seja o de uma diferença (e aí decidimos o que fazer a esse respeito) ou o compartilhamento de modos de produção de significados, de objetos e de significados (bem mais reconfortante para todos)." Isto é, a beleza/utilidade do Modelo dos Campos Semânticos está nas interações em sala de aula, interações estas que levam ao *compartilhamento* de algo, conforme Lins deixa bem claro, "minha teoria do conhecimento se dirige à manutenção da interação (ou de espaços comunicativos), declaradamente."(LINS, 2008)

As interações em sala de aula podem levar ao compartilhamento de modos de produção de significados e ao compartilhamento de diferenças.

> "No compartilhamento da diferença está, eu penso, a mais intensa oportunidade de aprendizagem (para ambos): é apenas no momento em que posso dizer "eu acho que entendo como você está pensando" que se torna legítimo e simétrico dizer, à continuação, "pois eu estou pensando diferente, e gostaria que você tentasse entender como eu estou pensando"(e, note, o "eu" não fica definido, nisso, se é o do professor ou o do aluno ...). Quer dizer, o que se aprende (ou o que se internaliza, no sentido de Vygotsky) não são conteúdos, técnicas, regras, e sim legitimidades. O que se aprende é a legitimidade de certos modos de produção de significados." (LINS, 2008)

Algumas vezes, o processo de produção de significado em sala de aula pode tomar outros caminhos, como se deparar com limites epistemológicos, entendidos como a impossibilidade de produzir significado para o que é dito pelo outro. Então, para evitar que um estranhamento se torne um limite epistemológico, é necessário realizar o *descentramento*, que requer fazer a leitura do outro, tentar se colocar no lugar do outro.

Santos e Lins (2016), a respeito do processo de descentramento, afirma: "O descentramento é o processo pelo qual você tenta mudar de lugar no mundo, mudar de interlocutor. (...). O cara tenta se colocar como um outro que escreveu aquilo achando que aquilo poderia ser dito. Então o descentramento é mudar o centro, é você sair de você como centro e tentar ir para o lugar onde o outro está como centro."

Outro possível rumo para um processo de produção de significado refere-se ao processo de *impermeabilização*, no qual Silva (2012) destaca: "Chamaremos de impermeabilização ao processo que leva os alunos a não compartilharem novos interlocutores em situação de interação face a face, diferente daqueles para o qual eles estavam voltados; de não se propor a produzir significados numa outra direção."

Diferente do estranhamento, em que o significado produzido pelo sujeito é da ordem da negação - "isso não poderia ser dito" -, no processo de impermeabilização o sujeito produz significado em uma direção não estando "disposto" a produzir em outra.

Silva (2012) explica que há diversas possibilidades que levam à impermeabilização, algumas delas são: acreditar na legitimidade do que diz, de tal forma que é desnecessário dizê-lo de outra forma; não poder produzir significados em outras direções por estar diante de um limite epistemológico; ou entender "ilegítimo" falar em determinada direção.

Essas ideias principais relativas ao Modelo dos Campos Semânticos têm como objetivo situar o leitor - para que o mesmo saiba de onde falamos - no que se refere à produção de significados em relação às análises que serão feitas por nós.

Agora, fica estabelecido a triangulação teórica que sustenta a intervenção metodoló-

gica e a análise da produção de significado na resolução dos problemas de contagem. A articulação dos referenciais não é acidental, mas sim uma construção deliberada para criar um quadro de referência robusto, capaz de investigar o problema de contagem em suas dimensões didática, cognitiva e epistemológica.

O ponto de partida pedagógico foi estabelecido pelos princípios didáticos de Cerioli e Viana (2012), que defendem a Didática dos Princípios de Contagem como o núcleo do raciocínio. A necessidade de materialização da lógica de contagem em Python é justificada pela teoria cognitiva de Lockwood (2011), que enfatiza a conexão entre os Processos de Contagem e a visualização dos Conjuntos de Resultados. O código atua como o dispositivo que torna visível a estrutura da contagem, expondo, por exemplo, o excesso de contagem em um problema de contagem. Finalmente, o Modelo dos Campos Semânticos é a lente analítica primária, pois o Modelo define que o conhecimento é a crença-afirmação junto com uma justificação e a análise da resolução dos problemas de contagem foca na qualidade da enunciação, buscando evidências de que o aluno foi capaz de produzir um significado fundamentado.

Portanto, a convergência desses referenciais permite que o trabalho transcenda a mera descrição da técnica computacional. A materialização algorítmica em Python é o meio que força o aluno a utilizar os Princípios Aditivo e k para 1 da Didática dos Problemas de Contagem, se for necessário. Este confronto é o fenômeno que será analisado para determinar se houve a produção de significado, provando se o aluno é capaz de passar da justificação memorizada (“a fórmula manda dividir”, por exemplo) para uma justificativa lógica e consciente (a divisão é uma ação corretiva que trata o excesso de permutações semanticamente idênticas).

3 Sequência Didática

A seguir estão as atividades, dividimo-as em quatro encontros, cada encontro com duração de 150 minutos. Vale ressaltar que tais atividades devem ser aplicadas após os alunos terem contato com o mínimo sobre programação com Python. Acreditamos que pode ser muito proveitoso aos professores que desejem aplicá-las identificar possíveis adaptações e/ou desenvolvimento de atividades complementares buscando assim oferecer o material que melhor se adeque aos seus alunos.

As atividades propostas têm como objetivo desenvolver as seguintes habilidades presentes em (BRASIL, 2018):

(EM13MAT310) Resolver e elaborar problemas de contagem envolvendo agrupamentos ordenáveis ou não de elementos, por meio dos princípios multiplicativo e aditivo, recorrendo a estratégias diversas, como o diagrama de árvore.

(EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

De acordo com as habilidades propostas, esta atividade é destinada ao $2^{\underline{o}}$ ou $3^{\underline{o}}$ anos do Ensino Médio, dependendo do currículo de cada estado, e requer o conhecimento básico de programação em Python, como já mencionado.

3.1 Primeiro encontro

Objetivos: As atividades propostas no primeiro encontro têm como objetivos principais: reconhecer os objetos dos problemas; reconhecer as configurações desejadas; construir a árvore de possibilidades; familiarizar com o código em Python; reconhecer a função do laço *for* e do condicional *if*; enunciar o Princípio Fundamental da Contagem.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 1:** Considere o seguinte problema: "Quatro estradas A, B, C e D conduzem

ao topo de um morro. De quantos modos diferentes uma pessoa pode subir e descer este morro?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
1  estradas = ['A','B','C','D']
2  contagem = 0
3
4  print('Passeio' '(','Subida',',','Descida',')',':')
5
6  for subida in estradas:
7      for descida in estradas:
8          print('(',subida, ',',descida,')')
9          contagem = contagem + 1
10
11 print('Quantidade de passeios: ', contagem)
```

Código 3.1: Subindo e descendo o morro

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 3.1.

g) Observe o Código 3.1, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

***Atividade* 2:** Considere o seguinte problema: "Suponhamos que na Atividade 1 a pessoa não queira descer o morro pela estrada que subiu. De quantos modos diferentes ela pode subir e descer este morro?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
estradas = ['A','B','C','D']
contagem = 0

print('Passeio' '(','Subida',',','Descida',')',':')

for subida in estradas:
    for descida in estradas:
        if subida != descida:
            print('(',subida, ',',descida,')')
            contagem = contagem + 1

print('Quantidade de passeios: ', contagem)
```

Código 3.2: Subindo e descendo o morro por caminhos diferentes

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 3.2.

g) Observe o Código 3.2, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo da linha de código *if subida != descida*?

***Atividade* 3:** Considere o seguinte problema: "Uma moeda é lançada três vezes. Qual o número de sequências possíveis *cara* e *coroa*?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
moeda = ['CARA','COROA']
contagem = 0

```

```
print('Resultados',':  ' )

for lancamento1 in moeda:
    for lancamento2 in moeda:
        for lancamento3 in moeda:
            print('(',lancamento1, ',',lancamento2, ',', lancamento3,')')
            contagem = contagem + 1

print('Quantidade de sequências: ', contagem)
```

Código 3.3: Sequências de cara e coroa

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 3.3.

g) Observe o Código 3.3, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

***Atividade* 4:** Considere o seguinte problema: "Uma bandeira é formada por quatro listras, que devem ser coloridas usando-se apenas as cores amarelo, rosa e verde, não devendo listras adjacentes ter a mesma cor. De quantos modos pode ser colorida a bandeira?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
cores = ['amarelo','rosa', 'verde']
contagem = 0

for cor1 in cores:
    for cor2 in cores:
        if cor2 != cor1:
            for cor3 in cores:
                if cor3 != cor2:
                    for cor4 in cores:
                        if cor4 != cor3:
                            print('(',cor1, ',',cor2, ',', cor3, ',', cor4,')')
                            contagem = contagem + 1
```

```
13
14 print('Quantidade de maneiras de colorir a bandeira: ', contagem)
```

Código 3.4: Colorindo a bandeira

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 3.4.

g) Observe o Código 3.4, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo das linhas de código *if cor2 != cor1*, *if cor3 != cor2* e *if cor4 != cor3*?

De acordo com as respostas obtidas ao resolver os problemas das atividades do Primeiro Encontro, chegamos no enunciado do Princípio Fundamental da Contagem, conforme encontramos em (CERIOLI; VIANA, 2012):

Princípio Fundamental da Contagem(PFC). *Seja A um conjunto finito. Se cada elemento de A pode ser formado pela tomada de n decisões, $d_1, d_2, \ldots, d_n$, de maneira que:*

- *a decisão d_1 pode ser tomada de m_1 maneiras;*
- *para cada maneira como a decisão d_1 pode ser tomada, a decisão d_2 pode ser tomada de m_2 maneiras;*
- *para cada maneira como as decisões d_1 e d_2 podem ser tomadas (nesta ordem), a decisão d_3 pode ser tomada de m_3 maneiras;*

  ...;
- *para cada maneira como as decisões $d_1, d_2, \ldots, d_{i-1}$ podem ser tomadas (nesta ordem), a decisão d_i pode ser tomada de m_i maneiras;*

  ...;
- *para cada maneira como as decisões $d_1, d_2, \ldots, d_{i-1}, d_i, \ldots, d_{n-1}$ podem ser tomadas (nesta ordem), a decisão d_n pode ser tomada de m_n maneiras.*

Então, $n(A) = m_1 \times m_2 \times m_3 \times \ldots \times m_i \times \ldots \times m_{n-1} \times m_n$, onde $n(A)$ indica a quantidade de elementos do conjunto A.

3.2 Segundo encontro

Objetivos: As atividades propostas no segundo encontro têm como objetivos principais: reconhecer o "funcionamento" do Princípio Fundamental da Contagem no código em Python; utilizar o Princípio Aditivo junto com o Princípio Fundamental da Contagem, quando a resolução do problema divide-se em casos; determinar o número de arranjos por uma aplicação direta do Princípio Fundamental da Contagem.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 5:** Considere o seguinte problema: "Quantos números pares de dois algarismos podem ser formados no sistema decimal?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
algarismos = [0,1,2,3,4,5,6,7,8,9]
contagem = 0

print('Números pares formados: ')
for unidades in algarismos:
    if unidades % 2 == 0:
        for dezenas in algarismos:
            if dezenas != 0:
                print(dezenas,unidades)
                contagem = contagem + 1

print('Quantidade de números pares com dois algarismos: ', contagem)
```

Código 3.5: Números pares com dois algarismos

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Observe o Código 3.5, qual é o objetivo das linhas de código *if dezenas!= 0* e *if unidades % 2 = 0*?

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

***Atividade* 6:** Considere o seguinte problema: "Quantos números pares de dois algarismos distintos podem ser formados no sistema decimal?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual?

c) Execute o seguinte código em Python:

```
algarismos = [0,1,2,3,4,5,6,7,8,9]
contagem = 0

print('Números pares formados: ')
for unidades in algarismos:
    if unidades % 2 == 0:
        for dezenas in algarismos:
            if dezenas != 0 and dezenas != unidades:
                print(dezenas,unidades)
                contagem = contagem + 1

print('Quantidade de números pares com dois algarismos distintos: ', contagem)
```

Código 3.6: Números pares com dois algarismos distintos

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Observe o Código 3.6, qual é o objetivo das linhas de código *if unidades % 2 = 0* e *if dezenas!= 0 and dezenas != unidades*?

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

Você deve ter notado, observando as configurações obtidas no item c) do problema da Atividade 6, que há a necessidade da utilização de um outro Princípio de Contagem, junto com o Princípio Fundamental da Contagem, para encontrar a quantidade de configurações. Tal Princípio é conhecido como Princípio Aditivo e pode ser enunciado da seguinte forma, conforme podemos encontrar em (CERIOLI; VIANA, 2012):

Sejam A um conjunto finito e $A_1, A_2, \cdots, A_n$ subconjuntos não vazios de A.

1. Dizemos que $A_1, A_2, \cdots, A_n$ *exaurem* A se, para cada elemento a de A, existe i, com $1 \leq i \leq n$, tal que $a \in A_i$; ou seja, se $A_1 \cup A_2 \cup \cdots \cup A_n = A$.

2. Dizemos que $A_1, A_2, \cdots, A_n$ são *disjuntos dois a dois* se, quando examinados aos pares, eles não possuem elementos em comum; ou seja, $A_i \cap A_j = \emptyset$, para todos i, j com $1 \leq i \neq j \leq n$.

3. Dizemos que $A_1, A_2, \cdots, A_n$ são uma partição de A se $A_1, A_2, \cdots, A_n$ exaurem A e são disjuntos dois a dois.

Princípio Aditivo. *Seja A um conjunto finito.*

Se $A_1, A_2, \cdots, A_n$ são uma partição de A, então $n(A) = n(A_1) + n(A_2) + \cdots + n(A_n)$.

***Atividade* 7:** Considere o seguinte problema: "De quantos modos 3 pessoas podem sentar-se em 5 cadeiras em fila?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
cadeiras = ['c1','c2','c3','c4','c5']
cadeiras_ocupadas = []

for cadeira_p1 in cadeiras:
    for cadeira_p2 in cadeiras:
            if cadeira_p2 != cadeira_p1:
                for cadeira_p3 in cadeiras:
                    if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:
                        cadeira_ocupada = cadeira_p1,cadeira_p2,cadeira_p3
                        cadeiras_ocupadas.append(cadeira_ocupada)

colunas = 5

print(f"\nCadeiras ocupadas pelas 3 pessoas:  ")
for i, cadeira_ocupada in enumerate(cadeiras_ocupadas):
    print(f"{cadeira_ocupada}", end=",")
    if (i + 1) % colunas == 0:
        print()

if len(cadeiras_ocupadas) % colunas != 0:
    print()

print('Quantidade de maneiras de organizar as 3 pessoas: ', len(cadeiras_ocupadas))
```

Código 3.7: Três pessoas em cinco cadeiras em fila

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

***Atividade* 8:** Considere o seguinte problema: "De quantos modos 6 pessoas podem sentar-se em 10 cadeiras em fila?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido. Se estiver e inseguro, em relação à sua solução, construa um código em Python para resolver o problema.

Observe que as configurações obtidas nos problemas das Atividades 7 e 8 são bem parecidos. Cerioli e Viana (2012) afirmam que: "Quando, para resolver um problema de contagem, podemos aplicar o Princípio Fundamental da Contagem de modo que todas as escolhas envolvidas são feitas em um mesmo conjunto e, a cada escolha a ser feita, os objetos já escolhidos anteriormente não podem mais ser considerados, dizemos que a configuração que estamos contando é um *arranjo simples*."

Será que nos problemas das atividades anteriores já contamos arranjos simples? Se sim, identifique-os.

Definição. Seja A um conjunto com n elementos e $k \in \mathbb{N}$ tal que $1 \leq k \leq n$. Um *arranjo simples* dos n elementos de A, tomados k a k, é uma **sequência** de k elementos distintos de A.

Denotando por $A_{n,k}$ o número de arranjos simples dos n elementos de A, tomados k a k. Temos que:

$$A_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1)$$

.

Note que nem a noção de arranjo simples nem a fórmula para a determinação da quantidade de arranjos simples são muito importantes na resolução dos problemas de

contagem, com esse tipo de configuração, vistos até aqui. Como vimos, podemos determinar o número de arranjos simples por uma aplicação direta do Princípio Fundamental da Contagem.

3.3 Terceiro encontro

Objetivos: As atividades propostas no terceiro encontro têm como objetivos principais: resolver problemas de contagem onde as configurações que estamos contando são permutações simples; utilizar código Python recursivo para contar permutações simples.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 9:** Considere o seguinte código em Python que resolve um determinado problema de contagem e execute-o:

```
letras = ['N','U','M','E','R','O']
anagramas = []

for letra_1 in letras:
    for letra_2 in letras:
            if letra_2 != letra_1:
                for letra_3 in letras:
                    if letra_3 != letra_2 and letra_3 != letra_1:
                        for letra_4 in letras:
                            if letra_4 != letra_3 and letra_4 != letra_2 and letra_4 !=
    letra_1:
                                for letra_5 in letras:
                                    if letra_5 != letra_4 and letra_5 != letra_3 and
    letra_5 != letra_2 and letra_5 != letra_1:
                                        for letra_6 in letras:
                                            if letra_6 != letra_5 and letra_6!= letra_4
    and letra_6 != letra_3 and letra_6 != letra_2 and letra_6 != letra_1:
                                                anagrama = (letra_1 + letra_2 + letra_3 +
     letra_4 + letra_5 + letra_6)
                                                anagramas.append(anagrama)


colunas = 20
largura_coluna = 20

print(f"\nAnagramas da palavra NÚMERO: ")
for i, anagrama in enumerate(anagramas):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(anagramas) % colunas != 0:
    print()

```

```
31 print('Quantidade de anagramas: ', len(anagramas))
```

Código 3.8: Código da Atividade 9

a) Pesquise o que são anagramas.

b) Quais são os objetos básicos do problema?

c) Quais são as configurações formadas?

d) Quantas decisões foram tomadas para resolver o problema? Quais foram essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão foi tomada?

e) Quantas configurações foram obtidas? Com isso, em que consiste o problema resolvido pelo Código 3.8?

f) Utilizando o Princípio Fundamental da Contagem, obtenha a solução do problema resolvido pelo Código 3.8.

No problema da Atividade 9, queremos determinar o número de arranjos simples de 6 letras, tomadas 6 a 6, quando isso ocorre dizemos que a configuração que estamos contando é uma *permutação simples*.

Definição. Seja A um conjunto com n elementos. Uma *permutação simples* dos n elementos de A é uma **sequência** sem repetições de todos os n elementos de A.

Denotando por P_n o número de permutações simples dos n elementos de A. Temos que:

$$P_n = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

.

O número $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$ é denotado por $n!$ e é chamado *fatorial* de n.

***Atividade* 10:** Considere o seguinte problema: "Quantos são os anagramas da palavra MOSTRA que começam por vogal e terminam por consoante?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
letras = ['M','O','S','T','R','A']
vogais = ['A','O']
consoantes = ['M','R','S','T']
anagramas = []

for letra_1 in letras:
    if letra_1 in vogais:
        for letra_6 in letras:
            if letra_6 in consoantes:
                for letra_2 in letras:
                    if letra_2 != letra_1 and letra_2 != letra_6:
                        for letra_3 in letras:
                            if letra_3 != letra_2 and letra_3 != letra_1 and
    letra_3 != letra_6:
                                for letra_4 in letras:
                                    if letra_4 != letra_3 and letra_4 != letra_2
    and letra_4 != letra_1 and letra_4 != letra_6:
                                        for letra_5 in letras:
                                            if letra_5 != letra_4 and letra_5 !=
    letra_3 and letra_5 != letra_2 and letra_5 != letra_1 and letra_5 != letra_6:
                                                anagrama = (letra_1 + letra_2 +
    letra_3 + letra_4 + letra_5 + letra_6)
                                                anagramas.append(anagrama)

colunas = 12

print(f"\nAnagramas da palavra MOSTRA que começam por vogal e teminam por consoante
    : ")
for i, anagrama in enumerate(anagramas):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(anagramas) % colunas != 0:
    print()

print('Quantidade de anagramas: ', len(anagramas))
```

Código 3.9: Anagramas com restrições

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

***Atividade* 11:** Considere o seguinte código recursivo em Python que resolve um determinado problema de contagem:

```python
def permutacoes(palavra):
    if len(palavra) == 1:
        return [palavra]

    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado

palavra = 'NUMERO'

todas_permutacoes = permutacoes(palavra)

colunas = 20

print(f"\nAnagramas da palavra NÚMERO: ")
for i, anagrama in enumerate(todas_permutacoes):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(todas_permutacoes) % colunas != 0:
    print()

print('Quantidade de anagramas: ', len(todas_permutacoes))
```

Código 3.10: Anagramas de NÚMERO e recursividade

a) Execute o código acima. O que esse código fornece como resultado?

b) Compare-o com o Código 3.9. Qual é a diferença?

c) Considere uma palavra com uma quantidade menor de letras, por exemplo ALO, e execute o Código 3.10 passo a passo manualmente, para entender o seu funcionamento.

***Atividade* 12:** Considere a variação do Código 3.10 em Python que determina os anagramas de uma palavra:

```python
def permutacoes(palavra):
    if len(palavra) == 1:
        return [palavra]

```

```
    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado

palavra = str(input("Digite uma palavra: ")).upper()

todas_permutacoes = permutacoes(palavra)

colunas = 8

print(f"\nAnagramas da palavra {palavra}: ")
for i, anagrama in enumerate(todas_permutacoes):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(todas_permutacoes) % colunas != 0:
    print()

print('Quantidade de anagramas: ', len(todas_permutacoes))
```

Código 3.11: Permutações simples e recursividade

a) Execute o código acima, usando como exemplo a palavra AMOR.

b) Usando o Princípio Fundamental da Contagem, determine o número de anagramas da palavra AMOR. Compare o seu resultado com o aquele fornecido pelo Código 3.11.

c) Execute o código usando, agora, como exemplo a palavra AMAR.

d) Analise, atentamente, a saída de execução do código. Você percebeu alguma coisa estranha? O quê?

e) Então, quantos são os anagramas da palavra AMAR?

f) Repita os itens c), d) e e) considerando a palavra BABA.

g) Repita os itens c), d) e e) considerando a palavra AAZA(força em arábico).

h) Por quê o Código 3.11 não funciona com as palavras AMAR, BABA e AAZA?

3.4 Quarto encontro

Objetivos: As atividades propostas no quarto encontro têm como objetivos principais: mostrar que o Princípio Fundamental da Contagem gera um excesso na resolução de problemas onde os objetos que serão permutados não são todos distintos e quando da determinação da quantidade de grupos(conjuntos); diferenciar uma sequência de um conjunto; corrigir a superestimação, construída pelo Princípio Fundamental da Contagem, utilizando o Princípio k para 1.

Tempo para execução: 3 aulas de 50 minutos.

Na Atividade 12, deve-se ter notado que o código que calcula os anagramas de uma palavra só funciona corretamente quando a mesma possui letras distintas, ou seja, o código só calcula a quantidade de permutações simples.

O código da Atividade 13 tem como objetivo corrigir esse problema, isto é, calcular o número de anagramas de qualquer palavra.

***Atividade* 13:** Considere o seguinte código recursivo em Python que determina os anagramas de uma palavra mas, agora, de uma forma mais organizada, resolvendo os problemas que tivemos na Atividade 12:

```
def permutacoes(palavra):

    if len(palavra) == 1:
        return [palavra]

    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado


def remover_duplicatas_sort(lista):

    lista_ordenada = sorted(lista)

    lista_de_listas = []
    grupo_atual = []
    for i in range(len(lista_ordenada)):
        if i == 0 or lista_ordenada[i] == lista_ordenada[i - 1]:
            grupo_atual.append(lista_ordenada[i])
        else:
```

```
            lista_de_listas.append(grupo_atual)
            grupo_atual = [lista_ordenada[i]]
    lista_de_listas.append(grupo_atual)

    for i, uplas in enumerate(lista_de_listas, 1):
        print(f"{i:3}. {uplas}")
    print()
    print("Número de anagramas: ", len(lista_ordenada))
    print()
    print("Número de anagramas repetidos em cada caixa: ", len(grupo_atual))
    print()


    resultado = [lista_ordenada[0]]

    for i in range(1, len(lista_ordenada)):
        if lista_ordenada[i] != lista_ordenada[i-1]:
            resultado.append(lista_ordenada[i])
    return resultado

palavra = str(input("Digite uma palavra qualquer: ")).upper()

todas_permutacoes = permutacoes(palavra)
permutacoes_com_elemetos_repetidos = remover_duplicatas_sort(todas_permutacoes)


print(f"\nAnagramas da palavra {palavra}: ")
for i, perm in enumerate(permutacoes_com_elemetos_repetidos, 1):
    print(f"{i:3}. {perm}")
print(f"Quantidade de anagramas da palavra {palavra}: {len(
    permutacoes_com_elemetos_repetidos)}")
```

Código 3.12: Permutações com elementos repetidos

a) Execute o código acima, usando como exemplo a palavra AMAR.

b) O resultado obtido é, de fato, a quantidade de anagramas da palavra AMAR?

c) Execute o código usando, agora, como exemplo a palavra AAZA.

d) O resultado obtido é, de fato, a quantidade de anagramas da palavra AAZA?

e) Execute o Código 3.12, usando a palavra SOSSEGO. Quantos anagramas essa palavra possui?

Na execução do código da Atividade 13, perceba que para descobrir a quantidade de anagramas das palavras com letras repetidas, houve necessidade de "jogar fora" excessos. Note que, em palavras com letras repetidas, para k anagramas obtidos(repetidos) apenas

1 é considerado anagrama da palavra em questão. Temos, assim, a utilização do Princípio k para 1 em conjunto com o Princípio Fundamental da Contagem.

Definição. Sejam A e B conjuntos finitos. Uma *função k para 1* de A em B associa elementos de A a elementos de B, de modo que:

- Cada k elementos de A estão associados a 1 elemento de B.
- Cada elemento de B é o associado de exatamente k elementos de A.

Princípio k para 1. *Sejam A e B conjuntos finitos. Se existe uma função k para 1 de A em B, então $n(A) = k \cdot n(B)$.*

Com isso, podemos concluir, que a quantidade de permutações de n objetos nem todos distintos, em que um deles aparece n_1 vezes(fazendo com que contemos cada anagrama $n_1!$ vezes), outro n_2 vezes(fazendo com que contemos cada anagrama $n_2!$ vezes), e assim por diante, é dada por

$$P_n^{n_1,n_2,\cdots} = \frac{n!}{n_1! \cdot n_2! \cdot \ldots}.$$

f) Determine o número de anagramas das palavras MANIA, CORRER e PASSISTA, usando o Princípio Fundamental da Contagem e o Princípio k para 1.

***Atividade* 14:** Vamos ao seguinte problema: "Considere 5 pessoas A, B, C, D e E. Quantas filas com 3 dessas 5 pessoas podemos formar?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
def formar_filas(pessoas):

    print("\n" + "=" * 60)
    print("PASSO 1: Forma as filas.")
    print("=" * 60)

    resultado = []
    contador_filas = 0

```

```
    for pessoa_1 in pessoas:
        for pessoa_2 in pessoas:
            if pessoa_2 != pessoa_1:
                for pessoa_3 in pessoas:
                    if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:
                        contador_filas += 1
                        fila_formada = [pessoa_1,pessoa_2,pessoa_3]
                        resultado.append(fila_formada)
                        print(f"Fila {contador_filas:3d}: {fila_formada}")

    print(f"\nTotal de filas diferentes: {contador_filas}")
    return resultado



pessoas = ['A','B','C','D','E']

filas_obtidas = formar_filas(pessoas)
```

Código 3.13: Filas com três pessoas dadas cinco pessoas

d) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

f) Esse problema é parecido com algum problema que já foi resolvido? Qual?

g) E quantas filas com 4 pessoas podemos formar?

***Atividade* 15:** Agora, vamos a esse problema: "Considere 5 pessoas A, B, C, D e E. Quantos grupos com 3 dessas 5 pessoas podemos formar?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) A solução desse problema é igual à solução do problema da Atividade 14? Ou seja, a quantidade de filas é igual à quantidade de grupos? Justifique.

d) Se a resposta do item anterior for NÃO, corrija a resposta, obtendo, assim, a resposta correta.

No problema da Atividade 14, estamos, claramente, buscando a quantidade de arranjos simples de 5 pessoas tomadas 3 a 3. Já no problema da Atividade 15, mais uma vez, se utilizarmos o Código 3.13 teremos excesso e, para corrigir a solução, necessitamos utilizar o Princípio k para 1.

No problema da Atividade 15 temos 5 pessoas e devemos efetuar uma escolha na qual 3 pessoas são tomados simultaneamente, dizemos, então, que a configuração que estamos contando é uma *combinação simples*.

Antes de definir formalmente *combinação simples*, vejamos a Atividade 16.

***Atividade* 16:** Considere o seguinte código em Python que resolve um determinado problema de contagem:

```
def formar_filas(pessoas):

    print("\n" + "=" * 60)
    print("PASSO 1: Forma as filas.")
    print("=" * 60)

    resultado = []
    contador_filas = 0

    for pessoa_1 in pessoas:
        for pessoa_2 in pessoas:
            if pessoa_2 != pessoa_1:
                for pessoa_3 in pessoas:
                    if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:
                        contador_filas += 1
                        fila_formada = [pessoa_1,pessoa_2,pessoa_3]
                        resultado.append(fila_formada)
                        print(f"Fila {contador_filas:3d}: {fila_formada}")

    print(f"\nTotal de filas diferentes: {contador_filas}")
    return resultado

def ordenar_e_imprimir_filas(filas):

    print("\n" + "=" * 60)
    print("PASSO 2: Ordena cada fila individualmente")
    print("=" * 60)

    filas_ordenadas = []
    ordenadas = []

    print("\nTodas as filas ORDENADAS:")
    print("-" * 60)

    for idx, fila in enumerate(filas, 1):
        fila_ordenada = sorted(fila)
        filas_ordenadas.append(fila_ordenada)
        print(f"Fila {idx}: {fila_ordenada}")
```

```
39
40      return filas_ordenadas
41
42  def remover_duplicatas_sort(lista):
43
44      print("\n" + "=" * 60)
45      print("PASSO 3: Coloca as filas iguais em uma mesma caixa")
46      print("=" * 60)
47
48      lista_ordenada = sorted(lista)
49
50      lista_de_listas = []
51      grupo_atual = []
52      for i in range(len(lista_ordenada)):
53          if i == 0 or lista_ordenada[i] == lista_ordenada[i - 1]:
54              grupo_atual.append(lista_ordenada[i])
55          else:
56              lista_de_listas.append(grupo_atual)
57              grupo_atual = [lista_ordenada[i]]
58
59      lista_de_listas.append(grupo_atual)
60
61      for i, uplas in enumerate(lista_de_listas, 1):
62          print(f"{i:3}. {uplas}")
63      print()
64      print("Número de filas: ", len(lista_ordenada))
65      print()
66      print("Depois que as filas são ordenadas, temos: ", len(grupo_atual) , " filas em
        cada caixa.")
67      print()
68
69      resultado = [lista_ordenada[0]]
70
71      for i in range(1, len(lista_ordenada)):
72          if lista_ordenada[i] != lista_ordenada[i-1]:
73              resultado.append(lista_ordenada[i])
74      return resultado
75
76
77  pessoas = ['A','B','C','D','E']
78
79  filas_obtidas = formar_filas(pessoas)
80  ordenadas = ordenar_e_imprimir_filas(filas_obtidas)
81  grupos = remover_duplicatas_sort(ordenadas)
82
83  print("Todas os grupos:")
84  for i, perm in enumerate(grupos, 1):
85      print(f"{i:3}. {perm}")
86  print(f"Número de grupos: {len(grupos)}")
```

Código 3.14: Código da Atividade 16

a) Execute o código acima.

b) O que esse código fornece como resultado?

c) Compare-o com o Código 3.13. Qual é a diferença?

d) Esse código fornece a solução do problema da Atividade 15?

e) Usando a ideia do código acima, dadas cinco pessoas quantos grupos com quatro pessoas podemos formar? E, dadas dez pessoas quantos grupos com quatro pessoas podemos formar?

Finalmente, temos a definição formal para Combinações Simples.

Definição. Seja A um conjunto com n elementos e $k \in \mathbb{N}$ tal que $1 \leq k \leq n$. Uma *combinação simples* dos n elementos de A, tomados k a k, é um **conjunto** de k elementos distintos de A.

Pela definição, como uma combinação é um **conjunto** com elementos de A. A ordem em que os elementos estão listados não é levada em conta na determinação da combinação.

Denotando por $C_{n,k}$ o número de combinações simples dos n elementos de A, tomados k a k. O número de combinações simples dos n elementos de A, tomados k a k, é dado pela fórmula

$$C_{n,k} = \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-(k-1))}{k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1} = \frac{A_{n,k}}{P_k}.$$

4 A ação

Esse capítulo tem como objetivo apresentar as atividades e os resultados obtidos em sua aplicação.

A presente pesquisa alinha-se aos pressupostos da pesquisa qualitativa, adotando como estratégia metodológica o estudo de caso. O foco central desta abordagem não reside na quantificação de acertos ou falhas na resolução de problemas combinatórios, mas sim na compreensão profunda dos processos interacionais e cognitivos que ocorrem durante a aprendizagem.

A escolha pelo estudo de caso justifica-se pela necessidade de investigar um fenômeno complexo dentro de seu contexto real de ocorrência. Ao adotar essa perspectiva interpretativa, a pesquisa busca mapear o percurso de construção do conhecimento nas entrelinhas das interações verbais e práticas. A linguagem Python é assumida, neste cenário, não apenas como uma ferramenta algorítmica ou de cálculo, mas como um artefato cultural e mediador que propõe uma nova sintaxe para a estruturação do pensamento matemático.

A nossa investigação foi realizada em uma escola pública da esfera estadual localizada no município de Itaboraí. A escolha desta escola vem do fato de já lecionarmos lá, especificamente em turmas de terceiro ano do Ensino Médio, e termos recebido autorização da direção para realizar o trabalho dentro das próprias aulas da turma e respeitando o conteúdo previsto pela Secretaria de Educação do Estado do Rio de Janeiro(SEEDUC-RJ).

Tínhamos o seguinte planejamento trimestral para a turma escolhida do terceiro ano do ensino médio .

Data	Conteúdo
17/09/25	Algoritmo
24/09/25	Python(Input e Output; variáveis numéricas e strings)
01/10/25	Python(Variáveis lógicas e listas)
08/10/25	Python(Condicional e for em listas)
22/10/25	Aplicação das atividades do 1º encontro
29/10/25	Projeto da Escola
05/11/25	Aplicação das atividades do 2º encontro
12/11/25	Aplicação das atividades do 3º encontro
19/11/25	Aplicação das atividades do 4º encontro
26/11/25	Prova do Trimestre
03/12/25	Recuperação do trimestre

Mas, vários eventos ocorreram no trimestre, prejudicando a aplicação das atividades. Dentre os eventos vale destacar: Projeto Redação(ENEM), Simulado para o SAEB, Feira de profissões, Sarau Literário, Prova do SAEB, Projeto trimestral da escola. Somados a esses eventos ainda temos a problemática que alguns alunos, no final do ano, não se envolvem com o ensino. Com isso, tivemos apenas os dias 19/11/25 e 03/12/25 para aplicar as atividades da Sequência Didática. No dia 19/11 aplicamos as Atividades 1, 2 e 3 do Primeiro Encontro e no dia 03/12 aplicamos a Atividade 9 do Terceiro Encontro e as Atividades 14 e 15 do Quarto Encontro. Vale ressaltar que nos dias 17/09 e 01/10 apresentamos de forma muito breve o conceito de algoritmos e a linguagem Python, pré requisitos necessários para o desenvolvimento de nossa pesquisa.

Durante a realização das atividades do Primeiro e Segundo encontros, estiveram presentes cerca de 10 estudantes e, desses 10 alunos, 3 ou 4 participaram ativamente, mesmo a turma tendo 27 alunos no total. O motivo da quantidade de estudantes ser tão baixa deve-se ao seguinte fato, após a data da aplicação da prova do SAEB, que ocorreu no dia 12/11/25, os alunos, que já estavam aprovados, não precisaram fazer a prova trimestral, ou seja, já estavam de férias. No escopo de um estudo de caso qualitativo, essa delimitação numérica longe de ser uma limitação constitui uma condição vantajosa e necessária, pois viabiliza um acompanhamento denso e minucioso da produção de significado de cada indivíduo. Além do número reduzido, garantiu-se a consistência do corpus analítico: foram exatamente os mesmos alunos que participaram tanto do primeiro quanto do segundo encontro. Essa continuidade longitudinal permitiu acompanhar a evolução empírica do grupo, observando como o contato com a lógica de programação introduzida no primeiro encontro reverberou e alterou suas estratégias de raciocínio.

O registro empírico das interações foi realizado primordialmente por meio de gravações de áudio. A escolha por este instrumento justifica-se pela necessidade de captar integralmente as falas espontâneas, as discussões entre os pares e as mediações dialógicas com o pesquisador no exato momento em que as hipóteses são formuladas, testadas ou refutadas.

Para investigar a dinâmica de aprendizagem que veio à tona durante as resoluções das atividades, os diálogos transcritos não serão analisados de forma fragmentada, isolando cada item do questionário. Em vez disso, a análise adota uma abordagem narrativa centrada em episódios de aprendizagem, acompanhando o fluxo contínuo das discussões em sala de aula, discussões essas muito poucas, devido a pouca quantidade de alunos.

Os dados serão examinados com toda a atenção sob a articulação de duas perspectivas teóricas complementares. Por um lado, o Modelo dos Campos Semânticos permitirá investigar a legitimidade da produção de significado dos estudantes, observando como os resíduos de enunciação advindos de suas práticas cotidianas entram em negociação com os textos propostos (o enunciado matemático, o diagrama de árvore e a sintaxe do código Python), gerando deslocamentos sucessivos em suas leituras. Por outro lado, o Modelo de Lockwood fornecerá o arcabouço para mapear o desenvolvimento do raciocínio combinatório em si. Observaremos como os estudantes transitam e coordenam as três instâncias do modelo — a definição das configurações válidas (Conjunto de Resultados), a sistematização visual e computacional das escolhas (Processos de Contagem) e a dedução da operação aritmética subjacente (Expressões Matemáticas).

4.1 Primeiro encontro

Objetivos: As atividades propostas no primeiro encontro têm como objetivos principais: reconhecer os objetos dos problemas; reconhecer as configurações desejadas; construir a árvore de possibilidades; familiarizar com o código em Python; reconhecer a função do laço *for* e do condicional *if*; enunciar o Princípio Fundamental da Contagem.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 1:** Considere o seguinte problema: "Quatro estradas A, B, C e D conduzem ao topo de um morro. De quantos modos diferentes uma pessoa pode subir e descer este morro?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

Diálogo

Professor: Depois da leitura atenta do problema. “Então, quais são os objetos básicos?”

A resposta unânime foi: “São as estradas do morro. A, B, C e D.”

Professor: “Quais são as configurações desejadas? O que queremos contar?”

Aluno A: “As configurações são os caminhos que vou usar para subir e descer o morro.”

Aluno B: “Isso mesmo professor. Sair de baixo, chegar lá no topo e depois voltar para baixo.” E, mostrando quatro dedos de cada mão e juntando-os, o aluno continua. “E, olha só, são 8 caminhos que temos no total.”

Professor: “Ok, entendi a sua resposta. Vamos ao item b) para tentar estruturá-la.”

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Diálogo

Professor: “Para fazer esse passeio completo, quantas decisões nós precisamos tomar?”

Aluno B: “Uma só! A decisão é escolher o caminho que vou percorrer.”

Professor: “Vou tentar ser mais claro. Para você escolher um caminho, você precisa cumprir etapas, certo?"

Aluno B: “Sim. Claro, subir e descer o morro.”

Professor: “Então, quantas decisões você precisará tomar para formar o seu caminho, ou melhor, a sua configuração?”

Aluno B: “Agora, entendi. São duas decisões. Uma é escolher o caminho para subir o morro e a outra é escolher o caminho para descer o morro.”

Professor: “Perfeito. Há alguma restrição na tomada de decisões?”

Aluno C: “Acho que sim. Se eu subir pelo caminho A eu não posso descer por ele também.”

Aluno A: "Claro que não, cara! De onde você tirou isso? Eu posso subir pelo caminho A e descer por ele também. Não é isso professor?"
Aluno B: "Tá dando nó na minha cabeça."
Aluno A: "Olha para o problema, tá escrito de quantos modos diferentes uma pessoa pode subir e descer este morro."
Aluno C: "Não, não. Agora, entendi. A palavra diferente é em relação ao percurso que faço de subida e descida, certo?"
Professor: "Isso aí. Então, temos duas decisões e não temos restrições. Agora, de quantas maneiras cada decisão pode ser tomada?"
Aluno B: "Não entendi."
Professor: "Certo. Qual é a primeira decisão ?"
Aluno B: "Escolher a estrada de subida."
Professor: "Então, quantas opções de estrada temos para tomar essa decisão?"
Aluno B: "Uma só."
Aluno C: "Claro que não. Você vai escolher uma estrada. Mas, você tem 4 opções de estrada: A, B, C ou D."
Professor: "Isso mesmo. Vamos adiantar o item seguinte, vamos começar a fazer a árvore de possibilidades(desenhei a árvore de possibilidades com os quatro ramos, representando as 4 opções de subida). Agora, precisamos escolher a estrada de subida. Qual vocês escolhem?"
Aluno A: "C."
Professor: "Subimos o morro pela estrada C, quantas opções de estrada temos, agora, para descer o morro?"
Aluno A: "São 4 opções."
Professor: "Essa quantidade é a mesma, se eu escolher outra estrada de subida, por exemplo, A?"
Aluno A: "Sim."
Desenhei todos os ramos de descida na árvore e para concluir fiz a seguinte pergunta:
Professor: "Então, para concluir, tomada a primeira decisão, quantas opções de estrada temos para a segunda decisão?
Aluno A: "16."
Professor: "Não entendi. Poderia explicar?"
Aluno A: "Professor, para subir, a árvore tem 4 galhos e para descer tem 16 galhos. Vê a árvore."

Aluno C: "Cara, como pode ter 16 opções de estrada, se só tem 4 estradas no problema? O que o professor perguntou foi o seguinte: se eu já subi o morro por uma estrada, tenho quantas maneiras de descer?"
Aluno A: "Errei. É mesmo. Entendi."

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

A seguinte árvore de possibilidades foi desenhada no quadro, ao resolver o item anterior.

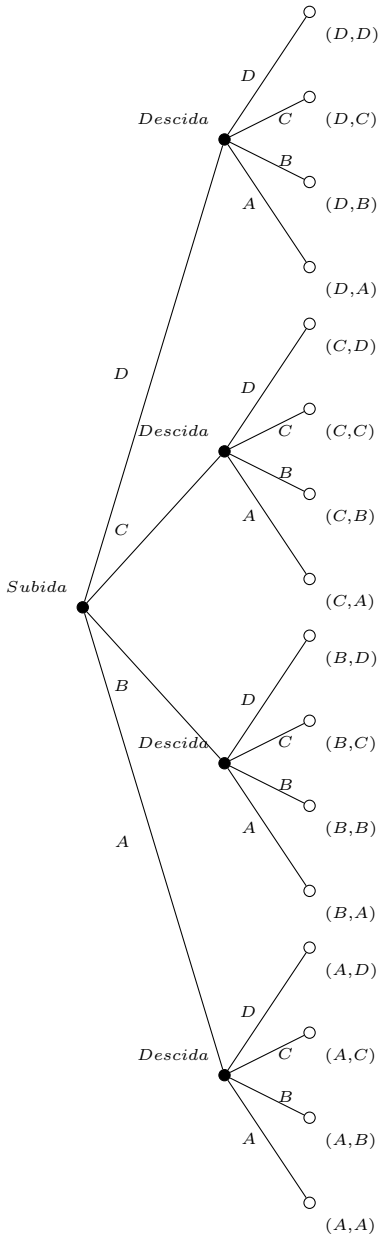


Figura 3: Árvore de Possibilidades da Atividade.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Diálogo

Professor: "De acordo com a árvore que desenhamos, quantas configurações existem?"

Aluno B: "São 16 configurações. Ficou mais fácil de ver assim. Eu tava somando quando disse que tinha 8 caminhos."

Aluno C: "Se cada um dos 4 caminhos de subida abriu 4 caminhos de descida... são 4 vezes 4. Dá 16 caminhos diferentes! Professor, vai ser sempre assim? Se fossem cinco estradas, o resultado seria 5×5?"

Professor: "Ótima pergunta. Sim. Aumentaremos um caminho de subida e, para cada caminho de subida, teremos, também mais uma opção de caminho para a descida. Ou seja, teremos $5 + 5 + 5 + 5 + 5 = 5 \times 5 = 25$ caminhos."

Professor: "Agora, vamos abrir o app **Pydroid** no celular, copiar esse código e executá-lo."

e) Agora, execute o seguinte código em Python:

```
1  estradas = ['A','B','C','D']
2  contagem = 0
3
4  print('Passeio' '(','Subida',',','Descida',')',':')
5
6  for subida in estradas:
7      for descida in estradas:
8          print('(',subida, ',',descida,')')
9          contagem = contagem + 1
10
11 print('Quantidade de passeios: ', contagem)
```

Código 4.1: Código da Atividade

Os alunos, nesse momento, copiaram e executaram o código, obtendo o resultado abaixo. Alguns erros de execução aconteceram, principalmente aqueles ocasionados pela má indentação(falta de recuo).

```
Passeio( Subida , Descida ) :
( A , A )
( A , B )
( A , C )
( A , D )
( B , A )
( B , B )
( B , C )
( B , D )
( C , A )
( C , B )
( C , C )
( C , D )
( D , A )
( D , B )
( D , C )
( D , D )
Quantidade de passeios:  16
>>>
```

Figura 4: Saída do Código 4.1.
Fonte: Autoria própria(2026).

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 4.1.

Diálogo

Aluno A: "O programa imprimiu a mesma lista que a gente fez na árvore e o mesmo resultado da nossa conta de vezes! É como se ele tivesse construído a árvore dentro dele."

Professor: Exatamente. E no item g), o que os laços *for* do código têm a ver com a nossa árvore de possibilidades?

g) Observe o Código 4.1, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

Diálogo

Aluno A: "O primeiro *for* é a primeira decisão, a subida. E o segundo *for* é a segunda decisão, a descida. Só não entendi uma coisa: Por que não posso colocar um *for* embaixo do outro, tem que afastar? Parece que tá um dentro do outro."

Professor: É isso mesmo. Para cada subida, devemos fazer todas as descidas. Por exemplo, o primeiro *for* fixa a subida pela estrada A. Enquanto o primeiro tá parado no A, o segundo *for* roda as descidas A, B, C e D. É como se fosse o programa desenhando os galhinhos menores da nossa árvore!

4.1.1 Análise da Atividade 1: Do Raciocínio Aditivo à Multiplicação

A **Atividade 1** inaugura a sequência didática apresentando aos alunos um problema clássico de contagem: "Quatro estradas A, B, C e D conduzem ao topo de um morro. De quantos modos diferentes uma pessoa pode subir e descer este morro?". O objetivo pedagógico central desta tarefa não se restringe à obtenção da resposta numérica final, mas sim à mobilização dos conhecimentos prévios dos estudantes e à estruturação de um raciocínio que os conduza do pensamento puramente aditivo e empírico para a compreensão da estrutura multiplicativa inerente ao Princípio Fundamental da Contagem. Além disso, a atividade introduz o uso da programação em Python como ferramenta de validação e generalização algorítmica.

A dinâmica desta primeira atividade é dissecada em três episódios centrais que evidenciam os principais conflitos semânticos e saltos cognitivos vivenciados pela turma.

Episódio 1: O Cotidiano Versus o Enunciado Matemático

Neste primeiro episódio, que engloba os itens **a**, **b** e parte do **c**, o foco analítico recai sobre a negociação do que constitui um resultado válido para o problema. Antes de qualquer tentativa formal de contagem, os alunos precisam entrar em acordo sobre as regras do Conjunto de Resultados, momento em que a linguagem cotidiana entra em atrito com o contrato didático da matemática. O diálogo a seguir evidencia essa tensão:

Professor: "Perfeito. Há alguma restrição na tomada de decisões?"

Aluno C: "Acho que sim. Se eu subir pelo caminho A eu não posso descer por ele também."

Aluno A: "Claro que não, cara! De onde você tirou isso? [...] Olha para o problema, tá escrito de quantos modos diferentes uma pessoa pode subir e descer este morro."

Aluno C: "Não, não. Agora, entendi. A palavra diferente é em relação ao percurso que faço de subida e descida, certo?"

Pela lente do Modelo dos Campos Semânticos, a fala inicial do *Aluno C* não é vista como um erro processual, mas como uma produção de significado perfeitamente legítima dentro de um campo semântico ancorado em práticas cotidianas de deslocamento. O aluno mobiliza, a partir de sua experiência vivida, um modelo de "passeio" no qual percursos de ida e volta tendem a ser distintos - uma lógica coerente em muitos contextos extramatemáticos. Esse campo semântico entra em atrito com o contrato didático do problema escolar, que exige uma leitura estritamente literal das restrições enunciadas. O

Aluno A atua, então, como mediador desse conflito. Ao direcionar a atenção de volta para o texto ("Olha para o problema, tá escrito..."), ele força um deslocamento no *Aluno C*, trazendo-o para o campo semântico da matemática formal, onde as restrições devem estar explícitas no enunciado e não inferidas pelo senso comum.

Sob a ótica do Modelo de Lockwood, este episódio é o alicerce para a construção do Conjunto de Resultados. Lockwood (2011) e Cerioli e Viana (2012) enfatizam que os alunos frequentemente lutam com problemas combinatórios porque não compreendem plenamente a natureza dos elementos que estão tentando contar. A intervenção pragmática do *Aluno C*("não posso descer por ele também") alteraria drasticamente a estrutura do Conjunto de Resultados, reduzindo os passeios totais de 16 para 12. A negociação semântica mediada pelo *Aluno A* foi o que garantiu a estabilização correta das regras de formação desse conjunto - compreendendo que o resultado é um par ordenado independente -, condição sem a qual os Processos de Contagem subsequentes seriam construídos sobre premissas falhas.

Episódio 2: Árvore de Possibilidades e Deslocamento para o Raciocínio Multiplicativo

Neste episódio, analisamos o momento em que intervimos no processo, introduzindo uma nova representação visual - a árvore de possibilidades - e como isso impacta a produção de significado dos alunos, que até então operavam sob uma lógica ancorada na adição e em restrições pragmáticas do cotidiano. O trecho a seguir ilustra a culminância dessa intervenção:

Professor: "De acordo com a árvore que desenhamos, quantas configurações existem?"

Aluno B: "São 16 configurações. Ficou mais fácil de ver assim. Eu tava somando quando disse que tinha 8 caminhos."

Aluno C: "Se cada um dos 4 caminhos de subida abriu 4 caminhos de descida... são 4 vezes 4. Dá 16 passeios diferentes! Professor, vai ser sempre assim? Se fossem cinco estradas, o resultado seria 5 vezes 5?"

À luz do Modelo dos Campos Semânticos, observamos neste trecho a materialização de um claro deslocamento cognitivo. Anteriormente, o *Aluno B* operava em um campo semântico no qual o enunciado uma leitura aditiva: dois grupos de quatro estradas, um para a subida e outro para a descida, totalizando oito. Essa leitura se materializou gestualmente - ao mostrar quatro dedos de cada mão - evidenciando que o objeto construído pelo

aluno era composto por dois conjuntos independentes e paralelos e não por pares ordenados resultantes de uma composição de decisões. A enunciação é internamente coerente, o que difere é o objeto sobre o qual ela opera. Atuando como mediador, introduzimos um novo texto na dinâmica da sala: a árvore de possibilidades.

A estrutura ramificada da árvore oferece uma resistência à leitura anterior do *Aluno B*. Não é mais possível produzir significado aditivo olhando para galhos que se multiplicam a cada nó. Ocorre, portanto, um deslocamento: para dar sentido à figura, o *Aluno B* precisa abandonar seu campo semântico inicial e assumir uma nova lógica de leitura. Sua enunciação "Ficou mais fácil de ver assim. Eu tava somando..." é o reconhecimento verbal desse próprio deslocamento, legitimando a nova forma de produzir significado matemático.

Simultaneamente, a articulação evidenciada pelo *Aluno C* pode ser dissecada com precisão através do Modelo de Lockwood. Para Lockwood, a fluidez no raciocínio combinatório exige a coordenação entre o Conjunto de Resultados, os Processos de Contagem e as Expressões Matemáticas. A árvore de possibilidades atua, neste contexto, como um robusto Processo de Contagem estruturante.

Ao visualizar a árvore, o *Aluno C* consegue, finalmente, compreender a natureza composta do Conjunto de Resultados (o par ordenado de subida e descida). Mais importante ainda, é a regularidade visual desse Processo de Contagem que lhe permite dar o salto cognitivo para a Expressão Matemática. Ao enunciar "Se cada um dos 4 caminhos [...] abriu 4 caminhos... são 4 vezes 4", o aluno demonstra ter coordenado com sucesso a etapa de contagem sistemática com a operação aritmética correspondente, chegando inclusive a formular uma generalização hipotética ("Se fossem cinco estradas..."). A árvore serviu como a alavanca que conectou a visualização do problema à sua formalização algébrica.

Episódio 3: A Materialização Algorítmica da Árvore

Agora, a análise recai sobre a introdução da linguagem de programação Python como uma nova ferramenta de modelagem, avaliando como os alunos traduzem a abstração visual da árvore de possibilidades para a sintaxe de um código computacional.

Aluno A: "O primeiro *for* é a primeira decisão, a subida. E o segundo *for* é a segunda decisão, a descida. Só não entendi uma coisa: Por que não posso colocar um for embaixo do outro, tem que afastar? Parece que tá um dentro do outro."

Professor: "É isso mesmo. Para cada subida, devemos fazer todas as descidas. [...] Enquanto o primeiro tá parado no A, o segundo *for* roda as descidas A, B, C e D. É como se fosse o programa desenhando os galhinhos menores da nossa árvore!"

O Modelo dos Campos Semânticos nos permite analisar o estranhamento inicial do *Aluno A* com a indentação do código. Diante de um texto sintaticamente novo, o aluno produz significado a partir de um referencial puramente visual e espacial ("um embaixo do outro" versus "um dentro do outro"). A nossa intervenção pedagógica residiu no uso de uma metáfora para promover o deslocamento semântico. Não recorremos ao jargão da ciência da computação (como "escopo de variáveis" ou "iteração aninhada"); em vez disso, fixamos o funcionamento do código no campo semântico que acabamos de consolidar com a turma: a árvore de possibilidades. Ao enunciar que o programa está "desenhando os galhinhos menores", fornecemos a ponte de significação necessária para que os alunos leiam a estrutura algorítmica.

Para o Modelo de Lockwood, a introdução do código Python atua como a formalização de um Processo de Contagem automatizado. Quando os alunos executam o código e o *Aluno A* observa que "o programa imprimiu a mesma lista que a gente fez na árvore e o mesmo resultado da nossa conta de vezes", ocorre a consolidação máxima do modelo. O aluno transita de forma fluida e consciente pelas três instâncias: o Conjunto de Resultados (a lista impressa no terminal), o Processo de Contagem (a lógica iterativa dos laços *for* refletindo a árvore) e a Expressão Matemática (a operação multiplicativa validada pelo contador). O pensamento computacional, neste episódio, não operou como substituto do raciocínio matemático, mas como uma nova representação capaz de tornar visível a estrutura do problema de uma forma diferente da árvore. A observação do *Aluno A* — "o programa imprimiu a mesma lista que a gente fez na árvore" — indica o reconhecimento de uma correspondência entre as representações, o que é, em si, um resultado pedagógico relevante. Não é possível afirmar, a partir deste episódio isolado, que essa correspondência foi plenamente internalizada; o que se pode dizer é que a execução do código criou condições para que a transição entre representações fosse percebida pelos alunos como um processo coerente e verificável.

A análise da **Atividade 1** revela que a apropriação do Princípio Fundamental da Contagem pelos estudantes vai além de uma simples memorização de uma operação aritmética. Trata-se, fundamentalmente, de um processo complexo de negociação de significados e de transições representacionais.

Ao cruzarmos as lentes do Modelo dos Campos Semânticos e do Modelo de Lockwood, torna-se evidente que a estrutura combinatória almejada - a fluidez entre Conjuntos de Resultados, Processos de Contagem e Expressões Matemáticas - apenas se estabelece quando os obstáculos semânticos iniciais são superados. O Modelo dos Campos Semânticos nos

permitiu observar que os impasses iniciais dos alunos, como a imposição de restrições de trajeto baseadas no cotidiano ou o uso da soma isolada ($4 + 4 = 8$), não eram falhas de raciocínio lógico, mas sim produções de significado legítimas, ancoradas em resíduos de enunciação de suas vivências diárias e corpóreas.

***Atividade* 2:** Considere o seguinte problema: "Suponhamos que na Atividade 1 a pessoa não queira descer o morro pela estrada que subiu. De quantos modos diferentes ela pode subir e descer este morro?". Responda as questões a seguir, antes de resolvê-lo.

Diálogo:
Aproveitando toda a discussão feita para o problema anterior, após a leitura do problema da Atividade 2, um dos alunos levantou-se e apagou os galhos da árvore que não eram configurações válidas para o problema.
Aluno C: "Pronto. A resposta é 12."
Com essa atitude, os itens a) ao d) foram respondidos. E, todos os alunos concordaram com a resposta.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Professor: Então, vamos ao item e)

e) Agora, execute o seguinte código em Python:

```
estradas = ['A','B','C','D']
contagem = 0

print('Passeio' '(','Subida',',','Descida',')',':')

for subida in estradas:
    for descida in estradas:
        if subida != descida:
            print('(',subida, ',',descida,')')
            contagem = contagem + 1

```

```
12 print('Quantidade de passeios: ', contagem)
```

Código 4.2: Código da Atividade

Os alunos copiaram e executaram o código, obtendo o resultado abaixo.

```
Passeio( Subida , Descida ) :
( A , B )
( A , C )
( A , D )
( B , A )
( B , C )
( B , D )
( C , A )
( C , B )
( C , D )
( D , A )
( D , B )
( D , C )
Quantidade de passeios:  12
>>>
```

Figura 5: Saída do Código 4.2.
Fonte: Autoria própria(2026).

Durante a cópia do código eles perceberam a presença da linha *if subida != descida* e, o mesmo aluno que apagou as configurações não válidas na árvore, disse:

Diálogo

Aluno C : "Professor, essa nova linha que apareceu no código faz exatamente o que eu fiz na árvore."

Aluno A: "Como assim? Eu não entendi."

Aluno C: "Olha só. O programa escolhe o caminho de subida, sei lá, o caminho A. Eu subo pelo caminho A mas, agora, tenho que descer. Se eu escolher o caminho A, o programa me avisa que eu não posso fazer isso, por causa da linha *if subida != descida*, então jogo fora essa possibilidade e vou escolher outro caminho. E, isso a gente faz para cada opção de caminho de subida."

Com essa análise, feita pelo aluno, os itens f) e g) foram respondidos.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 4.2.

g) Observe o Código 4.2, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo da linha de código *if subida != descida*?

4.1.2 Análise da Atividade 2: Restrições, a poda da árvore

A **Atividade 2** introduz um elemento fundamental no estudo da combinatória: a restrição. Ao propor que "a pessoa não queira descer o morro pela estrada que subiu", o problema exige que os estudantes revisitem e modifiquem a estrutura que haviam acabado de consolidar. O objetivo pedagógico avança de simples enumeração de casos independentes para a compreensão de como restrições afetam o Conjunto de Resultados e como isso se traduz na representação visual (a árvore) quanto na lógica de programação (as estruturas condicionais no Python).

A análise a seguir está dividida em dois episódios centrais que evidenciam como a manipulação física da representação e a leitura do código atuaram na consolidação desta nova etapa do raciocínio combinatório.

Episódio 1: O Impacto da Restrição na Árvore de Possibilidades

O primeiro grande acontecimento desta atividade ocorre antes mesmo de qualquer discussão verbalizada: a atitude do *Aluno C* de ir ao quadro e apagar os galhos da árvore que representavam a subida e descida pela mesma estrada, declarando que "a resposta é 12". Sob a perspectiva do Modelo dos Campos Semânticos, esse acontecimento possui um peso analítico formidável. Na **Atividade 1**, o *Aluno C* havia acionado resíduos de enunciação do seu cotidiano para impor a restrição de não repetir o caminho. Naquele momento, ocorreu um atrito de leituras, e ele precisou deslocar seu raciocínio para atender ao enunciado do problema. Agora, na **Atividade 2**, o "texto-problema" valida a sua leitura inicial. Por isso, a ação dele é tão imediata: o campo semântico necessário para interpretar essa restrição já estava plenamente constituído. Ele não precisa recalcular; ele simplesmente materializa a restrição apagando os galhos da árvore, produzindo um significado de "retirada" ou "poda" das opções não válidas.

Pela lente do Modelo de Lockwood, a ação de apagar os galhos ilustra uma intervenção direta no Conjunto de Resultados. Os alunos demonstram compreender que a restrição afeta a natureza das configurações aceitáveis - eliminando os pares (A,A), (B,B), (C,C), (D,D). Ao intervir na árvore de possibilidades, o aluno está, na verdade, reestruturando o Processo de Contagem visual. A transição mental é clara: partindo do conjunto anterior de 16 possibilidades, a eliminação visual dos quatro galhos inválidos conduz imediatamente ao novo resultado$(16 - 4 = 12)$. Há uma articulação fluida e quase instantânea entre a modificação do Processo de Contagem e a determinação do novo Conjunto de Resultados.

Episódio 2: A Sintaxe Computacional como Espelho da Ação Física

O segundo momento central emerge durante a análise do código 4.2, especificamente quando os alunos se deparam com a estrutura condicional *if subida != descida:*. O diálogo que se segue revela a ancoragem do pensamento algorítmico na experiência vivenciada:

Aluno C: "Professor, essa nova linha que apareceu no código faz exatamente o que eu fiz na árvore."

Aluno A: "Como assim? Eu não entendi."

Aluno C: "Olha só. O programa escolhe o caminho de subida, sei lá, o caminho *A*. Eu subo pelo caminho *A* mas, agora, tenho que descer. Se eu escolher o caminho *A*, o programa me avisa que eu não posso fazer isso, por causa da linha *if subida != descida*, então jogo fora essa possibilidade e vou escolher outro caminho. E, isso a gente faz para cada opção de caminho de subida."

Pelo Modelo dos Campos Semânticos, observamos que o *Aluno C* assume o papel de mediador para a *Aluno A*. Para interpretar a sintaxe do operador relacional de diferença(*!=*) e da condicional *if* - texto estruturalmente novo -, o *Aluno C* não recorre ao jargão técnico da programação. Pelo contrário, ele produz significado ancorando a lógica do código na sua própria ação física prévia: a poda da árvore ("faz exatamente o que eu fiz"). Ele chega a personificar o algoritmo ("o programa me avisa", "jogo fora"), criando uma leitura procedimental que torna o código transparente. A programação deixa de ser um emaranhado de regras abstratas e passa a representar uma ação intencional com a qual o aluno já estabeleceu intimidade semântica.

Concomitantemente, sob o Modelo de Lockwood, este diálogo consolida a equivalência entre os Processos de Contagem manual e computacional. O *Aluno C* explica com clareza o funcionamento do processo algorítmico de filtragem: enquanto os laços de repetição (for) geram o Conjunto de Resultados completo, a condicional (if) atua ativamente para testar e descartar as configurações que violam a restrição do problema. O estudante demonstra coordenar de forma madura a lógica computacional com a listagem final das possibilidades, evidenciando a robustez da aprendizagem combinatória construída nesta etapa.

Sob a perspectiva do Modelo dos Campos Semânticos, conclui-se que o sucesso da **Atividade 2** ocorreu, fundamentalmente, pela legitimação do conhecimento prévio. O fato de o novo problema exigir a não repetição de caminhos permitiu que o aluno ancorasse a matemática escolar em seu próprio referencial cotidiano. Isso nos leva a inferir que a superação de obstáculos epistemológicos em sala de aula é potencializada quando o "texto-

problema" não reprime a produção de significado do estudante, mas oferece um contexto formal onde essa leitura se torna operante e correta.

E, através do Modelo de Lockwood, é possível concluir que os alunos participantes ultrapassaram a fase de simples enumeração para atingir uma compreensão estrutural avançada do Conjunto de Resultados. A fluidez com que os alunos traduziram a ação física e visual de intervir na árvore de possibilidades para a interpretação da sintaxe condicional no Python revela um amadurecimento cognitivo importante. O código deixou de ser interpretado como uma caixa preta de regras arbitrárias e passou a ser dominado como um Processo de Contagem algorítmico e dinâmico, desenhado especificamente para filtrar o Conjunto de Resultados.

Portanto, a principal tese extraída deste episódio é que a integração do pensamento computacional no ensino de combinatória atinge seu potencial analítico máximo quando o algoritmo é apresentado como um espelho da ação lógica do próprio sujeito. Ao permitir que a restrição computacional fosse lida através da ação física de "podar" ramificações, a sequência didática garantiu uma apropriação profunda do conceito, provando que os alunos não estão apenas calculando possibilidades, mas compreendendo e manipulando ativamente as estruturas que as compõem.

***Atividade* 3:** Considere o seguinte problema: "Uma moeda é lançada três vezes. Qual o número de sequências possíveis *cara* e *coroa*?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

Diálogo

Aluno B: "O objeto é a moeda."

Professor: "Mas, o que acontece quando eu lanço essa moeda?"

Aluno B: "Eu vejo se saiu Cara ou Coroa, quando ela cai."

Professor: "Então, quais são os objetos básicos do problema?"

Aluno B: "Os lados da moeda."

Aluno A: "Então, já sei quantos são os resultados possíveis, 6, 3×2. São 3 lançamentos e 2 lados."

Professor: "Calma! Vamos ver o que queremos contar primeiro. Quais são as configurações desejadas?"
Aluno C: "Eu fiz aqui, professor. Deu cara cara cara, cara cara coroa, cara coroa cara, cara coroa coroa, coroa cara cara, coroa cara coroa, coroa coroa cara e coroa coroa coroa. Deu 8. Então, não são apenas 6."
Professor: "Ótimo! Então, cada configuração é formada pelos resultados do 1º, 2º e 3º lançamentos. Vamos, ao item b)."

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Diálogo
Aluno A: "Eu não tomo decisão nenhuma, quem decide é a sorte. E a restrição é que a moeda não pode cair em pé."
Professor: "Entendo a sua resposta. Mas, aqui o termo decisão é usado para designar as etapas de um processo de escolha."
Aluno A: "Ok. Então, são três decisões? Escolher o resultado do 1º lançamento, que pode ser cara ou coroa, 2 opções. Depois, o resultado do 2º, que também tem 2 opções, cara ou coroa. E, finalmente, o resultado do 3º, que também pode ser cara ou coroa, 2 opções também. E, não vai ter restrição."
Professor: "Ok. Agora, vamos ver se são 8 sequências possíveis. Para isso, vamos montar a árvore de possibilidades e, depois, vamos responder ao item d)."

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Desenhamos a árvore de possibilidades no quadro.

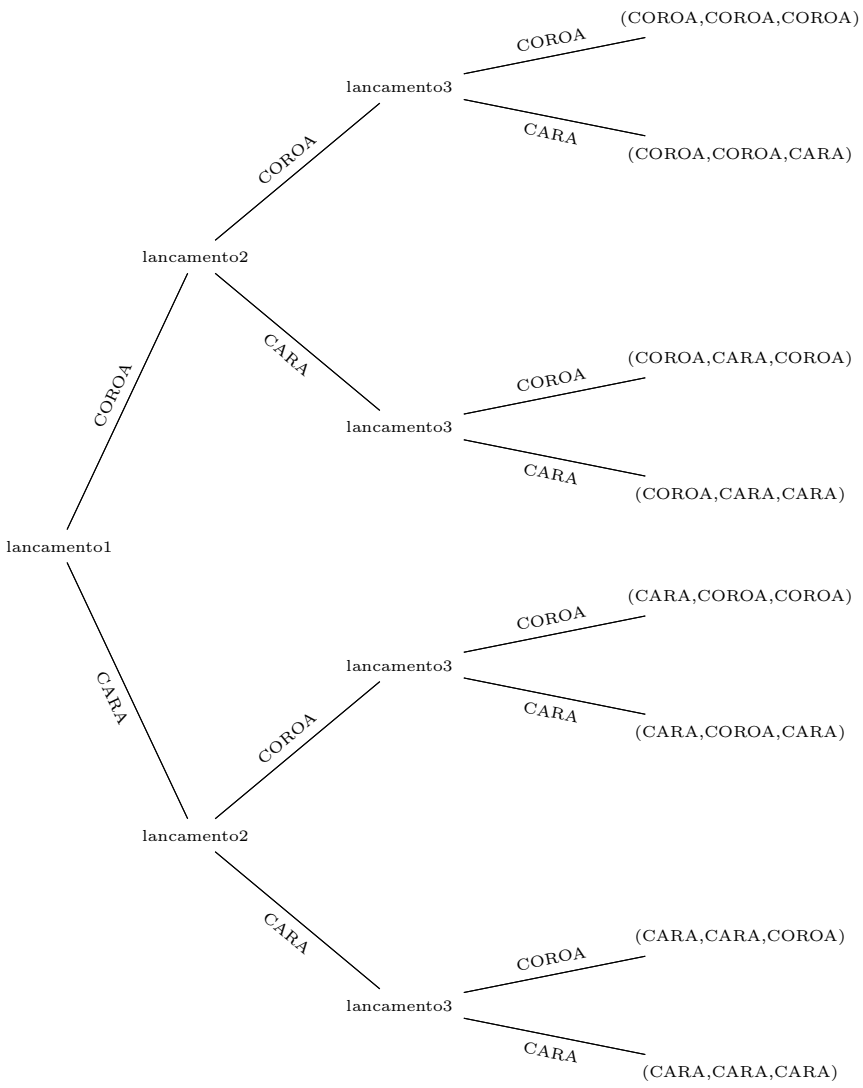


Figura 6: Árvore de Possibilidades da Atividade 3.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

> *Aluno C:* Pela árvore, eu contei 8 pontas, então são 8 configurações, como eu fiz. Faz sentido porque são 3 decisões de 2 opções, então a conta é $2 \times 2 \times 2 = 8$."
> *Professor:* "Perfeito. Vamos, agora ao código. Copiem o código e execute-o."

e) Agora, execute o seguinte código em Python:

```
moeda = ['CARA','COROA']
contagem = 0

print('Resultados',':  ' )

for lancamento1 in moeda:
    for lancamento2 in moeda:
```

```
8         for lancamento3 in moeda:
9             print('(',lancamento1, ',',lancamento2, ',', lancamento3,')')
10            contagem = contagem + 1
11
12 print('Quantidade de sequências: ', contagem)
```

Código 4.3: Código da Atividade 3

Diálogo

Aluno A: "O código é bem parecido com o código da Atividade 1. Tem mais uma linha com o *for* pois, agora, tem mais uma decisão."

Obtemos o seguinte resultado ao executar o Código 4.3.

```
Resultados :
( CARA , CARA , CARA )
( CARA , CARA , COROA )
( CARA , COROA , CARA )
( CARA , COROA , COROA )
( COROA , CARA , CARA )
( COROA , CARA , COROA )
( COROA , COROA , CARA )
( COROA , COROA , COROA )
Quantidade de sequências:  8
```

Figura 7: Saída do Código 4.3.
Fonte: Autoria própria(2026).

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 4.3.

Diálogo

Aluno A: "É claro, que obtemos as configurações e a quantidade igual à árvore. Professor, se eu colocar mais uma linha *for* no código, terei uma sequência com os resultados possíveis em quatro lançamentos?"

Professor: "Isso. Cresceríamos nossa árvore incluindo mais uma decisão. E, teríamos quantos sequências possíveis?"

Aluno A: "16. Ficaria $2 \times 2 \times 2 \times 2$."

Professor: "Então, quais serão as respostas para o item g)?"

g) Observe o Código 4.3, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

Diálogo

Aluno A: "Tem 3 *for* um dentro do outro. O primeiro *for* é o primeiro lançamento, e para cada face que posso ter no 1º lançamento, o segundo *for* roda as faces da moeda no 2º lançamento, e assim por diante. Cada laço é uma daquelas 3 decisões."

Finalizamos o Primeiro Encontro, enunciando o Princípio Fundamental da Contagem, conforme encontramos em (CERIOLI; VIANA, 2012):

Princípio Fundamental da Contagem(PFC). *Seja A um conjunto finito. Se cada elemento de A pode ser formado pela tomada de n decisões, $d_1, d_2, \ldots, d_n$, de maneira que:*

- *a decisão d_1 pode ser tomada de m_1 maneiras;*
- *para cada maneira como a decisão d_1 pode ser tomada, a decisão d_2 pode ser tomada de m_2 maneiras;*
- *para cada maneira como as decisões d_1 e d_2 podem ser tomadas (nesta ordem), a decisão d_3 pode ser tomada de m_3 maneiras;*

  $\ldots$;
- *para cada maneira como as decisões $d_1, d_2, \ldots, d_{i-1}$ podem ser tomadas (nesta ordem), a decisão d_i pode ser tomada de m_i maneiras;*

  $\ldots$;
- *para cada maneira como as decisões $d_1, d_2, \ldots, d_{i-1}, d_i, \ldots, d_{n-1}$ podem ser tomadas (nesta ordem), a decisão d_n pode ser tomada de m_n maneiras.*

Então, $n(A) = m_1 \times m_2 \times m_3 \times \ldots \times m_i \times \ldots \times m_{n-1} \times m_n$, onde $n(A)$ indica a quantidade de elementos do conjunto A.

4.1.3 Análise da Atividade 3: A Formalização do Princípio Fundamental da Contagem

A **Atividade 3** propõe um deslocamento significativo no cenário da modelagem: a transição de um problema de trajetos físicos para um experimento aleatório clássico, o

lançamento de uma moeda três vezes. O objetivo pedagógico desta etapa é verificar se a estrutura multiplicativa construída nas atividades anteriores possui solidez suficiente para ser transposta a um novo contexto, culminando na capacidade de abstração e na formalização do Princípio Fundamental da Contagem.

Para investigar a apropriação desse conhecimento e os desafios inerentes a essa transposição contextual, a análise a seguir organiza-se em dois episódios centrais.

Episódio 1: A Semântica do "Acaso" e o Embate na Estruturação dos Resultados

O início das discussões revela o retorno de tensões semânticas importantes, evidenciando que a mudança de contexto traz consigo novos resíduos de enunciação. Logo no item a), o *Aluno A* lança o seguinte resíduo de enunciação equivocado, tentando encurtar o processo de contagem: "já sei quantos são os resultados possíveis, 6, 3×2." Em seguida, no item b), ao ser questionado sobre as decisões necessárias para resolver o problema, o mesmo aluno argumenta: "Eu não tomo decisão nenhuma, quem decide é a sorte. E a restrição é que a moeda não pode cair em pé".

Pela ótica do Modelo dos Campos Semânticos, essas falas ilustram perfeitamente o atrito entre a linguagem cotidiana e o contrato didático da matemática. O cálculo $3 \times 2 = 6$ demonstra uma produção de significado ancorada na simples manipulação dos dados numéricos visíveis no enunciado - três lançamentos e duas faces, caracterizando uma leitura superficial. Contudo, o obstáculo epistemológico mais revelador surge na interpretação da palavra decisão. O aluno mobiliza o campo semântico de sua vivência: no mundo real, lançar uma moeda é o ato deliberado de delegar uma escolha ao acaso; logo, a ação humana inexiste nesse processo. A nossa intervenção atua como um mediador essencial para renegociar esse termo, deslocando-o do sentido de "arbítrio" para o sentido matemático de "etapa de um processo combinatório". A menção à moeda "cair em pé" reforça essa leitura que considera o contexto e a possível "ideia prática" do mundo físico, que precisa ser contornada para a abstração matemática.

A partir da perspectiva do Modelo de Lockwood, a superação desse obstáculo semântico é o que permite a correta estruturação do problema. Enquanto o *Aluno A* se equivocava na formulação de uma Expressão prematura (3×2), o *Aluno C* adotou a estratégia de base recomendada pelo modelo: ele construiu o Conjunto de Resultados de forma empírica e exaustiva ("Deu cara cara cara, cara cara coroa ..."), provando que o total era 8. Uma vez que o professor clarifica o termo decisão, os alunos conseguem, finalmente, estruturar o Processo de Contagem, dividindo o processo em três decisões independentes

com duas opções cada.

Episódio 2: A Árvore, o Código e a Abstração Algorítmica

Neste episódio evidencia-se a consolidação da aprendizagem através da articulação entre a árvore de possibilidades e o código Python, culminando na capacidade de generalização dos estudantes. Ao observar a árvore construída no quadro, o *Aluno C* realiza uma síntese estrutural notável: "Pela árvore, eu contei 8 pontas, então são 8 configurações (...) Faz sentido porque são 3 decisões de 2 opções, então a conta é $2 \times 2 \times 2 = 8$." Nesta fala, o Modelo de Lockwood materializa-se em sua totalidade. O estudante transita com autonomia pelas três instâncias cognitivas: ele valida seu Conjunto de Resultados inicial através do Processo de Contagem visual - a árvore de possibilidades - e deduz a Expressão Matemática correta que rege o fenômeno.

Sob a lente do Modelo dos Campos Semânticos, a subsequente introdução do código Python não gera mais o estranhamento sintático ou visual observado na **Atividade 1**. O *Aluno A* lê a estrutura de repetição com total fluência procedimental: "Tem 3 *for* um dentro do outro. O primeiro *for* é o primeiro lançamento (...) Cada laço é uma daquelas 3 decisões." A prova definitiva dessa apropriação conceitual ocorre quando instigamos uma generalização hipotética para quatro lançamentos. O *Aluno A*, prescindindo do apoio visual de uma nova árvore ou da execução de um novo código, projeta mentalmente o algoritmo ampliado e enuncia: "16. Ficaria $2 \times 2 \times 2 \times 2$.

A análise da **Atividade 3** encerra este ciclo de intervenções demonstrando como os estudantes evoluíram de um raciocínio aditivo e dependente da representação espacial para o domínio abstrato e generalizável da estrutura multiplicativa. A intersecção teórica adotada revela-se fundamental para a compreensão desse percurso. O Modelo dos Campos Semânticos explica como o avanço cognitivo não decorreu da imposição de fórmulas matemáticas, mas da negociação ativa de significados e da superação de atritos de linguagem. Simultaneamente, o Modelo de Lockwood mapeia a arquitetura dessa vitória de aprendizagem: a turma abandonou a busca por operações aritméticas superficiais para coordenar, de forma consciente e integrada, o Processo de Contagem, o Conjunto de Resultados e as Expressões.

Por fim, vale destacar o impacto positivo da inserção do pensamento computacional. O código em Python não serviu apenas para conferir a resposta final, mas funcionou como um modelo lógico manipulável pelos estudantes. Quando eles perceberam que cada etapa de escolha do problema correspondia a um laço de repetição *for*, e que colocar um laço dentro do outro multiplicava as possibilidades totais, o salto cognitivo aconteceu.

Esse nível de compreensão fez com que o Princípio Fundamental da Contagem surgisse como uma conclusão natural das investigações da própria turma, e não como uma fórmula pronta imposta pelo professor.

4.2 Segundo encontro

Tivemos que fazer modificações para o Segundo Encontro, pois o ano letivo estava terminando. Neste Encontro propomos a Atividade 9 presente na Sequência Didática naquele que seria o Terceiro Encontro e as Atividades 14 e 15 presentes naquele que seria o Quarto Encontro.

Objetivos: As atividades propostas no segundo encontro têm como objetivos principais: aplicar o Princípio Fundamental da Contagem na resolução de problemas e corrigir a superestimação construída pelo Princípio Fundamental da Contagem, quando da resolução de problemas cujos elementos do Conjunto de Resultados são Combinações Simples.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 14:** Vamos ao seguinte problema: "Considere 5 pessoas A, B, C, D e E. Quantas filas com 3 dessas 5 pessoas podemos formar?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

Diálogo

Aluno A: "Acho que os objetos básicos são as pessoas. E as configurações são as filas formadas."

Aluno B: "Professor, entendi quais são os objetos e quais são as configurações. Mas, fiquei com uma dúvida, se eu colocar o A, o B e o C na fila, o D e o E ficam de fora. A fila fica incompleta."

Professor: "Mas, o problema pede uma fila com quantas pessoas?"

Aluno B: "Com 3 pessoas. Mas por que o problema me deu 5 pessoas se eu só vou usar 3? Na matemática, se o problema dá um número, a gente tem que usar ele na conta final. Não é justo deixar o D e o E de fora, eles também fazem parte do problema."

Professor: "Entendo. Imagine numa corrida com 5 atletas, como o A, B, C, D e E. Quantos sobem no pódio no final?"
Aluno B: "Só três. O primeiro, o segundo e o terceiro lugar."
Professor: "E os outros dois?"
Aluno B: "Eles correm, mas não sobem no pódio. Ficam sem medalha."
Professor: "Perfeito. O nosso problema da fila é como esse pódio. Nós temos 5 candidatos, mas só 3 "vagas" na fila. Vamos ao item b)."

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Diálogo
Aluno A: "Acho que o código na letra c) resolve o problema, então como tem três *for*, o problema tem três decisões. E faz sentido, já que a fila vai ter três pessoas."
Aluno B: "Então, na primeira posição, tenho 5 pessoas para escolher. Na segunda, tenho 5. E na terceira, tenho 5. Logo, vamos ter $5 \times 5 \times 5 = 125$ filas."
Aluno A: "Essa conta tá errada! Uma pessoa não pode ocupar dois lugares na fila. Na primeira posição, temos 5 pessoas para escolher. Na segunda, temos 4. E na terceira, temos 3. Não é isso professor?"
Professor: "É isso, perfeito. Essa é a restrição do problema. Logo, temos 5 opções de escolha para ser a primeira da fila, escolhida a primeira da fila, restam 4 opções de escolha para ser a segunda da fila e, escolhida a segunda da fila, restam 3 opções de escolha para ser a última da fila. Agora, copiem o código e executem no celular."

c) Execute o seguinte código em Python:

```
def formar_filas(pessoas):

    print("\n" + "=" * 60)
    print("PASSO 1: Forma as filas.")
    print("=" * 60)

    resultado = []
    contador_filas = 0

    for pessoa_1 in pessoas:
```

```
        for pessoa_2 in pessoas:
            if pessoa_2 != pessoa_1:
                for pessoa_3 in pessoas:
                    if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:
                        contador_filas += 1
                        fila_formada = [pessoa_1,pessoa_2,pessoa_3]
                        resultado.append(fila_formada)
                        print(f"Fila {contador_filas:3d}: {fila_formada}")

    print(f"\nTotal de filas diferentes: {contador_filas}")
    return resultado



pessoas = ['A','B','C','D','E']

filas_obtidas = formar_filas(pessoas)
```

Código 4.4: Código da Atividade 14

Na Figura 8 temos a saída após a execução do código.

```
============================================================
PASSO 1: Forma as filas.
============================================================
Fila   1: ['A', 'B', 'C']
Fila   2: ['A', 'B', 'D']
Fila   3: ['A', 'B', 'E']
Fila   4: ['A', 'C', 'B']
Fila   5: ['A', 'C', 'D']
Fila   6: ['A', 'C', 'E']
Fila   7: ['A', 'D', 'B']
Fila   8: ['A', 'D', 'C']
Fila   9: ['A', 'D', 'E']
Fila  10: ['A', 'E', 'B']
Fila  11: ['A', 'E', 'C']
Fila  12: ['A', 'E', 'D']
Fila  13: ['B', 'A', 'C']
Fila  14: ['B', 'A', 'D']
Fila  15: ['B', 'A', 'E']
Fila  16: ['B', 'C', 'A']
Fila  17: ['B', 'C', 'D']
Fila  18: ['B', 'C', 'E']
Fila  19: ['B', 'D', 'A']
Fila  20: ['B', 'D', 'C']
Fila  21: ['B', 'D', 'E']
Fila  22: ['B', 'E', 'A']
Fila  23: ['B', 'E', 'C']
Fila  24: ['B', 'E', 'D']
Fila  25: ['C', 'A', 'B']
Fila  26: ['C', 'A', 'D']
Fila  27: ['C', 'A', 'E']
Fila  28: ['C', 'B', 'A']
Fila  29: ['C', 'B', 'D']
Fila  30: ['C', 'B', 'E']
Fila  31: ['C', 'D', 'A']
Fila  32: ['C', 'D', 'B']
Fila  33: ['C', 'D', 'E']
Fila  34: ['C', 'E', 'A']
Fila  35: ['C', 'E', 'B']
Fila  36: ['C', 'E', 'D']
Fila  37: ['D', 'A', 'B']
Fila  38: ['D', 'A', 'C']
Fila  39: ['D', 'A', 'E']
Fila  40: ['D', 'B', 'A']
Fila  41: ['D', 'B', 'C']
Fila  42: ['D', 'B', 'E']
Fila  43: ['D', 'C', 'A']
Fila  44: ['D', 'C', 'B']
Fila  45: ['D', 'C', 'E']
Fila  46: ['D', 'E', 'A']
Fila  47: ['D', 'E', 'B']
Fila  48: ['D', 'E', 'C']
Fila  49: ['E', 'A', 'B']
Fila  50: ['E', 'A', 'C']
Fila  51: ['E', 'A', 'D']
Fila  52: ['E', 'B', 'A']
Fila  53: ['E', 'B', 'C']
Fila  54: ['E', 'B', 'D']
Fila  55: ['E', 'C', 'A']
Fila  56: ['E', 'C', 'B']
Fila  57: ['E', 'C', 'D']
Fila  58: ['E', 'D', 'A']
Fila  59: ['E', 'D', 'B']
Fila  60: ['E', 'D', 'C']

Total de filas diferentes: 60
```

Figura 8: Saída do Código 4.4.
Fonte: Autoria própria(2026).

d) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

Diálogo

Aluno C: "Sim, obtemos as filas. E, são 60 filas. Caramba! A árvore vai ficar grande, não vai caber no meu caderno."

Professor: "Já fizemos a árvore. Não perceberam? O código que vocês executaram montou a árvore dentro dele. Alguém disse isso na Atividade 1."

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

Diálogo

Aluno B: "Agora é fazer a conta, eu tenho 5 opções para a primeira posição. Como não posso repetir, sobram 4 opções para a segunda. E depois 3 opções para a terceira. Fazendo $5 \times 4 \times 3 = 60$. Bateu com o total do programa!"

f) Esse problema é parecido com algum problema que já foi resolvido? Qual?

Não apresentamos esse item.

g) E quantas filas com 4 pessoas podemos formar?

Diálogo

Aluno B: "Fácil! Temos quatro decisões a tomar. Na quarta decisão, temos 2 opções de escolha. Pelo Princípio Fundamental da Contagem, podemos formar $5 \times 4 \times 3 \times 2 = 120$ filas com quatro pessoas."

4.2.1 Análise da Atividade 14: A semântica da "sobra", o contrato didático e a árvore invisível

A **Atividade 14** introduz uma nova nuance na modelagem combinatória dos estudantes. Embora a ideia de selecionar elementos e "deixar outros de fora" já estivesse presente na forma implícita em problemas anteriores, a grande atração nesta atividade reside na formulação explícita dos tamanhos do conjunto formado pelos objetos básicos e de cada elemento do Conjunto de Resultados no próprio enunciado: "5 pessoas" para formar filas com "3".

Para analisar os resíduos de enunciação desta atividade, dividiremos as discussões em dois episódios centrais.

Episódio 1: A Quebra do Contrato Didático e a Legitimação da "Sobra"

O primeiro grande obstáculo revelado pelos diálogos não é aritmético, mas semântico e interpretativo. No item a), o *Aluno B* evidencia um profundo desconforto com a modelagem do problema: "Mas por que o problema me deu 5 pessoas se eu só vou usar 3? Na matemática, se o problema dá um número, a gente tem que usar ele na conta final. Não é justo deixar o D e o E de fora ...".

Sob a lente do Modelo dos Campos Semânticos, essa enunciação é bastante interessante por revelar o choque entre dois campos semânticos. O primeiro é aquele que é um tipo "contrato didático tradicional": o estudante cristalizou a "regra escolar" de que todos os dados numéricos explícitos em um problema devem ser diretamente operados (somados, multiplicados, etc.) para produzir o resultado. O segundo campo ativado é o da vivência social: diferentemente de "estradas" não escolhidas, a ideia de deixar "pessoas" de fora de uma fila desperta um esquema de exclusão que o aluno acha que é "injusto".

A nossa intervenção, nesse caso, em vez de apelar para o rigor de uma definição de "sublistas", introduz um novo texto ancorado na vivência dos alunos: o cenário de uma corrida ("Quantos sobem no pódio?"). Ao deslocar o problema para uma metáfora esportiva, conseguimos legitimar o significado da exclusão. O *Aluno B* produz imediatamente uma nova enunciação válida: "Eles correm, mas não sobem no pódio. Ficam sem medalha". A resistência ao modelo matemático é superada pela atribuição de sentido prático à "sobra" dos elementos.

Episódio 2: A Árvore Internalizada no Algoritmo

Agora a barreira é a estruturação do Conjunto de Resultados. No item b), o *Aluno B* tenta aplicar diretamente a ideia de decisões sucessivas, mas negligencia a restrição de dependência, propondo a operação $5 \times 5 \times 5 = 125$. Imediatamente, o *Aluno A* refuta: "Essa conta tá errada! Uma pessoa não pode ocupar dois lugares na fila...".

Através do Modelo de Lockwood, observamos aqui a autocorreção mediada pelos pares. O *Aluno B* havia formulado uma Expressão prematura, baseada na repetição irrestrita. O *Aluno A*, contudo, recorre à natureza física do Conjunto de Resultados - uma pessoa é indivisível - para reestruturar o Processo de Contagem: as opções decrescem a cada decisão tomada. Esse movimento prova que os alunos não estão apenas multiplicando números aleatoriamente, mas avaliando constantemente se a Expressão reflete a realidade

das configurações desejadas.

O auge do amadurecimento algorítmico do grupo, no entanto, ocorre após a execução do código. Ao ver as 60 filas impressas na tela, o *Aluno C* exclama: "Caramba! A árvore vai ficar grande, não vai caber no meu caderno", ao que nós respondemos: "O código que vocês executaram montou a árvore dentro dele".

Pelo Modelo dos Campos Semânticos, esta interação demonstra a total fluidez e internalização das representações. O aluno já não precisa desenhar a árvore de possibilidades para saber que ela existe e qual é o seu formato; ele consegue visualizá-la mentalmente e projeta suas dimensões inviáveis no papel. O algoritmo em Python, com seus três laços for aninhados e seus condicionais, assume definitivamente o lugar do diagrama manual.

Finalmente, no item g), quando questionado sobre filas com 4 pessoas, o *aluno B* generaliza o Princípio Fundamental da Contagem de forma autônoma: "Fácil! Temos 4 decisões a tomar... $5 \times 4 \times 3 \times 2 = 120$ filas". Sob a ótica de Lockwood, a Expressão Matemática está agora perfeitamente alinhada e validada, demonstrando que a turma dominou o raciocínio de agrupamentos ordenados sem precisar memorizar nenhuma fórmula pré-fabricada.

A análise da **Atividade 14** permite concluir que a introdução de problemas de agrupamentos parciais ordenados impõe aos estudantes obstáculos que extrapolam a complexidade aritmética. O desafio central desta atividade não residiu na execução do cálculo em si, mas na necessidade de romper com expectativas implícitas da cultura escolar e de lidar com restrições que afetam a estrutura do Conjunto de Resultados.

Sob a perspectiva do Modelo dos Campos Semânticos, conclui-se que o "contrato didático" atua, muitas vezes, como um fator de resistência ao raciocínio combinatório. A crença cristalizada de que todos os dados numéricos de um enunciado devem ser operados gerou um bloqueio inicial. É notável que a superação desse atrito não se deu por meio da imposição de uma definição formal de agrupamento, mas sim pela ancoragem em uma metáfora social (o pódio de uma corrida). Isso nos leva a deduzir que a legitimação do modelo matemático em sala de aula depende profundamente da capacidade do professor de ofertar contextos onde a exclusão ou a "sobra"de elementos faça sentido prático e afetivo para o aluno.

Concomitantemente, através do Modelo de Lockwood, é possível observar um amadurecimento estrutural expressivo. A rápida autocorreção mediada pelos pares — do cálculo irrestrito $(5 \times 5 \times 5)$ para a expressão decrescente $(5 \times 4 \times 3)$ — comprova que os estudantes

mantêm o Processo de Contagem ativamente subordinado à realidade física do Conjunto de Resultados. Eles compreendem que as restrições da vida real (a impossibilidade de uma pessoa ocupar dois lugares) devem ditar a forma da Expressão Matemática.

Por fim, a principal tese consolidada nesta atividade é a da internalização algorítmica. A exclamação de que "a árvore vai ficar grande" ao observar o código em Python demonstra que a representação visual, antes desenhada no quadro, converteu-se em um modelo cognitivo interno. Os estudantes não precisam mais ver os galhos para saber como eles se multiplicam ou onde são podados pela restrição (if). Essa fluidez representacional é o que os capacitou a generalizar o Princípio Fundamental da Contagem para novos agrupamentos de forma autônoma (como o cálculo para 4 pessoas), provando que um conceito (o de Arranjo Simples) foi construído com significado, dispensando o uso prematuro de fórmulas prontas.

***Atividade* 9:** Considere o seguinte código em Python que resolve um determinado problema de contagem e execute-o:

```
letras = ['N','U','M','E','R','O']
anagramas = []

for letra_1 in letras:
    for letra_2 in letras:
            if letra_2 != letra_1:
                for letra_3 in letras:
                    if letra_3 != letra_2 and letra_3 != letra_1:
                        for letra_4 in letras:
                            if letra_4 != letra_3 and letra_4 != letra_2 and letra_4 !=
    letra_1:
                                for letra_5 in letras:
                                    if letra_5 != letra_4 and letra_5 != letra_3 and
    letra_5 != letra_2 and letra_5 != letra_1:
                                        for letra_6 in letras:
                                            if letra_6 != letra_5 and letra_6!= letra_4
    and letra_6 != letra_3 and letra_6 != letra_2 and letra_6 != letra_1:
                                                anagrama = (letra_1 + letra_2 + letra_3 +
     letra_4 + letra_5 + letra_6)
                                                anagramas.append(anagrama)


colunas = 20
largura_coluna = 20

print(f"\nAnagramas da palavra NÚMERO: ")
for i, anagrama in enumerate(anagramas):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(anagramas) % colunas != 0:
```

```
29      print()
30
31  print('Quantidade de anagramas: ', len(anagramas))
```

Código 4.5: Código da Atividade 9

a) Pesquise o que são anagramas.

> As várias ordenações de letras de uma determinada palavra são chamadas de **Anagramas**. Seriam como palavas, não necessariamente com sentido, que podem ser formadas a partir de um dado conjunto de letras.(SPREAFICO; SILVA, 2021)

b) Quais são os objetos básicos do problema?

> **Diálogo**
> *Aluno A:* "Os objetos básicos são as letras separadas: N, U, M, E, R, O. O código até coloca elas separadas numa lista logo na primeira linha. Igual nos códigos dos problemas anteriores."
> *Professor:* "Exato. As letras são nossos objetos básicos. E o que seriam as configurações formadas, que a próxima pergunta pede?"

c) Quais são as configurações formadas?

> **Diálogo**
> *Aluno A:* "Ah, as configurações são as palavras estranhas que vamos formar, os anagramas!"

d) Quantas decisões foram tomadas para resolver o problema? Quais foram essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão foi tomada?

Diálogo

Professor: "Vamos olhar para a estrutura do código. Quantas decisões foram tomadas para formar cada anagrama e de quantas maneiras podemos tomar cada uma?"

Aluno B: "Foram tomadas 6 decisões, porque tem 6 *for*. E como a lista *letras* tem 6 letras, cada decisão pode ser tomada de 6 maneiras. A gente escolhe 6 letras sempre."

Aluno C: "Mas se eu posso escolher de 6 maneiras todas as vezes, eu poderia escolher a letra "**R**" seis vezes e formar a palavra "RRRRRR". Isso não é um anagrama da palavra **NÚMERO**."

Aluno A: "Não pode! O primeiro *if* no código diz *letra_ 2 != letra_ 1.* Ou seja, a segunda letra tem que ser diferente da primeira."

Aluno B: "Ah, entendi. Então a primeira decisão tem 6 opções. A segunda decisão não tem 6, tem 5, porque não pode ser igual à primeira. E a terceira vai ter 4 opções, e assim vai."

Professor: "Vamos juntar todas as informações.

Primeiro, observando que há seis linhas de comando *for*, seis decisões serão tomadas.

Segundo, dadas as letras N,U,M,E,R,O, temos as seguintes decisões: 1ª) Escolher a 1ª letra do anagrama(*for letra_ 1 in letras*); 2ª) Escolher a 2ª letra do anagrama(*for letra_ 2 in letras*); 3ª) Escolher a 3ª letra do anagrama(*for letra_ 3 in letras*); 4ª) Escolher a 4ª letra do anagrama(*for letra_ 4 in letras*); 5ª) Escolher a 5ª letra do anagrama(*for letra_ 5 in letras*) e 6ª) Escolher a 6ª letra do anagrama(*for letra_ 6 in letras*).

Terceiro, temos uma restrição no problema, cada letra só pode ser usada uma única vez. Basta ver as linhas *if* presentes no código.

Então, temos para a 1ª decisão: 6 maneiras de escolha, pois antes de iniciarmos a formação do anagrama, temos 6 letras disponíveis em *letras*; 2ª decisão: 5 maneiras de escolha, pois após a 1ª decisão ter sido realizada, para qualquer letra escolhida na 1ª decisão , temos 5 letras disponíveis(*if letra_ 2 != letra_ 1:*); 3ª decisão: 4 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª e 2ª decisões, temos 4 letras disponíveis(*if letra_ 3 != letra_ 2 and letra_ 3 != letra_ 1:*); e, assim por diante."

e) Quantas configurações foram obtidas? Com isso, em que consiste o problema resol-

vido pelo Código 4.5?

> Dessa vez executei o código no meu celular e mostrei aos alunos o resultado abaixo.
>
> **Diálogo**
>
> *Professor:* "O código rodou e nos deu a quantidade total de anagramas da palavra *NÚMERO*. Mas como podemos calcular isso no papel, usando o Princípio Fundamental da Contagem, sem precisar rodar o Python?"

```
Anagramas da palavra NÚMERO:
NUMERO,NUMEOR,NUMREO,NUMROE,NUMOER,NUMORE,NUEMRO,NUEMOR,NUERMO,NUEROM,NUEOMR,NUEORM,NURMEO,NURMOE,NUREMO,NUREOM,NUROME,NUROEM,NUOMER,NUOMRE,
NUOEMR,NUOERM,NUORME,NUOREM,NMUERO,NMUEOR,NMUREO,NMUROE,NMUOER,NMUORE,NMEURO,NMEUOR,NMERUO,NMEROU,NMEOUR,NMEORU,NMRUEO,NMRUOE,NMREUO,NMREOU,
NMROUE,NMROEU,NMOUER,NMOURE,NMOEUR,NMOERU,NMORUE,NMOREU,NEUMRO,NEUMOR,NEURMO,NEUROM,NEUOMR,NEUORM,NEMURO,NEMUOR,NEMRUO,NEMROU,NEMOUR,NEMORU,
NERUMO,NERUOM,NERMUO,NERMOU,NEROUM,NEROMU,NEOUMR,NEOURM,NEOMUR,NEOMRU,NEORUM,NEORMU,NRUMEO,NRUMOE,NRUEMO,NRUEOM,NRUOME,NRUOEM,NRMUEO,NRMUOE,
NRMEUO,NRMEOU,NRMOUE,NRMOEU,NREUMO,NREUOM,NREMUO,NREMOU,NREOUM,NREOMU,NROUME,NROUEM,NROMUE,NROMEU,NROEUM,NROEMU,NOUMER,NOUMRE,NOUEMR,NOUERM,
NOURME,NOUREM,NOMUER,NOMURE,NOMEUR,NOMERU,NOMRUE,NOMREU,NOEUMR,NOEURM,NOEMUR,NOEMRU,NOERUM,NOERMU,NORUME,NORUEM,NORMUE,NORMEU,NOREUM,NOREMU,
UNMERO,UNMEOR,UNMREO,UNMROE,UNMOER,UNMORE,UNEMRO,UNEMOR,UNERMO,UNEROM,UNEOMR,UNEORM,UNRMEO,UNRMOE,UNREMO,UNREOM,UNROME,UNROEM,UNOMER,UNOMRE,
UNOEMR,UNOERM,UNORME,UNOREM,UMNERO,UMNEOR,UMNREO,UMNROE,UMNOER,UMNORE,UMENRO,UMENOR,UMERNO,UMERON,UMEONR,UMEORN,UMRNEO,UMRNOE,UMRENO,UMREON,
UMRONE,UMROEN,UMONER,UMONRE,UMOENR,UMOERN,UMORNE,UMOREN,UENMRO,UENMOR,UENRMO,UENROM,UENOMR,UENORM,UEMNRO,UEMNOR,UEMRNO,UEMRON,UEMONR,UEMORN,
UERNMO,UERNOM,UERMNO,UERMON,UERONM,UEROMN,UEONMR,UEONRM,UEOMNR,UEOMRN,UEORNM,UEORMN,URNMEO,URNMOE,URNEMO,URNEOM,URNOME,URNOEM,URMNEO,URMNOE,
URMENO,URMEON,URMONE,URMOEN,URENMO,URENOM,UREMNO,UREMON,UREONM,UREOMN,URONME,URONEM,UROMEN,UROMNE,UROENM,UROEMN,UONMER,UONMRE,UONEMR,UONERM,
UONRME,UONREM,UOMNER,UOMNRE,UOMENR,UOMERN,UOMRNE,UOMREN,UOENMR,UOENRM,UOEMNR,UOEMRN,UOERNM,UOERMN,UORNME,UORNEM,UORMNE,UORMEN,UORENM,UOREMN,
MNUERO,MNUEOR,MNUREO,MNUROE,MNUOER,MNUORE,MNEURO,MNEUOR,MNERUO,MNEROU,MNEOUR,MNEORU,MNRUEO,MNRUOE,MNREUO,MNREOU,MNROUE,MNROEU,MNOUER,MNOURE,
MNOEUR,MNOERU,MNORUE,MNOREU,MUNERO,MUNEOR,MUNREO,MUNROE,MUNOER,MUNORE,MUENRO,MUENOR,MUERNO,MUERON,MUEONR,MUEORN,MURNEO,MURNOE,MURENO,MUREON,
MURONE,MUROEN,MUONER,MUONRE,MUOENR,MUOERN,MUORNE,MUOREN,MENURO,MENUOR,MENRUO,MENROU,MENOUR,MENORU,MEUNRO,MEUNOR,MEURNO,MEURON,MEUONR,MEUORN,
MERNUO,MERNOU,MERUNO,MERUON,MERONU,MEROUN,MEONUR,MEONRU,MEOUNR,MEOURN,MEORNU,MEORUN,MRNUEO,MRNUOE,MRNEUO,MRNEOU,MRNOUE,MRNOEU,MRUNEO,MRUNOE,
MRUENO,MRUEON,MRUONE,MRUOEN,MRENUO,MRENOU,MREUNO,MREUON,MREONU,MREOUN,MRONUE,MRONEU,MROUNE,MROUEN,MROENU,MROEUN,MONUER,MONURE,MONEUR,MONERU,
MONRUE,MONREU,MOUNER,MOUNRE,MOUENR,MOUERN,MOURNE,MOUREN,MOENUR,MOENRU,MOEUNR,MOEURN,MOERNU,MOERUN,MORNUE,MORNEU,MORUNE,MORUEN,MORENU,MOREUN,
ENUMRO,ENUMOR,ENURMO,ENUROM,ENUOMR,ENUORM,ENMURO,ENMUOR,ENMRUO,ENMROU,ENMOUR,ENMORU,ENRUMO,ENRUOM,ENRMUO,ENRMOU,ENROUM,ENROMU,ENOUMR,ENOURM,
ENOMUR,ENOMRU,ENORUM,ENORMU,EUNMRO,EUNMOR,EUNRMO,EUNROM,EUNOMR,EUNORM,EUMNRO,EUMNOR,EUMRNO,EUMRON,EUMONR,EUMORN,EURNMO,EURNOM,EURMNO,EURMON,
EURONM,EUROMN,EUONMR,EUONRM,EUOMNR,EUOMRN,EUORNM,EUORMN,EMNURO,EMNUOR,EMNRUO,EMNROU,EMNOUR,EMNORU,EMUNRO,EMUNOR,EMURNO,EMURON,EMUONR,EMUORN,
EMRNUO,EMRNOU,EMRUNO,EMRUON,EMRONU,EMROUN,EMONUR,EMONRU,EMOUNR,EMOURN,EMORNU,EMORUN,ERNUMO,ERNUOM,ERNMUO,ERNMOU,ERNOUM,ERNOMU,ERUNMO,ERUNOM,
ERUMNO,ERUMON,ERUONM,ERUOMN,ERMNUO,ERMNOU,ERMUNO,ERMUON,ERMONU,ERMOUN,ERONUM,ERONMU,EROUNM,EROUMN,EROMNU,EROMUN,EONUMR,EONURM,EONMUR,EONMRU,
EONRUM,EONRMU,EOUNMR,EOUNRM,EOUMNR,EOUMRN,EOURNM,EOURMN,EOMNUR,EOMNRU,EOMUNR,EOMURN,EOMRNU,EOMRUN,EORNUM,EORNMU,EORUNM,EORMNU,EORMUN,
RNUMEO,RNUMOE,RNUEMO,RNUEOM,RNUOME,RNUOEM,RNMUEO,RNMUOE,RNMEUO,RNMEOU,RNMOUE,RNMOEU,RNEUMO,RNEUOM,RNEMUO,RNEMOU,RNEOUM,RNEOMU,RNOUME,RNOUEM,
RNOMUE,RNOMEU,RNOEUM,RNOEMU,RUNMEO,RUNMOE,RUNEMO,RUNEOM,RUNOME,RUNOEM,RUMNEO,RUMNOE,RUMENO,RUMEON,RUMONE,RUMOEN,RUENMO,RUENOM,RUEMNO,RUEMON,
RUEONM,RUEOMN,RUONME,RUONEM,RUOMNE,RUOMEN,RUOENM,RUOEMN,RMNUEO,RMNUOE,RMNEUO,RMNEOU,RMNOUE,RMNOEU,RMUNEO,RMUNOE,RMUENO,RMUEON,RMUONE,RMUOEN,
RMENUO,RMENOU,RMEUNO,RMEUON,RMEONU,RMEOUN,RMONUE,RMONEU,RMOUNE,RMOUEN,RMOENU,RMOEUN,RENUMO,RENUOM,RENMUO,RENMOU,RENOUM,RENOMU,REUNMO,REUNOM,
REUMNO,REUMON,REUONM,REUOMN,REMNUO,REMNOU,REMUNO,REMUON,REMONU,REMOUN,REONUM,REONMU,REOUNM,REOUMN,REOMNU,REOMUN,RONUME,RONUEM,RONMUE,RONMEU,
RONEUM,RONEMU,ROUNME,ROUNEM,ROUMNE,ROUMEN,ROUENM,ROUEMN,ROMNUE,ROMNEU,ROMUNE,ROMUEN,ROMENU,ROMEUN,ROENUM,ROENMU,ROEUNM,ROEUMN,ROEMNU,ROEMUN,
ONUMER,ONUMRE,ONUEMR,ONUERM,ONURME,ONUREM,ONMUER,ONMURE,ONMEUR,ONMERU,ONMRUE,ONMREU,ONEUMR,ONEURM,ONEMUR,ONEMRU,ONERUM,ONERMU,ONRUME,ONRUEM,
ONRMUE,ONRMEU,ONREUM,ONREMU,OUNMER,OUNMRE,OUNEMR,OUNERM,OUNRME,OUNREM,OUMNER,OUMNRE,OUMENR,OUMERN,OUMRNE,OUMREN,OUENMR,OUENRM,OUEMNR,OUEMRN,
OUERNM,OUERMN,OURNME,OURNEM,OURMNE,OURMEN,OURENM,OUREMN,OMNUER,OMNURE,OMNEUR,OMNERU,OMNRUE,OMNREU,OMUNER,OMUNRE,OMUENR,OMUERN,OMURNE,OMUREN,
OMENUR,OMENRU,OMEUNR,OMEURN,OMERNU,OMERUN,OMRNUE,OMRNEU,OMRUNE,OMRUEN,OMRENU,OMREUN,OENUMR,OENURM,OENMUR,OENMRU,OENRUM,OENRMU,OEUNMR,OEUNRM,
OEUMNR,OEUMRN,OEURNM,OEURMN,OEMNUR,OEMNRU,OEMUNR,OEMURN,OEMRNU,OEMRUN,OERNUM,OERNMU,OERUNM,OERUMN,OERMNU,OERMUN,ORNUME,ORNUEM,ORNMUE,ORNMEU,
ORNEUM,ORNEMU,ORUNME,ORUNEM,ORUMNE,ORUMEN,ORUENM,ORUEMN,ORMNUE,ORMNEU,ORMUNE,ORMUEN,ORMENU,ORMEUN,ORENUM,ORENMU,OREUNM,OREUMN,OREMNU,OREMUN,
Quantidade de anagramas:  720
```

Figura 9: Saída do Código 4.5.
Fonte: Autoria própria(2026).

f) Utilizando o Princípio Fundamental da Contagem, obtenha a solução do problema resolvido pelo Código 4.5.

> **Diálogo**
>
> *Aluno A:* "Os comandos *if* faz com que as nossas opções de escolha a cada nova posição vão diminuindo... A primeira posição tinha 6, a segunda 5, depois 4... . Então no Princípio Fundamental da Contagem a gente multiplica essas opções, né? Ficaria $6 \times 5 \times 4 \times 3 \times 2 \times 1$, que dá 720. O mesmo número que o código imprimiu no final!"

4.2.2 Análise da Atividade 9: Anagramas e a Leitura Matemática do Algoritmo

A **Atividade 9** propõe a investigação dos anagramas da palavra NÚMERO. O objetivo pedagógico avança para o estudo das permutações simples, onde a restrição de não repetição atua sobre a totalidade dos elementos disponívies. Nesta etapa metodológica invertemos a apresentação: em vez de os alunos construírem o código do zero, eles são convidados a ler e interpretar um algoritmo pronto repleto de laços de repetição *for* e estruturas condicionais *if* para, a partir dele, extrair a lógica combinatória.

A análise das interações obtidas nesta atividade será dividida em dois episódios centrais.

Episódio 1: O Choque Interpretativo e a Avaliação pelo Conjunto de Resultados

O diálogo mais rico ocorre no item d), quando os estudantes são desafiados a identificar as decisões e restrições do problema olhando para a estrutura do código. Inicialmente, o *Aluno B* faz uma leitura direta e quantitativa dos laços de repetição: "Foram tomadas 6 decisões, porque temos 6 *for*... cada decisão pode ser tomada de 6 maneiras." Sob a ótica do Modelo de Lockwood, o *Aluno B* estrutura um Processo de Contagem, que geraria a Expressão 6^6. Contudo, o erro é imediatamanente refutado pelo *Aluno C* através de um mecanismo essencial previsto pelo modelo: a avaliação baseada no Conjunto de Resultados. Ao argumentar que "eu poderia escolher a letra 'R' seis vezes e formar a palavra 'RRRRRR'. Isso não é um anagrama ...", o *Aluno C* materializa uma configuração inválida para provar a falha no processo de contagem do colega. Ele não refuta a conta com outra conta, mas com a visualização concreta do objeto matemático que está sendo formado.

Pela lente do Modelo dos Campos Semânticos, esse embate ilustra uma correta renegociação de significados . O *Aluno B* produziu focando apenas na sintaxe dos laços *for*. O *Aluno C*, por sua vez, acionou o campo semântica da palavra "anagrama" - cujo significado exige a permutação das letras originais, sem repetições extras - para impor um limite lógico. A ancoragem final que resolve o conflito é fornecida pelo *Aluno A*, que traduz a restrição teórica para a linguagem do algoritmo: "Não pode! O primeiro *if* no código diz *letra_ 2 != letra_ 1*". Nesse instante, a sintaxe relaconal do Python *!=* adquire pleno significado matemático para o grupo.

Episódio 2: A Tradução da Sintaxe Computacional para a Expressão Ma-

temática

O segundo episódio revela a consolidação da aprendizagem através da síntese do cálculo. No item f), instigado pelo professor a resolver o problema no papel usando o Princípio Fundamental da Contagem, o *Aluno A* enuncia a solução demonstrando total domínio estrutural: "Os comandos *if* faz com que as nossas opções de escolha a cada nova posição vão diminuindo... A primeira posição tinha 6, a segunda 5, depois 4... Ficaria $6 \times 5 \times 4 \times 3 \times 2 \times 1$, que dá 720."

Por meio do Modelo de Lockwood, observa-se que a Expressão Matemática não surge como uma tentativa de adivinhar uma fórmula (como $P_n = n!$), mas como uma consequência natural e justificada do Processo de Contagem. A expressão reflete perfeitamente a diminuição de opções a cada nova etapa de seleção.

Sob o olhar do Modelo dos Campos Semânticos, a leitura que o *Aluno A* faz do código é reveladora. Ele não lê o *if* apenas como um filtro que o computador usa para esconder resultados errados; ele produz um significado operatório para o comando, entendendo que o *if* altera a própria árvore de possibilidades - "faz com que as nossas opções de escolha (...) vão diminuindo". O código deixou de ser uma "caixa preta" que joga o número 720 para se tornar uma vitrine transparente do Princípio Fundamental da Contagem em ação.

***Atividade* 15:** Agora, vamos a esse problema: "Considere 5 pessoas A, B, C, D e E. Quantos grupos com 3 dessas 5 pessoas podemos formar?". Responda as questões a seguir, antes de resolvê-lo.

Diálogo

Aluno A: "De novo o mesmo problema? Esse problema é igual ao que resolvemos no começo da aula."

Aluno B: "Deve ter alguma pegadinha aí!"

Professor: "Leiam o problema com atenção e vejam se encontram alguma coisa diferente."

Aluno C: "Achei! Agora aparece a palavra "grupos" ao invés de "filas"."

Aluno A: "Mas, não é a mesma coisa?"

Professor: "Calma! Vamos responder as questões abaixo. Depois voltamos para essa discussão."

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

Diálogo

Professor: "Lendo o item (a) dessa atividade, o que muda em relação ao problema da primeira atividade?"

Aluno A: "Não muda nada. Os objetos básicos continuam sendo as 5 pessoas, e as configurações desejadas são juntar 3 delas para formar os grupos."

Professor: "Todos concordam?"

Aluno B: "Tem pegadinha com certeza."

Professor: "Então, para vocês, a solução do itens (b) e (c) é igual ao do 1º problema de hoje?"

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) A solução desse problema é igual à solução do problema da Atividade 14? Ou seja, a quantidade de filas é igual à quantidade de grupos? Justifique.

Diálogo

Aluno A: "Sim. Se eu escolher o A, o B e o C para formar um grupo, eu tenho que tomar 3 decisões de novo. Na primeira pessoa do grupo eu tenho 5 opções, depois 4, depois 3. Vai dar 60 de novo. É o mesmo problema com palavras diferentes."

Aluno B: "Espera, achei a pegadinha. Pensa num trabalho de escola. Se o professor disser "o grupo é João, Samuel e Yasmin" ou disser "o grupo é Yasmin, Samuel e João", você vai fazer o trabalho com as mesmas pessoas, não vai?"

Aluno A: "É, o trabalho é com os mesmos três."

Aluno B: "No 1º problema, a fila do banco mudava porque quem estava na frente era atendido primeiro. No grupo de estudo, não tem primeiro nem último."

Professor: "Perfeito! Agora, precisamos corrigir a nossa resposta, não são 60 grupos."

d) Se a resposta do item anterior for NÃO, corrija a resposta, obtendo, assim, a resposta correta.

Diálogo

Aluno C: "A minha cabeça tá quase explodindo. Eu acho que a gente tem que subtrair as repetições. Se deu 60, e eu sei que a fila ABC e a fila CBA representam o mesmo grupo, eu tiro as cópias. Deve ser 60 menos umas 10 repetições."

Aluno A: "Vamos escrever todas as repetições para o grupo formado por A, B e C para ver quantas são."

Aluno C: "Beleza. Tem ABC, ACB, BAC, BCA, CAB, CBA. São 6 jeitos de escrever o mesmo grupo."

Aluno B: "Vai dar um trabalho danado! Vamos pegar as filas do 1º problema e ver quais representam o mesmo grupo."

Aluno A: "Vamos lá!"

Deixei os alunos separarem as filas que representam o mesmo grupo. Depois de um tempo, veio a resposta.

Aluno C: "Professor, a gente fez aqui. Deu 10 grupos no total."

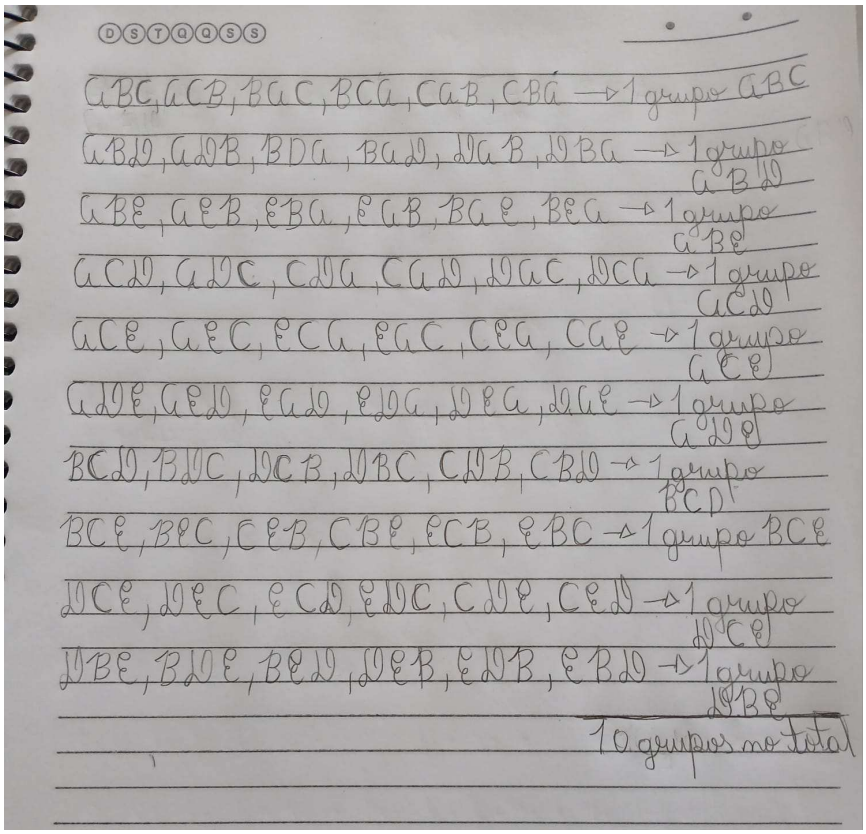


Figura 10: Resposta obtida pelos alunos para a atividade.
Fonte: Autoria própria(2026).

Aluno B: "Só não entendi ainda, o motivo de ter dado 10. Também achava que tinha que subtrair."
Aluno A: "Acho que entendi o porquê. Para cada grupo verdadeiro que a gente quer contar, a nossa conta de 60 contou ele 6 vezes, então tenho que dividir os 60 por 6. Aí, o resultado é 10."
Professor: "É isso! Agora, de onde vem esse 6?"
Aluno B: "Só piora!"
Professor: "Revejam o problema anterior, talvez ajude."
Aluno C: "O 6 é a quantidade de anagramas da palavra formada pelas iniciais dos "caras" que tão no grupo."
Aluno B: "Então, é a quantidade de jeitos que a gente tem de embaralhar as pessoas de um grupo que a gente formou. É isso?"
Professor: "Perfeito! E, se nós tivéssemos 7 pessoas e quiséssemos formar grupos com 4 pessoas. Quantos grupos poderemos formar?"
Aluno A: "Calma aí. Tem que fazer aqui."
Aluno A: "São 4 decisões. Então, temos $7 \times 6 \times 5 \times 4$ maneiras de formar filas com 4 pessoas. Que dá 840 filas."
Aluno B: "Mas, tem que ver quantas filas representam o mesmo grupo, certo?"
Professor: "Isso. E são quantas?"
Aluno B: "Como cada grupo tem 4 pessoas, é só fazer $4 \times 3 \times 2 \times 1$. Que é igual a 24."
Aluno A: "Agora, é só dividir 840 por 24, que dá, peraí... 35 grupos. Pronto!"

4.2.3 Análise da Atividade 15: A Ordem e a Superestimação

Na **Atividade 15** introduzimos o conceito fundamental das Combinações Simples. O desafio matemático central consiste em modelar a formação de agrupamentos não ordenados, situação em que a aplicação direta do Princípio Fundamental da Contagem gera uma superestimação do Conjunto de Resultados. Cabe ressaltar que, por limitações de tempo durante a intervenção, esta foi a única atividade em que os alunos não executaram o algoritmo em Python.

A análise desta atividade será dividida em dois episódios centrais.

Episódio 1: O Atrito Semântico e a Transição da Subtração para a Divisão

O primeiro choque cognitivo emerge quando os estudantes percebem que o cálculo

multiplicativo, o qual operava perfeitamente na **Atividade 14**, agora produz um resultado incorreto.

Pela lente do Modelo dos Campos Semânticos, a superação desse conflito ocorre por meio de uma renegociação de significados ancorada no cotidiano. O *Aluno B* estabelece a diferença estrutural entre agrupamentos ordenados e não ordenados recorrendo a vivências sociais: "Pensa num trabalho de escola. Se o professor disser o grupo é João, Samuel e Yasmin ou (...) Yasmin, Samuel e João, você vai fazer o trabalho com as mesmas pessoas (...) No 1º problema, a fila do banco mudava porque quem estava na frente era atendido primeiro". O conceito de irrelevância da ordem ganha, assim, materialidade e legitimidade. O texto matemático é lido através do filtro da experiência escolar.

Sob a perspectiva do Modelo de Lockwood, esse momento marca o reconhecimento de que a Expressão Matemática inicial (60) superestimou o Conjunto de Resultados. A correção desse erro, no entanto, não é imediata. O *Aluno C* manifesta um raciocínio aditivo inicial, propondo que se deve subtrair as repetições - "60 menos umas 10". Para resolver a dúvida, os alunos recorrem à estratégia mais elementar e poderosa do Modelo de Lockwood: o retorno à linguagem física do Conjunto de Resultados. Ao escreverem à mão as permutações do grupo *ABC* na Figura 10, eles visualizam fisicamente que 6 arranjos colapsam em apenas 1 grupo válido. É essa constatação empírica que permite ao *Aluno A* promover a reestruturação do Processo de Contagem, abandonando a ideia de subtração para adotar a razão proporcional: "a nossa conta de 60 contou ele 6 vezes, então tenho que dividir os 60 por 6".

Episódio 2: A Ancoragem nos Anagramas e a Autonomia na Generalização

Este episódio revela como a rede de significados da sequência didática se conecta para fundamentar a divisão combinatória. Quando questionamos a origem do divisor "6", o *Aluno C* não recorre a regras arbitrárias, mas aciona o conhecimento recém-estabelecido na **Atividade 9**: "O 6 é a quantidade de anagramas da palavra formada pelas iniciais dos caras que estão no grupo". Pelo Modelo dos Campos Semânticos, a palavra "anagrama" deixa de pertencer exclusivamente ao campo linguístico - embaralhar letras - e passa a produzir significado como um operador matemático genérico de permutação - "embaralhar pessoas".

Um certo grau de maturidade estrutural é evidenciado quando propomos o problema

generalizado de escolher 4 pessoas entre 7. O grupo articula perfeitamente os três componentes do Modelo de Lockwood sem recorrer a diagramas de árvores extensos, algoritmos em Python ou fórmulas. O *Aluno A* estrutura a Expressão Matemática $7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 840$, o *Aluno B* determina o fator de correção baseado na permutação simples do agrupamento, $4 \times 3 \times 2 \times 1 = 24$, e a divisão finaliza o Processo de Contagem - $840/24 = 35$. A abstração da fórmula de combinação ocorreu de maneira puramente orgânica.

A análise da **Atividade 15** conclui os objetivos do Segundo Encontro demonstrando que a superação da superestimação combinatória é, antes de tudo, um processo de refinamento semântico e estrutural.

Pelo Modelo dos Campos Semânticos, conclui-se que o conceito de "ordem não importa" só se torna operante quando o aluno o associa em metáforas de sua vivência - o "trabalho de escola" versus "a fila do banco". O sucesso da intervenção residiu na nossa postura dialógica, que estimulou essa produção de significado em vez de impor um novo axioma.

Pelo Modelo de Lockwood, a principal tese consolidada é a de que a operação matemática de divisão, tão complexa no ensino de agrupamentos não ordenados, só ganha sentido para o aluno quando justificada pela listagem exaustiva como na Figura 10. Ao visualizarem que cada grupo real possuía "múltiplas cópias", os estudantes conseguiram coordenar o Conjunto de Resultados com um novo Processo de Contagem, estabelecendo uma rede cognitiva estável.

5 Considerações Finais

O presente trabalho propôs-se a investigar o processo de ensino e aprendizagem dos Problemas de Contagem por meio da integração do pensamento computacional, utilizando a linguagem Python. Longe de buscar a mera validação de algoritmos ou a automação de cálculos, o objetivo central desta pesquisa foi examinar com atenção como os estudantes produzem significados, superam obstáculos epistemológicos e estruturam o raciocínio multiplicativo em um ambiente onde a programação atua como ferramenta de modelagem cognitiva. Para analisar isso, unimos duas teorias: o Modelo dos Campos Semânticos e o Modelo de Lockwood.

A análise do Primeiro Encontro evidenciou a transição fundamental do raciocínio aditivo e espacial para a consolidação da estrutura multiplicativa. Observou-se que a apropriação do Princípio Fundamental da Contagem não ocorreu pela imposição vertical de axiomas. Pelo contrário, o Modelo de Lockwood permitiu constatar que os alunos coordenaram exaustivamente o Conjunto de Resultados - como a construção manual de árvores de possibilidades - com o Processo de Contagem até que a Expressão Matemática surgisse como uma generalização natural. Concomitantemente, o Modelo dos Campos Semânticos lançou luz sobre o papel mediador da linguagem: os atritos iniciais, como a interpretação da palavra “decisão” associada à “sorte” no lançamento de moedas, foram ativamente renegociados, ancorando o rigor matemático em contextos familiares aos estudantes.

No Segundo Encontro, o desafio aumentou com um problema onde a ordem de escolha não importava, cenário em que a aplicação direta do Princípio Fundamental da Contagem gera superestimação do Conjunto de Resultados. Vimos, também, que a regra escolar de “usar todos os números do enunciado” atrapalhou no início, pois os alunos achavam "injusto"deixar pessoas de fora de uma fila. Esses problemas só foram resolvidos quando trouxemos exemplos do dia a dia, fazendo-os relacionar a formação de uma fila com a formação de um pódio esportivo e entender a diferença entre uma fila de banco e um grupo de trabalho escolar.

O código em Python não serviu apenas para dar a resposta final, mas ajudou os alunos a enxergarem a árvore de possibilidades. Eles entenderam que cada laço de repetição (for) era uma decisão e cada condição (if) era uma restrição do problema. Com isso, conseguiram deduzir sozinhos como fazer os cálculos de permutação e de divisão para corrigir a contagem, sem precisar decorar regras prontas.

Retomando a questão central que norteou este estudo, evidenciou-se que a materialização algorítmica do Princípio Fundamental da Contagem em Python influenciou a produção de significado de maneira estruturante: o algoritmo ofereceu um modelo cognitivo que permitiu aos estudantes visualizarem as restrições invisíveis do Conjunto de Resultados. Sob a ótica do Modelo dos Campos Semânticos, essa materialização só se efetivou porque o código foi lido e renegociado à luz das vivências cotidianas dos alunos. Como resultado direto dessa rede de significados, a justificação para os processos de ajuste de contagem - em especial a transição para as combinações simples - deixou de ser uma regra imposta e passou a ser uma necessidade lógica compreendida e verbalizada pela própria turma.

Apesar desses resultados positivos, a aplicação das atividades na escola enfrentou dificuldades reais que prejudicaram o alcance total dos objetivos que queríamos atingir. A principal delas foi o tempo muito curto para a execução das aulas, aliado à quantidade pequena de alunos participantes. O pouco tempo impediu, por exemplo, que o código em Python fosse testado na última atividade, cortando uma etapa importante de visualização que havia sido planejada. Além disso, o baixo número de alunos diminuiu muito a riqueza das discussões e as trocas de ideias entre eles, algo que é fundamental para a negociação de significados defendida pelo Modelo dos Campos Semânticos.

Por causa dessas limitações, não foi possível explorar todo o potencial da sequência de atividades que criamos. Pensando nisso e no que aprendemos ao longo do caminho, deixamos as seguintes propostas para trabalhos futuros:

- testar toda a sequência didática proposta com um número maior de alunos e com a carga horária normal de aulas da escola. Isso permitirá investigar como a troca de ideias acontece em grupos maiores e com mais tempo para discussões, proporcionará estudar outros tópicos como Permutações com Elementos Repetidos e a resolução de Problemas de Contagem usando a recursividade;

- investigar os obstáculos e os aprendizados que surgem quando os próprios estudantes precisam escrever o código em Python do zero para resolver o problema de conta-

gem, tendo em vista que na nossa pesquisa, os alunos apenas leram, executaram e modificaram códigos prontos;

- pesquisar como preparar outros professores de matemática para utilizar a linguagem Python de forma simples em suas aulas, ajudando-os a mediar os erros e as falas dos alunos com base no Modelo dos Campos Semânticos.

Conclui-se, portanto, que ensinar combinatória mediada pela programação e pelo diálogo é permitir que o estudante deixe a posição de mero receptor de fórmulas para assumir o papel de construtor e auditor de seus próprios modelos matemáticos.

Os códigos desenvolvidos neste trabalho podem ser acessados em https://github.com/sidneyferreiragraph/Codigos_Dissertacao.git .

Referências

BARICHELLO, Leonardo. **Livro Aberto de Matemática**: Pensamento Computacional. Rio de Janeiro: IMPA-OS, 2023. Disponível em https://umlivroaberto.impa.br/producao/pensamento-computacional-2/. Acesso em: 16 maio 2026.

BATANERO, Carmen; NAVARRO-PELAYO, Virginia; GODINO, Juan D. Effect of the Implicit Combinatorial Model on Combinatorial Reasoning in Secondary School Pupils. **Educational Studies in Mathematics**, v. 32, p. 181–199, Fevereiro 1997.

BLACK, Anderson Luis Aimi. **Scratch como Ferramenta para o Ensino de Frações**: Aprendizagem Criativa e Desenvolvimento de Pilares do Pensamento Computacional. 2024. Mestrado Profissional em Matemática em Rede Nacional – Universidade Federal da Fronteira Sul, Chapecó.

BORGES, Vanessa Henriques. **Combinatória e pensamento computacional**: Conexões para a Educação Básica no Século XXI. 2021. Mestrado Profissional em Matemática em Rede Nacional – Colégio Pedro II, Rio de Janeiro.

BRACKMANN, Christian Puhlmann. **Desenvolvimento do pensamento computacional através de atividades desplugadas na educação básica**. Agosto 2017. Tese de Doutorado – Universidade Federal do Rio Grande do Sul, Porto Alegre.

BRASIL. **Lei nº 14 533, de 11 de janeiro de 2023**. Brasília, DF: [*s. n.*], 2023. Dispõe sobre Instituição da Política Nacional de Educação Digital. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2023-2026/2023/lei/l14533.htm. Acesso em: 26 maio 2026.

BRASIL. **Base Nacional Comum Curricular (BNCC)**. Brasília, 2018. Disponível em: https://basenacionalcomum.mec.gov.br.Acesso em: 16 jul. 2025.

BRASIL. **Computação - Complemento à BNCC**. Brasília, 2022. Disponível em: https://www.gov.br/mec/pt-br/escolas-conectadas/documentos/documentos. Acesso em: 26 maio 2026.

CAVALCANTE, Rogério Silva. **Aritmética com Python**. 2018. Mestrado Profissional em Matemática em Rede Nacional – Universidade Federal de Goiás, Goiânia.

CERIOLI, Márcia R.; VIANA, Petrucio. **Minicurso de Combinatória de Contagem**. II Colóquio de Matemática da Região Sul. Universidade Estadual de Londrina, PR, Abril 2012.

COQUI, Alexandre Marquez. **O Pensamento Computacional no Ensino de Volume de Sólidos Geométricos**. 2025. Mestrado Profissional em Matemática em Rede Nacional – Universidade Estadual do Norte Fluminense, Campos dos Goytacazes.

CORRÊA, Kenderson Geane. **A Inserção do Pensamento Computacional nas aulas de Matemática no Ensino Básico**. 2022. Mestrado Profissional em Matemática em Rede Nacional – Universidade Federal de São João del-Rei, São João del-Rei.

DANTE, Luiz Roberto; VIANA, Fernando. **Do Seu Jeito**: Matemática. Volume 3. São Paulo: Ática, 2024.

DIAS, Ana Isabel de Azevedo Spinola. Pensamento Computacional na Educação Matemática. *In:* KALEFF, Ana Maria Martensen Roland (Ed.). **Educação Matemática: desafios do pensamento computacional e da inclusão na formação de professores**. Divinópilis, MG: Meus Ritmos, 2024.

DURO, Mariana Lima. **Análise Combinatória e Construção de Possibilidades**: O Raciocínio Formal no Ensino Médio. 2012. Mestrado em Educação Matemática – Universidade Federal do Rio Grande do Sul, Porto Alegre.

DUTRA, Enexandro Nobre. **Análise Combinatória**: Uma Proposta para o seu Ensino na Educação Básica com Ênfase no Princípio Fundamental da Contagem. 2014. Mestrado Profissional em Matemática em Rede Nacional – Universidade Estadual de Santa Cruz, Ilhéus.

ENGLISH, Lyn D. Combinatorics and the Development of Children's Combinatorial Reasoning. **Exploring probability in school: Challenges for teaching and learning**, v. 40, p. 121–141, jan. 2005.

LINS, Romulo Campos. A diferença como oportunidade para aprender. **Processos de ensinar e aprender: sujeitos, currıculos e cultura: livro**, v. 3, p. 530–550, 2008.

LINS, Romulo Campos. Epistemologia, História e Educação Matemática: tornando mais sólidas as bases da pesquisa. **Revista de Educação Matemática**, v. 1, n. 1, p. 75–91, 1993.

LINS, Romulo Campos. Matemática, monstros, significados e educação matemática. **Educação matemática: pesquisa em movimento. São Paulo: Cortez**, p. 92–120, 2004.

LINS, Romulo Campos. O Modelo dos Campos Semânticos: Estabelecimentos e Notas de Teorizações. *In:* ANGELO, Claudia Laus *et al.* (Ed.). **Modelo dos Campos Semânticos e Educação Matemática: 20 anos de História**. São Paulo: Midiograf, 2012.

LINS, Romulo Campos. O modelo teórico dos campos semânticos: uma análise epistemológica da álgebra e do pensamento algébrico. **Revista Dynamis, Blumenau**, v. 1, n. 7, p. 29–39, 1994.

LINS, Romulo Campos. Por que discutir teoria do conhecimento é relevante para a Educação Matemática. **Pesquisa em educação matemática: concepções e perspectivas. São Paulo: Editora da UNESP**, p. 75–94, 1999.

LIRA, Gabriel Costa Borba de. **Pensamento Computacional como ferramenta no processo de ensino-aprendizagem da matemática do ensino básico.** 2024. Mestrado Profissional em Matemática em Rede Nacional – Universidade Federal da Paraíba, João Pessoa.

LOCKWOOD, Elise; CHENNE, Adaline De. Enriching Students' Combinatorial Reasoning through the Use of Loops and Conditional Statements in Python. **International Journal of Research in Undergraduate Mathematics Education**, v. 6, p. 303–346, Outubro 2020.

LOCKWOOD, Elise; GIBSON, Bryan R. Combinatorial Tasks and Outcome Listing: Examining Productive Listing among Undergraduate Students. **Educational Studies in Mathematics**, v. 91, p. 247–270, Fevereiro 2016.

LOCKWOOD, Elise; SWINYARD, Craig A.; CAUGHMAN, John S. Patterns, Sets of Outcomes, and Combinatorial Justification: Two Students' Reinvention of Counting Formulas. **International Journal of Research in Undergraduate Mathematics Education**, v. 1, p. 27–62, Abril 2015.

LOCKWOOD, Elise Nicole. **Student Approaches to Combinatorial Enumeration**: The Role of Set-Oriented Thinking. Jan. 2011. Tese de Doutorado – Portland State University, United States of America. Disponível em https://pdxscholar.library.pdx.edu/open_access_etds.

MARTIN, George E. **Counting**: The Art of Enumerative Combinatorics. United States Of America: Springer Science Business Media, 2001.

MENEZES, Nilo Ney Coutinho. **Introdução à Programação com Python**: Algoritmos e Lógica de Programação para Iniciantes. São Paulo: Novatec Editora, 2024.

MORGADO, Augusto César *et al.* **Análise Combinatória e Probabilidade**. Rio de Janeiro: SBM, 2006.

PACHECO, Karina Gomez. **Resolução de Sistemas de Equações Lineares**: O Pensamento Computacional no Escalonamento da Matriz Ampliada. 2024. Mestrado Profissional em Matemática em Rede Nacional – Universidade Federal de Santa Catarina, Florianópolis.

PAIVA, Fábio Augusto Procópio de *et al.* **Introdução a Python com Aplicações de Sistemas Operacionais**. Natal: IFRN, 2019.

PYTHON. **Python Software Foundation**. 2025. Disponível em: https://www.python.org/. Acesso em: 1 jul. 2025.

RECH, Nicole Cristine. **Pensamento Computacional**: Uma Proposta de como introduzi-lo na Educação Básica relacionando com a Matemática. 2024. Mestrado Profissional em Matemática em Rede Nacional – Universidade do Estado de Santa Catarina, Joinville.

SANTOS, João Ricardo Viola dos; LINS, Romulo Campos. Movimentos de teorizações em Educação Matemática. **Bolema: Boletim de Educação Matemática**, v. 30, n. 55, p. 325–367, 2016.

SILVA, Amarildo Melchiades da. Impermeabilização no processo de produção de significados para a Álgebra Linear. *In:* ANGELO, Claudia Laus *et al.* (Ed.). **Modelo dos Campos Semânticos e Educação Matemática: 20 anos de História**. São Paulo: Midiograf, 2012.

SPREAFICO, Elen Viviani Pereira; SILVA, Kenia Cristina Pereira. **Livro Aberto de Matemática**: Análise Combinatória. Rio de Janeiro: IMPA-OS, 2021. Disponível em https://umlivroaberto.impa.br/producao/analise-combinatoria/. Acesso em: 16 maio 2026.

STURM, Wilton. **As Possibilidades de um Ensino de Análise Combinatória sob uma Abordagem Alternativa**. 1999. Mestrado em Educação Matemática – Universidade Estadual de Campinas, Campinas.

WING, Jeannette M. Computational Thinking. **Communications of the ACM**, 49,No.3, p. 33–35, mar. 2006.

WING, Jeannette M. Computational thinking's influence on research and education for all. **Italian journal of educational technology**, Edizioni Menabò-Menabò srl, v. 25, n. 2, p. 7–14, 2017.

APÊNDICE A – Breve introdução ao Python

Este capítulo tem como principal objetivo apresentar uma brevíssima introdução à linguagem Python com aquilo que vamos utilizar na confecção dos códigos que resolverão nossos problemas de contagem. Alertamos que, antes da apresentação da linguagem Python, o professor precisa apresentar ao aluno alguma ideia sobre algoritmos; para isso, recomendamos os livros (BARICHELLO, 2023) e (DANTE; VIANA, 2024).

A.1 Por que Python?

Python é uma linguagem de programação criada pelo matemático e programador Guido van Rossum em 1991. Embora muita gente faça associação direta com cobras, por causa das serpentes píton ou pitão, a origem do nome se deve ao grupo humorístico britânico Monty Python.

Mas, por que resolvemos trabalhar com a linguagem de programação Python? Eis alguns motivos:

Seguindo Paiva *et al.* (2019), "escolhemos a linguagem Python porque é simples, fácil de aprender e rápida para codificar."

Conforme descreve Menezes (2024):

> A linguagem de programação Phyton é muito interessante como primeira linguagem de programação devido a sua simplicidade e clareza. Embora simples, é também uma linguagem poderosa, podendo ser usada para administrar sistemas e desenvolver grandes projetos. É uma linguagem clara e objetiva, pois vai direto ao ponto, sem rodeios.

Além disso:

- Python é um software multiplataforma, isto é, você pode utilizá-lo em praticamente qualquer arquitetura de computadores ou sistema operacional, como Linux

(*https://kernel.org* ou *https://ubuntu.com*), FreeBSD (*https://freebsd.org*), Microsoft Windows ou Mac OS X (*https://www.apple.com/os/macos/*);

- Python é livre, ou seja, pode ser utilizada gratuitamente, graças ao trabalho da Python Software Foundation (PYTHON, 2025) e de inúmeros colaboradores;
- Python é uma linguagem que vem crescendo nos últimos anos, ou seja, a comunidade Python não pára de crescer, logo, possui uma comunidade bem conectada, que oferece bastante suporte, algo fundamental para aqueles que estão aprendendo Python como a primeira linguagem de programação, quanto para aqueles mais experientes.

A.2 Configurando o ambiente Python

A.2.1 Python no Windows

Visite o site (PYTHON, 2025), na aba Downloads, baixe a última versão do Python, que era a versão 3.13.5, quando este texto foi escrito.

Cuidado!!! Quando for dada a opção para você desmarcar componentes opcionais, não desmaque nenhum! Alguns destes componentes podem ser úteis para você, especialmente IDLE (utilitário para fazer o papel de terminal e editor de python).

A.2.2 Python no Linux

Se você está usando uma distribuição Linux como Ubuntu, Fedora ou OpenSUSE, então é provável que você já tenha o Python instalado em seu sistema.

Para testar se o Python já está instalado em seu Linux, abra um shell (como *konsole* ou $gnome - terminal$) e então entre com o comando $python - V$ ou $python3 - V$ como mostrado abaixo:

Figura 11: Versão do Python no terminal Linux.
Fonte: Autoria própria(2026).

Se você obter uma mensagem como esta: *bash: Python: command not found*. Então o Python não está instalado. Neste caso, você tem dois meios para instalar o Python em seu sistema:

- Você pode compilar o código fonte do Python e então instalá-lo. As instruções para a compilação são informadas no website do Python (PYTHON, 2025).
- Instalar os pacotes binários usando um gerenciador de pacotes que vem com o seu sistema operacional, tal como $apt-get$ no Ubuntu/Debian e outros Linux baseados em Debian, *yum* no Fedora, etc.

Podemos, também, instalar o IDLE no Linux, o IDLE é um ambiente integrado de desenvolvimento que acompanha a instalação do interpretador Python em sistemas operacionais Windows. Para fazer isso, basta abrir uma janela de terminal e executar o comando nela:

sudo apt-get install idle

Para executá-lo basta digitar no terminal *idle*. Veja o IDLE na Figura 12.

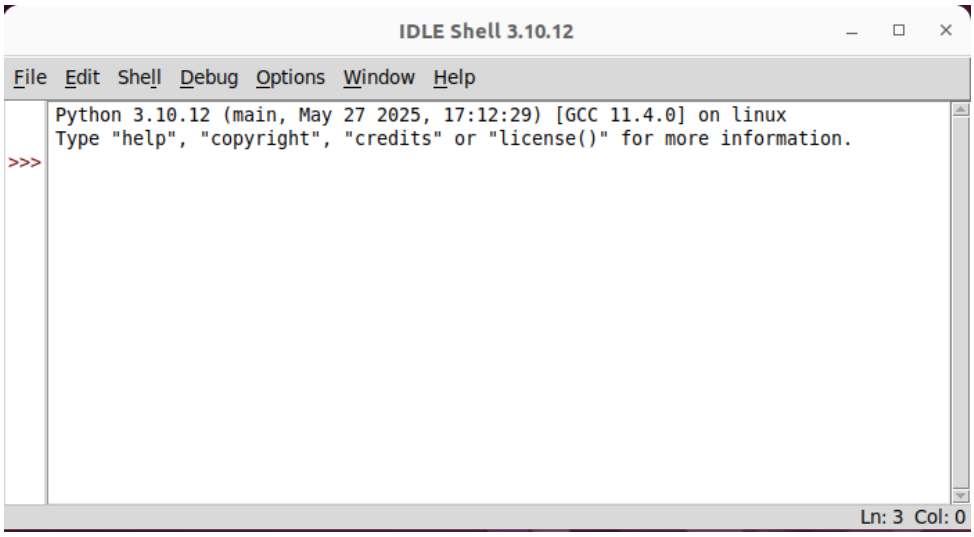


Figura 12: O ambiente IDLE.
Fonte: Autoria própria(2026).

Após instalação da linguagem Python, para que vejamos o primeiro programa funcionando, vamos escrever um "Alô Mundo!!", conforme exemplo no Código A.1.

```
1 >>> print("Alo Mundo!!")
2 Alo Mundo!!
```

Código A.1: Alô Mundo!

O programa acima foi executado no console da linguagem Python, ou seja, um local onde digitamos comandos, pressionamos ENTER e eles são executados. Para criar um módulo Python, que nada mais é do que uma sequência de comandos em um arquivo,

clique em “File”, em seguida “New File” e comece a programar, conforme Figura 14. Quando concluir o programa, pressione a tecla **F5**, nomeie o arquivo e observe o resultado do programa.

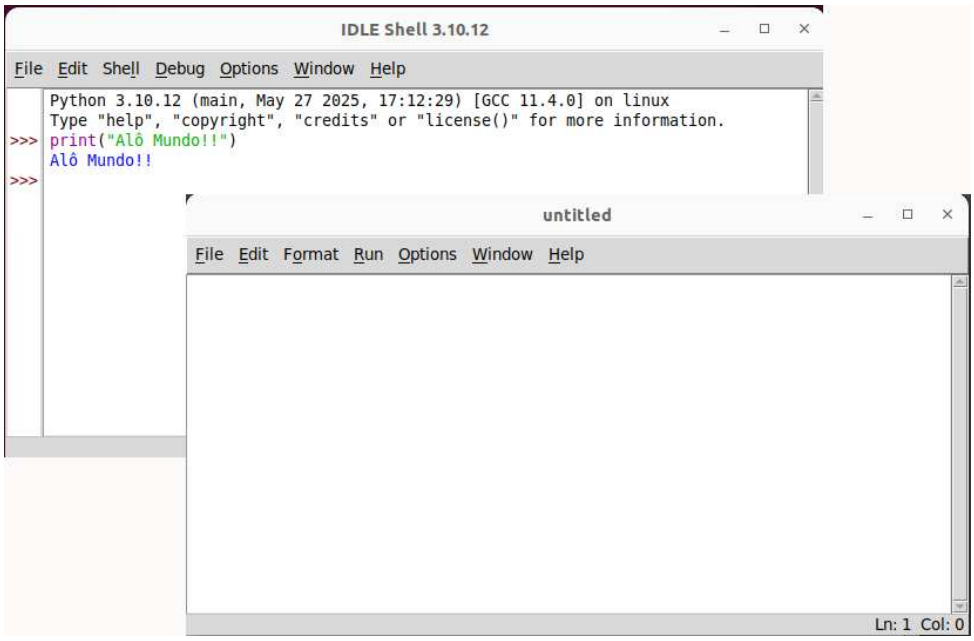


Figura 13: Local onde os arquivos Python são criados na ferramenta IDLE.
Fonte: Autoria própria(2026).

A.2.3 Python no Android

O *Pydroid 3* é uma IDE Python para dispositivos Android que permite escrever e executar código Python no celular ou tablet. O Pydroid 3 suporta Python 3.8 e conta com um editor de código simples, um emulador de terminal e um navegador web capaz de executar scripts Python.

O *Pydroid 3* possui um interpretador Python 3 offline, o que significa que você pode executar programas Python sem conexão com a internet.

Figura 14: IDE *Pydroid 3*
Fonte: https://play.google.com/store/apps/details?id=ru.iiec.pydroid3(2026).

O Pydroid 3 está disponível gratuitamente na Google Play Store.

A.3 Usando o Python como calculadora

O interpretador Python pode ser utilizado como uma poderosa calculadora.

Em Python, os símbolos das operações básicas são $+$, $-$, $*$, $/$, $**$, $//$ e $\%$. Dados a, $b \in \mathbb{R}$, vemos na Tabela 1, o que cada um desses símbolos retorna quando acionado com os números a e b. (CAVALCANTE, 2018)

Símbolo	Comando	Retorna
$+$	$a+b$	Adição
$-$	$a-b$	Subtração
$*$	$a*b$	Multiplicação
$/$	a/b	Divisão
$**$	$a**b$	Potência
$//$	$a//b$	Quaociente em $\mathbb{Z}$
$\%$	$a\%b$	Resto da divisão em $\mathbb{Z}$

Tabela 1: Operações Básicas

No Código A.2, vemos a utilização do Python como uma calculadora.

```
>>> 17 + 3 #Apos apertar ENTER teremos a SOMA na tela.
20
>>> 17 - 3 # A subtração
14
>>> 17 * 3 # A multiplicação
51
>>> 17 ** 3 # Aqui temos a potência, nesse caso 17 elevado ao cubo
4913
>>> 17/3 # A divisão
5.666666666666667
>>> 17//3 # Temos aqui o quociente da divisão inteira
5
>>> 17%3 # E aqui o resto da divisão inteira
2
```

Código A.2: Calculadora no Python

A.4 Variáveis

Conforme definido por Paiva *et al.* (2019):

> No contexto das linguagens de programação, uma *variável* é um espaço alocado na memória de um computador. O papel de uma variável é armazenar dados ou expressões que estão sendo utilizados por um programa em execução. Variáveis existem durante um determinado espaço de tempo, isto é, do início até o fim da execução do programa que as declarou.

Independentemente da linguagem de programação, na computação em geral trabalhamos com dados, e os classificamos conforme nossa necessidade. Por exemplo uma **string**, que é o termo reservado para qualquer tipo de dado alfanumérico (qualquer tipo de palavra/texto que contenha letras e números). Já quando vamos fazer qualquer operação aritmética precisamos tratar os números conforme seu tipo, por exemplo o número 8, que para programação é um **int** (número inteiro), enquanto o número 8.2 é um **float** (número com casa decimal).

Na Tabela 2 seguem os tipos de dados mais comuns que usamos em Python:

Tipo	Descrição	Exemplo
int	Número inteiro	25
float	Número com casas decimais	25.76
bol	Armazena valores lógicos (True ou False)	0(False) (ou 1(True))
string	Texto com qualquer caracter alfanumérico	'macarrão', "perigo"
list	Armazena conjuntos de elementos que podem ser acessados por meio de um índice.(Lista)	[35, "Maria", 1.69]

Tabela 2: Tipos de Dados Comuns

A.4.1 Variáveis Numéricas

Uma variável é numérica quando armazena números inteiros ou de ponto flutuante.

Os números inteiros são aqueles sem parte decimal, como 1, −8, 9999. Os números de ponto flutuante ou decimais são aqueles com parte decimal, como 4.0, 5.78987, 102.98.

Em Python, e na maioria das linguagens de programação, utiliza-se o ponto, e não a vírgula, como separador entre a parte inteira e fracionária de um número. Não utilizamos a vírgula como separador de milhar. Exemplo: 10.000.000 (dez milhões) é escrito 10000000 ou 10_000_000 (uma forma alternativa de separar números grandes é utilizar o sublinha entre os dígitos, a partir do Python 3.6) (MENEZES, 2024).

Python possui outros tipos de dados numéricos para representar números complexos (tipo complex, módulo cmath), ponto fixo (Decimal) e mesm o frações (Fraction). Se você precisar usar esses tipos de dados, verifique a documentação da linguagem.

No Código A.3 estamos manipulando e declarando variáveis numéricas.

```
1 >>> x = 7  # Declarando a variável x e atribuindo-lhe valor igual a 7
2 >>> x
```

```
7
>>> 16 + x # Operando com a variável x
23
>>> 6 ** x
279936
>>> y = 4 # Declarando a variável y
>>> y
4
>>> x * y # Operando com as variáveis x e y
28
>>> x / y
1.75
>>> x // y
1
>>> z = 3*x - 8*y # Declarando a variável z em função das variáveis x e y
>>> z
-11
>>> x + y - z
22
```

Código A.3: Operações Básicas no Python

A.4.2 Variáveis Lógicas

Muitas vezes, estamos interessados em armazenar um conteúdo simples: verdadeiro ou falso em uma variável. para isso, utilizamos um tipo de variável chamado tipo lógico ou booleano. Em Python, escreveremos **True** para verdadeiro e **False** para falso.

Para realizarmos comparações lógicas utilizamos operadores relacionais. Os operadores relacionais suportados em Python são apresentados na Tabela 3.

Operador	Operação	Símbolo matemático
==	Igualdade	$=$
>	Maior que	$>$
<	Menor que	$<$
!=	Diferente	$\neq$
>=	Maior ou igual	$\geq$
<=	Menor ou igual	$\leq$

Tabela 3: Operadores Relacionais

No Código A.4 mostramos como uma variável de tipo lógico é utilizada para armazenar o resultado de uma comparação.

```
>>> nota = 7.5 #Declarando a variável nota
>>> media = 7.0 #Declarando a variável media
>>> aprovado = nota >= media #Declarando a variável aprovado
>>> print(aprovado)
```

```
5 True
```

Código A.4: Variável Lógica no Python

Python, assim como outras linguagens, possui três operadores lógicos para realização de testes compostos. Veja quem são esses operadores, conforme Tabela **??**.

Operador	Expressão	Operação
and	X **and** Y	e
or	X **or** Y	ou
not	**not** X	não

Tabela 4: Operadores Lógicos

Assumindo T1 como Primeiro Teste e T2 como Segundo Teste, observe a tabela verdade para cada um dos operadores lógicos.

NOT	
T1	**not** T1
True	**False**
False	**True**

Tabela 5: Tabela verdade do operador lógico **not** em Python.

AND		
T1	T2	T1 **and** T2
True	**True**	**True**
True	**False**	**False**
False	**True**	**False**
False	**False**	**False**

Tabela 6: Tabela verdade do operador lógico **and** em Python.

OR		
T1	T2	T1 **or** T2
True	**True**	**True**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

Tabela 7: Tabela verdade do operador lógico **or** em Python.

No Código A.5 exemplificamos o uso dos operadores lógicos.

```
>>> #Definindo algumas variáveis
>>> a = True
>>> b = False
>>> x = 5
>>> y = 10

>>> #Usando o operador AND
>>> a and b
False
>>> x < 10 and y > 5
True

>>> #Usando o operador OR
>>> a or b
True
>>> x > 10 or y < 5
False

>>> #Usando o operador NOT
>>> not a
False
>>> not (x == y)
True

>>> #Combinação de operadores
>>> a and not b
True
>>> (x > 0) and (y < 15) or (x + y == 20)
True
```

Código A.5: Uso dos operadores lógicos no Python

A.4.3 Variáveis Strings

String é um tipo de variável que lida com uma sequência de caracteres, sejam números ou letras. Para definir uma string basta associar uma variável a qualquer sequência de caracteres entre aspas duplas (“ ”), ou simples (’ ’).

Toda string é indexada, ou seja, todos os caracteres contidos numa string possuem uma posição, o primeiro caractere tem índice 0, o segundo tem índice 1, e assim por diante. Para chamar os caracteres de uma string pelo seu índice basta acrescentar seu índice entre colchetes.

O tamanho de uma string pode ser obtido utilizando-se a função (len). Essa função retorna o número de caracteres na string.

Algumas operações com strings podem ser feitas com o “+” (também chamada de

concatenação), como também com o "∗".

Composição de strings (ou formatação de strings) em Python refere-se ao processo de criar strings dinâmicas combinando texto fixo com valores de variáveis ou expressões. Para isso, usamos os marcadores **%s**, **%d**, **%f**. Esses caracteres(marcadores) representam, respectivamente, uma string, um inteiro e um float.

O fatiamento em Python é muito poderoso e permite dividirmos uma string em pedaços menores. O fatiamento funciona com a utilização de dois pontos no índice da string. O número à esquerda dos dois pontos indica a posição de início da fatia; e o à direita, do fim.

No Código A.6 mostramos algumas "operações" com strings.

```
>>> #Declarando variáveis do tipo String
>>> nome = "Sidney Ferreira"
>>> a = "Breve"
>>> b = " Introdução"
>>> c = " Python"

>>> nome[0] # Caracter da posição 0 da variável nome
'5'
>>> nome[6] # Caracter da posição 6 da variável nome
' '
>>> nome[9] # Caracter da posição 9 da variável nome
'r'
>>> a + b + c  # Concatenação das variáveis a, b e c
'Breve Introdução Python'
>>> c * 5  # Concatenação da variável c cinco vezes
'Python Python Python Python Python'
>>> len(nome)  # Número de caracteres da variável nome
15
>>> nome[0:3] # Fatia de caracteres da variável nome da posição 0 até a posição 3, sem
    incluí-la
'Sid'
>>> nome[2:9] # Fatia de caracteres da variável nome da posição 2 até a posição 9, sem
    incluí-la
'dney Fe'
>>> nome[:4] # Fatia de caracteres da variável nome da posição 0 até a posição 4, sem
    incluí-la
'Sidn'
>>> nome[2:] # Fatia de caracteres da variável nome da posição 2 até a posição final
'dney Ferreira'
>>>
>>> cachorro = 'Bibi'
>>> racao = 1.3
>>> tempo = 6
>>> "%s comeu %f kg de ração em %d dias." %(cachorro, racao, tempo) # Composição
'Bibi comeu 1.300000 kg de ração em 6 dias.'
```

Código A.6: Strings no Python

A.4.4 Variáveis Lists

Como uma string, uma lista(**list**) é uma sequência de valores. Em uma string, os valores são caracteres; em uma lista, eles podem ser de qualquer tipo. Os valores em lista são chamados elementos.

Existem várias maneiras de criar uma nova lista; o mais simples é incluir os elementos entre colchetes ([ ]). Os elementos de uma lista não precisam ser do mesmo tipo.

Uma lista que não contém elementos é chamada de lista vazia; criamos uma lista vazia com colchetes vazios.

A sintaxe para acessar os elementos de uma lista é a mesma usada para acessar os caracteres de uma string, o operador colchete. A expressão dentro dos colchetes especifica o índice. Lembremos que os índices começam em 0.

Ao contrário das strings, as listas são mutáveis porque você pode alterar a ordem dos itens ou atribuir outro valor a um item. Quando o operador colchete aparece no lado esquerdo de uma atribuição, ele identifica o elemento da lista que será modificado.

O **range()** é uma ferramenta essencial para trabalhar com listas em Python, especialmente quando você precisa controlar índices ou gerar sequências numéricas específicas. Veremos a função **range()** melhor quando estivermos estudando a estrutura de repetição **for**.

No Código A.7 mostramos algumas "operações" com listas.

```
>>> vazio = [] # Variável vazio é uma lista vazia
>>> nomes = ['Ester', 'Guilherme', 'Gabriel', 'Emmanuel'] # Lista cujos elementos são strings
>>> idades = [1,35,37,7,8] # Lista cujos elementos são números
>>> mista = ['Sidney', 13, 45.7, [12,98]] # Lista com elementos diversificados
>>>
>>> nomes[0] # Elemento da lista nomes que ocupa a posição 0
'Ester'
>>> idades[2] = 3 # O elemento da posição 2 da lista idades será igual a 3, agora.
>>> idades
[1, 35, 3, 7, 8]
>>> sum(idades) # Soma os elementos da lista idades.
54
>>> list(range(len(nomes))) # Lista com os índices da lista nomes
[0, 1, 2, 3]
>>> c = nomes + idades # Concatenação de listas
>>> c
['Ester', 'Guilherme', 'Gabriel', 'Emmanuel', 1, 35, 3, 7, 8]
>>> idades * 6 # Concatenação da lista idade 6 vezes
[1, 35, 3, 7, 8, 1, 35, 3, 7, 8, 1, 35, 3, 7, 8, 1, 35, 3, 7, 8, 1, 35, 3, 7, 8]
```

```
>>> nomes[1:3]
['Guilherme', 'Gabriel']
>>> nomes[2:]
['Gabriel', 'Emmanuel']
```

Código A.7: Listas no Python

Python fornece métodos que operam em listas. Por exemplo:

- **append** adiciona um novo elemento ao final de uma lista;
- **extend** leva uma lista como um argumento para o método **append** e adiciona todos os elementos no final de uma lista;
- **sort** ordena os elementos da lista do menor para o maior.

Veja o funcionamento de tais métodos no Código A.8.

```
>>> x = ['f', 'g', 'a','b','c']
>>> y = ['d', 'h', 'f', 'e']
>>> x.append('d') # Adicionando o elemento d no final da lista x
>>> x
['f', 'g', 'a', 'b', 'c', 'd']
>>> y.extend(x)
>>> y
['d', 'h', 'f', 'e', 'f', 'g', 'a', 'b', 'c', 'd']
>>> y.sort() # Ordenando os elementos da lista y
>>> y
['a', 'b', 'c', 'd', 'd', 'e', 'f', 'f', 'g', 'h']
>>> x.sort()
>>> x
['a', 'b', 'c', 'd', 'f', 'g']
```

Código A.8: Métodos de Listas no Python

Existem várias maneiras de excluir elementos de uma lista. Se você sabe o índice do elemento que você quer eliminar, você pode usar **pop**. O método **pop** modifica a lista e retorna o elemento que foi removido. Se o índice não for fornecido, ele deleta e retorna o último elemento da lista.

Se você não precisar do valor removido, você pode usar o operador **del**. Se você sabe o elemento que quer remover (mas não o índice), você pode usar **remove**. Para remover mais de um elemento, você pode usar **del** com um índice de fatia.

Além do método **.sort()** o Python possui a função **sorted()**. Em **sort** você altera a lista em si, em **sorted** você tem um valor que pode utilizar em uma variável nova. Veja o funcionamento no Código A.9.

```
>>> a=[100,6,2]
>>> a.sort()
>>> a
[2, 6, 100]
>>> b=a
>>> b
[2, 6, 100]
>>> a=[100,6,2]
>>> b=sorted(a)
>>> b
[2, 6, 100]
>>>
```

Código A.9: Função **sorted()**.

No Código A.10 vemos os métodos **pop**, **del** e **remove** em funcionamento.

```
>>> x = [2,3,5,7,11,13,17,19,23,5]
>>> y = x.pop(2) # Remove o elemento da posição 2 da lista x e o guarda na variável y
>>> x
[2, 3, 7, 11, 13, 17, 19, 23, 5]
>>> y
5
>>> del x[1] #Remove o elemento da posição 1 da lista x
>>> x
[2, 7, 11, 13, 17, 19, 23, 5]
>>> del x[2:5]
>>> x
[2, 7, 19, 23, 5]
```

Código A.10: Excluindo elementos de Listas no Python.

A.5 Entrada e Saída de Dados

A forma mais comum de exibir dado é a função **print()**. Usamos essa função no Código A.1 e no Código A.11 abaixo.

```
mensagem = "Eu gosto de Combinatória."
x = 17
pi = 3.1415926535897931

print (mensagem)
print (x)
print (pi)
```

Código A.11: Saída de Dados no Python

Na Figura 15 vemos a saída do Código A.11.

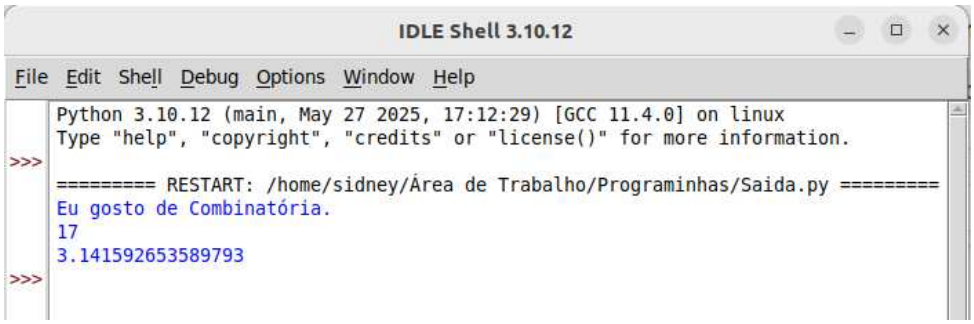


Figura 15: Saída do Código A.11
Fonte: Autoria própria(2026).

Há casos em que preferimos que o valor para a variável seja inserido pelo usuário, por meio do seu teclado. Python possui uma função embutida chamada **input** que promove a entrada de dados via teclado. Quando essa função é chamada, o programa pausa e espera o usuário digitar algo. Quando ele pressionar **Enter**, o programa volta à execução e o **input** retorna o que foi digitado como uma string. Se a intenção do usuário tenha sido digitar um número inteiro, um número real e um valor lógico basta usar as funções de conversão **int()**, **float()** e **bool()**, respectivamente.

Vejamos no Código A.12 um exemplo de utilização das funções **input** e **print**.

```
# Programa que solicita ao usuário o seu nome e dois números um inteiros
#e, após isso, imprime a soma, o produto, a diferença e o quociente deles.

nome = input("Digite seu nome:\n") #Solicita o nome do usuário
x = int(input("Digite o primeiro número:\n")) #Solicita o primeiro número
y = int(input("Digite o segundo número:\n")) #Solicita o segundo número

# A sequência \n no fim da string representa newline (nova linha),
# que é um caractere especial que causa a quebra de linha.

print("Seu nome é " + nome)
print("Você digitou os números: ", x, " e ", y)
print("A soma dos números é: ", x + y) #Informa a soma de x e y
print("O produto entre os números é: ", x * y)
print("A diferença entre o primeiro e o segundo número é: ", x - y)
print("O quociente entre o primeiro e o segundo número é: ", x / y)
```

Código A.12: Saída e Entrada de Dados no Python

Na Figura 16 vemos a execução do Código A.12.

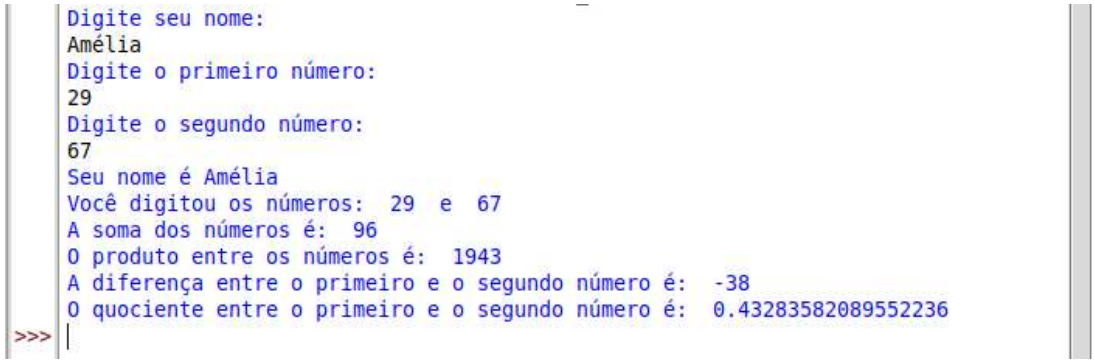


Figura 16: Execução do Código A.12
Fonte: Autoria própria(2026).

A.6 Estruturas Condicionais

Frequentemente, a programação envolve analisar um conjunto de condições e decidir qual ação deve ser executada de acordo com essas condições. *Estruturas Condicionais* nos dão essa habilidade. A forma mais simples é a declaração **if**.

A expressão booleana depois da declaração **if** é chamada de *condição*. Terminamos essa declaração com símbolo de dois pontos (:) e a(s) linha(s) depois da declaração **if** são indentadas.

Se a condição lógica é verdadeira, então a declaração indentada é executada. Se a condição lógica for falsa, a condição indentada é ignorada.

Se você digitar uma declaração **if** no interpretador Python, o prompt mudará de três chevrons para três pontos de modo a indicar que você está no meio de um bloco de declarações, como é mostrado no Código A.13:

```
1 >>> x = -4
2 >>> if x < 0:
3 ...     print ("x é negativo.")
4 ...
5 x é negativo.
6 >>>
```

Código A.13: Declaração **if** no Python

Com frequência vamos querer executar uma ação quando um teste condicional passar, e uma ação diferente em todos os demais casos. A sintaxe **if-else** do Python torna isso possível. Um bloco **if-else** é semelhante a uma instrução **if** simples, porém a instrução **else**

permite definir uma ação ou um conjunto de ações executado quando o teste condicional falhar. No Código A.14 vemos um exemplo.

```
>>> nota_final = 8.0
>>> if nota_final <= 7.0 :
...    print ("Você está reprovado.")
... else :
...    print ("Você está aprovado.")
...
...
Você está aprovado.
>>>
```

Código A.14: Declaração **if-else** no Python

Nem sempre nossos programas serão tão simples. Muitas vezes, precisaremos aninhar vários **if** para obter o comportamento desejado do programa. Aninhar, nesse caso, é utilizar um **if** dentro de outro (MENEZES, 2024).

```
tipo = int(input("Digite o tipo da mercadoria:"))
if tipo == 1:
     preco = 25
else :
     if tipo == 2:
          preco = 50
     else :
          if tipo == 3:
               preco = 75
          else :
               if tipo == 4:
                    preco = 100
               else :
                    if tipo == 5:
                         preco = 125
                    else :
                         print("Tipo inválido, digite de 1 a 5.")
                         preco = 0

print('O preço da mercadoria é: ', preco)
```

Código A.15: Declaração **if-else** no Python

Python apresenta uma solução muito interessante ao problema de múltiplos **ifs** aninhados. A cláusula **elif** substitui um par **else if** , mas sem criar outro nível de estrutura, evitando problemas de deslocamentos desnecessários à direita, como aconteceu no Código A.15. Um **if** pode ter vários **elif** , mas apenas um **else**.

Revisitamos o Código A.15, dessa vez usando **elif**. Veja o resultado no Código A.16.

```
tipo = int(input("Digite o tipo da mercadoria:"))
if tipo == 1:
     preco = 25
```

```
4  elif tipo == 2:
5         preco = 50
6  elif tipo == 3:
7         preco = 75
8  elif tipo == 4:
9         preco = 100
10 elif tipo == 5:
11        preco = 125
12 else :
13       print("Tipo inválido, digite de 1 a 5.")
14       preco = 0
15
16 print('O preço da mercadoria é: ', preco)
```

Código A.16: Código A.15 com **elif** no Python

A.7 Estruturas de Repetição

Em um programa, pode ser necessário repetir um trecho diversas vezes até que uma determinada condição seja satisfeita. Em programação, para casos como esse, são usadas estruturas conhecidas como iteração (não é interação!), repetição, laço ou loop. Python implementa duas estruturas de repetição: **while** e **for**.

Nesse trabalho só veremos a estrutura **for**.

O comando **for** no Python é um pouco diferente de outras linguagens. No Python ele percorre elementos de um variável. Por exemplo, ele pode percorrer os elementos de uma lista, as letras de uma string e etc. Para usar o comando **for** será criado uma variável auxiliar, com um nome qualquer, de tal forma que a cada iteração ela irá ser igualada a um elemento daquele objeto na ordem.

No Código A.17 vemos a execução da estrutura **for**.

```
1  >>> lista = ["a", "b", 34, "Sidney", 6543.6] # Declaração da variável lista.
2  >>> for j in lista:  #Laço for com criação da variável j.
3  ...     print(j)
4  ...
5  ...
6  a
7  b
8  34
9  Sidney
10 6543.6
11 >>>
```

Código A.17: Estrutura **for** no Python

Também é possível usar o **for** da mesma forma que em outras linguagens, onde são feitos alguns comandos por uma determinada quantidade de vezes que seja necessária. Para esse fim pode-se usar a função **range()** que gera uma sequência de números dentro de uma faixa especificada. A instrução **range(*N*)** gera uma lista com N números, onde o primeiro número é 0 e o último é $N-1$. No Código A.18 vemos o **range()** e o **for** em ação.

```
>>> range(10) #Faixa de 0 até 9
range(0,10)

>>> list(range(10)) #Lista cujos elementos estão no intervalo de 0 até 9, inclusive.
[0,1,2,3,4,5,6,7,8,9]

>>>for i in range(10): #Imprime Olá! 10 vezes.
...    print("Olá!")
...
Olá!
Olá!
Olá!
Olá!
Olá!
Olá!
Olá!
Olá!
Olá!
Olá!

>>> a = 1
>>> for j in range(11):
...     a = a*2
...     print(a)
... else:
...     print("Acabou!")
...
2
4
8
16
32
64
128
256
512
1024
2048

>>> a=1
>>> for j in range(10):
...     a=a*2
...     print(a)
... else:   #O for também possui a condição else. Essa última será executada apenas uma
    vez, assim que terminar a condição do for.
```

```
...
...     print("Acabou!")
...
...
2
4
8
16
32
64
128
256
512
1024
Acabou!
>>>
>>> x = ["Arroz", "Feijão", "Batata", "Cenoura", "Beterraba"]
>>> for i, j in enumerate(x): #Ao ser percorrida uma sequencia pode-se obter o índice da posição atual e seu valor correspondente através da função enumerate().
...
...     print(i,j)
...
...
0 Arroz
1 Feijão
2 Batata
3 Cenoura
4 Beterraba
```

Código A.18: Função **range()** e **enumerate()** no Python

A.8 Funções

Conforme define Paiva *et al.* (2019):

> [...] função é um nome para um conjunto de comandos que pode ser chamado várias vezes em pontos diferentes do programa, sem a necessidade de repetição de código. [...]Pense em uma função como sendo uma caixa preta, onde você envia um valor e recebe um resultado.

Já utilizamos várias funções em nossos programas Python. Lembre-se, por exemplo, de **input()**, de **print()**, de **range()**.

Funções são definidas usando a palavra chave **def**. Após essa palavra chave, vem um nome identificador para a função, seguido por um par de parênteses que pode incluir alguns nomes de variáveis e pelos dois pontos finais que encerram a linha. Em seguida,

vem o bloco de instruções que fazem parte dessa função. Na Figura 17, retirada de (PAIVA *et al.*, 2019), vemos como definir uma função.

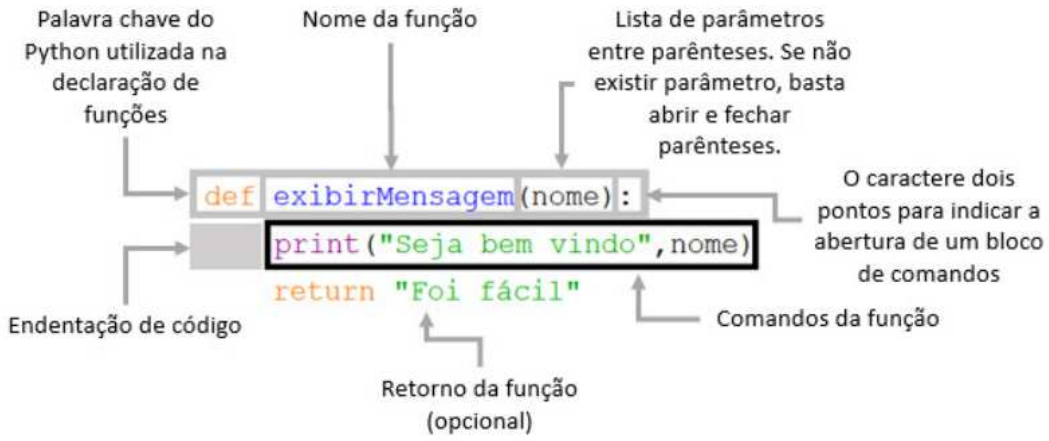


Figura 17: Definindo uma função no Python
Fonte: Paiva *et al.* (2019).

No Códigos A.19 e A.20 vemos exemplos de declarações de funções:

```
>>> def soma(a, b):
...     print(a+b)
...
>>> soma(9,12)
21
>>> soma(25,79)
104
>>>
```

Código A.19: Definindo uma função que soma dois números

```
>>> def par(a):
...     return a % 2 == 0
...
>>> def par_impar(a):
...     if par(a):
...         return "Par"
...     else:
...         return "Impar"
...
...
>>> print(par_impar(78))
Par
>>> print(par_impar(49))
Impar
>>>
```

Código A.20: Definindo uma função que verifica a paridade de um número

Uma função pode chamar a si mesma. Quando isso ocorre, temos uma função recursiva.

Um clássico exemplo de função recursiva é a função que calcula o fatorial de um número, como vemos no Código A.21.

```
>>> def fatorial(a):
...     if a == 0 or a == 1:
...         return 1
...     else:
...         return a * fatorial(a-1)
...
...
>>> print(fatorial(5))
120
>>>
```

Código A.21: Função recursiva do fatorial

Nos próximos capítulos utilizaremos, conforme a necessidade, o que vimos aqui sobre a linguagem Python com o intuito de resolver os problemas de contagem propostos.

APÊNDICE B – Gabarito das Atividades

Nesse capítulo apresentamos as resoluções das atividades que não conseguimos aplicar em sala de aula. Essas resoluções não foram alcançadas pelos alunos. O objetivo deste anexo é auxiliar aos professores que, por ventura, tiverem a chance de aplicar as atividades em sala de aula.

B.1 Primeiro encontro

***Atividade* 4:** Considere o seguinte problema: "Uma bandeira é formada por quatro listras, que devem ser coloridas usando-se apenas as cores amarelo, rosa e verde, não devendo listras adjacentes ter a mesma cor. De quantos modos pode ser colorida a bandeira?". Responda as questões a seguir, antes de resolvê-lo.

Esta atividade, a princípio, produz um estranhamento em alguns alunos: "Como vou colorir a bandeira com quatro listras, se possuo apenas três cores para usar?". Ou seja, eles acham que uma das listras ficará sem colorir.
Por isso, é interessante propor que o aluno desenhe uma bandeira com quatro listras e pinte-a, obedecendo as regras do problema. Com isso, eles percebem que, pelo menos, uma das cores pode se repetir.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: Aqui, são as 3 cores disponíveis: $\{amarelo, rosa, verde\}$.
- Configurações: A configuração é uma bandeira inteira pintada corretamente, respeitando a regra de que listras adjacentes possuem cores diferentes. Matematicamente, é uma quádrupla ordenada $(cor1, cor2, cor3, cor4)$, onde $cor1 \neq cor2$, $cor2 \neq cor3$ e $cor3 \neq cor4$.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Precisamos tomar 4 decisões: 1ª) Escolher a cor da primeira listra a ser colorida; 2ª) Escolher a cor da segunda listra a ser colorida; 3ª) Escolher a cor da terceira listra a ser colorida; 4ª) Escolher a cor da quarta listra a ser colorida.
Há uma restrição no problema. O enunciado diz "não devendo listras adjacentes ter a mesma cor". Isto é, há uma restrição de dependência local.
Sem perda de generalidade, escolheremos a seguinte bandeira para representar a bandeira que queremos colorir:

1ª listra
2ª listra
3ª listra
4ª listra

Agora, precisamos começar a colorir a bandeira.
O aluno poderia, agora, fazer a seguinte enunciação:"Vou colorir primeiro a 1ª listra da bandeira e depois a 3ª listra, não estou querendo me preocupar com a restrição do problema agora, e, depois vou colorir a 2ª e 4ª listras."
Nesse caso, o aluno pode enunciar que:
"1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
2ª decisão(cor da terceira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *amarelo*;
3ª decisão(cor da segunda listra): 1 maneira $\rightarrow \{verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;
4ª decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."
Mas, também, poderíamos ter a seguinte enunciação:
"1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
2ª decisão(cor da terceira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
3ª decisão(cor da segunda listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;

$4^{\underline{a}}$ decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na $3^{\underline{a}}$ listra."

A partir das duas enunciações, observemos que a resposta da $3^{\underline{a}}$ decisão depende da cor escolhida para colorir a $3^{\underline{a}}$ listra, a realização da $2^{\underline{a}}$ decisão. Com isso, temos dois casos a resolver: o caso em que as cores das $1^{\underline{a}}$ e $3^{\underline{a}}$ listras são iguais e o caso em que as cores dessas listras são diferentes.

A estratégia escolhida pelo aluno resolve o problema mas, chega nessa indecisão, obrigando que o problema seja resolvido em dois casos, torna-se mais trabalhoso. Em momento oportuno, voltaremos a esse problema e utilizaremos a estratégia acima para resolvê-lo.

Mas, tem como contornar, por enquanto, essa coisa "desagradável". Para isso, Morgado *et al.* (2006) recomendam a seguinte estratégia: "Pequenas dificuldades adiadas costumam transformar-se em grandes dificuldades. Se alguma decisão é mais complicada que as demais, ela deve ser tomada em primeiro lugar."

Ou seja, usando a estratégia acima, temos que a primeira listra a ser colorida na bandeira pode ser qualquer uma das quatro existentes. Devemos ter atenção na escolha das demais listras, a segunda listra escolhida, por exemplo, deve ser adjacente à listra anteriormente colorida. Então, podemos ter o seguinte:

$1^{\underline{a}}$ decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$;

$2^{\underline{a}}$ decisão(cor da segunda listra): O número de cores disponíveis depende da escolha anterior. Temos então, 2 cores. Por exemplo, se a cor da primeira listra é *rosa* então a cor da segunda listra pode ser $\{amarelo, verde\}$.

$3^{\underline{a}}$ decisão(cor da terceira listra): O número de cores disponíveis depende da escolha anterior. Temos então, 2 cores. Por exemplo, se a cor da segunda listra é *verde* então a cor da terceira listra pode ser $\{amarelo, rosa\}$.

$4^{\underline{a}}$ decisão(cor da quarta listra): O número de cores disponíveis depende da escolha anterior. Temos então, 2 cores. Por exemplo, se a cor da terceira listra é *amarelo* então a cor da quarta listra pode ser $\{verde, rosa\}$.

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Temos as configurações elencadas na árvore de possibilidades da Figura 18.

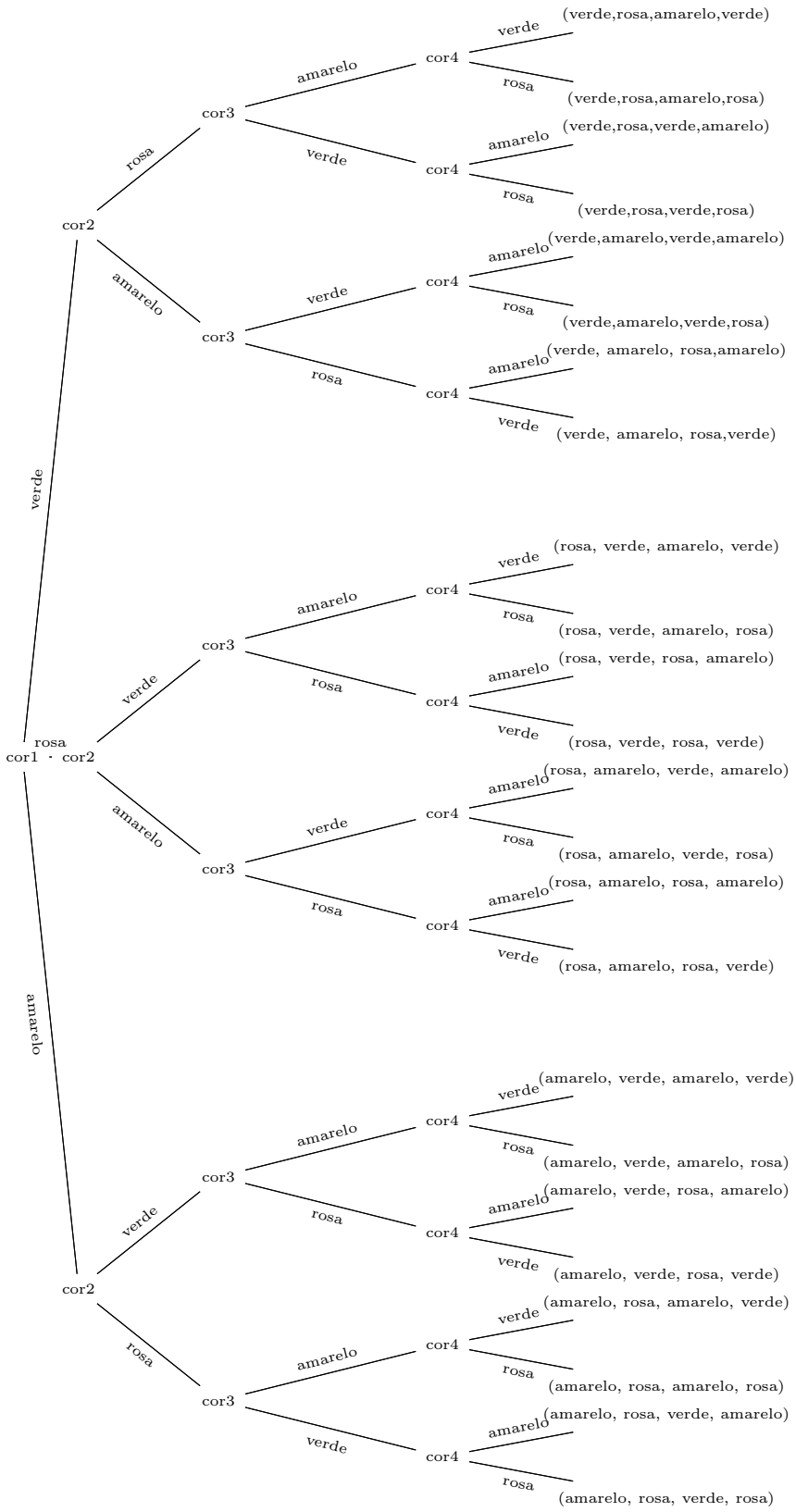


Figura 18: Árvore de Possibilidades da Atividade 4.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

> Obtemos 24 configurações.

e) Agora, execute o seguinte código em Python:

```
cores = ['amarelo','rosa', 'verde']
contagem = 0

for cor1 in cores:
    for cor2 in cores:
        if cor2 != cor1:
            for cor3 in cores:
                if cor3 != cor2:
                    for cor4 in cores:
                        if cor4 != cor3:
                            print('(',cor1, ',',cor2, ',', cor3, ',', cor4,')')
                            contagem = contagem + 1

print('Quantidade de maneiras de colorir a bandeira: ', contagem)
```

Código B.1: Código da Atividade 4

Obtemos o seguinte resultado ao executar o Código B.1.

```
( amarelo , rosa , amarelo , rosa )
( amarelo , rosa , amarelo , verde )
( amarelo , rosa , verde , amarelo )
( amarelo , rosa , verde , rosa )
( amarelo , verde , amarelo , rosa )
( amarelo , verde , amarelo , verde )
( amarelo , verde , rosa , amarelo )
( amarelo , verde , rosa , verde )
( rosa , amarelo , rosa , amarelo )
( rosa , amarelo , rosa , verde )
( rosa , amarelo , verde , amarelo )
( rosa , amarelo , verde , rosa )
( rosa , verde , amarelo , rosa )
( rosa , verde , amarelo , verde )
( rosa , verde , rosa , amarelo )
( rosa , verde , rosa , verde )
( verde , amarelo , rosa , amarelo )
( verde , amarelo , rosa , verde )
( verde , amarelo , verde , amarelo )
( verde , amarelo , verde , rosa )
( verde , rosa , amarelo , rosa )
( verde , rosa , amarelo , verde )
( verde , rosa , verde , amarelo )
( verde , rosa , verde , rosa )
Quantidade de maneiras de colorir a bandeira:  24
```

Figura 19: Saída do Código B.1.
Fonte: Autoria própria(2026).

Analisando o que o código faz:

I. **Linha 1:** *cores = ['amarelo','rosa','verde']* → Define os objetos básicos do problema.

II. **Linha 4:** *for cor1 in cores:* → Representa a 1ª Decisão. O computador "escolhe" uma cor para a primeira listra e "fixa" ela.

III. **Linha 5:** *for cor2 in cores:* → Representa a 2ª Decisão. Para cada cor que está "fixada" na 1ª decisão, o computador testa todas as outras cores possíveis.

IV. **Linha 6:** *if cor2 != cor1* → Esta linha é a materialização da restrição de que a cor da segunda listra não pode ser a mesma cor da primeira listra. Ela atua como um filtro. Ela diz: "Eu gerei o par (*amarelo*,*amarelo*), mas a restrição do problema diz que ele não serve. Descarte-o."

V. **Linha 7:** *for cor3 in cores:* → Representa a 3ª Decisão. Para cada cor que está "fixada" na 2ª decisão, o computador testa todas as outras cores possíveis.

VI. **Linha 8:***if cor3 != cor2* → Esta linha é a materialização da restrição de que a cor da terceira listra não pode ser a mesma cor da segunda listra. Ela atua como um filtro. Ela diz: "Eu gerei o par (*amarelo*,*verde*,*verde*), mas a restrição do problema diz que ele não serve. Descarte-o."

VII. **Linha 9:** *for cor4 in cores:* → Representa a 4ª Decisão. Para cada cor que está "fixada" na 3ª decisão, o computador testa todas as outras cores possíveis.

VIII. **Linha 10:***if cor4 != cor3* → Esta linha é a materialização da restrição de que a cor da quarta listra não pode ser a mesma cor da terceira listra. Ela atua como um filtro. Ela diz: "Eu gerei o par (*amarelo*,*verde*,*amarelo*,*amarelo*), mas a restrição do problema diz que ele não serve. Descarte-o."

IX. **Linha 11:** *print(...)* → Materializa a configuração (a quádrupla ordenada).

X. **Linha 12:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

XI. **Resultado da execução do código:** O programa imprimirá linha por linha:

('amarelo','rosa','amarelo','rosa'),('amarelo','rosa','amarelo','verde')... *até ...('verde','rosa','verde','rosa').*

E ao final: *Quantidade de maneiras de colorir a bandeira: 24*

Resumidamente, visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

1) $Cor1 = Amarelo$

2) $Cor2 = Amarelo \rightarrow if$ **bloqueia (iguais). Descarte.**

2) $Cor2 = Rosa \rightarrow if$ **permite**.

3) $Cor3 = Rosa \rightarrow if$ **bloqueia**.

3) $Cor3 = Verde \rightarrow if$ **permite**.

4) $Cor4 = Verde \rightarrow if$ **bloqueia**.

4) $Cor4 = Amarelo \rightarrow if$ **permite** $\rightarrow (amarelo, rosa, verde, amarelo)$.

4) $Cor4 = Rosa \rightarrow if$ **permite** $\rightarrow (amarelo, rosa, verde, rosa)$ ... e assim por diante até testar tudo.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código B.1.

A Árvore mostra visualmente todos os resultados possíveis. O Python "prova" a resposta gerando concretamente todos os 24 resultados.

g) Observe o Código B.1, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo das linhas de código *if cor2 != cor1*, *if cor3 != cor2* e *if cor4 != cor3*?

Foram utilizados 4 laços for. E cada um desses laços representa uma decisão.
O laço externo *for cor1 in cores:* representa a 1ª Decisão.
O 1º laço interno *for cor2 in cores:* representa a 2ª Decisão.
O 2º laço interno *for cor3 in cores:* representa a 3ª Decisão.
O 3º laço interno *for cor4 in cores:* representa a 3ª Decisão.
Nesses quatro comandos for aninhados, o algoritmo se propõe a visitar todas as configurações, independentemente de serem válidas ou não, ou seja, o código está percorrendo um espaço de $3 \times 3 \times 3 \times 3 = 81$ possibilidades totais.
Mas, as linhas de comando *if* são a materialização da restrição de que a repetição de cores em listras adjacentes é proibida.
Por exemplo, enquanto na construção da árvore de possibilidades a restrição do problema está "escondida" na quantidade de maneiras de tomar a 2ª Decisão(2 maneiras), no código Python a restrição está escrita e visível na linha *if cor2 != cor1:*.

B.2 Segundo encontro

***Atividade* 5:** Considere o seguinte problema: "Quantos números pares de dois algarismos podem ser formados no sistema decimal?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: São os algarismos do sistema numérico decimal: {0, 1,2,3,4,5,6,7,8,9}.
- Configurações: A configuração é um número par com dois algarismos. Matematicamente, é um par ordenado $(Dezena, Unidade)$ de modo que a Unidade é igual a 0,2,4,6 ou 8.

Uma Enunciação comum dos alunos, porém equivocada, é: "A Dezena tem que ser um dos algarismos, 0,2,4,6 ou 8."

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Como queremos formar números com dois algarismos, temos que tomar duas decisões para resolver esse problema. Como visto acima, precisamos formar o par ordenado $(Dezena, Unidade)$.
Temos duas restrições nesse problema.

- **Primeira restrição**: O par $(0, 7)$, por exemplo, representa o número 7, que é um número formado por apenas um algarismo. Portanto, o par ordenado $(0, a)$ não é uma configuração válida neste problema, isto é, o algarismo das dezenas não pode ser igual a zero.
- **Segunda restrição**: O algarismo das unidades deve ser um múltiplo de 2, ou seja, 0,2,4,6 ou 8.

Temos uma decisão mais restrita nesse problema, que é o número formado deve ser par. Por isso, como *não podemos adiar dificuldades*, temos:
1ª decisão: Escolher o algarismo das unidades;
2ª decisão: Escolher o algarismo das dezenas.
Então:
1ª decisão(algarismo das unidades): 5 maneiras $\rightarrow$ {0,2,4,6,8};
2ª decisão(algarismo das dezenas): 9 maneiras $\rightarrow \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
Quando o professor perguntar "Por que 9 maneiras?", a Justificação do aluno será: "Porque se o algarismo das dezenas for igual a zero, o número só tem um algarismo.". Essa fala explicita a restrição do problema. E, se a pergunta for "Por que 5 maneiras?", a Justificação do aluno será: "Porque o problema diz que o número formado deve ser par."

c) Execute o seguinte código em Python:

```
algarismos = [0,1,2,3,4,5,6,7,8,9]
contagem = 0

print('Números pares formados: ')
for unidades in algarismos:
    if unidades % 2 == 0:
        for dezenas in algarismos:
            if dezenas != 0:
                print(dezenas,unidades)
```

```
10                  contagem = contagem + 1
11
12 print('Quantidade de números pares com dois algarismos: ', contagem)
```

Código B.2: Código da Atividade 5

Analisando o código:

I. **Linha 1:** *algarismos = [0,1,2,3,4,5,6,7,8,9]* → Define os objetos básicos do problema.

II. **Linha 5:** *for unidades in algarismos:* → Representa a 1ª Decisão. O computador "escolhe" um algarismo para as unidades e "fixa" ele.

III. **Linha 6:** *if unidades % 2 == 0* → Esta linha é a materialização da restrição de que o algarismo das unidades tem que ser um múltiplo de dois. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o par (?,1), mas a restrição do problema diz que ele não serve. Descarte-o."

IV. **Linha 7:** *for dezenas in algarismos:* → Representa a 2ª Decisão. Para cada algarismo que está "fixado" na 1ª decisão, o computador testa todos os outros algarismos possíveis para as dezenas.

V. **Linha 8:** *if dezenas != 0* → Esta linha é a materialização da restrição de que o algarismo das dezenas não pode ser igual a zero. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o par (0,2), mas a restrição do problema diz que ele não serve. Descarte-o."

VI. **Linha 9:** *print(...)* → Materializa a configuração (o par ordenado).

VII. **Linha 10:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

VIII. **Resultado da execução do código:** O programa imprimirá linha por linha:10,20,30,...,88,98.

E ao final: *Quantidade de números pares com dois algarismos: 45*

Resumidamente, visualizemos o que acontece na "cabeça" do Python:

1) $unidades = 0 \rightarrow if$ **Permite.**

2) $dezenas = 0 \rightarrow if$ **Bloqueia. Descarte.**

2) $dezenas = 1 \rightarrow if$ **Permite.** $\rightarrow 10$

2) $dezenas = 2 \rightarrow if$ **Permite.** $\rightarrow 20$

2) $dezenas = 3 \rightarrow if$ **Permite.** $\rightarrow 30 \ldots$ e assim por diante até testar tudo.

Ou seja, se o algarismo das unidades for igual a 0, o algarismo das dezenas pode ser 1,2,3,4,5,6,7,8 ou 9, o que lhe fornece 9 números diferentes:

10,20,30,40,50,60,70,80,90

Se o algarismo das unidades for 2, há novamente 9 maneiras diferentes de escolher o algarismo das dezenas:

12,22,32,42,52,62,72,82,92

Analogamente, se o algarismo das unidades for 4:

14,24,34,44,54,64,74,84,94

E, o mesmo ocorre se o algarismo das unidades for 6 ou 8.
Com isso, temos o seguinte resultado ao executar o Código B.2.

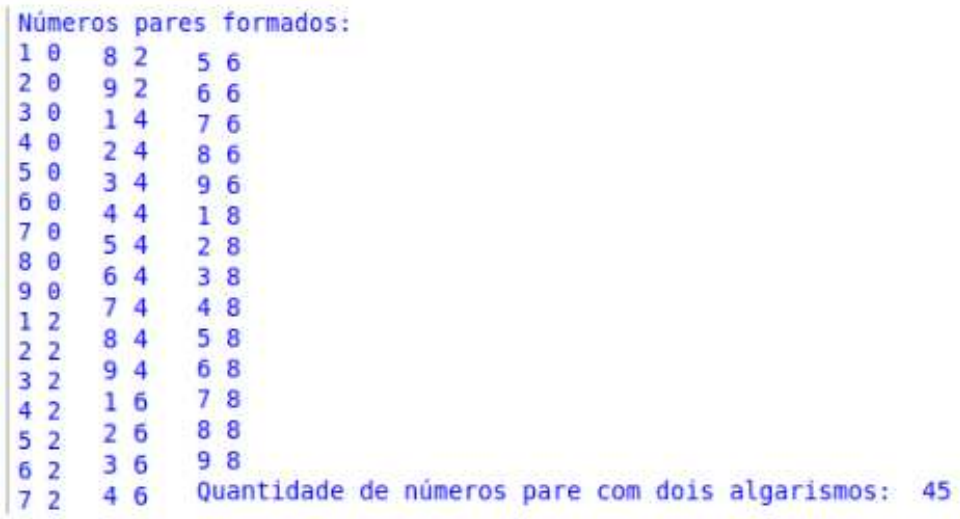


Figura 20: Saída do Código B.2.
Fonte: Autoria própria(2026).

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

> As configurações satisfazem as condições do problema.
> E o código fornece 45 configurações.

d) Observe o Código B.2, qual é o objetivo das linhas de código *if dezenas!= 0* e *if unidades % 2 = 0*?

> A linha *if unidades % 2 == 0* é a materialização da restrição de que o algarismo das unidades tem que ser um múltiplo de dois.
> A linha *if dezenas != 0* é a materialização da restrição de que o algarismo das dezenas não pode ser igual a zero.

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

> Pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher um algarismo para as unidades e, feito isso, há nove maneiras de escolher um algarismo para as dezenas, temos que podemos formar $5 \times 9 = 45$ números pares com dois algarismos. Resultado também encontrado ao executar o Código B.2.

***Atividade* 6:** Considere o seguinte problema: "Quantos números pares de dois algarismos distintos podem ser formados no sistema decimal?". Responda as questões a seguir, antes

de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos são os mesmos da Atividade 5.
- Configurações: A configuração é um par ordenado $(Dezena, Unidade)$ de modo que a Unidade é igual a 0,2,4,6 ou 8 e $Dezena \neq Unidade$.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual?

Agora, temos três restrições nesse problema.

- **Primeira restrição**: O par ordenado $(0, a)$ não é uma configuração válida neste problema, isto é, o algarismo das dezenas não pode ser igual a zero.
- **Segunda restrição**: O algarismo das unidades deve ser um múltiplo de 2.
- **Terceira restrição**: Os algarismos são distintos.

Precisamos tomar 2 decisões:
1ª decisão: Escolher o algarismo das unidades;
2ª decisão: Escolher o algarismo das dezenas.

c) Execute o seguinte código em Python:

```
algarismos = [0,1,2,3,4,5,6,7,8,9]
contagem = 0

print('Números pares formados: ')
for unidades in algarismos:
    if unidades % 2 == 0:
        for dezenas in algarismos:
            if dezenas != 0 and dezenas != unidades:
                print(dezenas,unidades)
                contagem = contagem + 1

print('Quantidade de números pares com dois algarismos distintos: ', contagem)
```

Código B.3: Código da Atividade 6

Analisando o código, observemos que ele é bem parecido com o Código B.2. A diferença está na linha **Linha 8:** *if dezenas != 0 and dezenas != unidades:*, que é a materialização de duas restrições, de que o algarismo das dezenas não pode ser igual a zero e de que o algarismo da dezenas não pode ser igual ao algarismo escolhido para as unidades. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o par (0,2), mas a restrição do problema diz que ele não serve. Descarte-o."E, "Eu estou prestes a gerar o par (4,4), mas a restrição do problema diz que ele não serve. Descarte-o."
Resumidamente, visualizemos o que acontece na "cabeça" do Python:

1) $unidades = 0 \rightarrow if$ **Permite.**

2) $dezenas = 0 \rightarrow if$ **Bloqueia. Descarte.**

2) $dezenas = 1 \rightarrow if$ **Permite.** $\rightarrow 10$

2) $dezenas = 2 \rightarrow if$ **Permite.** $\rightarrow 20$

2) $dezenas = 3 \rightarrow if$ **Permite.** $\rightarrow 30 \ldots$ e assim por diante até testar tudo.

Ou seja, se o algarismo das unidades for igual a 0, o algarismo das dezenas pode ser 1,2,3,4,5,6,7,8 ou 9, o que lhe fornece 9 números diferentes:

10,20,30,40,50,60,70,80,90

Se o algarismo das unidades for 2, há 8 maneiras diferentes de escolher o algarismo das dezenas:

12,32,42,52,62,72,82,92

Analogamente, se o algarismo das unidades for 4:

14,24,34,54,64,74,84,94

E, o mesmo ocorre se o algarismo das unidades for 6 ou 8.
Com isso, temos o seguinte resultado ao executar o Código B.3.

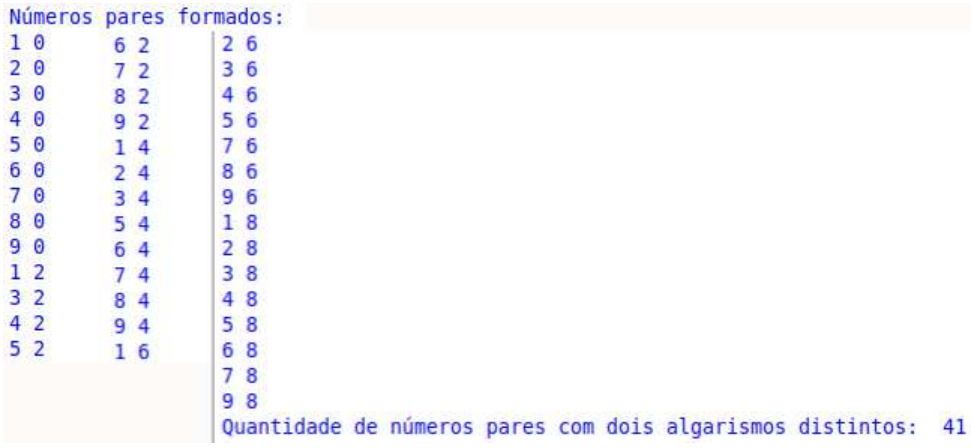


Figura 21: Saída do Código B.3.
Fonte: Autoria própria(2026).

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

> As configurações satisfazem as condições do problema.
> Obtemos 41 configurações.

d) Observe o Código B.3, qual é o objetivo das linhas de código *if unidades % 2 = 0* e *if dezenas!= 0 and dezenas != unidades*?

> A linha *if unidades % 2 == 0* é a materialização da restrição de que o algarismo das unidades tem que ser um múltiplo de dois.
> A linha *if dezenas != 0 and dezenas != unidades* é a materialização de duas restrições: o algarismo das dezenas não pode ser igual a zero e o algarismo da dezenas não pode ser igual ao algarismo escolhido para as unidades.

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

Observando a execução do Código B.3, temos que:

- quando o algarismo das unidades é igual a 0, o algarismo das dezenas pode ser 1,2,3,4,5,6,7,8 ou 9, o que fornece 9 números pares diferentes:

  10,20,30,40,50,60,70,80,90;

- quando o algarismo das unidades for 2, o algarismo das dezenas 1,3,4,5,6,7,8 ou 9, o que fornece 8 números pares diferentes:

  12,32,42,52,62,72,82,92

  Analogamente, quando o algarismo das unidades for 4, teremos:

  14,24,34,54,64,74,84,94

  E, o mesmo ocorre se o algarismo das unidades for 6 ou 8.

Logo, não é possível obter a resposta para esse problema usando, apenas, o Princípio Fundamental da Contagem.
A execução do Código B.3 mostra-nos que existem $9 + 4 \cdot 8 = 9 + 32 = 41$ números pares com 2 algarismos distintos. Tal resultado nos mostra que há "dois tipos" de números pares formados pelo código, aqueles em que o zero ocupa as unidades e aqueles que o zero não ocupa o algarismo das unidades.
Então, para aplicar o Princípio Fundamental da Contagem na resolução do problema, precisamos resolver dois problemas, que são:

- Quantos números pares com dois algarismos distintos terminam em zero?

  Para esse problema, temos 1 única maneira de tomar a decisão d_1, que é escolher o zero, uma vez tomada a decisão d_1, temos 9 maneiras de tomar a decisão d_2. Logo, pelo Princípio Fundamental da Contagem, há $1 \cdot 9 = 9$ números desse tipo.

- Quantos números pares com dois algarismos distintos não terminam em zero?

  Para esse problema, temos 4 maneiras de tomar a decisão d_1, que é escolher um dos algarismos 2, 4, 6 ou 8, uma vez tomada a decisão d_1, temos 8 maneiras de tomar a decisão d_2, pois o zero não pode ocorrer nas dezenas e não podemos usar o algarismo usado nas unidades. Logo, pelo Princípio Fundamental da Contagem, há $4 \cdot 8 = 32$ desse tipo.

Portanto, há $9 + 32 = 41$ números pares com dois algarismos distintos.
Temos então: um conjunto A formado pelos números pares com algarismos distintos; um conjunto A_1 formado pelos números pares com algarismos distintos com o algarismo das unidades igual a zero e um conjunto A_2 formado pelos números pares com algarismos distintos com o algarismo das unidades diferente de zero. Então, pelo Princípio Aditivo,$n(A) = n(A_1) + n(A_2) = 9 + 32 = 41$.
Agora, podemos voltar à enunciação feita por um aluno ao resolver o problema da Atividade 4(colorir a bandeira).
O aluno enunciou que:
"1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
2ª decisão(cor da terceira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *amarelo*;
3ª decisão(cor da segunda listra): 1 maneira $\rightarrow \{verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;
4ª decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."
Mas, também, poderíamos ter a seguinte enunciação:
"1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
2ª decisão(cor da terceira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
3ª decisão(cor da segunda listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;

$4^{\underline{a}}$ decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na $3^{\underline{a}}$ listra."

A partir das duas enunciações, observemos que a resposta da $3^{\underline{a}}$ decisão depende da cor escolhida para colorir a $3^{\underline{a}}$ listra, a realização da $2^{\underline{a}}$ decisão. Com isso, temos dois casos a resolver:

$1^{\underline{o}}$ caso: as cores das $1^{\underline{a}}$ e $3^{\underline{a}}$ listras são iguais;

$1^{\underline{a}}$ decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

$2^{\underline{a}}$ decisão(cor da terceira listra): 1 maneira $\rightarrow \{rosa\}$, vamos supor que o aluno escolheu *rosa*;

$3^{\underline{a}}$ decisão(cor da segunda listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a segunda listra não pode ter a mesma cor utilizada nas $1^{\underline{a}}$ e $3^{\underline{a}}$ listras, vamos supor que o aluno escolheu *amarelo*;

$4^{\underline{a}}$ decisão(cor da quarta listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a quarta listra não pode ter a mesma cor utilizada na $3^{\underline{a}}$ listra.

Pelo Princípio Fundamental da Contagem, conseguimos colorir de $3 \times 1 \times 2 \times 2 = 12$ maneiras diferentes de colorir a bandeira.

$2^{\underline{o}}$ caso: as cores das $1^{\underline{a}}$ e $3^{\underline{a}}$ listras são diferentes e as cores das $2^{\underline{a}}$ e $4^{\underline{a}}$ listras são iguais.

$1^{\underline{a}}$ decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

$2^{\underline{a}}$ decisão(cor da terceira listra): 2 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *amarelo*;

$3^{\underline{a}}$ decisão(cor da segunda listra): 1 maneira $\rightarrow \{verde\}$, a segunda listra não pode ter a mesma cor utilizada nas $1^{\underline{a}}$ e $3^{\underline{a}}$ listras;

$4^{\underline{a}}$ decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa,verde\}$, a quarta listra não pode ter a mesma cor utilizada na $3^{\underline{a}}$ listra.

Pelo Princípio Fundamental da Contagem, conseguimos colorir de $3 \times 2 \times 1 \times 2 = 12$ maneiras diferentes de colorir a bandeira.

Portanto, pelo Princípio Aditivo, há $12 + 12 = 24$ maneiras de colorir a bandeira, conforme determinamos no problema da Atividade 4.

Atividade 7: Considere o seguinte problema: "De quantos modos 3 pessoas podem sentar-se em 5 cadeiras em fila?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são os elementos disponíveis para a ação. Nesse caso, são as pessoas: $\{p1,p2,p3\}$ e as cadeiras: $\{c1,c2,c3,c4,c5\}$.
- Configurações: Cada configuração é formada por 3 pessoas sentadas em 3 cadeiras e 2 cadeiras vazias, ou seja, cada configuração é formada por uma tripla ordenada onde cada coordenada é uma cadeira ocupada pelas pessoas $p1$, $p2$ e $p3$, nessa ordem.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Temos uma restrição implícita no problema, uma cadeira para cada pessoa.
Precisamos tomar 3 decisões: 1ª) Escolher a cadeira que a pessoa $p1$ vai sentar; 2ª) Escolher a cadeira que a pessoa $p2$ vai sentar; 3ª) Escolher a cadeira que a pessoa $p3$ vai sentar.
Temos, então:
1ª decisão: 5 maneiras de escolher uma cadeira para a pessoa $p1$;
2ª decisão: 4 maneiras de escolher uma cadeira para a pessoa $p2$;
3ª decisão: 3 maneiras de escolher uma cadeira para a pessoa $p3$.

c) Execute o seguinte código em Python:

```
cadeiras = ['c1','c2','c3','c4','c5']
cadeiras_ocupadas = []

for cadeira_p1 in cadeiras:
    for cadeira_p2 in cadeiras:
            if cadeira_p2 != cadeira_p1:
                for cadeira_p3 in cadeiras:
                    if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:
                        cadeira_ocupada = cadeira_p1,cadeira_p2,cadeira_p3
                        cadeiras_ocupadas.append(cadeira_ocupada)

colunas = 5

print(f"\nCadeiras ocupadas pelas 3 pessoas:  ")
for i, cadeira_ocupada in enumerate(cadeiras_ocupadas):
```

```
16      print(f"{cadeira_ocupada}", end=",")
17      if (i + 1) % colunas == 0:
18          print()
19
20  if len(cadeiras_ocupadas) % colunas != 0:
21      print()
22
23  print('Quantidade de maneiras de organizar as 3 pessoas: ', len(cadeiras_ocupadas))
```

Código B.4: Código da Atividade 7

Analisando o código:

I. **Linha 1:** *cadeiras* = [*c*1,*c*2,*c*3,*c*4,*c*5] → Define o objeto básico cadeiras.

II. **Linha 2:** *cadeiras_ocupadas* = [] → Define uma lista onde cada elemento será as cadeiras ocupadas pelas pessoas *p*1, *p*2 e *p*3.

III. **Linha 4:** *for cadeira_p1 in cadeiras:* → Representa a 1ª Decisão. O computador "escolhe" uma cadeira para a pessoa *p*1 e "fixa" ela.

IV. **Linha 5:** *for cadeira_p2 in cadeiras:* → Representa a 2ª Decisão. O computador "escolhe" uma cadeira para a pessoa *p*2 e "fixa" ela.

V. **Linha 6:** *if cadeira_p2 != cadeira_p1 :* → Esta linha é a materialização da restrição de que cada pessoa senta em uma única cadeira. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar a tripla (*c*1,*c*1,?), mas a restrição do problema diz que ele não serve. Descarte-o."

VI. **Linha 7:** *for cadeira_p3 in cadeiras:* → Representa a 3ª Decisão. Para cada cadeira que está "fixada" na 1ª e 2ª decisões, o computador testa todas as outras cadeiras possíveis para a pessoa *p*3.

VII. **Linha 8:** *if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:* → Esta linha é a materialização da restrição de que que cada pessoa senta em uma única cadeira. Ela diz: "Eu estou prestes a gerar a tripla (*c*1,*c*2,*c*1), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."E, "Eu estou prestes a gerar a tripla (*c*1,*c*2,*c*2), mas a restrição do problema diz que ele não serve. Descarte-o."

VIII. **Linha 9:** $cadeira\_ocupada = cadeira\_p1$,$cadeira\_p2$,$cadeira\_p3 \rightarrow$ Materializa a configuração (a tripla ordenada).

IX. **Linha 10:** $cadeiras\_ocupadas.append(cadeira\_ocupada) \rightarrow$ Realiza a inserção da tripla formada na lista $cadeiras\_ocupadas$.

X. **Resultado da execução do código:** O programa imprimirá as triplas formadas.

E ao final: *Quantidade de maneiras de organizar as 3 pessoas: 60*

Obtemos, então, a seguinte saída:

```
Cadeiras ocupadas pelas 3 pessoas:
('c1', 'c2', 'c3'),('c1', 'c2', 'c4'),('c1', 'c2', 'c5'),('c1', 'c3', 'c2'),('c1', 'c3', 'c4'),
('c1', 'c3', 'c5'),('c1', 'c4', 'c2'),('c1', 'c4', 'c3'),('c1', 'c4', 'c5'),('c1', 'c5', 'c2'),
('c1', 'c5', 'c3'),('c1', 'c5', 'c4'),('c2', 'c1', 'c3'),('c2', 'c1', 'c4'),('c2', 'c1', 'c5'),
('c2', 'c3', 'c1'),('c2', 'c3', 'c4'),('c2', 'c3', 'c5'),('c2', 'c4', 'c1'),('c2', 'c4', 'c3'),
('c2', 'c4', 'c5'),('c2', 'c5', 'c1'),('c2', 'c5', 'c3'),('c2', 'c5', 'c4'),('c3', 'c1', 'c2'),
('c3', 'c1', 'c4'),('c3', 'c1', 'c5'),('c3', 'c2', 'c1'),('c3', 'c2', 'c4'),('c3', 'c2', 'c5'),
('c3', 'c4', 'c1'),('c3', 'c4', 'c2'),('c3', 'c4', 'c5'),('c3', 'c5', 'c1'),('c3', 'c5', 'c2'),
('c3', 'c5', 'c4'),('c4', 'c1', 'c2'),('c4', 'c1', 'c3'),('c4', 'c1', 'c5'),('c4', 'c2', 'c1'),
('c4', 'c2', 'c3'),('c4', 'c2', 'c5'),('c4', 'c3', 'c1'),('c4', 'c3', 'c2'),('c4', 'c3', 'c5'),
('c4', 'c5', 'c1'),('c4', 'c5', 'c2'),('c4', 'c5', 'c3'),('c5', 'c1', 'c2'),('c5', 'c1', 'c3'),
('c5', 'c1', 'c4'),('c5', 'c2', 'c1'),('c5', 'c2', 'c3'),('c5', 'c2', 'c4'),('c5', 'c3', 'c1'),
('c5', 'c3', 'c2'),('c5', 'c3', 'c4'),('c5', 'c4', 'c1'),('c5', 'c4', 'c2'),('c5', 'c4', 'c3'),
Quantidade de maneiras de organizar as 3 pessoas:  60
>>>
```

Figura 22: Saída do Código B.4.
Fonte: Autoria própria(2026).

d) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

As configurações satisfazem as condições do problema.
Obtemos 60 configurações.

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

Pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher uma cadeira para a pessoa $p1$ e, feito isso, há quatro maneiras de escolher uma cadeira para a pessoa $p2$ e, feito isso, há três maneiras de escolher uma cadeira para a pessoa $p3$, temos que, podemos organizar as três pessoas de $5 \times 4 \times 3 = 60$ maneiras diferentes. O Código B.4 é a materialização algorítmica do Princípio Fundamental da Contagem.

***Atividade* 8:** Considere o seguinte problema: "De quantos modos 6 pessoas podem sentar-se em 10 cadeiras em fila?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido. Se estiver e inseguro, em relação à sua solução, construa um código em Python para resolver o problema.

O problema proposto nessa atividade é bem parecido com o problema da Atividade 7.
O código abaixo fornece as configurações e a quantidade de configurações.

```
cadeiras = ['c1','c2','c3','c4','c5','c6','c7','c8','c9','c10']
cadeiras_ocupadas = []

for cadeira_p1 in cadeiras:
    for cadeira_p2 in cadeiras:
            if cadeira_p2 != cadeira_p1:
                for cadeira_p3 in cadeiras:
                    if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:
                        for cadeira_p4 in cadeiras:
                            if cadeira_p4 != cadeira_p3 and cadeira_p4 != cadeira_p2 and
    cadeira_p4 != cadeira_p1:
                                for cadeira_p5 in cadeiras:
                                    if cadeira_p5 != cadeira_p4 and cadeira_p5 !=
    cadeira_p3 and cadeira_p5 != cadeira_p2 and cadeira_p5 != cadeira_p1:
                                        for cadeira_p6 in cadeiras:
                                            if cadeira_p6 != cadeira_p5 and cadeira_p6 !=
     cadeira_p4 and cadeira_p6 != cadeira_p3 and cadeira_p6 != cadeira_p2 and cadeira_p6
    != cadeira_p1:
                                                cadeira_ocupada = cadeira_p1,cadeira_p2,
    cadeira_p3,cadeira_p4,cadeira_p5,cadeira_p6
                                                cadeiras_ocupadas.append(cadeira_ocupada)

colunas = 4

print(f"\nCadeiras ocupadas pelas 3 pessoas:  ")
for i, cadeira_ocupada in enumerate(cadeiras_ocupadas):
    print(f"{cadeira_ocupada}", end=",")
    if (i + 1) % colunas == 0:
        print()

```

```
26 if len(cadeiras_ocupadas) % colunas != 0:
27     print()
28
29 print('Quantidade de maneiras de organizar as 3 pessoas: ', len(cadeiras_ocupadas))
```

Código B.5: Código da Atividade 8

B.3 Terceiro encontro

***Atividade* 10:** Considere o seguinte problema: "Quantos são os anagramas da palavra MOSTRA que começam por vogal e terminam por consoante?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são as letras: $\{M,O,S,T,R,A\}$.
- Configurações: Cada configuração é um anagrama da palavra MOSTRA que começa por vogal e termina por consoante.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Como cada configuração é um anagrama da palavra MOSTRA, precisamos tomar 6 decisões.
Temos três restrições no problema: cada letra só pode ser usada uma única vez; a 1ª letra do anagrama deve ser uma vogal; a 6ª letra do anagrama deve ser uma consoante.
Levando em conta a estratégia, *"se alguma decisão é mais complicada que as demais, ela deve ser tomada em primeiro lugar"*, dadas as letras M,O,S,T,R,A, temos as seguintes decisões:

1ª) Escolher a 1ª letra do anagrama(**que deve ser vogal**);
2ª) Escolher a 6ª letra do anagrama(**que deve ser consoante**);
3ª) Escolher a 2ª letra do anagrama;
4ª) Escolher a 3ª letra do anagrama;
5ª) Escolher a 4ª letra do anagrama;
6ª) Escolher a 5ª letra do anagrama.

Temos, então:

1ª decisão: 2 maneiras, pois como essa letra deve ser vogal e nenhuma vogal foi selecionada, temos 2 letras disponíveis;

2ª decisão: 4 maneiras, pois como essa letra deve ser consoante e nenhuma consoante foi selecionada, temos 4 letras disponíveis;

3ª decisão: 4 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª e 2ª decisões, temos 4 letras disponíveis;

4ª decisão: 3 maneiras, pois após a 1ª, 2ª e 3ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª e 3ª decisões, temos 3 letras disponíveis;

5ª decisão: 2 maneiras, pois após a 1ª, 2ª, 3ª e 4ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª e 4ª decisões, temos 2 letras disponíveis;

6ª decisão: 1 maneira, pois após a 1ª, 2ª, 3ª, 4ª e 5ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª, 4ª e 5ª decisões, temos 1 letra disponível.

c) Execute o seguinte código em Python:

```
letras = ['M','O','S','T','R','A']
vogais = ['A','O']
consoantes = ['M','R','S','T']
anagramas = []

for letra_1 in letras:
    if letra_1 in vogais:
        for letra_6 in letras:
            if letra_6 in consoantes:
                for letra_2 in letras:
                    if letra_2 != letra_1 and letra_2 != letra_6:
                        for letra_3 in letras:
                            if letra_3 != letra_2 and letra_3 != letra_1 and
    letra_3 != letra_6:
                                for letra_4 in letras:
                                    if letra_4 != letra_3 and letra_4 != letra_2
    and letra_4 != letra_1 and letra_4 != letra_6:
                                        for letra_5 in letras:
```

```
17                                              if letra_5 != letra_4 and letra_5 !=
    letra_3 and letra_5 != letra_2 and letra_5 != letra_1 and letra_5 != letra_6:
18                                                  anagrama = (letra_1 + letra_2 +
    letra_3 + letra_4 + letra_5 + letra_6)
19                                                  anagramas.append(anagrama)
20
21 colunas = 12
22
23 print(f"\nAnagramas da palavra MOSTRA que começam por vogal e teminam por consoante
    : ")
24 for i, anagrama in enumerate(anagramas):
25     print(f"{anagrama},", end="")
26     if (i + 1) % colunas == 0:
27         print()
28
29 if len(anagramas) % colunas != 0:
30     print()
31
32 print('Quantidade de anagramas: ', len(anagramas))
```

Código B.6: Código da Atividade 10

Analisando o código:

I. **Linha 1:** *letras* = [*M*,*O*,*S*,*T*,*R*,*A*] → Define o objeto básico.

II. **Linhas 2 e 3:** *vogais* = [*A*,*O*] e *consoantes* = [*M*,*R*,*S*,*T*] → Diferencia as letras do objeto básico em vogais e consoantes.

III. **Linha 4:** *anagramas* = [] → Define uma lista onde cada elemento será um anagrama.

IV. **Linha 6:** *for letra_1 in letras:* → Representa a 1ª Decisão. O computador "escolhe" uma letra para ser a 1ª do anagrama e "fixa" ela.

V. **Linha 7:** *if letra_1 in vogais* → Esta linha é a materialização da restrição de que a 1ª letra do anagrama deve ser uma vogal. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o anagrama, por exemplo, *M*?????, mas a restrição do problema diz que ele não serve. Descarte-o."

VI. **Linha 8:** *for letra_6 in letras:* → Representa a 2ª Decisão. O computador "escolhe" uma letra para ser a 6ª letra do anagrama e "fixa" ela.

VII. **Linha 9:** *if letra_6 in consoantes* → Esta linha é a materialização da restrição de que a 6ª letra do anagrama deve ser uma consoante. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o anagrama, por exemplo, $O????A$, mas a restrição do problema diz que ele não serve. Descarte-o."

VIII. **Linha 10:** *for letra_2 in letras:* → Representa a 3ª Decisão. O computador "escolhe" uma letra para ser a 2ª letra do anagrama e "fixa" ela.

IX. **Linha 11:** *if letra_2 != letra_1 and letra_2 != letra_6* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o anagrama $OM???M$, mas a restrição do problema diz que ele não serve. Descarte-o."

X. **Linha 12:** *for letra_3 in letras:* → Representa a 4ª Decisão. O computador "escolhe" uma letra para ser a 3ª letra do anagrama e "fixa" ela.

XI. **Linha 13:** *if letra_3 != letra_2 and letra_3 != letra_1 and letra_3 != letra_6:* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela diz: "Eu estou prestes a gerar o anagrama $OMO??S$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."

XII. **Linha 14:** *for letra_4 in letras:* → Representa a 5ª Decisão. O computador "escolhe" uma letra para ser a 4ª letra do anagrama e "fixa" ela.

XIII. **Linha 15:** *if letra_4 != letra_3 and letra_4 != letra_2 and letra_4 != letra_1 and letra_4 != letra_6:* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela diz: "Eu estou prestes a gerar o anagrama $OMRM?S$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."

XIV. **Linha 16:** *for letra_5 in letras:* → Representa a 6ª Decisão. O computador "escolhe" uma letra para ser a 5ª letra do anagrama e para cada letra que está "fixada" nas 1ª, 2ª, 3ª, 4ª e 5ª decisões, o computador testa todas as outras letras possíveis para ser a 5ª letra.

XV. **Linha 17:** *if letra_5 != letra_4 and letra_5 != letra_3 and letra_5 != letra_2 and letra_5 != letra_1 and letra_5 != letra_6:* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela diz: "Eu estou prestes a gerar o anagrama $OMRTRS$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."

XVI. **Linha 18:** $anagrama = letra\_1 + letra\_2 + letra\_3 + letra\_4 + letra\_5 + letra\_6$ → Materializa a configuração (o anagrama).

XVII. **Linha 19:** *anagramas.append(anagrama)* → Realiza a inserção do anagrama formado na lista *anagramas*.

XVIII. **Resultado da execução do código:** O programa imprimirá os anagramas.

E ao final: *Quantidade de anagramas: 192*

```
Anagramas da palavra MOSTRA que começam por vogal e teminam por consoante:
OSTRAM,OSTARM,OSRTAM,OSRATM,OSATRM,OSARTM,OTSRAM,OTSARM,OTRSAM,OTRASM,OTASRM,OTARSM,
ORSTAM,ORSATM,ORTSAM,ORTASM,ORASTM,ORATSM,OASTRM,OASRTM,OATSRM,OATRSM,OARSTM,OARTSM,
OMTRAS,OMTARS,OMRTAS,OMRATS,OMATRS,OMARTS,OTMRAS,OTMARS,OTRMAS,OTRAMS,OTAMRS,OTARMS,
ORMTAS,ORMATS,ORTMAS,ORTAMS,ORAMTS,ORATMS,OAMTRS,OAMRTS,OATMRS,OATRMS,OARMTS,OARTMS,
OMSRAT,OMSART,OMRSAT,OMRAST,OMASRT,OMARST,OSMRAT,OSMART,OSRMAT,OSRAMT,OSAMRT,OSARMT,
ORMSAT,ORMAST,ORSMAT,ORSAMT,ORAMST,ORASMT,OAMSRT,OAMRST,OASMRT,OASRMT,OARMST,OARSMT,
OMSTAR,OMSATR,OMTSAR,OMTASR,OMASTR,OMATSR,OSMTAR,OSMATR,OSTMAR,OSTAMR,OSAMTR,OSATMR,
OTMSAR,OTMASR,OTSMAR,OTSAMR,OTAMSR,OTASMR,OAMSTR,OAMTSR,OASMTR,OASTMR,OATMSR,OATSMR,
AOSTRM,AOSRTM,AOTSRM,AOTRSM,AORSTM,AORTSM,ASOTRM,ASORTM,ASTORM,ASTROM,ASROTM,ASRTOM,
ATOSRM,ATORSM,ATSORM,ATSROM,ATROSM,ATRSOM,AROSTM,AROTSM,ARSOTM,ARSTOM,ARTOSM,ARTSOM,
AMOTRS,AMORTS,AMTORS,AMTROS,AMROTS,AMRTOS,AOMTRS,AOMRTS,AOTMRS,AOTRMS,AORMTS,AORTMS,
ATMORS,ATMROS,ATOMRS,ATORMS,ATRMOS,ATROMS,ARMOTS,ARMTOS,AROMTS,AROTMS,ARTMOS,ARTOMS,
AMOSRT,AMORST,AMSORT,AMSROT,AMROST,AMRSOT,AOMSRT,AOMRST,AOSMRT,AOSRMT,AORMST,AORSMT,
ASMORT,ASMROT,ASOMRT,ASORMT,ASRMOT,ASROMT,ARMOST,ARMSOT,AROMST,AROSMT,ARSMOT,ARSOMT,
AMOSTR,AMOTSR,AMSOTR,AMSTOR,AMTOSR,AMTSOR,AOMSTR,AOMTSR,AOSMTR,AOSTMR,AOTMSR,AOTSMR,
ASMOTR,ASMTOR,ASOMTR,ASOTMR,ASTMOR,ASTOMR,ATMOSR,ATMSOR,ATOMSR,ATOSMR,ATSMOR,ATSOMR,
Quantidade de anagramas:  192
```

Figura 23: Saída do Código B.6.
Fonte: Autoria própria(2026).

c) As configurações obtidas satisfazem as condições do problema? Quantas configura-

ções o código forneceu?

> As configurações obtidas satisfazem as condições do problema.
> Obtemos 192 configurações.

d) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

> Pelo Princípio Fundamental da Contagem, como há 2 maneiras de escolher uma letra para ser 1ª letra do anagrama(pois deve ser vogal) e, feito isso, há 4 maneiras de escolher uma letra para ser a 6ª letra do anagrama(pois deve ser consoante) e, feito isso, há 4 maneiras de escolher uma letra para ser a 2ª letra do anagrama e, feito isso, há 3 maneiras de escolher uma letra para ser a 3ª letra do anagrama e, feito isso, há 2 maneiras de escolher uma letra para ser a 4ª letra do anagrama e, feito isso, há 1 maneira de escolher uma letra para ser a 5ª letra do anagrama, temos que, podemos formar $2\times4\times4\times3\times2\times1 = 192$ anagramas da palavra MOSTRA que começam por vogal e terminam por consoante.

***Atividade* 11:** Considere o seguinte código recursivo em Python que resolve um determinado problema de contagem:

```python
def permutacoes(palavra):
    if len(palavra) == 1:
        return [palavra]

    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado

palavra = 'NUMERO'

todas_permutacoes = permutacoes(palavra)

colunas = 20

print(f"\nAnagramas da palavra NÚMERO: ")
for i, anagrama in enumerate(todas_permutacoes):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

```

```
27 if len(todas_permutacoes) % colunas != 0:
28     print()
29
30 print('Quantidade de anagramas: ', len(todas_permutacoes))
```

Código B.7: Código da Atividade 11

a) Execute o código acima. O que esse código fornece como resultado?

> Ao, simplesmente, executar o Código B.7, ele listará os 720 anagramas da palavra NÚMERO.

```
Anagramas da palavra NÚMERO:
NUMERO,NUMEOR,NUMREO,NUMROE,NUMOER,NUMORE,NUEMRO,NUEMOR,NUERMO,NUEROM,NUEOMR,NUEORM,NURMEO,NURMOE,NUREMO,NUREOM,NUROME,NUROEM,NUOMER,NUOMRE,
NUOEMR,NUOERM,NUORME,NUOREM,NMUERO,NMUEOR,NMUREO,NMUROE,NMUOER,NMUORE,NMEURO,NMEUOR,NMERUO,NMEROU,NMEOUR,NMEORU,NMRUEO,NMRUOE,NMREUO,NMREOU,
NMROUE,NMROEU,NMOUER,NMOURE,NMOEUR,NMOERU,NMORUE,NMOREU,NEUMRO,NEUMOR,NEURMO,NEUROM,NEUOMR,NEUORM,NEMURO,NEMUOR,NEMRUO,NEMROU,NEMOUR,NEMORU,
NERUMO,NERUOM,NERMUO,NERMOU,NEROUM,NEROMU,NEOUMR,NEOURM,NEOMUR,NEOMRU,NEORUM,NEORMU,NRUMEO,NRUMOE,NRUEMO,NRUEOM,NRUOME,NRUOEM,NRMUEO,NRMUOE,
NRMEUO,NRMEOU,NRMOUE,NRMOEU,NREUMO,NREUOM,NREMUO,NREMOU,NREOUM,NREOMU,NROUME,NROUEM,NROMUE,NROMEU,NROEUM,NROEMU,NOUMER,NOUMRE,NOUEMR,NOUERM,
NOURME,NOUREM,NOMUER,NOMURE,NOMEUR,NOMERU,NOMRUE,NOMREU,NOEUMR,NOEURM,NOEMUR,NOEMRU,NOERUM,NOERMU,NORUME,NORUEM,NORMUE,NORMEU,NOREUM,NOREMU,
UNMERO,UNMEOR,UNMREO,UNMROE,UNMOER,UNMORE,UNEMRO,UNEMOR,UNERMO,UNEROM,UNEOMR,UNEORM,UNRMEO,UNRMOE,UNREMO,UNREOM,UNROME,UNROEM,UNOMER,UNOMRE,
UNOEMR,UNOERM,UNORME,UNOREM,UMNERO,UMNEOR,UMNREO,UMNROE,UMNOER,UMNORE,UMENRO,UMENOR,UMERNO,UMERON,UMEONR,UMEORN,UMRNEO,UMRNOE,UMRENO,UMREON,
UMRONE,UMROEN,UMONER,UMONRE,UMOENR,UMOERN,UMORNE,UMOREN,UENMRO,UENMOR,UENRMO,UENROM,UENOMR,UENORM,UEMNRO,UEMNOR,UEMRNO,UEMRON,UEMONR,UEMORN,
UERNMO,UERNOM,UERMNO,UERMON,UERONM,UEROMN,UEONMR,UEONRM,UEOMNR,UEOMRN,UEORNM,UEORMN,URNMEO,URNMOE,URNEMO,URNEOM,URNOME,URNOEM,URMNEO,URMNOE,
URMENO,URMEON,URMONE,URMOEN,URENMO,URENOM,UREMNO,UREMON,UREONM,UREOMN,URONME,URONEM,UROMNE,UROMEN,UROENM,UROEMN,UONMER,UONMRE,UONEMR,UONERM,
UONRME,UONREM,UOMNER,UOMNRE,UOMENR,UOMERN,UOMRNE,UOMREN,UOENMR,UOENRM,UOEMNR,UOEMRN,UOERNM,UOERMN,UORNME,UORNEM,UORMNE,UORMEN,UORENM,UOREMN,
MNUERO,MNUEOR,MNUREO,MNUROE,MNUOER,MNUORE,MNEURO,MNEUOR,MNERUO,MNEROU,MNEOUR,MNEORU,MNRUEO,MNRUOE,MNREUO,MNREOU,MNROUE,MNROEU,MNOUER,MNOURE,
MNOEUR,MNOERU,MNORUE,MNOREU,MUNERO,MUNEOR,MUNREO,MUNROE,MUNOER,MUNORE,MUENRO,MUENOR,MUERNO,MUERON,MUEONR,MUEORN,MURNEO,MURNOE,MURENO,MUREON,
MURONE,MUROEN,MUONER,MUONRE,MUOENR,MUOERN,MUORNE,MUOREN,MENURO,MENUOR,MENRUO,MENROU,MENOUR,MENORU,MEUNRO,MEUNOR,MEURNO,MEURON,MEUONR,MEUORN,
MERNUO,MERNOU,MERUNO,MERUON,MERONU,MEROUN,MEONUR,MEONRU,MEOUNR,MEOURN,MEORNU,MEORUN,MRNUEO,MRNUOE,MRNEUO,MRNEOU,MRNOUE,MRNOEU,MRUNEO,MRUNOE,
MRUENO,MRUEON,MRUONE,MRUOEN,MRENUO,MRENOU,MREUNO,MREUON,MREONU,MREOUN,MRONUE,MRONEU,MROUNE,MROUEN,MROENU,MROEUN,MONUER,MONURE,MONEUR,MONERU,
MONRUE,MONREU,MOUNER,MOUNRE,MOUENR,MOUERN,MOURNE,MOUREN,MOENUR,MOENRU,MOEUNR,MOEURN,MOERNU,MOERUN,MORNUE,MORNEU,MORUNE,MORUEN,MORENU,MOREUN,
ENUMRO,ENUMOR,ENURMO,ENUROM,ENUOMR,ENUORM,ENMURO,ENMUOR,ENMRUO,ENMROU,ENMOUR,ENMORU,ENRUMO,ENRUOM,ENRMUO,ENRMOU,ENROUM,ENROMU,ENOUMR,ENOURM,
ENOMUR,ENOMRU,ENORUM,ENORMU,EUNMRO,EUNMOR,EUNRMO,EUNROM,EUNOMR,EUNORM,EUMNRO,EUMNOR,EUMRNO,EUMRON,EUMONR,EUMORN,EURNMO,EURNOM,EURMNO,EURMON,
EURONM,EUROMN,EUONMR,EUONRM,EUOMNR,EUOMRN,EUORNM,EUORMN,EMNURO,EMNUOR,EMNRUO,EMNROU,EMNOUR,EMNORU,EMUNRO,EMUNOR,EMURNO,EMURON,EMUONR,EMUORN,
EMRNUO,EMRNOU,EMRUNO,EMRUON,EMRONU,EMROUN,EMONUR,EMONRU,EMOUNR,EMOURN,EMORNU,EMORUN,ERNUMO,ERNUOM,ERNMUO,ERNMOU,ERNOUM,ERNOMU,ERUNMO,ERUNOM,
ERUMNO,ERUMON,ERUONM,ERUOMN,ERMNUO,ERMNOU,ERMUNO,ERMUON,ERMONU,ERMOUN,ERONUM,ERONMU,EROUNM,EROUMN,EROMNU,EROMUN,EONUMR,EONURM,EONMUR,EONMRU,
EONRUM,EONRMU,EOUNMR,EOUNRM,EOUMNR,EOUMRN,EOURNM,EOURMN,EOMNUR,EOMNRU,EOMUNR,EOMURN,EOMRNU,EOMRUN,EORNUM,EORNMU,EORUNM,EORUMN,EORMNU,EORMUN,
RNUMEO,RNUMOE,RNUEMO,RNUEOM,RNUOME,RNUOEM,RNMUEO,RNMUOE,RNMEUO,RNMEOU,RNMOUE,RNMOEU,RNEUMO,RNEUOM,RNEMUO,RNEMOU,RNEOUM,RNEOMU,RNOUME,RNOUEM,
RNOMUE,RNOMEU,RNOEUM,RNOEMU,RUNMEO,RUNMOE,RUNEMO,RUNEOM,RUNOME,RUNOEM,RUMNEO,RUMNOE,RUMENO,RUMEON,RUMONE,RUMOEN,RUENMO,RUENOM,RUEMNO,RUEMON,
RUEONM,RUEOMN,RUONME,RUONEM,RUOMNE,RUOMEN,RUOENM,RUOEMN,RMNUEO,RMNUOE,RMNEUO,RMNEOU,RMNOUE,RMNOEU,RMUNEO,RMUNOE,RMUENO,RMUEON,RMUONE,RMUOEN,
RMENUO,RMENOU,RMEUNO,RMEUON,RMEONU,RMEOUN,RMONUE,RMONEU,RMOUNE,RMOUEN,RMOENU,RMOEUN,RENUMO,RENUOM,RENMUO,RENMOU,RENOUM,RENOMU,REUNMO,REUNOM,
REUMNO,REUMON,REUONM,REUOMN,REMNUO,REMNOU,REMUNO,REMUON,REMONU,REMOUN,REONUM,REONMU,REOUNM,REOUMN,REOMNU,REOMUN,RONUME,RONUEM,RONMUE,RONMEU,
RONEUM,RONEMU,ROUNME,ROUNEM,ROUMNE,ROUMEN,ROUENM,ROUEMN,ROMNUE,ROMNEU,ROMUNE,ROMUEN,ROMENU,ROMEUN,ROENUM,ROENMU,ROEUNM,ROEUMN,ROEMNU,ROEMUN,
ONUMER,ONUMRE,ONUEMR,ONUERM,ONURME,ONUREM,ONMUER,ONMURE,ONMEUR,ONMERU,ONMRUE,ONMREU,ONEUMR,ONEURM,ONEMUR,ONEMRU,ONERUM,ONERMU,ONRUME,ONRUEM,
ONRMUE,ONRMEU,ONREUM,ONREMU,OUNMER,OUNMRE,OUNEMR,OUNERM,OUNRME,OUNREM,OUMNER,OUMNRE,OUMENR,OUMERN,OUMRNE,OUMREN,OUENMR,OUENRM,OUEMNR,OUEMRN,
OUERNM,OUERMN,OURNME,OURNEM,OURMNE,OURMEN,OURENM,OUREMN,OMNUER,OMNURE,OMNEUR,OMNERU,OMNRUE,OMNREU,OMUNER,OMUNRE,OMUENR,OMUERN,OMURNE,OMUREN,
OMENUR,OMENRU,OMEUNR,OMEURN,OMERNU,OMERUN,OMRNUE,OMRNEU,OMRUNE,OMRUEN,OMRENU,OMREUN,OENUMR,OENURM,OENMUR,OENMRU,OENRUM,OENRMU,OEUNMR,OEUNRM,
OEUMNR,OEUMRN,OEURNM,OEURMN,OEMNUR,OEMNRU,OEMUNR,OEMURN,OEMRNU,OEMRUN,OERNUM,OERNMU,OERUNM,OERUMN,OERMNU,OERMUN,ORNUME,ORNUEM,ORNMUE,ORNMEU,
ORNEUM,ORNEMU,ORUNME,ORUNEM,ORUMNE,ORUMEN,ORUENM,ORUEMN,ORMNUE,ORMNEU,ORMUNE,ORMUEN,ORMENU,ORMEUN,ORENUM,ORENMU,OREUNM,OREUMN,OREMNU,OREMUN,
Quantidade de anagramas:  720
```

Figura 24: Saída do Código B.7.
Fonte: Autoria própria(2026).

b) Compare-o com o Código B.6. Qual é a diferença?

> Diferente do Código B.6 que usava vários *for* um dentro do outro, esse código introduz o conceito computacional conhecido como *Recursividade*.

c) Considere uma palavra com uma quantidade menor de letras, por exemplo ALO, e execute o Código B.7 passo a passo manualmente, para entender o seu funcionamento.

Analisando o código:

I. **Linha 1:** *def permutacoes(palavra):* → Define uma função chamada *permutacoes* que aceita um texto (*palavra*) como entrada.

II. **Linhas 2 e 3:** $if len(palavra) == 1 :$ e $return[palavra]$ → Se a palavra tiver apenas 1 letra, o anagrama dela é ela mesma. Toda recursão precisa de um ponto de parada, o chamado *Caso Base*. Sem isso, o computador ficaria num *loop* infinito tentando dividir a palavra para sempre.

III. **Linha 5:** *resultado = []* → Cria uma lista vazia para guardar os anagramas encontrados.

IV. **Linha 7:** *for i in range(len(palavra)):* → O laço *for* vai passar por cada letra da palavra original, uma por uma.

V. **Linha 8:** $char\_atual = palavra[i]$ → "Fixa" a letra da vez. Por exemplo, se a palavra for "ALO" e $i = 0$, fixa o "A".

VI. **Linha 9:** $resto = palavra[: i] + palavra[i + 1 :]$ → Cria uma nova palavra com tudo o que sobrou. Por exemplo, se tirou "A" de "ALO", o *resto* é "LO".

VII. **Linha 10:** *for p in permutacoes(resto):* → A função chama ela mesma, mas agora passando apenas o *resto*. É como se dissesse: "Eu não sei resolver "ALO" inteiro agora, mas se eu fixar o "A", você resolve "LO" para mim?"

VIII. **Linha 11:** $resultado.append(char\_atual + p)$ → Concatena a letra que estava fixa ($char\_atual$) na frente de cada anagrama que voltou da recursão (p).

IX. **Linhas 15 e 17:** *palavra = "ALO"* e $todas\_permutacoes = permutacoes(palavra)$ → O usuário define a palavra "ALO" e dispara o processo.

X. **Resultado da execução do código:** O programa imprimirá os anagramas. E ao final: *Quantidade de anagramas: 6*

Visualizemos o que acontece na "cabeça" do Python. Para o melhor entendimento

da recursão, imagine que cada vez que a função *permutacoes* é chamada, abre-se um novo "nível" dentro da anterior. O computador pausa o "nível" de fora para resolver o de dentro.

Início) **Nível 1**

- *permutacoes("ALO")*
- A função verifica: $len(\text{"}ALO\text{"}) == 1$? Não. $len(\text{"}ALO\text{"}) = 3$.
- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 1**→ (Fixa a letra "A")

- $i = 0$;
- $char\_atual = \text{"}A\text{"}$;
- $resto = \text{"}LO\text{"}$;
- Chamada para $permutacoes(\text{"}LO\text{"})$.

Início) **Nível 2**

- *permutacoes("LO")*
- A função verifica: $len(\text{"}LO\text{"}) == 1$? Não. $len(\text{"}LO\text{"}) = 2$.
- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 2** → (Fixa a letra "L")

- $i = 0$;
- $char\_atual = \text{"}L\text{"}$;
- $resto = \text{"}O\text{"}$;
- Chamada para $permutacoes(\text{"}O\text{"})$;

Início) **Nível 3**

* *permutacoes("O")*
* A função verifica: $len(\text{"}O\text{"}) == 1$? Sim.
* $resultado = [\text{"}O\text{"}]$;
* Fecha o Nível 3.

- Concatena $\text{"}L\text{"} + \text{"}O\text{"}$;
- $resultado = [\text{"}LO\text{"}]$.

2) **2ª iteração do Nível 2** → (Fixa a letra "O")

- $i = 1$;

- *char_atual* = *"O"*;
- *resto* = *"L"*;
- Chamada para *permutacoes*(*"L"*);

Início) **Nível 3**

* *permutacoes("L")*
* A função verifica: *len*(*"L"*) == 1? Sim.
* *resultado* = [*"L"*];
* Fecha o Nível 3.

- Concatena *"O"* + *"L"*;
- *resultado* = [*"OL"*].

Fim) **Nível 2**

- *resultado* = [*"LO"*,*"OL"*].
- Fecha o Nível 2.

- Concatena *"A"* + *"LO"*;
- Concatena *"A"* + *"OL"*;
- *resultado* = [*"ALO"*,*"AOL"*].

2) **2ª iteração do Nível 1**→ (Fixa a letra "L")

- $i = 1$;
- *char_atual* = *"L"*;
- *resto* = *"AO"*;
- Chamada para *permutacoes*(*"AO"*).

Início) **Nível 2**

- *permutacoes("AO")*
- A função verifica: *len*(*"AO"*) == 1? Não. *len*(*"AO"*) = 2.
- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 2** → (Fixa a letra "A")

- $i = 0$;
- *char_atual* = *"A"*;
- *resto* = *"O"*;
- Chamada para *permutacoes*(*"O"*);

Início) **Nível 3**

        * *permutacoes("O")*
        * A função verifica: $len(\text{"O"}) == 1$? Sim.
        * $resultado = [\text{"O"}]$;
        * Fecha o Nível 3.
      - Concatena *"A"* + *"O"*;
      - $resultado = [\text{"AO"}]$.

  2) **2ª iteração do Nível 2** → (Fixa a letra "O")
      - $i = 1$;
      - $char\_atual = \text{"O"}$;
      - $resto = \text{"A"}$;
      - Chamada para $permutacoes(\text{"A"})$;

  Início) **Nível 3**
        * *permutacoes("A")*
        * A função verifica: $len(\text{"A"}) == 1$? Sim.
        * $resultado = [\text{"A"}]$;
        * Fecha o Nível 3.
      - Concatena *"O"* + *"A"*;
      - $resultado = [\text{"OA"}]$.

  Fim) **Nível 2**
      - $resultado = [\text{"AO"},\text{"OA"}]$.
      - Fecha o Nível 2.

- Concatena *"L"* + *"AO"*;
- Concatena *"L"* + *"OA"*;
- $resultado = [\text{"ALO"},\text{"AOL"},\text{"LAO"},\text{"LOA"}]$.

3) **3ª iteração do Nível 1**→ (Fixa a letra "O")

- $i = 2$;
- $char\_atual = \text{"O"}$;
- $resto = \text{"AL"}$;
- Chamada para $permutacoes(\text{"AL"})$.

Início) **Nível 2**
    - *permutacoes("AL")*
    - A função verifica: $len(\text{"AL"}) == 1$? Não. $len(\text{"AL"}) = 2$.

- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 2** → (Fixa a letra "A")
   - $i = 0$;
   - $char\_atual$ = *"A"*;
   - $resto$ = *"L"*;
   - Chamada para $permutacoes$(*"L"*);

Início) **Nível 3**
   - *permutacoes("L")*
   - A função verifica: len(*"L"*) == 1? Sim.
   - $resultado$ = [*"L"*];
   - Fecha o Nível 3.
- Concatena *"A"* + *"L"*;
- $resultado$ = [*"AL"*].

2) **2ª iteração do Nível 2** → (Fixa a letra "L")
   - $i = 1$;
   - $char\_atual$ = *"L"*;
   - $resto$ = *"A"*;
   - Chamada para $permutacoes$(*"A"*);

Início) **Nível 3**
   - *permutacoes("A")*
   - A função verifica: len(*"A"*) == 1? Sim.
   - $resultado$ = [*"A"*];
   - Fecha o Nível 3.
- Concatena *"L"* + *"A"*;
- $resultado$ = [*"LA"*].

Fim) **Nível 2**
- $resultado$ = [*"AL"*,*"LA"*].
- Fecha o Nível 2.

- Concatena *"O"* + *"AL"*;
- Concatena *"O"* + *"LA"*;
- $resultado$ = [*"ALO"*,*"AOL"*,*"LAO"*,*"LOA"*,*"OAL"*,*"OLA"*].

Fim) **Nível 1**

- Fecha o Nível 1.
- $resultado = [“ALO”, “AOL”, “LAO”, “LOA”, “OAL”, “OLA”]$.

Temos o seguinte resultado ao executar o Código B.7, utilizando como entrada a palavra *ALO*:

```
Anagramas da palavra ALO:
ALO,AOL,LAO,LOA,OAL,OLA,
Quantidade de anagramas:  6
>>>
```

Figura 25: Saída do Código B.7.
Fonte: Autoria própria(2026).

Chegamos, assim, ao fim da iteração do Código B.7. Notemos que para resolver o problema grande *permutacoes*(“*ALO*”), o computador “empilhou” problemas menores até chegar no caso trivial (uma letra só), e depois veio “desempilhando” e montando as respostas.
O código prova de forma mecânica a fórmula $n! = n \times (n-1)!$.

***Atividade* 12:** Considere o seguinte código recursivo em Python que determina os anagramas de uma palavra:

```
def permutacoes(palavra):
    if len(palavra) == 1:
        return [palavra]

    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado

palavra = str(input("Digite uma palavra: ")).upper()

todas_permutacoes = permutacoes(palavra)

colunas = 8

print(f"\nAnagramas da palavra {palavra}: ")
for i, anagrama in enumerate(todas_permutacoes):
```

```
23      print(f"{anagrama},", end="")
24      if (i + 1) % colunas == 0:
25          print()
26
27  if len(todas_permutacoes) % colunas != 0:
28      print()
29
30  print('Quantidade de anagramas: ', len(todas_permutacoes))
```

Código B.8: Código da Atividade 12

a) Execute o código acima, usando como exemplo a palavra AMOR.

> Ao executar o Código B.8, que é idêntico ao Código B.7, obtemos os anagramas da palavra AMOR e a quantidade dos mesmos.

```
Anagramas da palavra AMOR:
AMOR,AMRO,AOMR,AORM,ARMO,AROM,MAOR,MARO,
MOAR,MORA,MRAO,MROA,OAMR,OARM,OMAR,OMRA,
ORAM,ORMA,RAMO,RAOM,RMAO,RMOA,ROAM,ROMA,
Quantidade de anagramas:  24
```

Figura 26: Saída do Código B.8 usando a palavra AMOR.
Fonte: Autoria própria(2026).

b) Usando o Princípio Fundamental da Contagem, determine o número de anagramas da palavra AMOR. Compare o seu resultado com o aquele fornecido pelo Código B.8.

> Usamos aqui a mesma ideia utilizada na Atividade anterior.

c) Execute o código usando, agora, como exemplo a palavra AMAR.

> Ao executar o Código B.8, devemos obter os anagramas da palavra AMAR e a quantidade dos mesmos.

```
Anagramas da palavra AMAR:
AMAR,AMRA,AAMR,AARM,ARMA,ARAM,MAAR,MARA,
MAAR,MARA,MRAA,MRAA,AAMR,AARM,AMAR,AMRA,
ARAM,ARMA,RAMA,RAAM,RMAA,RMAA,RAAM,RAMA,
Quantidade de anagramas:  24
```

Figura 27: Saída do Código B.8 usando a palavra AMAR.
Fonte: Autoria própria(2026).

d) Analise, atentamente, a saída de execução do código. Você percebeu alguma coisa estranha? O quê?

Ao analisar a lista de anagramas, percebemos que há anagramas repetidos. Por exemplo, o primeiro anagrama, AMAR, também é o décimo quinto anagrama na nossa lista.
O computador ficou maluco? Por que ele escreveu a mesma palavra duas vezes?
Devemos notar que trocar o A do início com o A do meio da palavra AMAR não gera uma nova palavra.
Então, o código gerou anagramas em excesso. Vamos corrigir?

e) Então, quantos são os anagramas da palavra AMAR?

Comecemos colocando os anagramas iguais em uma mesma "caixa":
[AMAR,AMAR]
[AMRA,AMRA]
[AAMR,AAMR]
[AARM,AARM]
[ARMA,ARMA]
[ARAM,ARAM]
[MAAR,MAAR]
[MARA,MARA]
[MRAA,MRAA]
[RAMA,RAMA]
[RAAM,RAAM]
[RMAA,RMAA]
Observemos que cada "caixa" possui dois anagramas iguais(como os "A" são iguais, contamos cada anagrama $2! = 2$ vezes), ou seja, para cada anagrama da palavra AMAR produzido, dois são repetidos. Com isso, podemos concluir que a palavra AMAR possui $\frac{24}{2} = 12$ anagramas.
Anagramas da palavra AMAR = [AMAR, AMRA, AAMR, AARM, ARMA, ARAM, MAAR, MARA, MRAA, RAMA, RAAM, RMAA].

f) Repita os itens c), d) e e) considerando a palavra BABA.

```
Anagramas da palavra BABA:
BABA,BAAB,BBAA,BBAA,BAAB,BABA,ABBA,ABAB,
ABBA,ABAB,AABB,AABB,BBAA,BBAA,BABA,BAAB,
BABA,BAAB,ABAB,ABBA,AABB,AABB,ABBA,ABAB,
Quantidade de anagramas:  24
```

Figura 28: Saída do Código B.8 usando a palavra BABA.
Fonte: Autoria própria(2026).

Ao analisar a lista de anagramas, percebamos que há anagramas repetidos. Por exemplo, o primeiro anagrama, BABA, também é o sexto, décimo quinto e décimo sétimo anagramas na nossa lista.

Devemos notar que trocar o B do início com o B do meio e o A do meio com o A do final da palavra BABA não gera uma nova palavra.

Dessa vez parece que o excesso foi maior. Vamos corrigir?

Comecemos colocando os anagramas iguais em uma mesma "caixa":

[BABA,BABA,BABA,BABA]

[BAAB,BAAB,BAAB,BAAB]

[BBAA,BBAA,BBAA,BBAA]

[ABBA,ABBA,ABBA,ABBA]

[ABAB,ABAB,ABAB,ABAB]

[AABB,AABB,AABB,AABB]

Observemos que cada "caixa" possui quatro anagramas iguais(como os "A" são iguais, contamos cada anagrama 2! vezes; analogamente, como os "B" são iguais, contamos cada anagrama 2! vezes), ou seja, para cada anagrama da palavra BABA produzido, quatro($2! \times 2!$) são repetidos. Com isso, podemos concluir que a palavra BABA possui $\frac{24}{4} = 6$ anagramas.

Anagramas da palavra BABA = [BABA, BAAB, BBAA, ABBA, ABAB, AABB].

g) Repita os itens c), d) e e) considerando a palavra AAZA(força em arábico).

```
Anagramas da palavra AAZA:
AAZA,AAAZ,AZAA,AZAA,AAAZ,AAZA,AAZA,AAAZ,
AZAA,AZAA,AAAZ,AAZA,ZAAA,ZAAA,ZAAA,ZAAA,
ZAAA,ZAAA,AAAZ,AAZA,AAAZ,AAZA,AZAA,AZAA,
Quantidade de anagramas:  24
```

Figura 29: Saída do Código B.8 usando a palavra AAZA.
Fonte: Autoria própria(2026).

Ao analisar a lista de anagramas, percebamos que há anagramas repetidos. Por exemplo, o primeiro anagrama, AAZA, também é o sexto, sétimo, décimo segundo, vigésimo e vigésimo segundo anagramas na nossa lista.
Devemos notar que trocar o A do início com o A do meio ou com o A do final da palavra AAZA não gera uma nova palavra.
Vamos corrigir?
Comecemos colocando os anagramas iguais em uma mesma "caixa":
[AAZA,AAZA,AAZA,AAZA,AAZA,AAZA]
[AAAZ,AAAZ,AAAZ,AAAZ,AAAZ,AAAZ]
[AZAA,AZAA,AZAA,AZAA,AZAA,AZAA]
[ZAAA,ZAAA,ZAAA,ZAAA,ZAAA,ZAAA]
Observemos que cada "caixa" possui seis anagramas iguais(como os "A" são iguais, contamos cada anagrama $3! = 6$ vezes), ou seja, para cada anagrama da palavra AAZA produzido, seis são repetidos. Com isso, podemos concluir que a palavra AAZA possui $\frac{24}{6} = 4$ anagramas.
Anagramas da palavra AAZA = [AAZA, AAAZ, AZAA,ZAAA].

h) Por quê o Código B.8 não funciona com as palavras AMAR, BABA e AAZA?

O computador não "lê" letras; ele "lê" índices de memória. Para o código recursivo, a *string* "AMAR", por exemplo, é tratada internamente como um vetor de posições: $[0, 1, 2, 3]$.
O algoritmo permuta os índices.

- **Permutação A:** Índices $[0, 1, 2, 3] \rightarrow$ Letras $A_1MA_2R \rightarrow$ "AMAR".
- **Permutação B:** Índices $[2, 1, 0, 3] \rightarrow$ Letras $A_2MA_1R \rightarrow$ "AMAR"

Para o computador, a **Permutação A** e a **Permutação B** são objetos diferentes porque vieram de índices diferentes. Ele não verifica o valor do conteúdo.

B.4 Quarto encontro

Na Atividade 12, deve-se ter notado que o código que calcula os anagramas de uma palavra só funciona corretamente quando a mesma possui letras distintas, ou seja, o código só calcula a quantidade de permutações simples.

O código da Atividade 13 tem como objetivo corrigir esse problema, isto é, calcular o número de anagramas de qualquer palavra.

***Atividade* 13:** Considere o seguinte código recursivo em Python que determina os anagramas de uma palavra mas, agora, de uma forma mais organizada, resolvendo os problemas que tivemos na Atividade 12:

```
def permutacoes(palavra):

    if len(palavra) == 1:
        return [palavra]

    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado


def remover_duplicatas_sort(lista):

    lista_ordenada = sorted(lista)

    lista_de_listas = []
    grupo_atual = []
    for i in range(len(lista_ordenada)):
        if i == 0 or lista_ordenada[i] == lista_ordenada[i - 1]:
            grupo_atual.append(lista_ordenada[i])
        else:
            lista_de_listas.append(grupo_atual)
            grupo_atual = [lista_ordenada[i]]
    lista_de_listas.append(grupo_atual)

    for i, uplas in enumerate(lista_de_listas, 1):
        print(f"{i:3}. {uplas}")
    print()
    print("Número de anagramas: ", len(lista_ordenada))
    print()
    print("Número de anagramas repetidos em cada caixa: ", len(grupo_atual))
    print()


    resultado = [lista_ordenada[0]]

    for i in range(1, len(lista_ordenada)):
        if lista_ordenada[i] != lista_ordenada[i-1]:
            resultado.append(lista_ordenada[i])
    return resultado

```

```
47 palavra = str(input("Digite uma palavra qualquer: ")).upper()
48
49 todas_permutacoes = permutacoes(palavra)
50 permutacoes_com_elemetos_repetidos = remover_duplicatas_sort(todas_permutacoes)
51
52
53 print(f"\nAnagramas da palavra {palavra}: ")
54 for i, perm in enumerate(permutacoes_com_elemetos_repetidos, 1):
55     print(f"{i:3}. {perm}")
56 print(f"Quantidade de anagramas da palavra {palavra}: {len(
    permutacoes_com_elemetos_repetidos)}")
```

Código B.9: Código da Atividade 13

A ideia por trás desse Código é a mesma que utilizamos nos itens e), f) e g) da Atividade 12.

- **Linhas 1 a 15:** *def permutacoes(palavra):* → O código gera todos os "anagramas" da palavra dada;
- **Linhas 17 a 45:** *def remover_duplicatas_sort(lista):* → Agora, o código começará a remover os excessos.
  - **Linha 19:** *lista_ordenada = sorted(lista)* → Para encontrar os excessos mais rápido, o primeiro passo dado é ordenar a lista de anagramas;
  - **Linhas 24 a 29:** *for i in range(len(lista_ordenada)):* → Neste bloco, o código agrupa e mostra as duplicatas antes de removê-las;
  - **Linhas 31 a 37:** *for i, uplas in enumerate(lista_de_listas,1):* → Neste bloco, o código imprime os grupos formados, mostrando que o computador contou em excesso;
  - **Linhas 40 a 45:** *resultado = [lista_ordenda[0]]* → Neste bloco, o código percorre a lista ordenada e só aceita um item se ele for diferente do anterior, aqui ocorre a retirada dos excessos.

a) Execute o código acima, usando como exemplo a palavra AMAR.

- A função *permutacoes* gera 24 anagramas(com os repetidos), conforme vemos na Figura 27;
- A função *remover_duplicatas_sort* ordena os anagramas, agrupa os iguais em "caixas"(vemos que cada "caixa" possui dois anagramas iguais), filtra os anagramas(lista os $\frac{24}{2} = 12$ anagramas).

```
Digite uma palavra qualquer: AMAR
  1. ['AAMR', 'AAMR']
  2. ['AARM', 'AARM']
  3. ['AMAR', 'AMAR']
  4. ['AMRA', 'AMRA']
  5. ['ARAM', 'ARAM']
  6. ['ARMA', 'ARMA']
  7. ['MAAR', 'MAAR']
  8. ['MARA', 'MARA']
  9. ['MRAA', 'MRAA']
 10. ['RAAM', 'RAAM']
 11. ['RAMA', 'RAMA']
 12. ['RMAA', 'RMAA']

Número de anagramas:  24

Número de anagramas repetidos em cada caixa:  2


Anagramas da palavra AMAR:
  1. AAMR
  2. AARM
  3. AMAR
  4. AMRA
  5. ARAM
  6. ARMA
  7. MAAR
  8. MARA
  9. MRAA
 10. RAAM
 11. RAMA
 12. RMAA
Quantidade de anagramas da palavra AMAR: 12
```

Figura 30: Saída do Código B.9 usando a palavra AMAR.
Fonte: Autoria própria(2026).

b) O resultado obtido é, de fato, a quantidade de anagramas da palavra AMAR?

O resultado obtido é a quantidade de anagramas da palavra AMAR $\rightarrow \frac{4 \times 3 \times 2 \times 1}{2 \times 1} = \frac{4!}{2!} = \frac{24}{2} = 12$, conforme visto no item e) da Atividade 12.

c) Execute o código usando, agora, como exemplo a palavra AAZA.

```
Digite uma palavra qualquer: AAZA
  1. ['AAAZ', 'AAAZ', 'AAAZ', 'AAAZ', 'AAAZ', 'AAAZ']
  2. ['AAZA', 'AAZA', 'AAZA', 'AAZA', 'AAZA', 'AAZA']
  3. ['AZAA', 'AZAA', 'AZAA', 'AZAA', 'AZAA', 'AZAA']
  4. ['ZAAA', 'ZAAA', 'ZAAA', 'ZAAA', 'ZAAA', 'ZAAA']

Número de anagramas:  24

Número de anagramas repetidos em cada caixa:  6


Anagramas da palavra AAZA:
  1. AAAZ
  2. AAZA
  3. AZAA
  4. ZAAA
Quantidade de anagramas da palavra AAZA: 4
```

Figura 31: Saída do Código B.9 usando a palavra AAZA.
Fonte: Autoria própria(2026).

d) O resultado obtido é, de fato, a quantidade de anagramas da palavra AAZA?

O resultado obtido é a quantidade de anagramas da palavra AAZA $\rightarrow \frac{4 \times 3 \times 2 \times 1}{3 \times 2 \times 1} = \frac{4!}{3!} = \frac{24}{6} = 4$, conforme visto no item g) da Atividade 12.

e) Execute o Código B.9, usando a palavra SOSSEGO. Quantos anagramas essa palavra possui?

- A função *permutacoes* gera 5040 anagramas(com os repetidos);
- A função *remover_duplicatas_sort* ordena os anagramas, agrupa os iguais em "caixas"(vemos que cada "caixa" possui seis anagramas iguais), filtra os anagramas(lista os $\frac{5040}{12} = 420$ anagramas).

Pelo que vimos, nos itens anteriores, o resultado obtido é a quantidade de anagramas da palavra SOSSEGO $\rightarrow \frac{7!}{3! \times 2!} = \frac{5040}{12} = 420$. Importante notar, que cada "caixa" possui doze anagramas iguais(como os "S" são iguais, contamos cada anagrama $3! = 6$ vezes e, como os "O" são iguais, contamos cada anagrama $2! = 2$ vezes), ou seja, para cada "anagrama" da palavra SOSSEGO produzido, doze são repetidos.

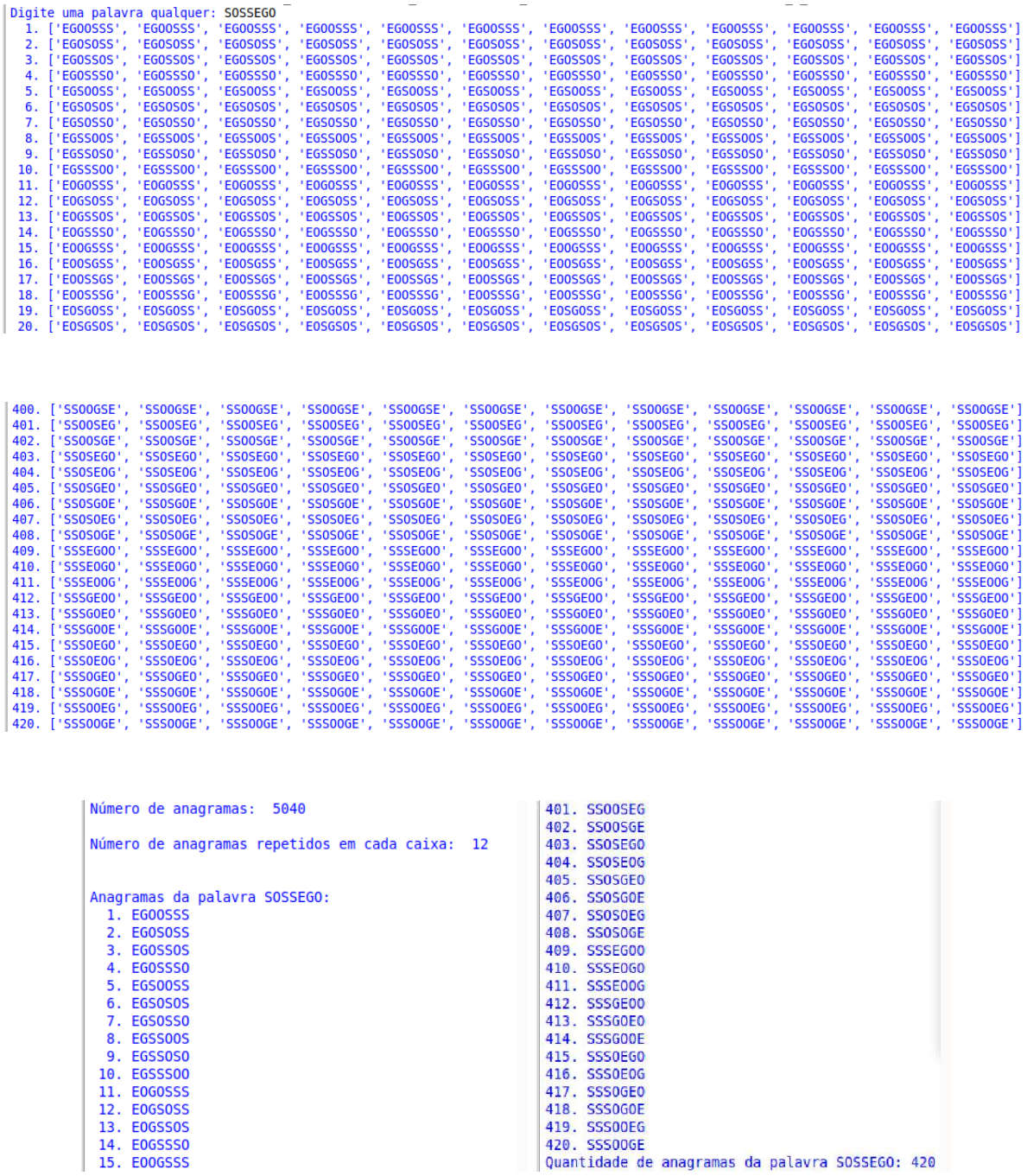


Figura 32: Resumo da Saída do Código B.9 usando a palavra SOSSEGO.
Fonte: Autoria própria(2026).

f) Determine o número de anagramas das palavras MANIA, CORRER e PASSISTA, usando o Princípio Fundamental da Contagem e o Princípio k para 1.

- **MANIA**
  - Supondo as letras distintas, pelo Princípio Fundamental da Contagem, há $5 \times 4 \times 3 \times 2 \times 1 = 5! = 120$ anagramas;
  - Como há duas letras "A", contamos cada anagrama $2! = 2$ vezes;
  - Logo, pelo Princípio k para 1 (nesse caso $k = 2!$), a palavra MANIA possui $\frac{120}{2!} = \frac{120}{2} = 60$ anagramas.
- **CORRER**
  - Supondo as letras distintas, pelo Princípio Fundamental da Contagem, há $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 6! = 720$ anagramas;
  - Como há três letras "R", contamos cada anagrama $3! = 6$ vezes;
  - Logo, pelo Princípio k para 1(nesse caso $k = 3!$), a palavra CORRER possui $\frac{720}{3!} = \frac{720}{6} = 120$ anagramas.
- **PASSISTA**
  - Supondo as letras distintas, pelo Princípio Fundamental da Contagem, há $8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 8! = 40320$ anagramas;
  - Como há três letras "S", contamos cada anagrama $3! = 6$ vezes e, há, também duas letras "A", contamos cada anagrama $2! = 2$ vezes, logo, estamos contando cada anagrama $2! \times 3! = 2 \times 6 = 12$ vezes;
  - Logo, pelo Princípio k para 1(nesse caso $k = 2! \times 3!$), a palavra CORRER possui $\frac{40320}{2! \times 3!} = \frac{40320}{2 \times 6} = \frac{40320}{12} = 3360$ anagramas.

***Atividade* 16:** Considere o seguinte código em Python que resolve um determinado problema de contagem:

```
def formar_filas(pessoas):

    print("\n" + "=" * 60)
    print("PASSO 1: Forma as filas.")
    print("=" * 60)

    resultado = []
    contador_filas = 0

    for pessoa_1 in pessoas:
        for pessoa_2 in pessoas:
```

```
            if pessoa_2 != pessoa_1:
                for pessoa_3 in pessoas:
                    if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:
                        contador_filas += 1
                        fila_formada = [pessoa_1,pessoa_2,pessoa_3]
                        resultado.append(fila_formada)
                        print(f"Fila {contador_filas:3d}: {fila_formada}")

    print(f"\nTotal de filas diferentes: {contador_filas}")
    return resultado

def ordenar_e_imprimir_filas(filas):

    print("\n" + "=" * 60)
    print("PASSO 2: Ordena cada fila individualmente")
    print("=" * 60)

    filas_ordenadas = []
    ordenadas = []

    print("\nTodas as filas ORDENADAS:")
    print("-" * 60)

    for idx, fila in enumerate(filas, 1):
        fila_ordenada = sorted(fila)
        filas_ordenadas.append(fila_ordenada)
        print(f"Fila {idx}: {fila_ordenada}")

    return filas_ordenadas

def remover_duplicatas_sort(lista):

    print("\n" + "=" * 60)
    print("PASSO 3: Coloca as filas iguais em uma mesma caixa")
    print("=" * 60)

    lista_ordenada = sorted(lista)

    lista_de_listas = []
    grupo_atual = []
    for i in range(len(lista_ordenada)):
        if i == 0 or lista_ordenada[i] == lista_ordenada[i - 1]:
            grupo_atual.append(lista_ordenada[i])
        else:
            lista_de_listas.append(grupo_atual)
            grupo_atual = [lista_ordenada[i]]

    lista_de_listas.append(grupo_atual)

    for i, uplas in enumerate(lista_de_listas, 1):
        print(f"{i:3}. {uplas}")
    print()
    print("Número de filas: ", len(lista_ordenada))
    print()
```

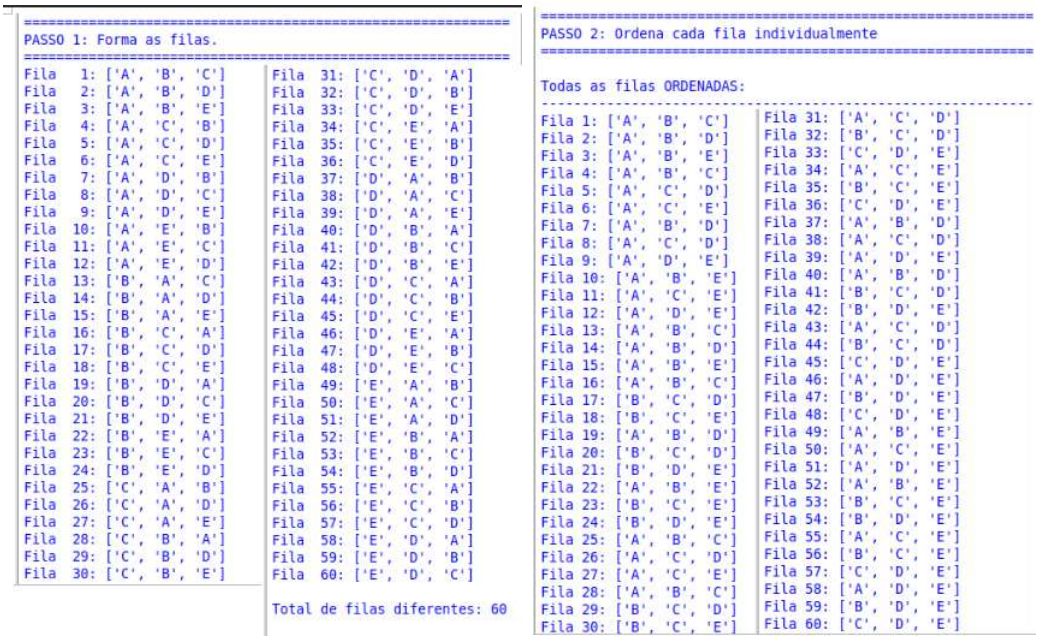
```
66      print("Depois que as filas são ordenadas, temos: ", len(grupo_atual) , " filas em
        cada caixa.")
67      print()
68
69      resultado = [lista_ordenada[0]]
70
71      for i in range(1, len(lista_ordenada)):
72          if lista_ordenada[i] != lista_ordenada[i-1]:
73              resultado.append(lista_ordenada[i])
74      return resultado
75
76
77  pessoas = ['A','B','C','D','E']
78
79  filas_obtidas = formar_filas(pessoas)
80  ordenadas = ordenar_e_imprimir_filas(filas_obtidas)
81  grupos = remover_duplicatas_sort(ordenadas)
82
83  print("Todas os grupos:")
84  for i, perm in enumerate(grupos, 1):
85      print(f"{i:3}. {perm}")
86  print(f"Número de grupos: {len(grupos)}")
```

Código B.10: Código da Atividade 16

a) Execute o código acima.

Analisando o código:
A ideia por trás desse Código é a mesma que utilizamos no item d) da Atividade 15.

- **Linhas 1 a 21:** *def formar_filas(pessoas):* → O código gera todos as filas formadas por três pessoas dentre cinco pessoas dadas;
- **Linhas 23 a 40:** *def ordenar_e_imprimir_filas(filas):* → O código ordena cada uma das filas formadas por três pessoas dentre cinco pessoas dadas;
- **Linhas 42 a 74:** *def remover_duplicatas_sort(lista):* → Neste bloco, o código agrupa as filas ordenadas e iguais em uma mesma "caixa" e, depois, remove as duplicatas.



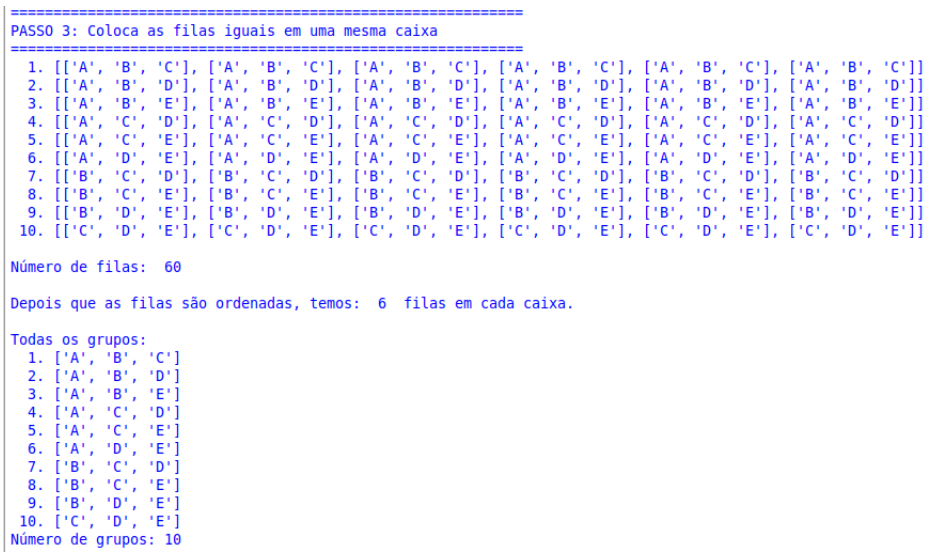


Figura 33: Saída do Código B.10.
Fonte: Autoria própria(2026).

b) O que esse código fornece como resultado?

Ao executar o código, teremos:

- A função *formar_filas* gera as 60 filas formadas por três pessoas, dadas cinco pessoas;
- A função *ordenar_e_imprimir_filas* ordena as filas.
- A função *remover_duplicatas_sort* agrupa as filas iguais em "caixas"(vemos que cada "caixa" possui seis filas iguais); filtra as filas(lista os $\frac{60}{6} = 10$ grupos com três pessoas, dadas cinco pessoas).Fornecendo, assim, a solução para o problema da Atividade 15.

e) Usando a ideia do código acima, dadas cinco pessoas quantos grupos com quatro pessoas podemos formar? E, dadas dez pessoas quantos grupos com quatro pessoas podemos formar?

- **Dadas cinco pessoas quantos grupos com quatro pessoas podemos formar?**
  - **Dadas cinco pessoas quantos filas com quatro pessoas podemos formar?**

    Precisamos tomar 4 decisões: 1ª) Escolher a pessoa do início da fila; 2ª) Escolher a segunda pessoa da fila; 3ª) Escolher a terceira pessoa da fila; 4ª) Escolher a pessoa do final da fila.

    Temos, então:

    1ª decisão: 5 maneiras, pois antes de iniciarmos a formação da fila, temos 5 pessoas disponíveis;

    2ª decisão: 4 maneiras, pois após a 1ª decisão ter sido realizada, para qualquer pessoa escolhida na 1ª decisão , temos 4 pessoas disponíveis;

    3ª decisão: 3 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer pessoas escolhidas nas 1ª e 2ª decisões, temos 3 pessoas disponíveis;

    4ª decisão: 2 maneiras, pois após a 1ª, 2ª e 3ª decisões terem sido feitas, para quaisquer pessoas escolhidas nas 1ª, 2ª e 3ª decisões, temos 2 pessoas disponíveis.

    Pelo Princípio Fundamental da Contagem, como há 5 maneiras de escolher uma pessoa para ser 1ª pessoa da fila e, feito isso, há 4 maneiras de escolher uma pessoa para ser a 2ª pessoa da fila e, feito isso, há 3 maneiras de

escolher uma pessoa para ser a $3^{\underline{a}}$ pessoa da fila e, feito isso, há 2 maneiras de escolher uma pessoa para ser a $4^{\underline{a}}$ pessoa da fila, temos que, podemos formar $5 \times 4 \times 3 \times 2 = 120$ filas.

– **Quantas filas "iguais" serão colocadas em cada caixa?**

Para resolver esse problema notemos, por exemplo, que 24 permutações simples das quatro pessoas $P1$,$P2$,$P3$,$P4$:

$$[P1, P2, P3, P4], [P1, P2, P4, P3], [P1, P3, P2, P4], [P1, P3, P4, P2],$$

$$[P1, P4, P2, P3], [P1, P4, P3, P2], [P2, P1, P3, P4], [P2, P1, P4, P3],$$

$$[P2, P3, P1, P4], [P2, P3, P4, P1], [P2, P4, P1, P3], [P2, P4, P3, P1],$$

$$[P3, P1, P2, P4], [P3, P1, P4, P2], [P3, P2, P1, P4], [P3, P2, P4, P1],$$

$$[P3, P4, P1, P2], [P3, P4, P2, P1], [P4, P1, P2, P3], [P4, P1, P3, P2],$$

$$[P4, P2, P1, P3], [P4, P2, P3, P1], [P4, P3, P1, P2], [P4, P3, P2, P1],$$

correspondem ao único grupo $\{P1,P2,P3,P4\}$.

Então, em cada caixa teremos 24 filas iguais, ou seja, a cada grupo formado por 4 pessoas dentre as 5 pessoas dadas, corresponde 24 permutações simples das mesmas 4 pessoas($4! = 24$).

Logo, pelo Princípio k para 1(nesse caso, $k = 4! = 24$), podemos formar $\frac{120}{24} = 5$ grupos com 4 pessoas escolhidas dentre 5 pessoas dadas.

- **Dadas dez pessoas quantos grupos com quatro pessoas podemos formar?**

  – **Dadas dez pessoas quantos filas com quatro pessoas podemos formar?**

Precisamos tomar 4 decisões: 1ª) Escolher a pessoa do início da fila; 2ª) Escolher a segunda pessoa da fila; 3ª) Escolher a terceira pessoa da fila; 4ª) Escolher a pessoa do final da fila.

Temos, então:

1ª decisão: 10 maneiras, pois antes de iniciarmos a formação da fila, temos 5 pessoas disponíveis;

2ª decisão: 9 maneiras, pois após a 1ª decisão ter sido realizada, para qualquer pessoa escolhida na 1ª decisão , temos 9 pessoas disponíveis;

3ª decisão: 8 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer pessoas escolhidas nas 1ª e 2ª decisões, temos 8 pessoas disponíveis;

4ª decisão: 7 maneiras, pois após a 1ª, 2ª e 3ª decisões terem sido feitas, para quaisquer pessoas escolhidas nas 1ª, 2ª e 3ª decisões, temos 7 pessoas disponíveis.

Pelo Princípio Fundamental da Contagem, como há 10 maneiras de escolher uma pessoa para ser 1ª pessoa da fila e, feito isso, há 9 maneiras de escolher uma pessoa para ser a 2ª pessoa da fila e, feito isso, há 8 maneiras de escolher uma pessoa para ser a 3ª pessoa da fila e, feito isso, há 7 maneiras de escolher uma pessoa para ser a 4ª pessoa da fila, temos que, podemos formar $10 \times 9 \times 8 \times 7 = 5040$ filas.

– **Quantas filas "iguais" serão colocadas em cada caixa?**

Como no problema anterior notemos, por exemplo, que 24 permutações simples das quatro pessoas $P1$,$P2$,$P3$,$P4$ correspondem ao único grupo $\{P1,P2,P3,P4\}$.

Então, em cada caixa teremos 24 filas iguais, ou seja, a cada grupo formado por 4 pessoas dentre as 10 pessoas dadas, corresponde 24 permutações simples das mesmas 4 pessoas($4! = 24$).

Logo, pelo Princípio k para 1(nesse caso, $k = 4! = 24$), podemos formar $\frac{5040}{24} = 210$ grupos com 4 pessoas escolhidas dentre 10 pessoas dadas.

APÊNDICE C – Recurso Educacional

Contagem com Python: Uma Abordagem pelo Conjunto de Resultados na Análise Combinatória

Sidney da Silva Ferreira

Recurso Educacional:

Contagem com Python: Uma Abordagem pelo Conjunto de Resultados na Análise Combinatória

SIDNEY DA SILVA FERREIRA

JONES COLOMBO

Niterói - 2026

Lista de Figuras

Lista de Códigos

Sumário

1 Introdução

Este recurso educacional é resultado da dissertação "A integração da Análise Combinatória e do Pensamento Computacional no Ensino de Matemática por meio da Linguagem Python" (FERREIRA, 2026), e consiste de uma sequência didática composta por dezesseis atividades, separadas em quatro encontros, sobre Análise Combinatória. A ideia central é usar a linguagem de programação Python para tornar visível aquilo que normalmente fica invisível na resolução de problemas de contagem: o conjunto de todos os resultados possíveis.

As atividades foram organizadas de forma progressiva avançando gradualmente para a abstração matemática, tendo o Princípio Fundamental da Contagem ou Princípio Multiplicativo como principal norteador para obter os resultados almejados. Pretende-se, assim, levar o aluno a pensar de verdade sobre o problema, não apenas a aplicar uma receita. Esta intervenção está fundamentada na seguinte triangulação teórica:

- A **Didática dos Princípios de Contagem (DPC)** que é utilizada em (CERIOLI; VIANA, 2012), a fim de apresentar uma abordagem analítica para a resolução de problemas de contagem.
- O **Modelo Cognitivo de Lockwood** desenvolvido na tese de doutorado (LOCKWOOD, 2011), que mostra o papel fundamental desempenhado pelo conjunto de resultados de um problema de contagem, papel esse percebido, principalmente, no trabalho (LOCKWOOD; GIBSON, 2016). A partir disso, utilizamos a enumeração gerada pelo código em Python para conectar o processo de contagem à fórmula.
- O **Modelo dos Campos Semânticos (MCS)** de Lins (2012), que fornece a base para analisar a produção de significado. O Modelo dos Campos Semânticos nos mostra se o aluno está apenas aplicando regras ou se ele realmente está consciente da maneira que está resolvendo o problema.

As atividades propostas no primeiro encontro trás os seguintes objetivos principais: reconhecer os objetos dos problemas; reconhecer as configurações desejadas; construir a árvore de possibilidades; familiarizar-se com o código em Python; reconhecer a função do laço *for* e do condicional *if* ; enunciar o Princípio Fundamental da Contagem.

No segundo encontro temos os seguintes objetivos: utilizar o Princípio Aditivo junto com o Princípio Fundamental da Contagem, quando a resolução do problema divide-se em casos; determinar o número de arranjos por uma aplicação direta do Princípio Fundamental da Contagem.

No terceiro encontro, propomos problemas de contagem onde as configurações que estamos contando são permutações simples e utilizamos código Python recursivo para contar permutações simples.

E, no quarto encontro, o Princípio Fundamental da Contagem gera um excesso na resolução dos problemas propostos, onde os objetos que serão permutados não são todos distintos e quando da determinação da quantidade de grupos(conjuntos), logo, o objetivo principal desse encontro é corrigir a superestimação, construída pelo Princípio Fundamental da Contagem, utilizando o Princípio k para 1.

Cada encontro deve ser aplicado em três aulas com cinquenta minutos cada. Propomos que tais atividades possam ser aplicadas em turmas do $2^{\underline{o}}$ ou $3^{\underline{o}}$ anos do Ensino Médio, dependendo do currículo de cada estado.

Vale ressaltar que as atividades propostas devem ser aplicadas após os alunos terem contato com o mínimo sobre programação com Python. Sugerimos a utilização do app *Pydroid 3* para dispositivos Android que permite escrever e executar código Python no celular ou tablet. O *Pydroid 3* possui um interpretador Python 3 offline, o que significa que é possível executar programas Python sem conexão com a internet.

2 Sequência Didática

A seguir estão as atividades, dividimo-as em quatro encontros, cada encontro com duração de 150 minutos. Vale ressaltar que tais atividades devem ser aplicadas após os alunos terem contato com o mínimo sobre programação com Python. Acreditamos que pode ser muito proveitoso aos professores que desejem aplicá-las identificar possíveis adaptações e/ou desenvolvimento de atividades complementares buscando assim oferecer o material que melhor se adeque aos seus alunos.

As atividades propostas têm como objetivo desenvolver as seguintes habilidades presentes em (BRASIL, 2018):

(EM13MAT310) Resolver e elaborar problemas de contagem envolvendo agrupamentos ordenáveis ou não de elementos, por meio dos princípios multiplicativo e aditivo, recorrendo a estratégias diversas, como o diagrama de árvore.

(EM13MAT405) Utilizar conceitos iniciais de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.

De acordo com as habilidades propostas, esta atividade é destinada ao 2^{o} ou 3^{o} anos do Ensino Médio, dependendo do currículo de cada estado, e requer o conhecimento básico de programação em Python, como já mencionado.

2.1 Primeiro encontro

Objetivos: As atividades propostas no primeiro encontro têm como objetivos principais: reconhecer os objetos dos problemas; reconhecer as configurações desejadas; construir a árvore de possibilidades; familiarizar com o código em Python; reconhecer a função do laço *for* e do condicional *if*; enunciar o Princípio Fundamental da Contagem.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 1:** Considere o seguinte problema: "Quatro estradas A, B, C e D conduzem

ao topo de um morro. De quantos modos diferentes uma pessoa pode subir e descer este morro?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
1  estradas = ['A','B','C','D']
2  contagem = 0
3
4  print('Passeio' '(','Subida',',','Descida',')',':')
5
6  for subida in estradas:
7      for descida in estradas:
8          print('(',subida, ',',descida,')')
9          contagem = contagem + 1
10
11 print('Quantidade de passeios: ', contagem)
```

Código 2.1: Subindo e descendo o morro

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.1.

g) Observe o Código 2.1, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

***Atividade* 2:** Considere o seguinte problema: "Suponhamos que na Atividade 1 a pessoa não queira descer o morro pela estrada que subiu. De quantos modos diferentes ela pode subir e descer este morro?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
estradas = ['A','B','C','D']
contagem = 0

print('Passeio' '(','Subida',',','Descida',')',':')

for subida in estradas:
    for descida in estradas:
        if subida != descida:
            print('(',subida, ',',descida,')')
            contagem = contagem + 1

print('Quantidade de passeios: ', contagem)
```

Código 2.2: Subindo e descendo o morro por caminhos diferentes

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.2.

g) Observe o Código 2.2, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo da linha de código *if subida != descida*?

***Atividade* 3:** Considere o seguinte problema: "Uma moeda é lançada três vezes. Qual o número de sequências possíveis *cara* e *coroa*?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
moeda = ['CARA','COROA']
contagem = 0

```

```
print('Resultados',':  ' )

for lancamento1 in moeda:
    for lancamento2 in moeda:
        for lancamento3 in moeda:
            print('(',lancamento1, ',',lancamento2, ',', lancamento3,')')
            contagem = contagem + 1

print('Quantidade de sequências: ', contagem)
```

Código 2.3: Sequências de cara e coroa

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.3.

g) Observe o Código 2.3, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

***Atividade* 4:** Considere o seguinte problema: "Uma bandeira é formada por quatro listras, que devem ser coloridas usando-se apenas as cores amarelo, rosa e verde, não devendo listras adjacentes ter a mesma cor. De quantos modos pode ser colorida a bandeira?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

e) Agora, execute o seguinte código em Python:

```
cores = ['amarelo','rosa', 'verde']
contagem = 0

for cor1 in cores:
    for cor2 in cores:
        if cor2 != cor1:
            for cor3 in cores:
                if cor3 != cor2:
                    for cor4 in cores:
                        if cor4 != cor3:
                            print('(',cor1, ',',cor2, ',', cor3, ',', cor4,')')
                            contagem = contagem + 1
```

```
13
14 print('Quantidade de maneiras de colorir a bandeira: ', contagem)
```

Código 2.4: Colorindo a bandeira

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.4.

g) Observe o Código 2.4, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo das linhas de código *if cor2 != cor1*, *if cor3 != cor2* e *if cor4 != cor3*?

De acordo com as respostas obtidas ao resolver os problemas das atividades do Primeiro Encontro, chegamos no enunciado do Princípio Fundamental da Contagem, conforme encontramos em (CERIOLI; VIANA, 2012):

Princípio Fundamental da Contagem(PFC). *Seja A um conjunto finito. Se cada elemento de A pode ser formado pela tomada de n decisões, $d_1, d_2, \ldots, d_n$, de maneira que:*

- *a decisão d_1 pode ser tomada de m_1 maneiras;*
- *para cada maneira como a decisão d_1 pode ser tomada, a decisão d_2 pode ser tomada de m_2 maneiras;*
- *para cada maneira como as decisões d_1 e d_2 podem ser tomadas (nesta ordem), a decisão d_3 pode ser tomada de m_3 maneiras;*

  ...;
- *para cada maneira como as decisões $d_1, d_2, \ldots, d_{i-1}$ podem ser tomadas (nesta ordem), a decisão d_i pode ser tomada de m_i maneiras;*

  ...;
- *para cada maneira como as decisões $d_1, d_2, \ldots, d_{i-1}, d_i, \ldots, d_{n-1}$ podem ser tomadas (nesta ordem), a decisão d_n pode ser tomada de m_n maneiras.*

Então, $n(A) = m_1 \times m_2 \times m_3 \times \ldots \times m_i \times \ldots \times m_{n-1} \times m_n$, onde $n(A)$ indica a quantidade de elementos do conjunto A.

2.2 Segundo encontro

Objetivos: As atividades propostas no segundo encontro têm como objetivos principais: reconhecer o "funcionamento" do Princípio Fundamental da Contagem no código em Python; utilizar o Princípio Aditivo junto com o Princípio Fundamental da Contagem, quando a resolução do problema divide-se em casos; determinar o número de arranjos por uma aplicação direta do Princípio Fundamental da Contagem.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 5:** Considere o seguinte problema: "Quantos números pares de dois algarismos podem ser formados no sistema decimal?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
algarismos = [0,1,2,3,4,5,6,7,8,9]
contagem = 0

print('Números pares formados: ')
for unidades in algarismos:
    if unidades % 2 == 0:
        for dezenas in algarismos:
            if dezenas != 0:
                print(dezenas,unidades)
                contagem = contagem + 1

print('Quantidade de números pares com dois algarismos: ', contagem)
```

Código 2.5: Números pares com dois algarismos

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Observe o Código 2.5, qual é o objetivo das linhas de código *if dezenas!= 0* e *if unidades % 2 = 0*?

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

***Atividade* 6:** Considere o seguinte problema: "Quantos números pares de dois algarismos distintos podem ser formados no sistema decimal?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual?

c) Execute o seguinte código em Python:

```
algarismos = [0,1,2,3,4,5,6,7,8,9]
contagem = 0

print('Números pares formados: ')
for unidades in algarismos:
    if unidades % 2 == 0:
        for dezenas in algarismos:
            if dezenas != 0 and dezenas != unidades:
                print(dezenas,unidades)
                contagem = contagem + 1

print('Quantidade de números pares com dois algarismos distintos: ', contagem)
```

Código 2.6: Números pares com dois algarismos distintos

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Observe o Código 2.6, qual é o objetivo das linhas de código *if unidades % 2 = 0* e *if dezenas!= 0 and dezenas != unidades*?

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

Você deve ter notado, observando as configurações obtidas no item c) do problema da Atividade 6, que há a necessidade da utilização de um outro Princípio de Contagem, junto com o Princípio Fundamental da Contagem, para encontrar a quantidade de configurações. Tal Princípio é conhecido como Princípio Aditivo e pode ser enunciado da seguinte forma, conforme podemos encontrar em (CERIOLI; VIANA, 2012):

Sejam A um conjunto finito e $A_1, A_2, \cdots, A_n$ subconjuntos não vazios de A.

1. Dizemos que $A_1, A_2, \cdots, A_n$ *exaurem* A se, para cada elemento a de A, existe i, com $1 \leq i \leq n$, tal que $a \in A_i$; ou seja, se $A_1 \cup A_2 \cup \cdots \cup A_n = A$.

2. Dizemos que $A_1, A_2, \cdots, A_n$ são *disjuntos dois a dois* se, quando examinados aos pares, eles não possuem elementos em comum; ou seja, $A_i \cap A_j = \emptyset$, para todos i, j com $1 \leq i \neq j \leq n$.

3. Dizemos que $A_1, A_2, \cdots, A_n$ são uma partição de A se $A_1, A_2, \cdots, A_n$ exaurem A e são disjuntos dois a dois.

Princípio Aditivo. *Seja A um conjunto finito.*

Se $A_1, A_2, \cdots, A_n$ são uma partição de A, então $n(A) = n(A_1) + n(A_2) + \cdots + n(A_n)$.

***Atividade* 7:** Considere o seguinte problema: "De quantos modos 3 pessoas podem sentar-se em 5 cadeiras em fila?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```python
cadeiras = ['c1','c2','c3','c4','c5']
cadeiras_ocupadas = []

for cadeira_p1 in cadeiras:
    for cadeira_p2 in cadeiras:
            if cadeira_p2 != cadeira_p1:
                for cadeira_p3 in cadeiras:
                    if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:
                        cadeira_ocupada = cadeira_p1,cadeira_p2,cadeira_p3
                        cadeiras_ocupadas.append(cadeira_ocupada)

colunas = 5

print(f"\nCadeiras ocupadas pelas 3 pessoas:  ")
for i, cadeira_ocupada in enumerate(cadeiras_ocupadas):
    print(f"{cadeira_ocupada}", end=",")
    if (i + 1) % colunas == 0:
        print()

if len(cadeiras_ocupadas) % colunas != 0:
    print()

print('Quantidade de maneiras de organizar as 3 pessoas: ', len(cadeiras_ocupadas))
```

Código 2.7: Três pessoas em cinco cadeiras em fila

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

***Atividade* 8:** Considere o seguinte problema: "De quantos modos 6 pessoas podem sentar-se em 10 cadeiras em fila?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido. Se estiver e inseguro, em relação à sua solução, construa um código em Python para resolver o problema.

Observe que as configurações obtidas nos problemas das Atividades 7 e 8 são bem parecidos. Cerioli e Viana (2012) afirmam que: "Quando, para resolver um problema de contagem, podemos aplicar o Princípio Fundamental da Contagem de modo que todas as escolhas envolvidas são feitas em um mesmo conjunto e, a cada escolha a ser feita, os objetos já escolhidos anteriormente não podem mais ser considerados, dizemos que a configuração que estamos contando é um *arranjo simples*."

Será que nos problemas das atividades anteriores já contamos arranjos simples? Se sim, identifique-os.

Definição. Seja A um conjunto com n elementos e $k \in \mathbb{N}$ tal que $1 \leq k \leq n$. Um *arranjo simples* dos n elementos de A, tomados k a k, é uma **sequência** de k elementos distintos de A.

Denotando por $A_{n,k}$ o número de arranjos simples dos n elementos de A, tomados k a k. Temos que:

$$A_{n,k} = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1)$$

.

Note que nem a noção de arranjo simples nem a fórmula para a determinação da quantidade de arranjos simples são muito importantes na resolução dos problemas de

contagem, com esse tipo de configuração, vistos até aqui. Como vimos, podemos determinar o número de arranjos simples por uma aplicação direta do Princípio Fundamental da Contagem.

2.3 Terceiro encontro

Objetivos: As atividades propostas no terceiro encontro têm como objetivos principais: resolver problemas de contagem onde as configurações que estamos contando são permutações simples; utilizar código Python recursivo para contar permutações simples.

Tempo para execução: 3 aulas de 50 minutos.

***Atividade* 9:** Considere o seguinte código em Python que resolve um determinado problema de contagem e execute-o:

```
letras = ['N','U','M','E','R','O']
anagramas = []

for letra_1 in letras:
    for letra_2 in letras:
            if letra_2 != letra_1:
                for letra_3 in letras:
                    if letra_3 != letra_2 and letra_3 != letra_1:
                        for letra_4 in letras:
                            if letra_4 != letra_3 and letra_4 != letra_2 and letra_4 !=
    letra_1:
                                for letra_5 in letras:
                                    if letra_5 != letra_4 and letra_5 != letra_3 and
    letra_5 != letra_2 and letra_5 != letra_1:
                                        for letra_6 in letras:
                                            if letra_6 != letra_5 and letra_6!= letra_4
    and letra_6 != letra_3 and letra_6 != letra_2 and letra_6 != letra_1:
                                                anagrama = (letra_1 + letra_2 + letra_3 +
     letra_4 + letra_5 + letra_6)
                                                anagramas.append(anagrama)


colunas = 20
largura_coluna = 20

print(f"\nAnagramas da palavra NÚMERO: ")
for i, anagrama in enumerate(anagramas):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(anagramas) % colunas != 0:
    print()

```

```
31 print('Quantidade de anagramas: ', len(anagramas))
```

Código 2.8: Código da Atividade 9

a) Pesquise o que são anagramas.

b) Quais são os objetos básicos do problema?

c) Quais são as configurações formadas?

d) Quantas decisões foram tomadas para resolver o problema? Quais foram essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão foi tomada?

e) Quantas configurações foram obtidas? Com isso, em que consiste o problema resolvido pelo Código 2.8?

f) Utilizando o Princípio Fundamental da Contagem, obtenha a solução do problema resolvido pelo Código 2.8.

No problema da Atividade 9, queremos determinar o número de arranjos simples de 6 letras, tomadas 6 a 6, quando isso ocorre dizemos que a configuração que estamos contando é uma *permutação simples*.

Definição. Seja A um conjunto com n elementos. Uma *permutação simples* dos n elementos de A é uma **sequência** sem repetições de todos os n elementos de A.

Denotando por P_n o número de permutações simples dos n elementos de A. Temos que:

$$P_n = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

.

O número $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$ é denotado por $n!$ e é chamado *fatorial* de n.

***Atividade* 10:** Considere o seguinte problema: "Quantos são os anagramas da palavra MOSTRA que começam por vogal e terminam por consoante?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
letras = ['M','O','S','T','R','A']
vogais = ['A','O']
consoantes = ['M','R','S','T']
anagramas = []

for letra_1 in letras:
    if letra_1 in vogais:
        for letra_6 in letras:
            if letra_6 in consoantes:
                for letra_2 in letras:
                    if letra_2 != letra_1 and letra_2 != letra_6:
                        for letra_3 in letras:
                            if letra_3 != letra_2 and letra_3 != letra_1 and
    letra_3 != letra_6:
                                for letra_4 in letras:
                                    if letra_4 != letra_3 and letra_4 != letra_2
    and letra_4 != letra_1 and letra_4 != letra_6:
                                        for letra_5 in letras:
                                            if letra_5 != letra_4 and letra_5 !=
    letra_3 and letra_5 != letra_2 and letra_5 != letra_1 and letra_5 != letra_6:
                                                anagrama = (letra_1 + letra_2 +
    letra_3 + letra_4 + letra_5 + letra_6)
                                                anagramas.append(anagrama)

colunas = 12

print(f"\nAnagramas da palavra MOSTRA que começam por vogal e teminam por consoante
    : ")
for i, anagrama in enumerate(anagramas):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(anagramas) % colunas != 0:
    print()

print('Quantidade de anagramas: ', len(anagramas))
```

Código 2.9: Anagramas com restrições

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

d) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

***Atividade* 11:** Considere o seguinte código recursivo em Python que resolve um determinado problema de contagem:

```python
def permutacoes(palavra):
    if len(palavra) == 1:
        return [palavra]

    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado

palavra = 'NUMERO'

todas_permutacoes = permutacoes(palavra)

colunas = 20

print(f"\nAnagramas da palavra NÚMERO: ")
for i, anagrama in enumerate(todas_permutacoes):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(todas_permutacoes) % colunas != 0:
    print()

print('Quantidade de anagramas: ', len(todas_permutacoes))
```

Código 2.10: Anagramas de NÚMERO e recursividade

a) Execute o código acima. O que esse código fornece como resultado?

b) Compare-o com o Código 2.9. Qual é a diferença?

c) Considere uma palavra com uma quantidade menor de letras, por exemplo ALO, e execute o Código 2.10 passo a passo manualmente, para entender o seu funcionamento.

***Atividade* 12:** Considere a variação do Código 2.10 em Python que determina os anagramas de uma palavra:

```python
def permutacoes(palavra):
    if len(palavra) == 1:
        return [palavra]

```

```
    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado

palavra = str(input("Digite uma palavra: ")).upper()

todas_permutacoes = permutacoes(palavra)

colunas = 8

print(f"\nAnagramas da palavra {palavra}: ")
for i, anagrama in enumerate(todas_permutacoes):
    print(f"{anagrama},", end="")
    if (i + 1) % colunas == 0:
        print()

if len(todas_permutacoes) % colunas != 0:
    print()

print('Quantidade de anagramas: ', len(todas_permutacoes))
```

Código 2.11: Permutações simples e recursividade

a) Execute o código acima, usando como exemplo a palavra AMOR.

b) Usando o Princípio Fundamental da Contagem, determine o número de anagramas da palavra AMOR. Compare o seu resultado com o aquele fornecido pelo Código 2.11.

c) Execute o código usando, agora, como exemplo a palavra AMAR.

d) Analise, atentamente, a saída de execução do código. Você percebeu alguma coisa estranha? O quê?

e) Então, quantos são os anagramas da palavra AMAR?

f) Repita os itens c), d) e e) considerando a palavra BABA.

g) Repita os itens c), d) e e) considerando a palavra AAZA(força em arábico).

h) Por quê o Código 2.11 não funciona com as palavras AMAR, BABA e AAZA?

2.4 Quarto encontro

Objetivos: As atividades propostas no quarto encontro têm como objetivos principais: mostrar que o Princípio Fundamental da Contagem gera um excesso na resolução de problemas onde os objetos que serão permutados não são todos distintos e quando da determinação da quantidade de grupos(conjuntos); diferenciar uma sequência de um conjunto; corrigir a superestimação, construída pelo Princípio Fundamental da Contagem, utilizando o Princípio k para 1.

Tempo para execução: 3 aulas de 50 minutos.

Na Atividade 12, deve-se ter notado que o código que calcula os anagramas de uma palavra só funciona corretamente quando a mesma possui letras distintas, ou seja, o código só calcula a quantidade de permutações simples.

O código da Atividade 13 tem como objetivo corrigir esse problema, isto é, calcular o número de anagramas de qualquer palavra.

***Atividade* 13:** Considere o seguinte código recursivo em Python que determina os anagramas de uma palavra mas, agora, de uma forma mais organizada, resolvendo os problemas que tivemos na Atividade 12:

```
def permutacoes(palavra):

    if len(palavra) == 1:
        return [palavra]

    resultado = []

    for i in range(len(palavra)):
        char_atual = palavra[i]
        resto = palavra[:i] + palavra[i+1:]
        for p in permutacoes(resto):
            resultado.append(char_atual + p)

    return resultado


def remover_duplicatas_sort(lista):

    lista_ordenada = sorted(lista)

    lista_de_listas = []
    grupo_atual = []
    for i in range(len(lista_ordenada)):
        if i == 0 or lista_ordenada[i] == lista_ordenada[i - 1]:
            grupo_atual.append(lista_ordenada[i])
        else:
```

```
27              lista_de_listas.append(grupo_atual)
28              grupo_atual = [lista_ordenada[i]]
29      lista_de_listas.append(grupo_atual)
30
31      for i, uplas in enumerate(lista_de_listas, 1):
32          print(f"{i:3}. {uplas}")
33      print()
34      print("Número de anagramas: ", len(lista_ordenada))
35      print()
36      print("Número de anagramas repetidos em cada caixa: ", len(grupo_atual))
37      print()
38
39
40      resultado = [lista_ordenada[0]]
41
42      for i in range(1, len(lista_ordenada)):
43          if lista_ordenada[i] != lista_ordenada[i-1]:
44              resultado.append(lista_ordenada[i])
45      return resultado
46
47 palavra = str(input("Digite uma palavra qualquer: ")).upper()
48
49 todas_permutacoes = permutacoes(palavra)
50 permutacoes_com_elemetos_repetidos = remover_duplicatas_sort(todas_permutacoes)
51
52
53 print(f"\nAnagramas da palavra {palavra}: ")
54 for i, perm in enumerate(permutacoes_com_elemetos_repetidos, 1):
55      print(f"{i:3}. {perm}")
56 print(f"Quantidade de anagramas da palavra {palavra}: {len(
      permutacoes_com_elemetos_repetidos)}")
```

Código 2.12: Permutações com elementos repetidos

a) Execute o código acima, usando como exemplo a palavra AMAR.

b) O resultado obtido é, de fato, a quantidade de anagramas da palavra AMAR?

c) Execute o código usando, agora, como exemplo a palavra AAZA.

d) O resultado obtido é, de fato, a quantidade de anagramas da palavra AAZA?

e) Execute o Código 2.12, usando a palavra SOSSEGO. Quantos anagramas essa palavra possui?

Na execução do código da Atividade 13, perceba que para descobrir a quantidade de anagramas das palavras com letras repetidas, houve necessidade de "jogar fora" excessos. Note que, em palavras com letras repetidas, para k anagramas obtidos(repetidos) apenas

1 é considerado anagrama da palavra em questão. Temos, assim, a utilização do Princípio k para 1 em conjunto com o Princípio Fundamental da Contagem.

Definição. Sejam A e B conjuntos finitos. Uma *função k para 1* de A em B associa elementos de A a elementos de B, de modo que:

- Cada k elementos de A estão associados a 1 elemento de B.
- Cada elemento de B é o associado de exatamente k elementos de A.

Princípio k para 1. *Sejam A e B conjuntos finitos. Se existe uma função k para 1 de A em B, então $n(A) = k \cdot n(B)$.*

Com isso, podemos concluir, que a quantidade de permutações de n objetos nem todos distintos, em que um deles aparece n_1 vezes(fazendo com que contemos cada anagrama $n_1!$ vezes), outro n_2 vezes(fazendo com que contemos cada anagrama $n_2!$ vezes), e assim por diante, é dada por

$$P_n^{n_1,n_2,\cdots} = \frac{n!}{n_1! \cdot n_2! \cdot \ldots}.$$

f) Determine o número de anagramas das palavras MANIA, CORRER e PASSISTA, usando o Princípio Fundamental da Contagem e o Princípio k para 1.

***Atividade* 14:** Vamos ao seguinte problema: "Considere 5 pessoas A, B, C, D e E. Quantas filas com 3 dessas 5 pessoas podemos formar?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) Execute o seguinte código em Python:

```
def formar_filas(pessoas):

    print("\n" + "=" * 60)
    print("PASSO 1: Forma as filas.")
    print("=" * 60)

    resultado = []
    contador_filas = 0

```

```
    for pessoa_1 in pessoas:
        for pessoa_2 in pessoas:
            if pessoa_2 != pessoa_1:
                for pessoa_3 in pessoas:
                    if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:
                        contador_filas += 1
                        fila_formada = [pessoa_1,pessoa_2,pessoa_3]
                        resultado.append(fila_formada)
                        print(f"Fila {contador_filas:3d}: {fila_formada}")

    print(f"\nTotal de filas diferentes: {contador_filas}")
    return resultado



pessoas = ['A','B','C','D','E']

filas_obtidas = formar_filas(pessoas)
```

Código 2.13: Filas com três pessoas dadas cinco pessoas

d) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

f) Esse problema é parecido com algum problema que já foi resolvido? Qual?

g) E quantas filas com 4 pessoas podemos formar?

***Atividade* 15:** Agora, vamos a esse problema: "Considere 5 pessoas A, B, C, D e E. Quantos grupos com 3 dessas 5 pessoas podemos formar?". Responda as questões a seguir, antes de resolvê-lo.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

c) A solução desse problema é igual à solução do problema da Atividade 14? Ou seja, a quantidade de filas é igual à quantidade de grupos? Justifique.

d) Se a resposta do item anterior for NÃO, corrija a resposta, obtendo, assim, a resposta correta.

No problema da Atividade 14, estamos, claramente, buscando a quantidade de arranjos simples de 5 pessoas tomadas 3 a 3. Já no problema da Atividade 15, mais uma vez, se utilizarmos o Código 2.13 teremos excesso e, para corrigir a solução, necessitamos utilizar o Princípio k para 1.

No problema da Atividade 15 temos 5 pessoas e devemos efetuar uma escolha na qual 3 pessoas são tomados simultaneamente, dizemos, então, que a configuração que estamos contando é uma *combinação simples*.

Antes de definir formalmente *combinação simples*, vejamos a Atividade 16.

***Atividade* 16:** Considere o seguinte código em Python que resolve um determinado problema de contagem:

```
def formar_filas(pessoas):

    print("\n" + "=" * 60)
    print("PASSO 1: Forma as filas.")
    print("=" * 60)

    resultado = []
    contador_filas = 0

    for pessoa_1 in pessoas:
        for pessoa_2 in pessoas:
            if pessoa_2 != pessoa_1:
                for pessoa_3 in pessoas:
                    if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:
                        contador_filas += 1
                        fila_formada = [pessoa_1,pessoa_2,pessoa_3]
                        resultado.append(fila_formada)
                        print(f"Fila {contador_filas:3d}: {fila_formada}")

    print(f"\nTotal de filas diferentes: {contador_filas}")
    return resultado

def ordenar_e_imprimir_filas(filas):

    print("\n" + "=" * 60)
    print("PASSO 2: Ordena cada fila individualmente")
    print("=" * 60)

    filas_ordenadas = []
    ordenadas = []

    print("\nTodas as filas ORDENADAS:")
    print("-" * 60)

    for idx, fila in enumerate(filas, 1):
        fila_ordenada = sorted(fila)
        filas_ordenadas.append(fila_ordenada)
        print(f"Fila {idx}: {fila_ordenada}")
```

```
39
40      return filas_ordenadas
41
42  def remover_duplicatas_sort(lista):
43
44      print("\n" + "=" * 60)
45      print("PASSO 3: Coloca as filas iguais em uma mesma caixa")
46      print("=" * 60)
47
48      lista_ordenada = sorted(lista)
49
50      lista_de_listas = []
51      grupo_atual = []
52      for i in range(len(lista_ordenada)):
53          if i == 0 or lista_ordenada[i] == lista_ordenada[i - 1]:
54              grupo_atual.append(lista_ordenada[i])
55          else:
56              lista_de_listas.append(grupo_atual)
57              grupo_atual = [lista_ordenada[i]]
58
59      lista_de_listas.append(grupo_atual)
60
61      for i, uplas in enumerate(lista_de_listas, 1):
62          print(f"{i:3}. {uplas}")
63      print()
64      print("Número de filas: ", len(lista_ordenada))
65      print()
66      print("Depois que as filas são ordenadas, temos: ", len(grupo_atual) , " filas em
        cada caixa.")
67      print()
68
69      resultado = [lista_ordenada[0]]
70
71      for i in range(1, len(lista_ordenada)):
72          if lista_ordenada[i] != lista_ordenada[i-1]:
73              resultado.append(lista_ordenada[i])
74      return resultado
75
76
77  pessoas = ['A','B','C','D','E']
78
79  filas_obtidas = formar_filas(pessoas)
80  ordenadas = ordenar_e_imprimir_filas(filas_obtidas)
81  grupos = remover_duplicatas_sort(ordenadas)
82
83  print("Todas os grupos:")
84  for i, perm in enumerate(grupos, 1):
85      print(f"{i:3}. {perm}")
86  print(f"Número de grupos: {len(grupos)}")
```

Código 2.14: Código da Atividade 16

a) Execute o código acima.

b) O que esse código fornece como resultado?

c) Compare-o com o Código 2.13. Qual é a diferença?

d) Esse código fornece a solução do problema da Atividade 15?

e) Usando a ideia do código acima, dadas cinco pessoas quantos grupos com quatro pessoas podemos formar? E, dadas dez pessoas quantos grupos com quatro pessoas podemos formar?

Finalmente, temos a definição formal para Combinações Simples.

Definição. Seja A um conjunto com n elementos e $k \in \mathbb{N}$ tal que $1 \leq k \leq n$. Uma *combinação simples* dos n elementos de A, tomados k a k, é um **conjunto** de k elementos distintos de A.

Pela definição, como uma combinação é um **conjunto** com elementos de A. A ordem em que os elementos estão listados não é levada em conta na determinação da combinação.

Denotando por $C_{n,k}$ o número de combinações simples dos n elementos de A, tomados k a k. O número de combinações simples dos n elementos de A, tomados k a k, é dado pela fórmula

$$C_{n,k} = \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-(k-1))}{k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1} = \frac{A_{n,k}}{P_k}.$$

3 Resultados Esperados

3.1 Primeiro encontro

***Atividade* 1:** Considere o seguinte problema: "Quatro estradas A, B, C e D conduzem ao topo de um morro. De quantos modos diferentes uma pessoa pode subir e descer este morro?".

> Geralmente, a palavra "diferentes" no enunciado do problema, pode causar um "estranhamento", mas, tal palavra refere-se ao par completo, subida e descida, e não necessariamente às estradas individuais. Assim, espera-se a seguinte Leitura Plausível, por exemplo, subir por B e descer por B é um "modo diferente" de subir por B e descer por D.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

> Nesse item temos a identificação dos elementos que formam o "mundo" do problema.
>
> - Objetos básicos: são os elementos disponíveis para a ação. Aqui, são as 4 estradas: $\{A, B, C, D\}$.
> - Configurações: A configuração é um trajeto de subida e descida. Matematicamente, é um par ordenado $(Subida, Descida)$.
>
> Ao resolver este item, o aluno é convidado a colocar-se no papel da pessoa que deve realizar a ação solicitada pelo problema, ou seja, o aluno deve se colocar no papel da pessoa que deve subir e descer o morro.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras

cada decisão pode ser tomada?

Precisamos tomar 2 decisões:1ª)Escolher a estrada de subida; 2ª) Escolher a estrada de descida.
Não há restrição. O enunciado diz "subir e descer", mas não diz "descer por uma estrada diferente". Se o aluno assumir que não pode repetir a estrada, ele inseriu uma regra que não existe no texto.
Temos, então:
1ª decisão(subida): 4 maneiras $\rightarrow \{A, B, C, D\}$;
2ª decisão(descida): 4 maneiras $\rightarrow \{A, B, C, D\}$.
Com a postura exercida pelo aluno, descrita no item anterior, ele consegue vislumbrar que o problema pode ser dividido em dois problemas menores, primeiro ele resolve o problema "escolher a estrada para subir o morro" e depois, já tendo subido o morro, resolve o segundo problema "escolher a estrada para descer do morro".

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Agora é o momento de montar os trajetos. Geralmente, o aluno fixa a estrada de subida e, a partir daí, escolhe a estrada de descida:

- **subida** $\rightarrow A \Rightarrow$ **Possíveis configurações:** $(A,A),(A,B),(A,C),(A,D)$.
- **subida** $\rightarrow B \Rightarrow$ **Possíveis configurações:** $(B,A),(B,B),(B,C),(B,D)$.
- **subida** $\rightarrow C \Rightarrow$ **Possíveis configurações:** $(C,A),(C,B),(C,C),(C,D)$.
- **subida** $\rightarrow D \Rightarrow$ **Possíveis configurações:** $(D,A),(D,B),(D,C),(D,D)$.

A partir daí, temos a seguinte árvore de possibilidades.

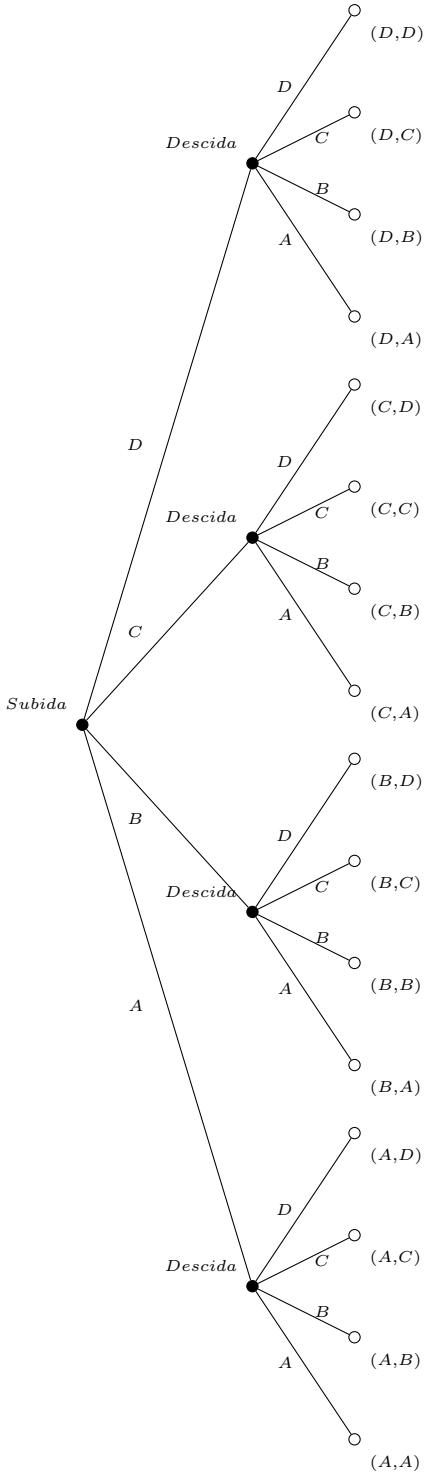


Figura 1: Árvore de Possibilidades da Atividade 1.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Obtemos 16 configurações.

Para esse item poderemos obter as seguintes enunciações como respostas:

$$4 + 4 + 4 + 4 = 16 \text{ ou } 4 \times 4 = 16.$$

e) Agora, execute o código 2.1.

Temos o seguinte resultado ao executar o Código 2.1.

```
Passeio( Subida , Descida ) :
( A , A )
( A , B )
( A , C )
( A , D )
( B , A )
( B , B )
( B , C )
( B , D )
( C , A )
( C , B )
( C , C )
( C , D )
( D , A )
( D , B )
( D , C )
( D , D )
Quantidade de passeios:  16
>>>
```

Figura 2: Saída do Código 2.1.
Fonte: Autoria própria(2026).

Mas, executar não é simplesmente "rodar" o código. Há a necessidade de analisar o que o código faz:

I. **Linha 1:** *estradas = ['A','B','C','D']* → Define os objetos básicos do problema.

II. **Linha 6:** *for subida in estradas:* → Representa a 1ª Decisão. O computador "escolhe" uma estrada e "fixa" ela.

III. **Linha 7:** *for descida in estradas:* → Representa a 2ª Decisão. Para cada subida que está "fixada", o computador testa todas as descidas possíveis.

O fato de um *for* estar dentro do outro (aninhado) é a representação computacional da multiplicação matemática (4×4) e da ramificação da árvore.

IV. **Linha 8:** *print(...)* $\rightarrow$ Materializa a configuração (o par ordenado).

V. **Linha 9:** *contagem = contagem + 1* $\rightarrow$ Realiza a contagem passo a passo.

VI. **Resultado da execução do código:** O programa imprimirá linha por linha:*('A','A'),('A','B')... até ...('D','D').*

E ao final: *Quantidade de passeios: 16*

Visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

- **Laço externo:** Escolhe *subida = 'A'*, o computador "fixa" o A.
- **Entra no Laço interno:**
  - **Laço interno:**Escolhe *descida = 'A'* $\rightarrow$ Imprime (A, A).
  - **Laço interno:** Escolhe *descida = 'B'* $\rightarrow$ Imprime (A, B). (Note que a subida continua $'A'$).
  - **Laço interno:** Escolhe *descida = 'C'* $\rightarrow$ Imprime (A, C).
  - **Laço interno:** Escolhe *descida = 'D'* $\rightarrow$ Imprime (A, D).
- **Fim do Laço interno:** Acabaram as opções de descida para o $'A'$.
- **Laço externo:** Escolhe *subida = 'B'*.
- **O Laço interno recomeça do zero:**
  - **Laço interno:** Escolhe *descida = 'A'* $\rightarrow$ Imprime (B, A).
  - **Laço interno:** Escolhe *descida = 'B'* $\rightarrow$ Imprime (B, B)
  - ... e assim por diante.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.1.

O aluno deve entender que "Subir por A e Descer por B" é um significado diferente de "Subir por B e Descer por A".
A Árvore mostra visualmente todos os passeios. O Python "prova" a resposta gerando concretamente todas as 16 possibilidades, usando a mesma ideia utilizada pela maioria dos alunos descrita no item c).

g) Observe o Código 2.1, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

Foram utilizados 2 laços for. E cada um desses laços representa uma decisão.
O laço externo *for subida in estradas:* representa a 1ª Decisão. Na árvore de possibilidades, isso representa os galhos principais que saem do tronco.
O laço interno *for descida in estradas:* representa a 2ª Decisão. Na árvore de possibilidades, estes são os ramos menores que saem de cada galho principal.

***Atividade* 2:** Considere o seguinte problema: "Suponhamos que na Atividade 1 a pessoa não queira descer o morro pela estrada que subiu. De quantos modos diferentes ela pode subir e descer este morro?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são os elementos disponíveis para a ação. Aqui, são as 4 estradas: $\{A, B, C, D\}$.
- Configurações: A configuração é um trajeto de subida e descida. Matematicamente, é um par ordenado $(Subida, Descida)$.

  Diferente da Atividade 1, agora o par (A, A), por exemplo, tornou-se ilegítimo. Ou seja, devemos ter que $Subida \neq Descida$.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Diferente do problema anterior, temos aqui a introdução de uma restrição de dependência (“não descer pela mesma estrada”).
Temos, então:
1ª decisão(subida): 4 maneiras $\rightarrow \{A, B, C, D\}$;
2ª decisão(descida): O número de maneiras depende da escolha anterior. Temos então, 3 maneiras. Por exemplo, se a estrada de subida é a A então a estrada de descida pode ser $\{B, C, D\}$.
Quando o professor perguntar “Por que 3?”, a Justificação do aluno será: “Porque eu já usei uma na subida e não posso repetir”. Essa fala explicita a restrição do problema.

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Fixando a estrada de subida e, a partir daí, escolhendo a estrada de descida:

- **subida** $\rightarrow A \Rightarrow$ **Possíveis configurações:** $(A,B),(A,C),(A,D)$.
- **subida** $\rightarrow B \Rightarrow$ **Possíveis configurações:** $(B,A),(B,C),(B,D)$.
- **subida** $\rightarrow C \Rightarrow$ **Possíveis configurações:** $(C,A),(C,B),(C,D)$.
- **subida** $\rightarrow D \Rightarrow$ **Possíveis configurações:** $(D,A),(D,B),(D,C)$.

A partir daí, temos a seguinte árvore de possibilidades.

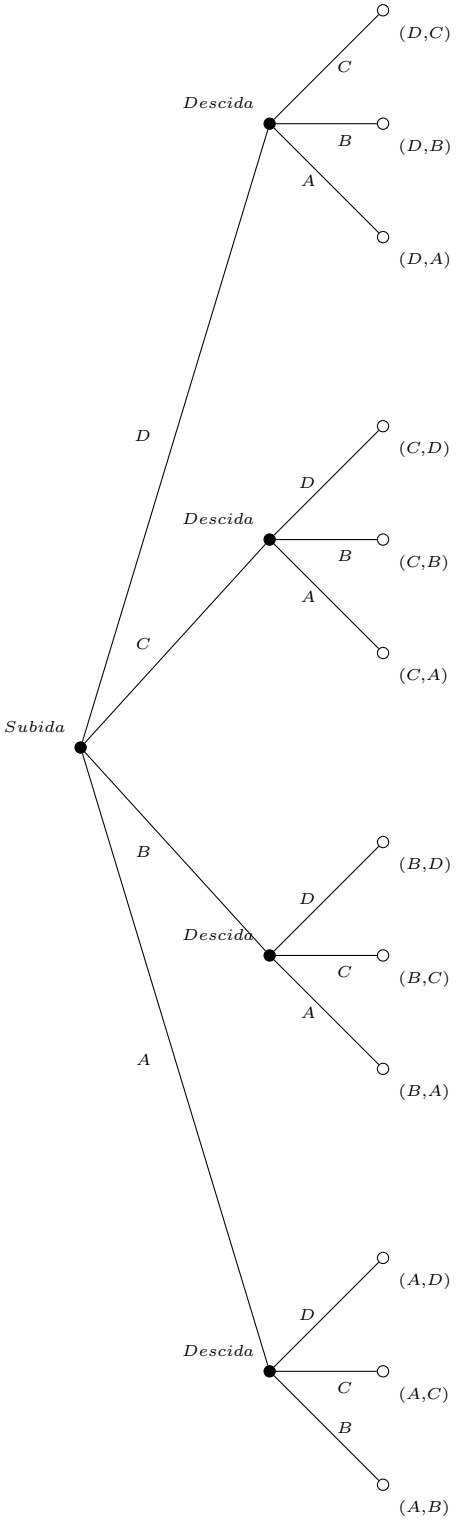


Figura 3: Árvore de Possibilidades da Atividade 2.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Obtemos 12 configurações.

Para esse item poderemos obter as seguintes enunciações como respostas:

$$3+3+3+3=12 \text{ ou } 4\times 3=12.$$

e) Agora, execute o código 2.2

Obtemos o seguinte resultado.

```
Passeio( Subida , Descida ) :
( A , B )
( A , C )
( A , D )
( B , A )
( B , C )
( B , D )
( C , A )
( C , B )
( C , D )
( D , A )
( D , B )
( D , C )
Quantidade de passeios:  12
>>>
```

Figura 4: Saída do Código 2.2.
Fonte: Autoria própria(2026).

I. **Linha 1:** *estradas = ['A','B','C','D']* → Define os objetos básicos do problema.

II. **Linha 6:** *for subida in estradas:* → Representa a 1ª Decisão. O computador "escolhe" uma estrada e "fixa" ela.

III. **Linha 7:** *for descida in estradas:* → Representa a 2ª Decisão. Para cada subida que está "fixada", o computador testa todas as descidas possíveis.

IV. **Linha 8:** *if subida != descida* → Esta linha é a materialização da restrição de que a repetição é proibida. Ela atua como um filtro. Ela diz: "Eu gerei o par (A,A), mas a restrição do problema diz que ele não serve. Descarte-o."

V. **Linha 9:** *print(...)* → Materializa a configuração (o par ordenado).

VI. **Linha 10:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

VII. **Resultado da execução do código:** O programa imprimirá linha por linha:*('A','B')... até ...('D','C')*.

E ao final: *Quantidade de passeios: 12*

Visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

- **Laço externo:** Escolhe *subida = 'A'*, o computador "fixa" o A.
- **Entra no Laço interno:**
  - **Laço interno:**Escolhe *descida = 'A'* → Não Imprime (A, A), devido a restrição.
  - **Laço interno:** Escolhe *descida = 'B'* → Imprime (A, B). (Note que a subida continua $'A'$).
  - **Laço interno:** Escolhe *descida = 'C'* → Imprime (A, C).
  - **Laço interno:** Escolhe *descida = 'D'* → Imprime (A, D).
- **Fim do Laço interno:** Acabaram as opções de descida para o $'A'$.
- **Laço externo:** Escolhe *subida = 'B'*.
- **O Laço interno recomeça do zero:**
  - **Laço interno:** Escolhe *descida = 'A'* → Imprime (B, A).
  - **Laço interno:** Escolhe *descida = 'B'* → Não Imprime (B, B), devido a restrição.
  - ... e assim por diante.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.2.

A Árvore mostra visualmente todos os passeios. O Python "prova" a resposta gerando concretamente todas as 12 possibilidades, usando a mesma ideia utilizada pela maioria dos alunos descrita no item c).

g) Observe o Código 2.2, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo da linha de código *if subida != descida*?

Foram utilizados 2 laços for. E cada um desses laços representa uma decisão. O laço externo *for subida in estradas:* representa a 1ª Decisão. Na árvore de possibilidades, isso representa os galhos principais que saem do tronco.
O laço interno *for descida in estradas:* representa a 2ª Decisão. Na árvore de possibilidades, estes são os ramos menores que saem de cada galho principal.
É importante notar que o código está percorrendo um espaço de $4 \times 4 = 16$ possibilidades totais, mas, a linha *if subida != descida*, como visto na resolução do item e), é a materialização da restrição de que a repetição é proibida. Essa linha exclui as 4 possibilidades (A,A),(B,B),(C,C),(D,D) na resolução do problema.
Enquanto na construção da árvore de possibilidades a restrição do problema está "escondida" na quantidade de maneiras de tomar a 2ª Decisão(3 maneiras), no código Python a restrição está escrita e visível na linha *if subida != descida:*.

***Atividade* 3:** Considere o seguinte problema: "Uma moeda é lançada três vezes. Qual o número de sequências possíveis *cara* e *coroa*?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são os elementos disponíveis para a ação. Aqui, são as 2 faces da moeda: $\{CARA, COROA\}$.
- Configurações: A configuração é um resultado do 1º lançamento, do 2º lançamento e do 3º lançamento. Matematicamente, é uma tripla ordenada $(lancamento1, lancamento2, lancamento3)$.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Precisamos tomar 3 decisões:1ª)Escolher a face vista no 1º lançamento; 2ª) Escolher a face vista no 2º lançamento; 3ª) Escolher a face vista no 3º lançamento.

Não há restrição no problema. Se o aluno assumir que não pode repetir a face vista em um lançamento anterior, ele inseriu uma regra que não existe no texto.

Temos, então:

1ª decisão(1º lançamento): 2 maneiras $\rightarrow \{CARA, COROA\}$;

2ª decisão(2º lançamento): 2 maneiras $\rightarrow \{CARA, COROA\}$;

3ª decisão(3º lançamento): 2 maneiras $\rightarrow \{CARA, COROA\}$.

Com a postura ativa assumida pelo aluno no item anterior, ele consegue vislumbrar que o problema pode ser dividido em três problemas menores (dividir para conquistar), primeiro ele resolve o problema "escolher a possível face ao lançar a moeda pela primeira vez", já tendo feito a primeira escolha, resolve o segundo problema "escolher a possível face ao lançar a moeda pela segunda vez" e, feito isso, resolve o terceiro problema "escolher a possível face ao lançar a moeda pela terceira vez".

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Podemos formar as seguintes configurações:

1ª e 2ª faces iguais a *cara* $\rightarrow$ (*cara,cara,cara*), (*cara,cara,coroa*);

1ª face *cara* e 2ª face *coroa* $\rightarrow$ (*cara,coroa,cara*), (*cara,coroa,coroa*);

1ª face *coroa* e 2ª face *cara* $\rightarrow$ (*coroa,cara,cara*), (*coroa,cara,coroa*);

1ª e 2ª faces iguais a *coroa* $\rightarrow$ (*coroa,coroa,cara*), (*coroa,coroa,coroa*).

Que nos fornece a seguinte árvore de possibilidades:

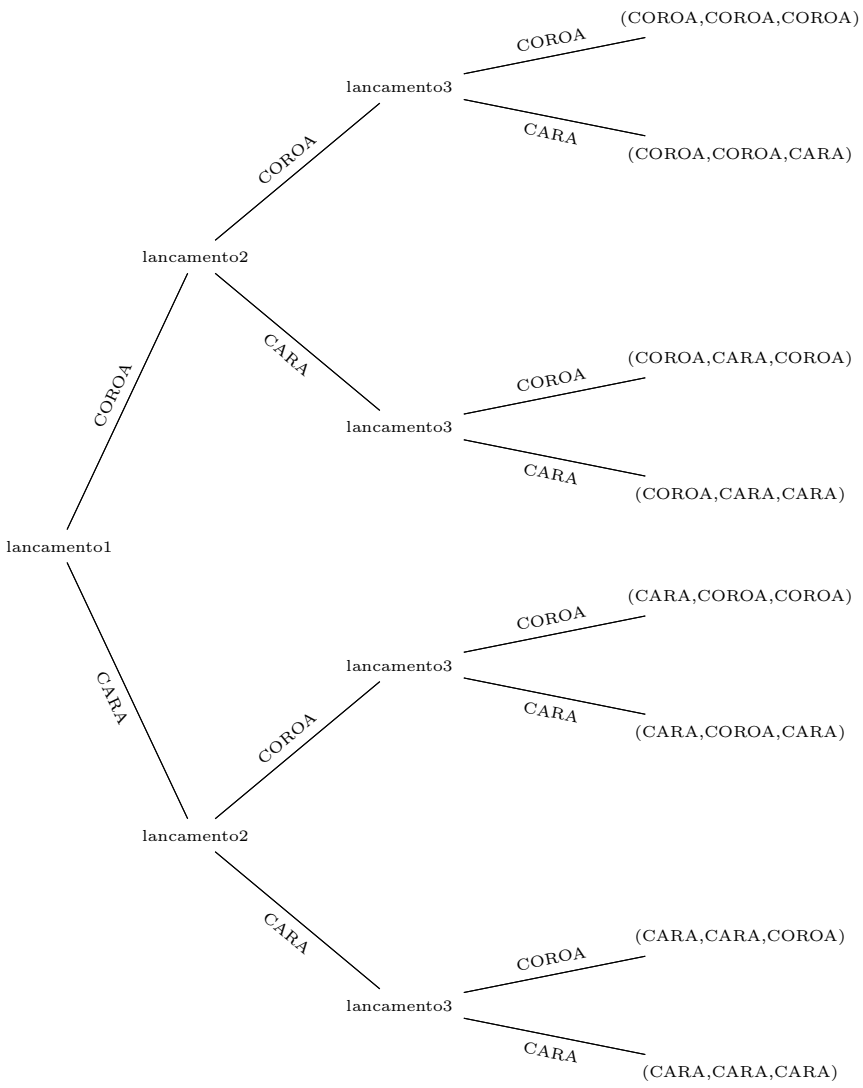


Figura 5: Árvore de Possibilidades da Atividade 3.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Obtemos 8 configurações.

Para esse item poderemos obter as seguintes enunciações como respostas:

$$2 + 2 + 2 + 2 = 8 \text{ ou } 2 \times 2 \times 2 = 12.$$

e) Agora, execute o código 2.3.

```
Resultados :
( CARA , CARA , CARA )
( CARA , CARA , COROA )
( CARA , COROA , CARA )
( CARA , COROA , COROA )
( COROA , CARA , CARA )
( COROA , CARA , COROA )
( COROA , COROA , CARA )
( COROA , COROA , COROA )
Quantidade de sequências:  8
```

Figura 6: Saída do Código 2.3.
Fonte: Autoria própria(2026).

Analisando o que o código faz:

I. **Linha 1:** *moeda = ['CARA','COROA']* → Define os objetos básicos do problema.

II. **Linha 6:** *for lancamento1 in moeda:* → Representa a 1ª Decisão. O computador "escolhe" uma face e "fixa" ela.

III. **Linha 7:** *for lancamento2 in moeda:* → Representa a 2ª Decisão. Para cada face que está "fixada" na 1ª decisão, o computador "escolhe" uma face e "fixa".

IV. **Linha 8:** *for lancamento3 in moeda:* → Representa a 3ª Decisão. Para cada face que será "fixada" na 2ª decisão, o computador testa todas as outras faces possíveis.

V. **Linha 9:** *print(...)* → Materializa a configuração (a tripla ordenada).

VI. **Linha 10:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

VI. **Resultado da execução do código:** O programa imprimirá linha por linha:*('CARA','CARA','CARA'),('CARA','CARA','COROA')...até ...('COROA','COROA','COROA').*

E ao final: *Quantidade de sequências: 8*

Visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

- **Laço externo:** Escolhe *lancamento1 = 'CARA'*, o computador "fixa" $CARA$.
- **Entra no 1º Laço interno:**
  - **Laço interno:** Escolhe *lancamento2 = 'CARA'*,o computador "fixa" $CARA$.
  - **Entra no 2º Laço interno:**
    - **Laço interno:** Escolhe *lancamento3 = 'CARA'* $\rightarrow$ Imprime ($CARA$,$CARA$,$CARA$)
    - **Laço interno:** Escolhe *lancamento3 = 'COROA'* $\rightarrow$ Imprime ($CARA$,$CARA$,$COROA$)
  - **Fim do 2º laço interno:** Acabaram as opções do 3º lançamento para $'CARA'$.
  - **Laço interno:** Escolhe *lancamento2 = 'COROA'*.
  - **Entra no 2º Laço interno:**
    - **Laço interno:** Escolhe *lancamento3 = 'CARA'* $\rightarrow$ Imprime ($CARA$,$COROA$,$CARA$)
    - **Laço interno:** Escolhe *lancamento3 = 'COROA'* $\rightarrow$ Imprime ($CARA$,$COROA$,$COROA$)
  - **Fim do 2º laço interno:** Acabaram as opções do 3º lançamento para $'COROA'$.
- **Fim do 1º Laço interno.**
- **Laço externo:** Escolhe *lancamento1 = 'COROA'*.
- **Entra no 1º Laço interno:**
  - **Recomeça todo o processo.**

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.3.

> A Árvore mostra visualmente todos os resultados possíveis. O Python "prova" a resposta gerando concretamente todos os 8 resultados.

g) Observe o Código 2.3, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão?

> Foram utilizados 3 laços for. E cada um desses laços representa uma decisão. O laço externo *for lancamento1 in moeda:* representa a 1ª Decisão. Na árvore de possibilidades, isso representa os galhos principais que saem do tronco.
> O 1º laço interno *for lancamento2 in moeda:* representa a 2ª Decisão. Na árvore de possibilidades, estes são os galhos menores que saem de cada galho principal.
> O 2º laço interno *for lancamento3 in moeda:* representa a 3ª Decisão. Na árvore de possibilidades, estes são os ramos menores que saem de cada galho menor.
> Se, por acaso, a moeda fosse lançada mais uma vez, teríamos no problema quatro decisões e o nosso código, como consequência, teria quatro comandos *for*.

***Atividade* 4:** Considere o seguinte problema: "Uma bandeira é formada por quatro listras, que devem ser coloridas usando-se apenas as cores amarelo, rosa e verde, não devendo listras adjacentes ter a mesma cor. De quantos modos pode ser colorida a bandeira?".

> Esta atividade, a princípio, produz um estranhamento em alguns alunos: "Como vou colorir a bandeira com quatro listras, se possuo apenas três cores para usar?". Ou seja, eles acham que uma das listras ficará sem colorir.
> Por isso, para esse problema, é interessante propor que o aluno desenhe uma bandeira com quatro listras e pinte-a, obedecendo as regras do problema. Com isso, eles percebem que, pelo menos, uma das cores pode se repetir.
> Então, o aluno começa a produzir o significado de que a *listra 1* e a *listra 3*, por exemplo, podem ter a mesma cor, mas a *listra 1* e a *listra 2* não.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são os elementos disponíveis para a ação. Aqui, são as 3 cores disponíveis: $\{amarelo, rosa, verde\}$.
- Configurações: A configuração é uma bandeira inteira pintada corretamente, respeitando a regra de que listras adjacentes possuem cores diferentes. Matematicamente, é uma quádrupla ordenada $(cor1, cor2, cor3, cor4)$, onde $cor1 \neq cor2$, $cor2 \neq cor3$ e $cor3 \neq cor4$.

O aluno deve se colocar no papel da pessoa que está pintando a bandeira.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Precisamos tomar 4 decisões:1ª) Escolher a cor da primeira listra a ser colorida; 2ª) Escolher a cor da segunda listra a ser colorida; 3ª) Escolher a cor da terceira listra a ser colorida; 4ª) Escolher a cor da quarta listra a ser colorida.
Há uma restrição no problema. O enunciado diz "não devendo listras adjacentes ter a mesma cor". Isto é, há uma restrição de dependência local.
Sem perda de generalidade, escolheremos a seguinte bandeira para representar a bandeira que queremos colorir:

1ª listra
2ª listra
3ª listra
4ª listra

Agora, precisamos começar a colorir a bandeira.
O aluno poderia, agora, fazer a seguinte enunciação:"Vou colorir primeiro a 1ª listra da bandeira e depois a 3ª listra, não estou querendo me preocupar com a restrição do problema agora, e, depois vou colorir a 2ª e 4ª listras."

Nesse caso, o aluno pode enunciar que:
"1ª decisão(cor da primeira listra): 3 maneiras → $\{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
2ª decisão(cor da terceira listra): 3 maneiras → $\{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *amarelo*;
3ª decisão(cor da segunda listra): 1 maneira → $\{verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;
4ª decisão(cor da quarta listra): 2 maneiras → $\{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."
Mas, também, poderíamos ter a seguinte enunciação:
"1ª decisão(cor da primeira listra): 3 maneiras → $\{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
2ª decisão(cor da terceira listra): 3 maneiras → $\{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
3ª decisão(cor da segunda listra): 2 maneiras → $\{amarelo, verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;
4ª decisão(cor da quarta listra): 2 maneiras → $\{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."
A partir das duas enunciações, observemos que a resposta da 3ª decisão depende da cor escolhida para colorir a 3ª listra, a realização da 2ª decisão. Com isso, temos dois casos a resolver: o caso em que as cores das 1ª e 3ª listras são iguais e o caso em que as cores dessas listras são diferentes.
A estratégia escolhida pelo aluno resolve o problema mas, chega nessa indecisão, obrigando que o problema seja resolvido em dois casos, torna-se mais trabalhoso. Em momento oportuno, voltaremos a esse problema e utilizaremos a estratégia acima para resolvê-lo.
Mas, tem como contornar, por enquanto, essa coisa "desagradável". Para isso, Morgado *et al.* (2006) recomendam a seguinte estratégia: "Pequenas dificuldades adiadas costumam transformar-se em grandes dificuldades. Se alguma decisão é mais complicada que as demais, ela deve ser tomada em primeiro lugar."

Ou seja, usando a estratégia acima, temos que a primeira listra a ser colorida na bandeira pode ser qualquer uma das quatro existentes. Devemos ter atenção na escolha das demais listras, a segunda listra escolhida, por exemplo, deve ser adjacente à listra anteriormente colorida. Então, podemos ter o seguinte:
1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$;
2ª decisão(cor da segunda listra): O número de cores disponíveis depende da escolha anterior. Temos então, 2 cores. Por exemplo, se a cor da primeira listra é *rosa* então a cor da segunda listra pode ser $\{amarelo, verde\}$.
Quando o professor perguntar "Por que 2?", a Justificação do aluno será: "Porque eu não posso usar a mesma cor que usei na listra anterior, que é adjacente a listra que estou colorindo agora". Essa fala explicita a restrição do problema.
3ª decisão(cor da terceira listra): O número de cores disponíveis depende da escolha anterior. Temos então, 2 cores. Por exemplo, se a cor da segunda listra é *verde* então a cor da terceira listra pode ser $\{amarelo, rosa\}$.
4ª decisão(cor da quarta listra): O número de cores disponíveis depende da escolha anterior. Temos então, 2 cores. Por exemplo, se a cor da terceira listra é *amarelo* então a cor da quarta listra pode ser $\{verde, rosa\}$.

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

Usando as mesmas estratégias utilizadas nas atividades anteriores, obtemos as configurações elencadas na árvore de possibilidades da Figura 7.

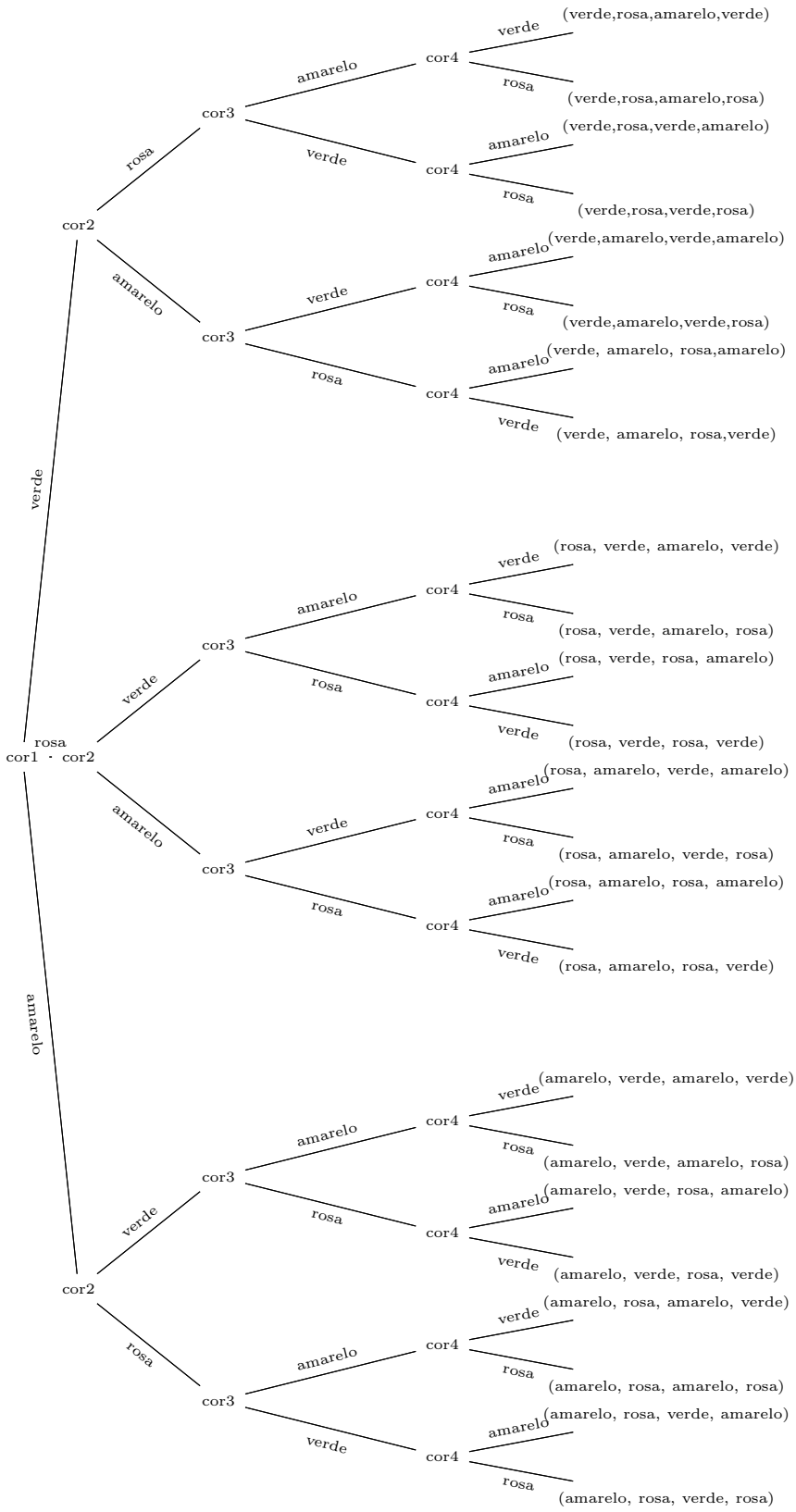


Figura 7: Árvore de Possibilidades da Atividade 4.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

Obtemos 24 configurações.

e) Agora, execute o código 2.4.

```
( amarelo , rosa , amarelo , rosa )
( amarelo , rosa , amarelo , verde )
( amarelo , rosa , verde , amarelo )
( amarelo , rosa , verde , rosa )
( amarelo , verde , amarelo , rosa )
( amarelo , verde , amarelo , verde )
( amarelo , verde , rosa , amarelo )
( amarelo , verde , rosa , verde )
( rosa , amarelo , rosa , amarelo )
( rosa , amarelo , rosa , verde )
( rosa , amarelo , verde , amarelo )
( rosa , amarelo , verde , rosa )
( rosa , verde , amarelo , rosa )
( rosa , verde , amarelo , verde )
( rosa , verde , rosa , amarelo )
( rosa , verde , rosa , verde )
( verde , amarelo , rosa , amarelo )
( verde , amarelo , rosa , verde )
( verde , amarelo , verde , amarelo )
( verde , amarelo , verde , rosa )
( verde , rosa , amarelo , rosa )
( verde , rosa , amarelo , verde )
( verde , rosa , verde , amarelo )
( verde , rosa , verde , rosa )
Quantidade de maneiras de colorir a bandeira:  24
```

Figura 8: Saída do Código 2.4.
Fonte: Autoria própria(2026).

Analisando o que o código faz:

I. **Linha 1:** *cores = ['amarelo','rosa','verde']* → Define os objetos básicos do problema.

II. **Linha 4:** *for cor1 in cores:* → Representa a 1ª Decisão. O computador "escolhe" uma cor para a primeira listra e "fixa" ela.

III. **Linha 5:** *for cor2 in cores:* → Representa a 2ª Decisão. Para cada cor que está "fixada" na 1ª decisão, o computador testa todas as outras cores possíveis.

IV. **Linha 6:** *if cor2 != cor1* → Esta linha é a materialização da restrição de que a cor da segunda listra não pode ser a mesma cor da primeira listra. Ela atua como um filtro. Ela diz: "Eu gerei o par (*amarelo*,*amarelo*), mas a restrição do problema diz que ele não serve. Descarte-o."

V. **Linha 7:** *for cor3 in cores:* → Representa a 3ª Decisão. Para cada cor que está "fixada" na 2ª decisão, o computador testa todas as outras cores possíveis.

VI. **Linha 8:***if cor3 != cor2* → Esta linha é a materialização da restrição de que a cor da terceira listra não pode ser a mesma cor da segunda listra. Ela atua como um filtro. Ela diz: "Eu gerei o par (*amarelo*,*verde*,*verde*), mas a restrição do problema diz que ele não serve. Descarte-o."

VII. **Linha 9:** *for cor4 in cores:* → Representa a 4ª Decisão. Para cada cor que está "fixada" na 3ª decisão, o computador testa todas as outras cores possíveis.

VIII. **Linha 10:***if cor4 != cor3* → Esta linha é a materialização da restrição de que a cor da quarta listra não pode ser a mesma cor da terceira listra. Ela atua como um filtro. Ela diz: "Eu gerei o par (*amarelo*,*verde*,*amarelo*,*amarelo*), mas a restrição do problema diz que ele não serve. Descarte-o."

IX. **Linha 11:** *print(...)* → Materializa a configuração (a quádrupla ordenada).

X. **Linha 12:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

XI. **Resultado da execução do código:** O programa imprimirá linha por linha:

('amarelo','rosa','amarelo','rosa'),('amarelo','rosa','amarelo','verde')...
até ...('verde','rosa','verde','rosa').

E ao final: *Quantidade de maneiras de colorir a bandeira: 24*

Resumidamente, visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

1) $Cor1 = Amarelo$

2) $Cor2 = Amarelo \rightarrow if$ **bloqueia (iguais). Descarte.**

2) $Cor2 = Rosa \rightarrow if$ **permite**.

3) $Cor3 = Rosa \rightarrow if$ **bloqueia**.

3) $Cor3 = Verde \rightarrow if$ **permite**.

4) $Cor4 = Verde \rightarrow if$ **bloqueia**.

4) $Cor4 = Amarelo \rightarrow if$ **permite** $\rightarrow (amarelo, rosa, verde, amarelo)$.

4) $Cor4 = Rosa \rightarrow if$ **permite** $\rightarrow (amarelo, rosa, verde, rosa) \ldots$ e assim por diante até testar tudo.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.4.

A Árvore mostra visualmente todos os resultados possíveis. O Python "prova" a resposta gerando concretamente todos os 24 resultados.

g) Observe o Código 2.4, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo das linhas de código *if cor2 != cor1*, *if cor3 != cor2* e *if cor4 != cor3*?

Foram utilizados 4 laços for. E cada um desses laços representa uma decisão.
O laço externo *for cor1 in cores:* representa a 1ª Decisão.
O 1º laço interno *for cor2 in cores:* representa a 2ª Decisão.
O 2º laço interno *for cor3 in cores:* representa a 3ª Decisão.
O 3º laço interno *for cor4 in cores:* representa a 3ª Decisão.
Nesses quatro comandos for aninhados, o algoritmo se propõe a visitar todas as configurações, independentemente de serem válidas ou não, ou seja, o código está percorrendo um espaço de $3 \times 3 \times 3 \times 3 = 81$ possibilidades totais.

Mas, as linhas de comando *if* são a materialização da restrição de que a repetição de cores em listras adjacentes é proibida.
Por exemplo, enquanto na construção da árvore de possibilidades a restrição do problema está "escondida" na quantidade de maneiras de tomar a 2ª Decisão(2 maneiras), no código Python a restrição está escrita e visível na linha *if cor2 != cor1:*.

3.2 Segundo encontro

***Atividade* 5:** Considere o seguinte problema: "Quantos números de dois algarismos podem ser formados no sistema decimal?".

Esta atividade é particularmente interessante porque mostra que nem sempre a restrição do problema aparece de forma explícita.

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são os elementos disponíveis para a ação. Aqui, são os algarismos do sistema numérico decimal: $\{0, 1,2,3,4,5,6,7,8,9\}$.
- Configurações: A configuração é um número com dois algarismos. Matematicamente, é um par ordenado $(Dezena, Unidade)$ que forma um número válido com dois algarismos.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Como queremos formar números com dois algarismos, temos que tomar duas decisões para resolver esse problema. Como visto acima, precisamos formar o par ordenado $(Dezena, Unidade)$.

Temos uma restrição nesse problema. O par $(0, 7)$, por exemplo, representa o número 7, que é um número formado por apenas um algarismo. Portanto, o par ordenado $(0, a)$ não é uma configuração válida neste problema, isto é, o algarismo das dezenas não pode ser igual a zero.

Como no problema anterior, usaremos a estratégia de *Não adiar dificuldades*. Então, teremos:

1ª decisão: Escolher o algarismo das dezenas(como o algarismo das dezenas não pode ser igual a zero, essa restrição nos força a escolher o algarismo das dezenas primeiro);

2ª decisão: Escolher o algarismo das unidades.

Portanto,

1ª decisão(algarismo das dezenas): 9 maneiras $\rightarrow \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$;

2ª decisão(algarismo das unidades): 10 maneiras $\rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Quando o professor perguntar "Por que 9?", a Justificação do aluno será: "Porque se o algarismo das dezenas for igual a zero, o número só tem um algarismo.". Essa fala explicita a restrição do problema. E, se a pergunta for "Por que 10 na 2ª decisão?", a Justificação do aluno será: "Porque o problema não diz que os algarismos que formam o número devem ser distintos."

c) Agora, liste as possíveis configurações. Faça uma árvore de possibilidades.

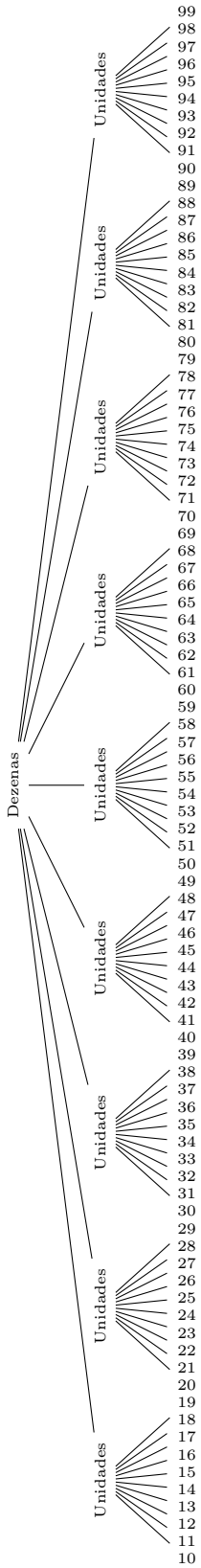


Figura 9: Árvore de Possibilidades da Atividade 5.
Fonte: Autoria própria(2026).

d) Quantas configurações você obteve? Isto é, qual é a resposta para o problema?

> O conjunto de resultados desse problema já começa a dar trabalho para ser obtido manualmente. A maioria dos alunos, já percebendo uma certa regularidade ao resolver os problemas anteriores, não montam a árvore de possibilidades toda, simplesmente, chegam a conclusão que há $9+9+9+9+9+9+9+9+9+9 = 9 \times 10 = 90$ configurações, pois para cada algarismo utilizado nas dezenas há 10 possibilidades de escolhas para as unidades.

e) Agora, execute o código 2.5.

```
Números formados:
1 0
1 1
1 2
1 3
1 4
1 5
1 6
1 7
1 8
1 9
2 0
2 1
2 2
2 3
2 4
2 5
2 6
2 7
2 8
2 9
3 0
3 1
3 2
3 3
3 4
3 5
3 6
3 7
3 8
3 9
4 0
4 1
4 2
4 3
4 4
4 5
4 6
4 7
4 8
4 9
5 0
5 1
5 2
5 3
5 4
5 5
5 6
5 7
5 8
5 9
6 0
6 1
6 2
6 3
6 4
6 5
6 6
6 7
6 8
6 9
7 0
7 1
7 2
7 3
7 4
7 5
7 6
7 7
7 8
7 9
8 0
8 1
8 2
8 3
8 4
8 5
8 6
8 7
8 8
8 9
9 0
9 1
9 2
9 3
9 4
9 5
9 6
9 7
9 8
9 9
Quantidade de números com dois algarismos:  90
>>>
```

Figura 10: Saída do Código 2.5.
Fonte: Autoria própria(2026).

I. **Linha 1:** *algarismos = [0,1,2,3,4,5,6,7,8,9]* → Define os objetos básicos do problema.

II. **Linha 6:** *for dezenas in algarismos:* → Representa a 1ª Decisão. O computador "escolhe" um algarismo para as dezenas e "fixa" ele.

III. **Linha 7:** *if dezenas != 0* → Esta linha é a materialização da restrição de que o algarismo das dezenas não pode ser igual a zero. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o par $(0,)$, mas a restrição do problema diz que ele não serve. Descarte-o."

IV. **Linha 8:** *for unidades in algarismos:* → Representa a 2ª Decisão. Para cada algarismo que está "fixado" na 1ª decisão, o computador testa todos os outros algarismos possíveis para as unidades.

V. **Linha 9:** *print(...)* → Materializa a configuração (o par ordenado).

VI. **Linha 10:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

VII. **Resultado da execução do código:** O programa imprimirá linha por linha:10,11,12,...,98,99.

E ao final: *Quantidade de números com dois algarismos: 90*

Resumidamente, visualizemos o que acontece na "cabeça" do Python, comparando com o desenho da árvore:

1) $dezenas = 0 \rightarrow if$ **bloqueia. Descarte.**

1) $dezenas = 1 \rightarrow if$ **Permite.**

2) $unidades = 0 \rightarrow if$ **Permite** $\rightarrow 10$

2) $unidades = 1 \rightarrow if$ **Permite** $\rightarrow 11 \ldots$ e assim por diante até testar tudo.

Ou seja, se o algarismo das dezenas for igual a 1, o algarismo das unidades pode ser 0,1,2,3,4,5,6,7,8 ou 9, o que lhe fornece 10 números diferentes:

10,11,12,13,14,15,16,17,18,19

Se o algarismo das dezenas for 2, há novamente 10 maneiras diferentes de escolher o algarismo das unidades:

20,21,22,23,24,25,26,27,28,29

Analogamente, se o algarismo das dezenas for 3:

30,31,32,33,34,35,36,37,38,39

E, o mesmo ocorre se o algarismo das dezenas for 4, 5, 6, 7, 8, ou 9.

f) Compare os resultados obtidos nos itens c) e d) com os resultados fornecidos pelo Código 2.5.

A Árvore mostra visualmente todos os resultados possíveis. O Python "prova" a resposta gerando concretamente todos os 90 resultados.

g) Observe o Código 2.5, quantos laços *for* foram utilizados? Será que cada laço *for* tem alguma associação com cada decisão? Qual é o objetivo da linha de código *if dezenas!= 0*?

Foram utilizados 2 laços for. E cada um desses laços representa uma decisão. O laço externo *for dezenas in algarismos:* representa a 1ª Decisão. O laço interno *for unidades in algarismos:* representa a 2ª Decisão. Nesses dois comandos for aninhados, o algoritmo se propõe a visitar todas as configurações, independentemente de serem válidas ou não, ou seja, o código está percorrendo um espaço de $10 \times 10 = 100$ possibilidades totais. Mas, a linha de comando *if dezenas != 0* é a materialização da restrição de que o algarismos das dezenas não pode ser igual a zero. Por exemplo, enquanto na construção da árvore de possibilidades a restrição do problema está "escondida" na quantidade de maneiras de tomar a 1ª Decisão(9 maneiras), no código Python a restrição está escrita e visível na linha *if dezenas != 0:*.

h) O que é mais trabalhoso, responder ao item c) ou construir o Código 2.5?

> O aluno começa a notar, na resolução desse problema, que desenhar 90 ramos é uma tarefa cansativa e propensa ao erro humano.
> Com isso, mesmo exigindo um certo nível de abstração, a escrita do código é operacionalmente mais eficiente. Além disso, faz com o aluno construa a lógica (os laços for e o if) que gera a prova que são 90 números com dois algarismos.

***Atividade* 6:** Considere o seguinte problema: "Quantos números pares de dois algarismos distintos podem ser formados no sistema decimal?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

> - Objetos básicos: são os algarismos do sistema numérico decimal: $\{0, 1,2,3,4,5,6,7,8,9\}$.
> - Configurações: A configuração é um número par com dois algarismos distintos. Matematicamente, é um par ordenado $(Dezena, Unidade)$ de modo que a Unidade é igual a 0,2,4,6 ou 8 e $Dezena \neq Unidade$.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual?

> Temos três restrições nesse problema.
>
> - **Primeira restrição**: O par ordenado $(0, a)$ não é uma configuração válida neste problema, isto é, o algarismo das dezenas não pode ser igual a zero.
> - **Segunda restrição**: O algarismo das unidades deve ser um múltiplo de 2.
> - **Terceira restrição**: Os algarismos são distintos.
>
> Precisamos tomar 2 decisões:
> 1ª decisão: Escolher o algarismo das unidades;
> 2ª decisão: Escolher o algarismo das dezenas.

c) Execute o código 2.6

Analisando o código:

I. **Linha 1:** *algarismos = [0,1,2,3,4,5,6,7,8,9]* → Define os objetos básicos do problema.

II. **Linha 5:** *for unidades in algarismos:* → Representa a 1ª Decisão. O computador "escolhe" um algarismo para as unidades e "fixa" ele.

III. **Linha 6:** *if unidades % 2 == 0:* → Esta linha é a materialização da restrição de que o algarismo das unidades tem que ser um múltiplo de dois. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o par (?,1), mas a restrição do problema diz que ele não serve. Descarte-o."

IV. **Linha 7:** *for dezenas in algarismos:* → Representa a 2ª Decisão. Para cada algarismo que está "fixado" na 1ª decisão, o computador testa todos os outros algarismos possíveis para as dezenas.

V. **Linha 8:** *if dezenas != 0 and dezenas != unidades:* → Esta linha é a materialização de duas restrições, de que o algarismo das dezenas não pode ser igual a zero e de que o algarismo da dezenas não pode ser igual ao algarismo escolhido para as unidades. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o par (0,2), mas a restrição do problema diz que ele não serve. Descarte-o."E, "Eu estou prestes a gerar o par (4,4), mas a restrição do problema diz que ele não serve. Descarte-o."

VI. **Linha 9:** *print(...)* → Materializa a configuração (o par ordenado).

VII. **Linha 10:** *contagem = contagem + 1* → Realiza a contagem passo a passo.

VIII. **Resultado da execução do código:** O programa imprimirá linha por linha:10,20,30, . . . ,78,98.

E ao final: *Quantidade de números pares com dois algarismos: 41*

Resumidamente, visualizemos o que acontece na "cabeça" do Python:

1) $unidades = 0 \rightarrow if$ **Permite.**

2) $dezenas = 0 \rightarrow if$ **Bloqueia. Descarte.**

2) $dezenas = 1 \rightarrow if$ **Permite.** $\rightarrow 10$

2) $dezenas = 2 \rightarrow if$ **Permite.** $\rightarrow 20$

2) $dezenas = 3 \rightarrow if$ **Permite.** $\rightarrow 30 \ldots$ e assim por diante até testar tudo.

Ou seja, se o algarismo das unidades for igual a 0, o algarismo das dezenas pode ser 1,2,3,4,5,6,7,8 ou 9, o que lhe fornece 9 números diferentes:

10,20,30,40,50,60,70,80,90

Se o algarismo das unidades for 2, há 8 maneiras diferentes de escolher o algarismo das dezenas:

12,32,42,52,62,72,82,92

Analogamente, se o algarismo das unidades for 4:

14,24,34,54,64,74,84,94

E, o mesmo ocorre se o algarismo das unidades for 6 ou 8.
Com isso, temos o seguinte resultado ao executar o Código 2.6.

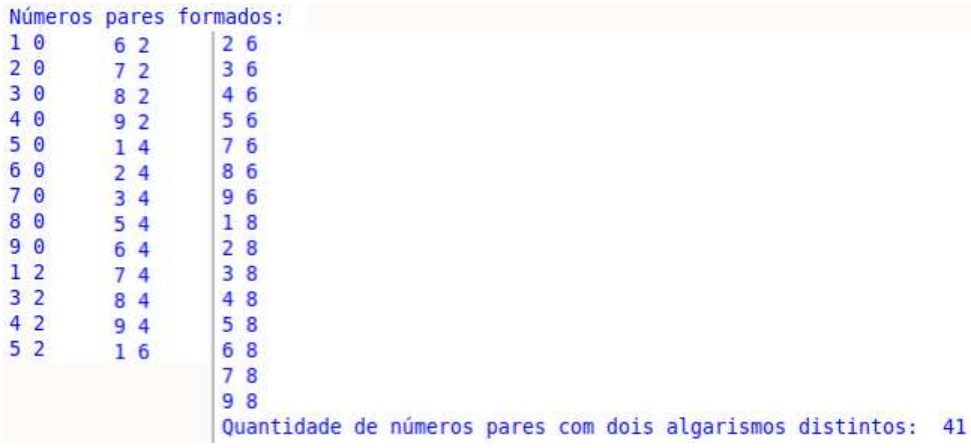


Figura 11: Saída do Código 2.6.
Fonte: Autoria própria(2026).

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

> As configurações satisfazem as condições do problema.
> Obtemos 41 configurações.

d) Observe o Código 2.6, qual é o objetivo das linhas de código *if unidades % 2 = 0* e *if dezenas!= 0 and dezenas != unidades*?

> A linha *if unidades % 2 == 0* é a materialização da restrição de que o algarismo das unidades tem que ser um múltiplo de dois.
> A linha *if dezenas != 0 and dezenas != unidades* é a materialização de duas restrições: o algarismo das dezenas não pode ser igual a zero e o algarismo da dezenas não pode ser igual ao algarismo escolhido para as unidades.

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema.

Vimos que, observando a execução do Código 2.6, temos que:

- quando o algarismo das unidades é igual a 0, o algarismo das dezenas pode ser 1,2,3,4,5,6,7,8 ou 9, o que fornece 9 números pares diferentes:

  10,20,30,40,50,60,70,80,90;

- quando o algarismo das unidades for 2, o algarismo das dezenas 1,3,4,5,6,7,8 ou 9, o que fornece 8 números pares diferentes:

  12,32,42,52,62,72,82,92

  Analogamente, quando o algarismo das unidades for 4, teremos:

  14,24,34,54,64,74,84,94

  E, o mesmo ocorre se o algarismo das unidades for 6 ou 8.

Logo, não é possível obter a resposta para esse problema usando, apenas, o Princípio Fundamental da Contagem.
A execução do Código 2.6 mostra-nos que existem $9 + 4 \cdot 8 = 9 + 32 = 41$ números pares com 2 algarismos distintos. Tal resultado nos mostra que há "dois tipos" de números pares formados pelo código, aqueles em que o zero ocupa as unidades e aqueles que o zero não ocupa o algarismo das unidades.
Então, para aplicar o Princípio Fundamental da Contagem na resolução do problema, precisamos resolver dois problemas, que são:

- Quantos números pares com dois algarismos distintos terminam em zero?

  Para esse problema, temos 1 única maneira de tomar a decisão d_1, que é escolher o zero, uma vez tomada a decisão d_1, temos 9 maneiras de tomar a decisão d_2. Logo, pelo Princípio Fundamental da Contagem, há $1 \cdot 9 = 9$ números desse tipo.

- Quantos números pares com dois algarismos distintos não terminam em zero?

  Para esse problema, temos 4 maneiras de tomar a decisão d_1, que é escolher um dos algarismos 2, 4, 6 ou 8, uma vez tomada a decisão d_1, temos 8 maneiras de tomar a decisão d_2, pois o zero não pode ocorrer nas dezenas e não podemos usar o algarismo usado nas unidades. Logo, pelo Princípio Fundamental da Contagem, há $4 \cdot 8 = 32$ desse tipo.

Portanto, há $9 + 32 = 41$ números pares com dois algarismos distintos.

Você deve ter notado, observando as configurações obtidas no item c) do problema da Atividade 6, que há a necessidade da utilização de um outro Princípio de Contagem, junto com o Princípio Fundamental da Contagem, para encontrar a quantidade de configurações. Tal Princípio é conhecido como Princípio Aditivo e pode ser enunciado da seguinte forma, conforme podemos encontrar em (CERIOLI; VIANA, 2012):

Sejam A um conjunto finito e $A_1, A_2, \cdots, A_n$ subconjuntos não vazios de A.

1. Dizemos que $A_1, A_2, \cdots, A_n$ *exaurem* A se, para cada elemento a de A, existe i, com $1 \leq i \leq n$, tal que $a \in A_i$; ou seja, se $A_1 \cup A_2 \cup \cdots \cup A_n = A$.

2. Dizemos que $A_1, A_2, \cdots, A_n$ são *disjuntos dois a dois* se, quando examinados aos pares, eles não possuem elementos em comum; ou seja, $A_i \cap A_j = \emptyset$, para todos i, j com $1 \leq i \neq j \leq n$.

3. Dizemos que $A_1, A_2, \cdots, A_n$ são uma partição de A se $A_1, A_2, \cdots, A_n$ exaurem A e são disjuntos dois a dois.

Princípio Aditivo. *Seja A um conjunto finito.*

Se $A_1, A_2, \cdots, A_n$ são uma partição de A, então $n(A) = n(A_1) + n(A_2) + \cdots + n(A_n)$.

No problema da Atividade 6, o conjunto A é formado pelos números pares com algarismos distintos, o conjunto A_1 é formados pelos números pares com algarismos distintos com o algarismo das unidades igual a zero e o conjunto A_2 é formados pelos números pares com algarismos distintos com o algarismo das unidades diferente de zero. Então, $n(A) = n(A_1) + n(A_2) = 9 + 32 = 41$.

Agora, podemos voltar a Enunciação feita por um aluno ao resolver o problema da Atividade 4(colorir a bandeira).

O aluno enunciou que:

"1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

2ª decisão(cor da terceira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *amarelo*;

3ª decisão(cor da segunda listra): 1 maneira $\rightarrow \{verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;

4ª decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."

Mas, também, poderíamos ter a seguinte enunciação:

"1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

2ª decisão(cor da terceira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

3ª decisão(cor da segunda listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a segunda listra não pode ter a mesma cor utilizada nas 1ª e 3ª listras;

4ª decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa, verde\}$, a quarta listra não pode ter a mesma cor utilizada na 3ª listra."

A partir das duas enunciações, observemos que a resposta da 3ª decisão depende da cor escolhida para colorir a 3ª listra, a realização da 2ª decisão. Com isso, temos dois casos a resolver:

1º caso: as cores das 1ª e 3ª listras são iguais;

1ª decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;

$2^{\underline{a}}$ decisão(cor da terceira listra): 1 maneira $\rightarrow \{rosa\}$, vamos supor que o aluno escolheu *rosa*;
$3^{\underline{a}}$ decisão(cor da segunda listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a segunda listra não pode ter a mesma cor utilizada nas $1^{\underline{a}}$ e $3^{\underline{a}}$ listras, vamos supor que o aluno escolheu *amarelo*;
$4^{\underline{a}}$ decisão(cor da quarta listra): 2 maneiras $\rightarrow \{amarelo, verde\}$, a quarta listra não pode ter a mesma cor utilizada na $3^{\underline{a}}$ listra.
Pelo Princípio Fundamental da Contagem, conseguimos colorir de $3 \times 1 \times 2 \times 2 = 12$ maneiras diferentes de colorir a bandeira.
$2^{\underline{o}}$ caso: as cores das $1^{\underline{a}}$ e $3^{\underline{a}}$ listras são diferentes e as cores das $2^{\underline{a}}$ e $4^{\underline{a}}$ listras são iguais.
$1^{\underline{a}}$ decisão(cor da primeira listra): 3 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *rosa*;
$2^{\underline{a}}$ decisão(cor da terceira listra): 2 maneiras $\rightarrow \{amarelo, rosa, verde\}$, vamos supor que o aluno escolheu *amarelo*;
$3^{\underline{a}}$ decisão(cor da segunda listra): 1 maneira $\rightarrow \{verde\}$, a segunda listra não pode ter a mesma cor utilizada nas $1^{\underline{a}}$ e $3^{\underline{a}}$ listras;
$4^{\underline{a}}$ decisão(cor da quarta listra): 2 maneiras $\rightarrow \{rosa,verde\}$, a quarta listra não pode ter a mesma cor utilizada na $3^{\underline{a}}$ listra.
Pelo Princípio Fundamental da Contagem, conseguimos colorir de $3 \times 2 \times 1 \times 2 = 12$ maneiras diferentes de colorir a bandeira.
Portanto, pelo Princípio Aditivo, há $12 + 12 = 24$ maneiras de colorir a bandeira, conforme determinamos no problema da Atividade 4.

***Atividade* 7:** Considere o seguinte problema: "De quantos modos 3 pessoas podem sentar-se em 5 cadeiras em fila?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são as pessoas: $\{p1,p2,p3\}$ e as cadeiras: $\{c1,c2,c3,c4,c5\}$.
- Configurações: Cada configuração é formada por 3 pessoas sentadas em 3 cadeiras e 2 cadeiras vazias, ou seja, cada configuração é formada por uma tripla ordenada onde cada coordenada é uma cadeira ocupada pelas pessoas $p1$, $p2$ e $p3$, nessa ordem.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Temos uma restrição implícita no problema, uma cadeira para cada pessoa.
Precisamos tomar 3 decisões: 1ª) Escolher a cadeira que a pessoa $p1$ vai sentar; 2ª) Escolher a cadeira que a pessoa $p2$ vai sentar; 3ª) Escolher a cadeira que a pessoa $p3$ vai sentar.
Temos, então:
1ª decisão: 5 maneiras de escolher uma cadeira para a pessoa $p1$;
2ª decisão: 4 maneiras de escolher uma cadeira para a pessoa $p2$;
3ª decisão: 3 maneiras de escolher uma cadeira para a pessoa $p3$.

c) Execute o código 2.7

Analisando o código:

I. **Linha 1:** *cadeiras* = [$c1$,$c2$,$c3$,$c4$,$c5$] → Define o objeto básico cadeiras.

II. **Linha 2:** *cadeiras_ocupadas* = [] → Define uma lista onde cada elemento será as cadeiras ocupadas pelas pessoas $p1$, $p2$ e $p3$.

III. **Linha 4:** *for cadeira_p1 in cadeiras:* → Representa a 1ª Decisão. O computador "escolhe" uma cadeira para a pessoa $p1$ e "fixa" ela.

IV. **Linha 5:** *for cadeira_p2 in cadeiras:* → Representa a 2ª Decisão. O computador "escolhe" uma cadeira para a pessoa $p2$ e "fixa" ela.

V. **Linha 6:** *if cadeira_p2 != cadeira_p1 :* → Esta linha é a materialização da restrição de que cada pessoa senta em uma única cadeira. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar a tripla ($c1$,$c1$,?), mas a restrição do problema diz que ele não serve. Descarte-o."

VI. **Linha 7:** *for cadeira_p3 in cadeiras:* → Representa a 3ª Decisão. Para cada cadeira que está "fixada" na 1ª e 2ª decisões, o computador testa todas as outras cadeiras possíveis para a pessoa $p3$.

VII. **Linha 8:** *if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:* → Esta linha é a materialização da restrição de que que cada pessoa senta em uma única cadeira. Ela diz: "Eu estou prestes a gerar a tripla ($c1$,$c2$,$c1$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."E, "Eu estou prestes a gerar a tripla ($c1$,$c2$,$c2$), mas a restrição do problema diz que ele não serve. Descarte-o."

VIII. **Linha 9:** $cadeira\_ocupada = cadeira\_p1$,$cadeira\_p2$,$cadeira\_p3$ → Materializa a configuração (a tripla ordenada).

IX. **Linha 10:** *cadeiras_ocupadas.append(cadeira_ocupada)* → Realiza a inserção da tripla formada na lista *cadeiras_ocupadas*.

X. **Resultado da execução do código:** O programa imprimirá as triplas formadas.

E ao final: *Quantidade de maneiras de organizar as 3 pessoas: 60*

Resumidamente, visualizemos o que acontece na "cabeça" do Python:

1) $cadeira\_p1 = c1$.

2) $cadeira\_p2 = c1 \rightarrow if$ **Bloqueia. Descarte.**

2) $cadeira\_p2 = c2 \rightarrow if$ **Permite.**

3) $cadeira\_p3 = c1 \rightarrow if$ **Bloqueia. Descarte.**

3) $cadeira\_p3 = c2 \rightarrow if$ **Bloqueia. Descarte.**

3) $cadeira\_p3 = c3 \rightarrow if$ **Permite.** → ($c1$,$c2$,$c3$) . . . e assim por diante até testar tudo.

Obtemos, então, a seguinte saída:

```
Cadeiras ocupadas pelas 3 pessoas:
('c1', 'c2', 'c3'),('c1', 'c2', 'c4'),('c1', 'c2', 'c5'),('c1', 'c3', 'c2'),('c1', 'c3', 'c4'),
('c1', 'c3', 'c5'),('c1', 'c4', 'c2'),('c1', 'c4', 'c3'),('c1', 'c4', 'c5'),('c1', 'c5', 'c2'),
('c1', 'c5', 'c3'),('c1', 'c5', 'c4'),('c2', 'c1', 'c3'),('c2', 'c1', 'c4'),('c2', 'c1', 'c5'),
('c2', 'c3', 'c1'),('c2', 'c3', 'c4'),('c2', 'c3', 'c5'),('c2', 'c4', 'c1'),('c2', 'c4', 'c3'),
('c2', 'c4', 'c5'),('c2', 'c5', 'c1'),('c2', 'c5', 'c3'),('c2', 'c5', 'c4'),('c3', 'c1', 'c2'),
('c3', 'c1', 'c4'),('c3', 'c1', 'c5'),('c3', 'c2', 'c1'),('c3', 'c2', 'c4'),('c3', 'c2', 'c5'),
('c3', 'c4', 'c1'),('c3', 'c4', 'c2'),('c3', 'c4', 'c5'),('c3', 'c5', 'c1'),('c3', 'c5', 'c2'),
('c3', 'c5', 'c4'),('c4', 'c1', 'c2'),('c4', 'c1', 'c3'),('c4', 'c1', 'c5'),('c4', 'c2', 'c1'),
('c4', 'c2', 'c3'),('c4', 'c2', 'c5'),('c4', 'c3', 'c1'),('c4', 'c3', 'c2'),('c4', 'c3', 'c5'),
('c4', 'c5', 'c1'),('c4', 'c5', 'c2'),('c4', 'c5', 'c3'),('c5', 'c1', 'c2'),('c5', 'c1', 'c3'),
('c5', 'c1', 'c4'),('c5', 'c2', 'c1'),('c5', 'c2', 'c3'),('c5', 'c2', 'c4'),('c5', 'c3', 'c1'),
('c5', 'c3', 'c2'),('c5', 'c3', 'c4'),('c5', 'c4', 'c1'),('c5', 'c4', 'c2'),('c5', 'c4', 'c3'),
Quantidade de maneiras de organizar as 3 pessoas:  60
>>>
```

Figura 12: Saída do Código 2.7.
Fonte: Autoria própria(2026).

d) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

> As configurações satisfazem as condições do problema.
> Obtemos 60 configurações.

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

> Pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher uma cadeira para a pessoa $p1$ e, feito isso, há quatro maneiras de escolher uma cadeira para a pessoa $p2$ e, feito isso, há três maneiras de escolher uma cadeira para a pessoa $p3$, temos que, podemos organizar as três pessoas de $5 \times 4 \times 3 = 60$ maneiras diferentes. O Código 2.7 é a materialização algorítmica do Princípio Fundamental da Contagem.

***Atividade* 8:** Considere o seguinte problema: "De quantos modos 6 pessoas podem sentar-se em 10 cadeiras em fila?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

> - Objetos básicos: são as cadeiras: $\{c1,c2,c3,c4,c5,c6,c7,c8,c9,c10\}$ e as pessoas: $\{p1,p2,p3,p4,p5,p6\}$.
> - Configurações: Cada configuração é formada por 6 pessoas sentadas em 6 cadeiras e 4 cadeiras vazias, ou seja, cada configuração é formada por uma 6-upla ordenada onde cada coordenada é uma cadeira ocupada pelas pessoas $p1$, $p2$, $p3$, $p4$, $p5$ e $p6$, nessa ordem.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Temos uma restrição implícita no problema, uma cadeira para cada pessoa. Precisamos tomar 6 decisões: 1ª) Escolher a cadeira que a pessoa $p1$ vai sentar; 2ª) Escolher a cadeira que a pessoa $p2$ vai sentar; 3ª) Escolher a cadeira que a pessoa $p3$ vai sentar; 4ª) Escolher a cadeira que a pessoa $p4$ vai sentar; 5ª) Escolher a cadeira que a pessoa $p5$ vai sentar e 6ª) Escolher a cadeira que a pessoa $p6$ vai sentar.

Temos, então:

1ª decisão: 10 maneiras de escolher uma cadeira para a pessoa $p1$;

2ª decisão: 9 maneiras de escolher uma cadeira para a pessoa $p2$;

3ª decisão: 8 maneiras de escolher uma cadeira para a pessoa $p3$;

4ª decisão: 7 maneiras de escolher uma cadeira para a pessoa $p4$;

5ª decisão: 6 maneiras de escolher uma cadeira para a pessoa $p5$;

6ª decisão: 5 maneiras de escolher uma cadeira para a pessoa $p6$.

c) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido. Se estiver e inseguro, em relação à sua solução, construa um código em Python para resolver o problema.

Pelo Princípio Fundamental da Contagem, como há 10 maneiras de escolher uma cadeira para a pessoa $p1$ e, feito isso, há 9 maneiras de escolher uma cadeira para a pessoa $p2$ e, feito isso, há 8 maneiras de escolher uma cadeira para a pessoa $p3$ e, feito isso, há 7 maneiras de escolher uma cadeira para a pessoa $p4$ e, feito isso, há 6 maneiras de escolher uma cadeira para a pessoa $p5$ e, feito isso, há 5 maneiras de escolher uma cadeira para a pessoa $p6$, temos que, podemos organizar as seis pessoas de $10 \times 9 \times 8 \times 7 \times 6 \times 5 = 151200$ maneiras diferentes.

O código abaixo fornece as configurações e a quantidade de configurações.

```
1 cadeiras = ['c1','c2','c3','c4','c5','c6','c7','c8','c9','c10']
2 cadeiras_ocupadas = []
3
4 for cadeira_p1 in cadeiras:
5     for cadeira_p2 in cadeiras:
6             if cadeira_p2 != cadeira_p1:
7                 for cadeira_p3 in cadeiras:
```

```
8                       if cadeira_p3 != cadeira_p2 and cadeira_p3 != cadeira_p1:
9                           for cadeira_p4 in cadeiras:
10                              if cadeira_p4 != cadeira_p3 and cadeira_p4 !=
    cadeira_p2 and cadeira_p4 != cadeira_p1:
11                                  for cadeira_p5 in cadeiras:
12                                      if cadeira_p5 != cadeira_p4 and cadeira_p5 !=
    cadeira_p3 and cadeira_p5 != cadeira_p2 and cadeira_p5 != cadeira_p1:
13                                          for cadeira_p6 in cadeiras:
14                                              if cadeira_p6 != cadeira_p5 and
    cadeira_p6 != cadeira_p4 and cadeira_p6 != cadeira_p3 and cadeira_p6 !=
    cadeira_p2 and cadeira_p6 != cadeira_p1:
15                                                  cadeira_ocupada = cadeira_p1,
    cadeira_p2,cadeira_p3,cadeira_p4,cadeira_p5,cadeira_p6
16                                                  cadeiras_ocupadas.append(
    cadeira_ocupada)
17
18 colunas = 4
19
20 print(f"\nCadeiras ocupadas pelas 3 pessoas:  ")
21 for i, cadeira_ocupada in enumerate(cadeiras_ocupadas):
22     print(f"{cadeira_ocupada}", end=",")
23     if (i + 1) % colunas == 0:
24         print()
25
26 if len(cadeiras_ocupadas) % colunas != 0:
27     print()
28
29 print('Quantidade de maneiras de organizar as 3 pessoas: ', len(cadeiras_ocupadas))
```

Código 3.1: Código da Atividade 8

3.3 Terceiro encontro

***Atividade* 9:** Considere o código 2.8 que resolve um determinado problema de contagem e execute-o:

a) Pesquise o que são anagramas.

> As várias ordenações de letras de uma determinada palavra são chamadas de **Anagramas**. Seriam como palavas, não necessariamente com sentido, que podem ser formadas a partir de um dado conjunto de letras.(SPREAFICO; SILVA, 2021)

b) Quais são os objetos básicos do problema?

> Observando a linha 1 do código, vemos que são as letras: $\{N,U,M,E,R,O\}$.

c) Quais são as configurações formadas?

Pelas linhas 2 e 22, as configurações são os anagramas da palavra NÚMERO.

d) Quantas decisões foram tomadas para resolver o problema? Quais foram essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão foi tomada?

Pelo o Código 2.8, observemos que há seis linhas de comando *for*. Logo, seis decisões foram tomadas.
Então, dadas as letras N,U,M,E,R,O, temos as seguintes decisões: 1ª) Escolher a 1ª letra do anagrama(*for letra_1 in letras*); 2ª) Escolher a 2ª letra do anagrama(*for letra_2 in letras*); 3ª) Escolher a 3ª letra do anagrama(*for letra_3 in letras*); 4ª) Escolher a 4ª letra do anagrama(*for letra_4 in letras*); 5ª) Escolher a 5ª letra do anagrama(*for letra_5 in letras*) e 6ª) Escolher a 6ª letra do anagrama(*for letra_6 in letras*).
Temos uma restrição no problema, cada letra só pode ser usada uma única vez. Basta ver as linhas de código *if* presentes no Código 2.8.
Temos, então:
1ª decisão: 6 maneiras, pois antes de iniciarmos a formação do anagrama, temos 6 letras disponíveis em *letras*;
2ª decisão: 5 maneiras, pois após a 1ª decisão ter sido realizada, para qualquer letra escolhida na 1ª decisão , temos 5 letras disponíveis(*if letra_2 != letra_1:*);
3ª decisão: 4 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª e 2ª decisões, temos 4 letras disponíveis(*if letra_3 != letra_2 and letra_3 != letra_1:*);
4ª decisão: 3 maneiras, pois após a 1ª, 2ª e 3ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª e 3ª decisões, temos 3 letras disponíveis(*if letra_4 != letra_3 and letra_4 != letra_2 and letra_4 != letra_1:*);
5ª decisão: 2 maneiras, pois após a 1ª, 2ª, 3ª e 4ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª e 4ª decisões, temos 2 letras disponíveis(*if letra_5 != letra_4 and letra_5 != letra_3 and letra_5 != letra_2 and letra_5 != letra_1:*);

> 6ª decisão: 1 maneira, pois após a 1ª, 2ª, 3ª, 4ª e 5ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª, 4ª e 5ª decisões, temos 1 letra disponível(*if letra_6 != letra_5 and letra_6 != letra_4 and letra_6 != letra_3 and letra_6 != letra_2 and letra_6 != letra_1:*).

e) Quantas configurações foram obtidas? Com isso, em que consiste o problema resolvido pelo Código 2.8?

> Após a execução do Código 2.8, obtemos os anagramas da palavra NÚMERO e a quantidade dos anagramas.

```
Anagramas da palavra NÚMERO:
NUMERO,NUMEOR,NUMREO,NUMROE,NUMOER,NUMORE,NUEMRO,NUEMOR,NUERMO,NUEROM,NUEOMR,NUEORM,NURMEO,NURMOE,NUREMO,NUREOM,NUROME,NUROEM,NUOMER,NUOMRE,
NUOEMR,NUOERM,NUORME,NUOREM,NMUERO,NMUEOR,NMUREO,NMUROE,NMUOER,NMUORE,NMEURO,NMEUOR,NMERUO,NMEROU,NMEOUR,NMEORU,NMRUEO,NMRUOE,NMREUO,NMREOU,
NMROUE,NMROEU,NMOUER,NMOURE,NMOEUR,NMOERU,NMORUE,NMOREU,NEUMRO,NEUMOR,NEURMO,NEUROM,NEUOMR,NEUORM,NEMURO,NEMUOR,NEMRUO,NEMROU,NEMOUR,NEMORU,
NERUMO,NERUOM,NERMUO,NERMOU,NEROUM,NEROMU,NEOUMR,NEOURM,NEOMUR,NEOMRU,NEORUM,NEORMU,NRUMEO,NRUMOE,NRUEMO,NRUEOM,NRUOME,NRUOEM,NRMUEO,NRMUOE,
NRMEUO,NRMEOU,NRMOUE,NRMOEU,NREUMO,NREUOM,NREMUO,NREMOU,NREOUM,NREOMU,NROUME,NROUEM,NROMUE,NROMEU,NROEUM,NROEMU,NOUMER,NOUMRE,NOUEMR,NOUERM,
NOURME,NOUREM,NOMUER,NOMURE,NOMEUR,NOMERU,NOMRUE,NOMREU,NOEUMR,NOEURM,NOEMUR,NOEMRU,NOERUM,NOERMU,NORUME,NORUEM,NORMUE,NORMEU,NOREUM,NOREMU,
UNMERO,UNMEOR,UNMREO,UNMROE,UNMOER,UNMORE,UNEMRO,UNEMOR,UNERMO,UNEROM,UNEOMR,UNEORM,UNRMEO,UNRMOE,UNREMO,UNREOM,UNROME,UNROEM,UNOMER,UNOMRE,
UNOEMR,UNOERM,UNORME,UNOREM,UMNERO,UMNEOR,UMNREO,UMNROE,UMNOER,UMNORE,UMENRO,UMENOR,UMERNO,UMERON,UMEONR,UMEORN,UMRNEO,UMRNOE,UMRENO,UMREON,
UMRONE,UMROEN,UMONER,UMONRE,UMOENR,UMOERN,UMORNE,UMOREN,UENMRO,UENMOR,UENRMO,UENROM,UENOMR,UENORM,UEMNRO,UEMNOR,UEMRNO,UEMRON,UEMONR,UEMORN,
UERNMO,UERNOM,UERMNO,UERMON,UERONM,UEROMN,UEONMR,UEONRM,UEOMNR,UEOMRN,UEORNM,UEORMN,URNMEO,URNMOE,URNEMO,URNEOM,URNOME,URNOEM,URMNEO,URMNOE,
URMENO,URMEON,URMONE,URMOEN,URENMO,URENOM,UREMNO,UREMON,UREONM,UREOMN,URONME,URONEM,UROMEN,UROMNE,UROEMN,UROENM,UONMER,UONMRE,UONEMR,UONERM,
UONRME,UONREM,UOMNER,UOMNRE,UOMENR,UOMERN,UOMRNE,UOMREN,UOENMR,UOENRM,UOEMNR,UOEMRN,UOERNM,UOERMN,UORNME,UORNEM,UORMNE,UORMEN,UORENM,UOREMN,
MNUERO,MNUEOR,MNUREO,MNUROE,MNUOER,MNUORE,MNEURO,MNEUOR,MNERUO,MNEROU,MNEOUR,MNEORU,MNRUEO,MNRUOE,MNREUO,MNREOU,MNROUE,MNROEU,MNOUER,MNOURE,
MNOEUR,MNOERU,MNORUE,MNOREU,MUNERO,MUNEOR,MUNREO,MUNROE,MUNOER,MUNORE,MUENRO,MUENOR,MUERNO,MUERON,MUEONR,MUEORN,MURNEO,MURNOE,MURENO,MUREON,
MURONE,MUROEN,MUONER,MUONRE,MUOENR,MUOERN,MUORNE,MUOREN,MENURO,MENUOR,MENRUO,MENROU,MENOUR,MENORU,MEUNRO,MEUNOR,MEURNO,MEURON,MEUONR,MEUORN,
MERNUO,MERNOU,MERUNO,MERUON,MERONU,MEROUN,MEONUR,MEONRU,MEOUNR,MEOURN,MEORNU,MEORUN,MRNUEO,MRNUOE,MRNEUO,MRNEOU,MRNOUE,MRNOEU,MRUNEO,MRUNOE,
MRUENO,MRUEON,MRUONE,MRUOEN,MRENUO,MRENOU,MREUNO,MREUON,MREONU,MREOUN,MRONUE,MRONEU,MROUNE,MROUEN,MROENU,MROEUN,MONUER,MONURE,MONEUR,MONERU,
MONRUE,MONREU,MOUNER,MOUNRE,MOUENR,MOUERN,MOURNE,MOUREN,MOENUR,MOENRU,MOEUNR,MOEURN,MOERNU,MOERUN,MORNUE,MORNEU,MORUNE,MORUEN,MORENU,MOREUN,
ENUMRO,ENUMOR,ENURMO,ENUROM,ENUOMR,ENUORM,ENMURO,ENMUOR,ENMRUO,ENMROU,ENMOUR,ENMORU,ENRUMO,ENRUOM,ENRMUO,ENRMOU,ENROUM,ENROMU,ENOUMR,ENOURM,
ENOMUR,ENOMRU,ENORUM,ENORMU,EUNMRO,EUNMOR,EUNRMO,EUNROM,EUNOMR,EUNORM,EUMNRO,EUMNOR,EUMRNO,EUMRON,EUMONR,EUMORN,EURNMO,EURNOM,EURMNO,EURMON,
EURONM,EUROMN,EUONMR,EUONRM,EUOMNR,EUOMRN,EUORNM,EUORMN,EMNURO,EMNUOR,EMNRUO,EMNROU,EMNOUR,EMNORU,EMUNRO,EMUNOR,EMURNO,EMURON,EMUONR,EMUORN,
EMRNUO,EMRNOU,EMRUNO,EMRUON,EMRONU,EMROUN,EMONUR,EMONRU,EMOUNR,EMOURN,EMORNU,EMORUN,ERNUMO,ERNUOM,ERNMUO,ERNMOU,ERNOUM,ERNOMU,ERUNMO,ERUNOM,
ERUMNO,ERUMON,ERUONM,ERUOMN,ERMNUO,ERMNOU,ERMUNO,ERMUON,ERMONU,ERMOUN,ERONUM,ERONMU,EROUNM,EROUMN,EROMNU,EROMUN,EONURM,EONURM,EONMUR,EONMRU,
EONRUM,EONRMU,EOUNMR,EOUNRM,EOUMNR,EOUMRN,EOURNM,EOURMN,EOMNUR,EOMNRU,EOMUNR,EOMURN,EOMRNU,EOMRUN,EORNUM,EORNMU,EORUNM,EORUMN,EORMNU,EORMUN,
RNUMEO,RNUMOE,RNUEMO,RNUEOM,RNUOME,RNUOEM,RNMUEO,RNMUOE,RNMEUO,RNMEOU,RNMOUE,RNMOEU,RNEUMO,RNEUOM,RNEMUO,RNEMOU,RNEOUM,RNEOMU,RNOUME,RNOUEM,
RNOMUE,RNOMEU,RNOEUM,RNOEMU,RUNMEO,RUNMOE,RUNEMO,RUNEOM,RUNOME,RUNOEM,RUMNEO,RUMNOE,RUMENO,RUMEON,RUMONE,RUMOEN,RUENMO,RUENOM,RUEMNO,RUEMON,
RUEONM,RUEOMN,RUONME,RUONEM,RUOMNE,RUOMEN,RUOENM,RUOEMN,RMNUEO,RMNUOE,RMNEUO,RMNEOU,RMNOUE,RMNOEU,RMUNEO,RMUNOE,RMUENO,RMUEON,RMUONE,RMUOEN,
RMENUO,RMENOU,RMEUNO,RMEUON,RMEONU,RMEOUN,RMONUE,RMONEU,RMOUNE,RMOUEN,RMOENU,RMOEUN,RENUMO,RENUOM,RENMUO,RENMOU,RENOUM,RENOMU,REUNMO,REUNOM,
REUMNO,REUMON,REUONM,REUOMN,REMNUO,REMNOU,REMUNO,REMUON,REMONU,REMOUN,REONUM,REONMU,REOUNM,REOUMN,REOMNU,REOMUN,RONUME,RONUEM,RONMUE,RONMEU,
RONEUM,RONEMU,ROUNME,ROUNEM,ROUMNE,ROUMEN,ROUENM,ROUEMN,ROMNUE,ROMNEU,ROMUNE,ROMUEN,ROMENU,ROMEUN,ROENUM,ROENMU,ROEUNM,ROEUMN,ROEMNU,ROEMUN,
ONUMER,ONUMRE,ONUEMR,ONUERM,ONURME,ONUREM,ONMUER,ONMURE,ONMEUR,ONMERU,ONMRUE,ONMREU,ONEUMR,ONEURM,ONEMUR,ONEMRU,ONERUM,ONERMU,ONRUME,ONRUEM,
ONRMUE,ONRMEU,ONREUM,ONREMU,OUNMER,OUNMRE,OUNEMR,OUNERM,OUNRME,OUNREM,OUMNER,OUMNRE,OUMENR,OUMERN,OUMRNE,OUMREN,OUENMR,OUENRM,OUEMNR,OUEMRN,
OUERNM,OUERMN,OURNME,OURNEM,OURMNE,OURMEN,OURENM,OUREMN,OMNUER,OMNURE,OMNEUR,OMNERU,OMNRUE,OMNREU,OMUNER,OMUNRE,OMUENR,OMUERN,OMURNE,OMUREN,
OMENUR,OMENRU,OMEUNR,OMEURN,OMERNU,OMERUN,OMRNUE,OMRNEU,OMRUNE,OMRUEN,OMRENU,OMREUN,OENUMR,OENURM,OENMUR,OENMRU,OENRUM,OENRMU,OEUNMR,OEUNRM,
OEUMNR,OEUMRN,OEURNM,OEURMN,OEMNUR,OEMNRU,OEMUNR,OEMURN,OEMRNU,OEMRUN,OERNUM,OERNMU,OERUNM,OERUMN,OERMNU,OERMUN,ORNUME,ORNUEM,ORNMUE,ORNMEU,
ORNEUM,ORNEMU,ORUNME,ORUNEM,ORUMNE,ORUMEN,ORUENM,ORUEMN,ORMNUE,ORMNEU,ORMUNE,ORMUEN,ORMENU,ORMEUN,ORENUM,ORENMU,OREUNM,OREUMN,OREMNU,OREMUN,
Quantidade de anagramas:  720
```

Figura 13: Saída do Código 2.8.
Fonte: Autoria própria(2026).

> Portanto, o Código 2.8 resolve o seguinte problema: "Quantos são os anagramas da palavra NÚMERO?"

f) Utilizando o Princípio Fundamental da Contagem, obtenha a solução do problema resolvido pelo Código 2.8.

Pelo Princípio Fundamental da Contagem, como há 6 maneiras de escolher uma letra para ser 1ª letra do anagrama e, feito isso, há 5 maneiras de escolher uma letra para ser a 2ª letra do anagrama e, feito isso, há 4 maneiras de escolher uma letra para ser a 3ª letra do anagrama e, feito isso, há 3 maneiras de escolher uma letra para ser a 4ª letra do anagrama e, feito isso, há 2 maneiras de escolher uma letra para ser a 5ª letra do anagrama e, feito isso, há 1 maneira de escolher uma letra para ser a 6ª letra do anagrama, temos que, podemos formar $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$ anagramas.

***Atividade* 10:** Considere o seguinte problema: "Quantos são os anagramas da palavra MOSTRA que começam por vogal e terminam por consoante?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: são as letras: $\{M,O,S,T,R,A\}$.
- Configurações: Cada configuração é um anagrama da palavra MOSTRA que começa por vogal e termina por consoante.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Como cada configuração é um anagrama da palavra MOSTRA, precisamos tomar 6 decisões.
Temos três restrições no problema: cada letra só pode ser usada uma única vez; a 1ª letra do anagrama deve ser uma vogal; a 6ª letra do anagrama deve ser uma consoante.
Levando em conta a estratégia, *"se alguma decisão é mais complicada que as demais, ela deve ser tomada em primeiro lugar"*, dadas as letras M,O,S,T,R,A, temos as seguintes decisões:

1ª) Escolher a 1ª letra do anagrama(**que deve ser vogal**);
2ª) Escolher a 6ª letra do anagrama(**que deve ser consoante**);
3ª) Escolher a 2ª letra do anagrama;
4ª) Escolher a 3ª letra do anagrama;
5ª) Escolher a 4ª letra do anagrama;
6ª) Escolher a 5ª letra do anagrama.
Temos, então:
1ª decisão: 2 maneiras, pois como essa letra deve ser vogal e nenhuma vogal foi selecionada, temos 2 letras disponíveis;
2ª decisão: 4 maneiras, pois como essa letra deve ser consoante e nenhuma consoante foi selecionada, temos 4 letras disponíveis;
3ª decisão: 4 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª e 2ª decisões, temos 4 letras disponíveis;
4ª decisão: 3 maneiras, pois após a 1ª, 2ª e 3ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª e 3ª decisões, temos 3 letras disponíveis;
5ª decisão: 2 maneiras, pois após a 1ª, 2ª, 3ª e 4ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª e 4ª decisões, temos 2 letras disponíveis;
6ª decisão: 1 maneira, pois após a 1ª, 2ª, 3ª, 4ª e 5ª decisões terem sido feitas, para quaisquer letras escolhidas nas 1ª, 2ª, 3ª, 4ª e 5ª decisões, temos 1 letra disponível.

c) Execute o código 2.9.

Analisando o código:

I. **Linha 1:** *letras* = [*M*,*O*,*S*,*T*,*R*,*A*] → Define o objeto básico.

II. **Linhas 2 e 3:** *vogais* = [*A*,*O*] e *consoantes* = [*M*,*R*,*S*,*T*] → Diferencia as letras do objeto básico em vogais e consoantes.

III. **Linha 4:** *anagramas* = [] → Define uma lista onde cada elemento será um anagrama.

IV. **Linha 6:** *for letra_1 in letras:* → Representa a 1ª Decisão. O computador "escolhe" uma letra para ser a 1ª do anagrama e "fixa" ela.

V. **Linha 7:** *if letra_1 in vogais* → Esta linha é a materialização da restrição de que a 1ª letra do anagrama deve ser uma vogal. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o anagrama, por exemplo, M?????, mas a restrição do problema diz que ele não serve. Descarte-o."

VI. **Linha 8:** *for letra_6 in letras:* → Representa a 2ª Decisão. O computador "escolhe" uma letra para ser a 6ª letra do anagrama e "fixa" ela.

VII. **Linha 9:** *if letra_6 in consoantes* → Esta linha é a materialização da restrição de que a 6ª letra do anagrama deve ser uma consoante. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o anagrama, por exemplo, $O????A$, mas a restrição do problema diz que ele não serve. Descarte-o."

VIII. **Linha 10:** *for letra_2 in letras:* → Representa a 3ª Decisão. O computador "escolhe" uma letra para ser a 2ª letra do anagrama e "fixa" ela.

IX. **Linha 11:** *if letra_2 != letra_1 and letra_2 != letra_6* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela atua como um filtro. Ela diz: "Eu estou prestes a gerar o anagrama $OM???M$, mas a restrição do problema diz que ele não serve. Descarte-o."

X. **Linha 12:** *for letra_3 in letras:* → Representa a 4ª Decisão. O computador "escolhe" uma letra para ser a 3ª letra do anagrama e "fixa" ela.

XI. **Linha 13:** *if letra_3 != letra_2 and letra_3 != letra_1 and letra_3 != letra_6:* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela diz: "Eu estou prestes a gerar o anagrama $OMO??S$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."

XII. **Linha 14:** *for letra_4 in letras:* → Representa a 5ª Decisão. O computador "escolhe" uma letra para ser a 4ª letra do anagrama e "fixa" ela.

XIII. **Linha 15:** *if letra_4 != letra_3 and letra_4 != letra_2 and letra_4 != letra_1 and letra_4 != letra_6:* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela diz: "Eu estou prestes a gerar o anagrama $OMRM?S$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."

XIV. **Linha 16:** *for letra_5 in letras:* → Representa a 6ª Decisão. O computador "escolhe" uma letra para ser a 5ª letra do anagrama e para cada letra que está "fixada" nas 1ª, 2ª, 3ª, 4ª e 5ª decisões, o computador testa todas as outras letras possíveis para ser a 5ª letra.

XV. **Linha 17:** *if letra_5 != letra_4 and letra_5 != letra_3 and letra_5 != letra_2 and letra_5 != letra_1 and letra_5 != letra_6:* → Esta linha é a materialização da restrição de que cada letra só pode ser usada uma única vez. Ela diz: "Eu estou prestes a gerar o anagrama $OMRTRS$), por exemplo, mas a restrição do problema diz que ele não serve. Descarte-o."

XVI. **Linha 18:** *anagrama = letra_1 + letra_2 + letra_3 + letra_4 + letra_5 + letra_6* → Materializa a configuração (o anagrama).

XVII. **Linha 19:** *anagramas.append(anagrama)* → Realiza a inserção do anagrama formado na lista *anagramas*.

XVIII. **Resultado da execução do código:** O programa imprimirá os anagramas.

E ao final: *Quantidade de anagramas: 192*

Resumidamente, visualizemos o que acontece na "cabeça" do Python:

1) $letra\_1 = M \rightarrow if$ **Bloqueia. Descarte.**

1) $letra\_1 = O \rightarrow if$ **Permite.**

2) $letra\_6 = M \rightarrow if$ **Permite.**

3) $letra\_2 = M \rightarrow if$ **Bloqueia. Descarte.**

3) $letra\_2 = O \rightarrow if$ **Bloqueia. Descarte.**

3) $letra\_2 = S \rightarrow if$ **Permite.**

4) $letra\_3 = M \rightarrow if$ **Bloqueia. Descarte.**

4) $letra\_3 = O \rightarrow if$ **Bloqueia. Descarte.**

4) $letra\_3 = S \rightarrow if$ **Bloqueia. Descarte.**

4) $letra\_3 = T \rightarrow if$ **Permite.**

5) $letra\_4 = M \rightarrow if$ **Bloqueia. Descarte.**

5) $letra\_4 = O \rightarrow if$ **Bloqueia. Descarte.**

5) $letra\_4 = S \rightarrow if$ **Bloqueia. Descarte.**

5) $letra\_4 = T \rightarrow if$ **Bloqueia. Descarte.**

5) $letra\_4 = R \rightarrow if$ **Permite.**

6) $letra\_5 = M \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = O \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = S \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = T \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = R \rightarrow if$ **Bloqueia. Descarte.**

6) $letra\_5 = A \rightarrow if$ **Permite.** $\rightarrow OSTRAM \ldots$ e assim por diante até testar tudo.

Obtendo, a saída abaixo:

```
Anagramas da palavra MOSTRA que começam por vogal e teminam por consoante:
OSTRAM,OSTARM,OSRTAM,OSRATM,OSATRM,OSARTM,OTSRAM,OTSARM,OTRSAM,OTRASM,OTASRM,OTARSM,
ORSTAM,ORSATM,ORTSAM,ORTASM,ORASTM,ORATSM,OASTRM,OASRTM,OATSRM,OATRSM,OARSTM,OARTSM,
OMTRAS,OMTARS,OMRTAS,OMRATS,OMATRS,OMARTS,OTMRAS,OTMARS,OTRMAS,OTRAMS,OTAMRS,OTARMS,
ORMTAS,ORMATS,ORTMAS,ORTAMS,ORAMTS,ORATMS,OAMTRS,OAMRTS,OATMRS,OATRMS,OARMTS,OARTMS,
OMSRAT,OMSART,OMRSAT,OMRAST,OMASRT,OMARST,OSMRAT,OSMART,OSRMAT,OSRAMT,OSAMRT,OSARMT,
ORMSAT,ORMAST,ORSMAT,ORSAMT,ORAMST,ORASMT,OAMSRT,OAMRST,OASMRT,OASRMT,OARMST,OARSMT,
OMSTAR,OMSATR,OMTSAR,OMTASR,OMASTR,OMATSR,OSMTAR,OSMATR,OSTMAR,OSTAMR,OSAMTR,OSATMR,
OTMSAR,OTMASR,OTSMAR,OTSAMR,OTAMSR,OTASMR,OAMSTR,OAMTSR,OASMTR,OASTMR,OATMSR,OATSMR,
AOSTRM,AOSRTM,AOTSRM,AOTRSM,AORSTM,AORTSM,ASOTRM,ASORTM,ASTORM,ASTROM,ASROTM,ASRTOM,
ATOSRM,ATORSM,ATSORM,ATSROM,ATROSM,ATRSOM,AROSTM,AROTSM,ARSOTM,ARSTOM,ARTOSM,ARTSOM,
AMOTRS,AMORTS,AMTORS,AMTROS,AMROTS,AMRTOS,AOMTRS,AOMRTS,AOTMRS,AOTRMS,AORMTS,AORTMS,
ATMORS,ATMROS,ATOMRS,ATORMS,ATRMOS,ATROMS,ARMOTS,ARMTOS,AROMTS,AROTMS,ARTMOS,ARTOMS,
AMOSRT,AMORST,AMSORT,AMSROT,AMROST,AMRSOT,AOMSRT,AOMRST,AOSMRT,AOSRMT,AORMST,AORSMT,
ASMORT,ASMROT,ASOMRT,ASORMT,ASRMOT,ASROMT,ARMOST,ARMSOT,AROMST,AROSMT,ARSMOT,ARSOMT,
AMOSTR,AMOTSR,AMSOTR,AMSTOR,AMTOSR,AMTSOR,AOMSTR,AOMTSR,AOSMTR,AOSTMR,AOTMSR,AOTSMR,
ASMOTR,ASMTOR,ASOMTR,ASOTMR,ASTMOR,ASTOMR,ATMOSR,ATMSOR,ATOMSR,ATOSMR,ATSMOR,ATSOMR,
Quantidade de anagramas:  192
```

Figura 14: Saída do Código 2.9.
Fonte: Autoria própria(2026).

c) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

> As configurações obtidas satisfazem as condições do problema.
> Obtemos 192 configurações.

d) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

> Pelo Princípio Fundamental da Contagem, como há 2 maneiras de escolher uma letra para ser 1ª letra do anagrama(pois deve ser vogal) e, feito isso, há 4 maneiras de escolher uma letra para ser a 6ª letra do anagrama(pois deve ser consoante) e, feito isso, há 4 maneiras de escolher uma letra para ser a 2ª letra do anagrama e, feito isso, há 3 maneiras de escolher uma letra para ser a 3ª letra do anagrama e, feito isso, há 2 maneiras de escolher uma letra para ser a 4ª letra do anagrama e, feito isso, há 1 maneira de escolher uma letra para ser a 5ª letra do anagrama, temos que, podemos formar $2 \times 4 \times 4 \times 3 \times 2 \times 1 = 192$ anagramas da palavra MOSTRA que começam por vogal e terminam por consoante.

***Atividade* 11:** Considere o código recursivo 2.10 que resolve um determinado problema de contagem:

a) Execute o código acima. O que esse código fornece como resultado?

Ao, simplesmente, executar o Código 2.10, ele listará os 720 anagramas da palavra NÚMERO.

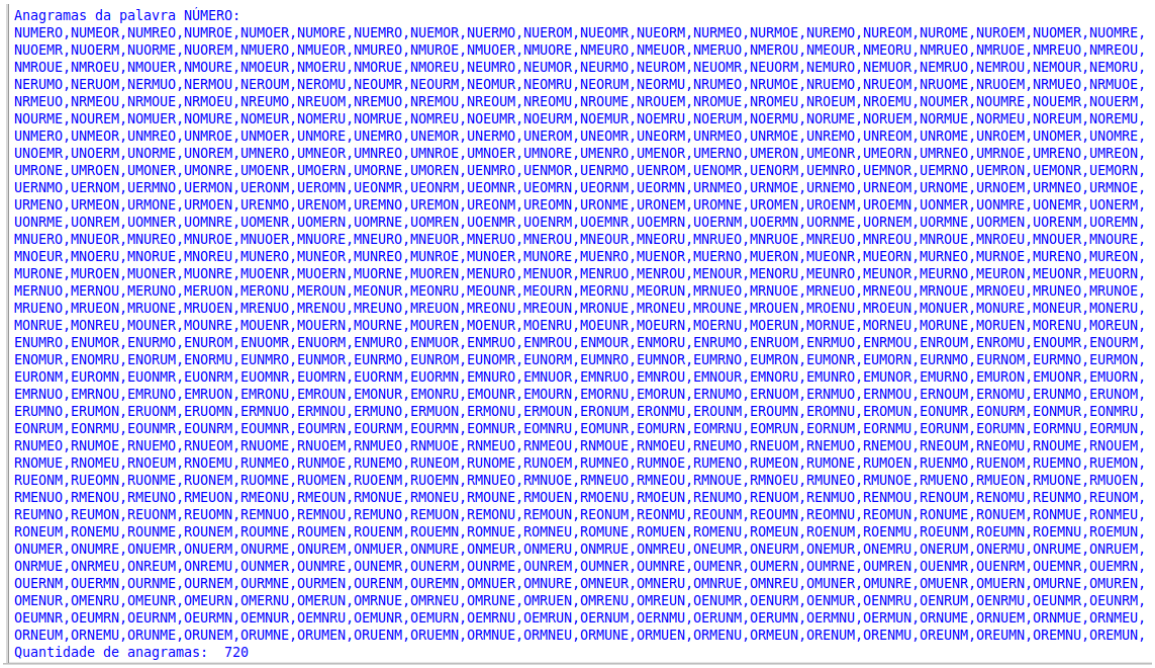


Figura 15: Saída do Código 2.10.
Fonte: Autoria própria(2026).

b) Compare-o com o Código 2.9. Qual é a diferença?

Diferente do Código 2.9 que usava vários *for* um dentro do outro, esse código introduz o conceito computacional conhecido como *Recursividade*.

c) Considere uma palavra com uma quantidade menor de letras, por exemplo ALO, e execute o Código 2.10 passo a passo manualmente, para entender o seu funcionamento.

Analisando o código:

I. **Linha 1:** *def permutacoes(palavra):* → Define uma função chamada *permutacoes* que aceita um texto (*palavra*) como entrada.

II. **Linhas 2 e 3:** $if len(palavra) == 1 :$ e $return[palavra] \rightarrow$ Se a palavra tiver apenas 1 letra, por exemplo *A*, qual é o anagrama dela? É ela mesma. Toda recursão precisa de um ponto de parada, o chamado *Caso Base*. Sem isso, o computador ficaria num *loop* infinito tentando dividir a palavra para sempre.

III. **Linha 5:** $resultado = [] \rightarrow$ Cria uma lista vazia para guardar os anagramas encontrados.

IV. **Linha 7:** *for i in range(len(palavra)):* $\rightarrow$ O laço *for* vai passar por cada letra da palavra original, uma por uma.

V. **Linha 8:** $char\_atual = palavra[i] \rightarrow$ "Fixa" a letra da vez. Por exemplo, se a palavra for "ALO" e $i = 0$, fixa o "A".

VI. **Linha 9:** $resto = palavra[: i] + palavra[i + 1 :] \rightarrow$ Cria uma nova palavra com tudo o que sobrou. Por exemplo, se tirou "A" de "ALO", o *resto* é "LO".

VII. **Linha 10:** *for p in permutacoes(resto):* $\rightarrow$ A função chama ela mesma, mas agora passando apenas o *resto*. É como se dissesse: "Eu não sei resolver "ALO" inteiro agora, mas se eu fixar o "A", você resolve "LO" para mim?"

VIII. **Linha 11:** $resultado.append(char\_atual + p) \rightarrow$ Concatena a letra que estava fixa ($char\_atual$) na frente de cada anagrama que voltou da recursão (p).

IX. **Linhas 15 e 17:** $palavra =$ *"ALO"* e $todas\_permutacoes = permutacoes(palavra) \rightarrow$ O usuário define a palavra "ALO" e dispara o processo.

X. **Resultado da execução do código:** O programa imprimirá os anagramas.

E ao final: *Quantidade de anagramas: 6*

Visualizemos o que acontece na "cabeça" do Python. Para o melhor entendimento da recursão, imagine que cada vez que a função *permutacoes* é chamada, abre-se um novo "nível" dentro da anterior. O computador pausa o "nível" de fora para resolver

o de dentro.

Início) **Nível 1**

- *permutacoes("ALO")*
- A função verifica: $len(\text{"ALO"}) == 1$? Não. $len(\text{"ALO"}) = 3$.
- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 1**→ (Fixa a letra "A")

- $i = 0$;
- $char\_atual = \text{"A"}$;
- $resto = \text{"LO"}$;
- Chamada para $permutacoes(\text{"LO"})$.

Início) **Nível 2**

- *permutacoes("LO")*
- A função verifica: $len(\text{"LO"}) == 1$? Não. $len(\text{"LO"}) = 2$.
- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 2** → (Fixa a letra "L")

- $i = 0$;
- $char\_atual = \text{"L"}$;
- $resto = \text{"O"}$;
- Chamada para $permutacoes(\text{"O"})$;

Início) **Nível 3**

* *permutacoes("O")*
* A função verifica: $len(\text{"O"}) == 1$? Sim.
* $resultado = [\text{"O"}]$;
* Fecha o Nível 3.

- Concatena $\text{"L"} + \text{"O"}$;
- $resultado = [\text{"LO"}]$.

2) **2ª iteração do Nível 2** → (Fixa a letra "O")

- $i = 1$;
- $char\_atual = \text{"O"}$;

- *resto* = *“L”*;
- Chamada para *permutacoes*(*“L”*);

Início) **Nível 3**

* *permutacoes(“L”)*
* A função verifica: *len*(*“L”*) == 1? Sim.
* *resultado* = [*“L”*];
* Fecha o Nível 3.

- Concatena *“O”* + *“L”*;
- *resultado* = [*“OL”*].

Fim) **Nível 2**

- *resultado* = [*“LO”*,*“OL”*].
- Fecha o Nível 2.

- Concatena *“A”* + *“LO”*;
- Concatena *“A”* + *“OL”*;
- *resultado* = [*“ALO”*,*“AOL”*].

2) **2ª iteração do Nível 1**→ (Fixa a letra “L”)

- $i = 1$;
- *char_atual* = *“L”*;
- *resto* = *“AO”*;
- Chamada para *permutacoes*(*“AO”*).

Início) **Nível 2**

- *permutacoes(“AO”)*
- A função verifica: *len*(*“AO”*) == 1? Não. *len*(*“AO”*) = 2.
- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 2** → (Fixa a letra “A”)

- $i = 0$;
- *char_atual* = *“A”*;
- *resto* = *“O”*;
- Chamada para *permutacoes*(*“O”*);

Início) **Nível 3**

* *permutacoes(“O”)*

        * A função verifica: $len(\text{“}O\text{”}) == 1$? Sim.
        * $resultado = [\text{“}O\text{”}]$;
        * Fecha o Nível 3.
    - Concatena *“A”* + *“O”*;
    - $resultado = [\text{“}AO\text{”}]$.

2) **2ª iteração do Nível 2** → (Fixa a letra “O”)

    - $i = 1$;
    - $char\_atual = \text{“}O\text{”}$;
    - $resto = \text{“}A\text{”}$;
    - Chamada para *permutacoes(“A”)*;

Início) **Nível 3**

    * *permutacoes(“A”)*
    * A função verifica: $len(\text{“}A\text{”}) == 1$? Sim.
    * $resultado = [\text{“}A\text{”}]$;
    * Fecha o Nível 3.
    - Concatena *“O”* + *“A”*;
    - $resultado = [\text{“}OA\text{”}]$.

Fim) **Nível 2**

    - $resultado = [\text{“}AO\text{”},\text{“}OA\text{”}]$.
    - Fecha o Nível 2.

- Concatena *“L”* + *“AO”*;
- Concatena *“L”* + *“OA”*;
- $resultado = [\text{“}ALO\text{”},\text{“}AOL\text{”},\text{“}LAO\text{”},\text{“}LOA\text{”}]$.

3) **3ª iteração do Nível 1**→ (Fixa a letra “O”)

- $i = 2$;
- $char\_atual = \text{“}O\text{”}$;
- $resto = \text{“}AL\text{”}$;
- Chamada para *permutacoes(“AL”)*.

Início) **Nível 2**

- *permutacoes(“AL”)*
- A função verifica: $len(\text{“}AL\text{”}) == 1$? Não. $len(\text{“}AL\text{”}) = 2$.

- Entra no *for i in range(len(palavra)):*→ Fixar cada letra como letra inicial em cada iteração.

1) **1ª iteração do Nível 2** → (Fixa a letra "A")
   - $i = 0$;
   - $char\_atual$ = *"A"*;
   - $resto$ = *"L"*;
   - Chamada para $permutacoes$(*"L"*);

Início) **Nível 3**
   - $permutacoes$(*"L"*)
   - A função verifica: len(*"L"*) == 1? Sim.
   - $resultado$ = [*"L"*];
   - Fecha o Nível 3.
   - Concatena *"A"* + *"L"*;
   - $resultado$ = [*"AL"*].

2) **2ª iteração do Nível 2** → (Fixa a letra "L")
   - $i = 1$;
   - $char\_atual$ = *"L"*;
   - $resto$ = *"A"*;
   - Chamada para $permutacoes$(*"A"*);

Início) **Nível 3**
   - $permutacoes$(*"A"*)
   - A função verifica: len(*"A"*) == 1? Sim.
   - $resultado$ = [*"A"*];
   - Fecha o Nível 3.
   - Concatena *"L"* + *"A"*;
   - $resultado$ = [*"LA"*].

Fim) **Nível 2**
   - $resultado$ = [*"AL"*,*"LA"*].
   - Fecha o Nível 2.

- Concatena *"O"* + *"AL"*;
- Concatena *"O"* + *"LA"*;
- $resultado$ = [*"ALO"*,*"AOL"*,*"LAO"*,*"LOA"*,*"OAL"*,*"OLA"*].

Fim) **Nível 1**

- Fecha o Nível 1.
- $resultado = [“ALO”,“AOL”,“LAO”,“LOA”,“OAL”,“OLA”]$.

Temos o seguinte resultado ao executar o Código 2.10, utilizando como entrada a palavra ALO:

```
Anagramas da palavra ALO:
ALO,AOL,LAO,LOA,OAL,OLA,
Quantidade de anagramas:  6
>>>
```

Figura 16: Saída do Código 2.10.
Fonte: Autoria própria(2026).

Chegamos, assim, ao fim da iteração do Código 2.10. Notemos que para resolver o problema grande $permutacoes(“ALO”)$, o computador “empilhou” problemas menores até chegar no caso trivial (uma letra só), e depois veio “desempilhando” e montando as respostas.
O código prova de forma mecânica a fórmula $n! = n \times (n-1)!$.

***Atividade* 12:** Considere o código 2.11, uma variação do Código 2.10, em Python que determina os anagramas de uma palavra:

a) Execute o código acima, usando como exemplo a palavra AMOR.

Ao executar o Código 2.11, que é idêntico ao Código 2.10, obtemos os anagramas da palavra AMOR e a quantidade dos mesmos.

```
Anagramas da palavra AMOR:
AMOR,AMRO,AOMR,AORM,ARMO,AROM,MAOR,MARO,
MOAR,MORA,MRAO,MROA,OAMR,OARM,OMAR,OMRA,
ORAM,ORMA,RAMO,RAOM,RMAO,RMOA,ROAM,ROMA,
Quantidade de anagramas:  24
>
```

Figura 17: Saída do Código 2.11 usando a palavra AMOR.
Fonte: Autoria própria(2026).

b) Usando o Princípio Fundamental da Contagem, determine o número de anagramas da palavra AMOR. Compare o seu resultado com o aquele fornecido pelo Código 2.11.

- **Objetos básicos.** letras: $\{A,M,O,R\}$.
- **Configurações.** Cada configuração é formada por um anagrama com as letras da palavra AMOR.
- **Restrições.** Temos uma única restrição, cada letra só pode ser usada uma única vez.
- **Decisões.** Há quatro decisões a serem tomadas: $1^{\underline{a}}$) Escolher a primeira letra do anagrama; $2^{\underline{a}}$) Escolher a segunda letra do anagrama; $3^{\underline{a}}$) Escolher a terceira letra do anagrama; $4^{\underline{a}}$) Escolher a quarta letra do anagrama.

  Temos, então:

  $1^{\underline{a}}$ decisão: 4 maneiras, pois antes de iniciarmos a formação do anagrama, temos 4 letras disponíveis;

  $2^{\underline{a}}$ decisão: 3 maneiras, pois após a $1^{\underline{a}}$ decisão ter sido realizada, para qualquer letra escolhida na $1^{\underline{a}}$ decisão , temos 3 letras disponíveis;

  $3^{\underline{a}}$ decisão: 2 maneiras, pois após a $1^{\underline{a}}$ e $2^{\underline{a}}$ decisões terem sido feitas, para quaisquer letras escolhidas nas $1^{\underline{a}}$ e $2^{\underline{a}}$ decisões, temos 2 letras disponíveis;

  $4^{\underline{a}}$ decisão: 1 maneira, pois após a $1^{\underline{a}}$, $2^{\underline{a}}$ e $3^{\underline{a}}$ decisões terem sido feitas, para quaisquer letras escolhidas nas $1^{\underline{a}}$, $2^{\underline{a}}$ e $3^{\underline{a}}$ decisões, temos 1 letra disponível.

- **Pelo Princípio Fundamental da Conagem**, a palavra AMOR possui $4 \times 3 \times 2 \times 1 = 4! = 24$ anagramas.

Obtemos, assim, o mesmo resultado encontrado na execução do Código 2.11 quando usamos a palavra AMOR como entrada.

c) Execute o código usando, agora, como exemplo a palavra AMAR.

Ao executar o Código 2.11, obtemos os anagramas da palavra AMAR e a quantidade dos mesmos.

```
Anagramas da palavra AMAR:
AMAR,AMRA,AAMR,AARM,ARMA,ARAM,MAAR,MARA,
MAAR,MARA,MRAA,MRAA,AAMR,AARM,AMAR,AMRA,
ARAM,ARMA,RAMA,RAAM,RMAA,RMAA,RAAM,RAMA,
Quantidade de anagramas:  24
```

Figura 18: Saída do Código 2.11 usando a palavra AMAR.
Fonte: Autoria própria(2026).

d) Analise, atentamente, a saída de execução do código. Você percebeu alguma coisa estranha? O quê?

Ao analisar a lista de anagramas, percebemos que há anagramas repetidos. Por exemplo, o primeiro anagrama, AMAR, também é o décimo quinto anagrama na nossa lista.
O computador ficou maluco? Por que ele escreveu a mesma palavra duas vezes?
Devemos notar que trocar o A do início com o A do meio da palavra AMAR não gera uma nova palavra.
Então, o Código gerou anagramas em excesso. Vamos corrigir?

e) Então, quantos são os anagramas da palavra AMAR?

Comecemos colocando os anagramas iguais em uma mesma "caixa":
[AMAR,AMAR]
[AMRA,AMRA]
[AAMR,AAMR]
[AARM,AARM]
[ARMA,ARMA]
[ARAM,ARAM]
[MAAR,MAAR]

[MARA,MARA]
[MRAA,MRAA]
[RAMA,RAMA]
[RAAM,RAAM]
[RMAA,RMAA]
Observemos que cada "caixa" possui dois anagramas iguais(como os "A" são iguais, contamos cada anagrama $2! = 2$ vezes), ou seja, para cada anagrama da palavra AMAR produzido, dois são repetidos. Com isso, podemos concluir que a palavra AMAR possui $\frac{24}{2} = 12$ anagramas.
Anagramas da palavra AMAR = [AMAR, AMRA, AAMR, AARM, ARMA, ARAM, MAAR, MARA, MRAA, RAMA, RAAM, RMAA].

f) Repita os itens c), d) e e) considerando a palavra BABA.

Ao executar o Código 2.11, obtemos os anagramas da palavra BABA e a quantidade dos mesmos.

```
Anagramas da palavra BABA:
BABA,BAAB,BBAA,BBAA,BAAB,BABA,ABBA,ABAB,
ABBA,ABAB,AABB,AABB,BBAA,BBAA,BABA,BAAB,
BABA,BAAB,ABAB,ABBA,AABB,AABB,ABBA,ABAB,
Quantidade de anagramas:  24
```

Figura 19: Saída do Código 2.11 usando a palavra BABA.
Fonte: Autoria própria(2026).

Ao analisar a lista de anagramas, percebamos que há anagramas repetidos. Por exemplo, o primeiro anagrama, BABA, também é o sexto, décimo quinto e décimo sétimo anagramas na nossa lista.
Devemos notar que trocar o B do início com o B do meio e o A do meio com o A do final da palavra BABA não gera uma nova palavra.
Dessa vez parece que o excesso foi maior. Vamos corrigir?
Comecemos colocando os anagramas iguais em uma mesma "caixa":
[BABA,BABA,BABA,BABA]
[BAAB,BAAB,BAAB,BAAB]
[BBAA,BBAA,BBAA,BBAA]

[ABBA,ABBA,ABBA,ABBA]
[ABAB,ABAB,ABAB,ABAB]
[AABB,AABB,AABB,AABB]
Observemos que cada "caixa" possui quatro anagramas iguais(como os "A" são iguais, contamos cada anagrama 2! vezes; analogamente, como os "B" são iguais, contamos cada anagrama 2! vezes), ou seja, para cada anagrama da palavra BABA produzido, quatro($2! \times 2!$) são repetidos. Com isso, podemos concluir que a palavra BABA possui $\frac{24}{4} = 6$ anagramas.
Anagramas da palavra BABA = [BABA, BAAB, BBAA, ABBA, ABAB, AABB].

g) Repita os itens c), d) e e) considerando a palavra AAZA(força em arábico).

Ao executar o Código 2.11, obtemos os anagramas da palavra AAZA e a quantidade dos mesmos.

```
Anagramas da palavra AAZA:
AAZA,AAAZ,AZAA,AZAA,AAAZ,AAZA,AAZA,AAAZ,
AZAA,AZAA,AAAZ,AAZA,ZAAA,ZAAA,ZAAA,ZAAA,
ZAAA,ZAAA,AAAZ,AAZA,AAAZ,AAZA,AZAA,AZAA,
Quantidade de anagramas:  24
```

Figura 20: Saída do Código 2.11 usando a palavra AAZA.
Fonte: Autoria própria(2026).

Ao analisar a lista de anagramas, percebamos que há anagramas repetidos. Por exemplo, o primeiro anagrama, AAZA, também é o sexto, sétimo, décimo segundo, vigésimo e vigésimo segundo anagramas na nossa lista.
Devemos notar que trocar o A do início com o A do meio ou com o A do final da palavra AAZA não gera uma nova palavra.
Vamos corrigir?
Comecemos colocando os anagramas iguais em uma mesma "caixa":
[AAZA,AAZA,AAZA,AAZA,AAZA,AAZA]
[AAAZ,AAAZ,AAAZ,AAAZ,AAAZ,AAAZ]
[AZAA,AZAA,AZAA,AZAA,AZAA,AZAA]
[ZAAA,ZAAA,ZAAA,ZAAA,ZAAA,ZAAA]

Observemos que cada "caixa" possui seis anagramas iguais(como os "A" são iguais, contamos cada anagrama $3! = 6$ vezes), ou seja, para cada anagrama da palavra AAZA produzido, seis são repetidos. Com isso, podemos concluir que a palavra AAZA possui $\frac{24}{6} = 4$ anagramas.
Anagramas da palavra AAZA = [AAZA, AAAZ, AZAA,ZAAA].

h) Por quê o Código 2.11 não funciona com as palavras AMAR, BABA e AAZA?

O computador não "lê" letras; ele "lê" índices de memória. Para o código recursivo, a *string* "AMAR", por exemplo, é tratada internamente como um vetor de posições: $[0, 1, 2, 3]$.
O algoritmo permuta os índices.

- **Permutação A:** Índices $[0, 1, 2, 3] \rightarrow$ Letras $A_1MA_2R \rightarrow$ "AMAR".
- **Permutação B:** Índices $[2, 1, 0, 3] \rightarrow$ Letras $A_2MA_1R \rightarrow$ "AMAR"

Para o computador, a **Permutação A** e a **Permutação B** são objetos diferentes porque vieram de índices diferentes. Ele não verifica o valor do conteúdo.

3.4 Quarto encontro

Na Atividade 12, deve-se ter notado que o código que calcula os anagramas de uma palavra só funciona corretamente quando a mesma possui letras distintas, ou seja, o código só calcula a quantidade de permutações simples.

O código da Atividade 13 tem como objetivo corrigir esse problema, isto é, calcular o número de anagramas de qualquer palavra.

***Atividade* 13:** Considere o código recursivo 2.12 em Python que determina os anagramas de uma palavra mas, agora, de uma forma mais organizada, resolvendo os problemas que tivemos na Atividade 12:

A ideia por trás desse Código é a mesma que utilizamos nos itens e), f) e g) da Atividade 12.

- **Linhas 1 a 15:** *def permutacoes(palavra):* → O código gera todos os "anagramas" da palavra dada;
- **Linhas 17 a 45:** *def remover_duplicatas_sort(lista):* → Agora, o código começará a remover os excessos.
  - **Linha 19:** *lista_ordenada = sorted(lista)* → Para encontrar os excessos mais rápido, o primeiro passo dado é ordenar a lista de anagramas;
  - **Linhas 24 a 29:** *for i in range(len(lista_ordenada)):* → Neste bloco, o código agrupa e mostra as duplicatas antes de removê-las;
  - **Linhas 31 a 37:** *for i, uplas in enumerate(lista_de_listas,1):* → Neste bloco, o código imprime os grupos formados, mostrando que o computador contou em excesso;
  - **Linhas 40 a 45:** *resultado = [lista_ordenda[0]]* → Neste bloco, o código percorre a lista ordenada e só aceita um item se ele for diferente do anterior, aqui ocorre a retirada dos excessos.

a) Execute o código acima, usando como exemplo a palavra AMAR.

- A função *permutacoes* gera 24 anagramas(com os repetidos), conforme vemos na Figura 18;
- A função *remover_duplicatas_sort* ordena os anagramas, agrupa os iguais em "caixas"(vemos que cada "caixa" possui dois anagramas iguais), filtra os anagramas(lista os $\frac{24}{2} = 12$ anagramas).

```
Digite uma palavra qualquer: AMAR
  1. ['AAMR', 'AAMR']
  2. ['AARM', 'AARM']
  3. ['AMAR', 'AMAR']
  4. ['AMRA', 'AMRA']
  5. ['ARAM', 'ARAM']
  6. ['ARMA', 'ARMA']
  7. ['MAAR', 'MAAR']
  8. ['MARA', 'MARA']
  9. ['MRAA', 'MRAA']
 10. ['RAAM', 'RAAM']
 11. ['RAMA', 'RAMA']
 12. ['RMAA', 'RMAA']

Número de anagramas:  24

Número de anagramas repetidos em cada caixa:  2


Anagramas da palavra AMAR:
  1. AAMR
  2. AARM
  3. AMAR
  4. AMRA
  5. ARAM
  6. ARMA
  7. MAAR
  8. MARA
  9. MRAA
 10. RAAM
 11. RAMA
 12. RMAA
Quantidade de anagramas da palavra AMAR: 12
```

Figura 21: Saída do Código 2.12 usando a palavra AMAR.
Fonte: Autoria própria(2026).

b) O resultado obtido é, de fato, a quantidade de anagramas da palavra AMAR?

O resultado obtido é a quantidade de anagramas da palavra AMAR $\rightarrow$ $\frac{4 \times 3 \times 2 \times 1}{2 \times 1} = \frac{4!}{2!} = \frac{24}{2} = 12$, conforme visto no item e) da Atividade 12.

c) Execute o código usando, agora, como exemplo a palavra AAZA.

- A função *permutacoes* gera 24 anagramas(com os repetidos), conforme vemos na Figura 20;
- A função *remover_duplicatas_sort* ordena os anagramas, agrupa os iguais em "caixas"(vemos que cada "caixa" possui seis anagramas iguais), filtra os anagramas(lista os $\frac{24}{6} = 4$ anagramas).

```
Digite uma palavra qualquer: AAZA
  1. ['AAAZ', 'AAAZ', 'AAAZ', 'AAAZ', 'AAAZ', 'AAAZ']
  2. ['AAZA', 'AAZA', 'AAZA', 'AAZA', 'AAZA', 'AAZA']
  3. ['AZAA', 'AZAA', 'AZAA', 'AZAA', 'AZAA', 'AZAA']
  4. ['ZAAA', 'ZAAA', 'ZAAA', 'ZAAA', 'ZAAA', 'ZAAA']

Número de anagramas:  24

Número de anagramas repetidos em cada caixa:  6


Anagramas da palavra AAZA:
  1. AAAZ
  2. AAZA
  3. AZAA
  4. ZAAA
Quantidade de anagramas da palavra AAZA: 4
```

Figura 22: Saída do Código 2.12 usando a palavra AAZA.
Fonte: Autoria própria(2026).

d) O resultado obtido é, de fato, a quantidade de anagramas da palavra AAZA?

O resultado obtido é a quantidade de anagramas da palavra AAZA $\rightarrow \frac{4 \times 3 \times 2 \times 1}{3 \times 2 \times 1} = \frac{4!}{3!} = \frac{24}{6} = 4$, conforme visto no item g) da Atividade 12.

e) Execute o Código 2.12, usando a palavra SOSSEGO. Quantos anagramas essa palavra possui?

- A função *permutacoes* gera 5040 anagramas(com os repetidos);
- A função *remover_duplicatas_sort* ordena os anagramas, agrupa os iguais em "caixas"(vemos que cada "caixa" possui seis anagramas iguais), filtra os anagramas(lista os $\frac{5040}{12} = 420$ anagramas).

Pelo que vimos, nos itens anteriores, o resultado obtido é a quantidade de anagramas da palavra SOSSEGO $\rightarrow \frac{7!}{3! \times 2!} = \frac{5040}{12} = 420$. Importante notar, que cada "caixa" possui doze anagramas iguais(como os "S" são iguais, contamos cada anagrama $3! = 6$ vezes e, como os "O" são iguais, contamos cada anagrama $2! = 2$ vezes), ou seja, para cada "anagrama" da palavra SOSSEGO produzido, doze são repetidos.

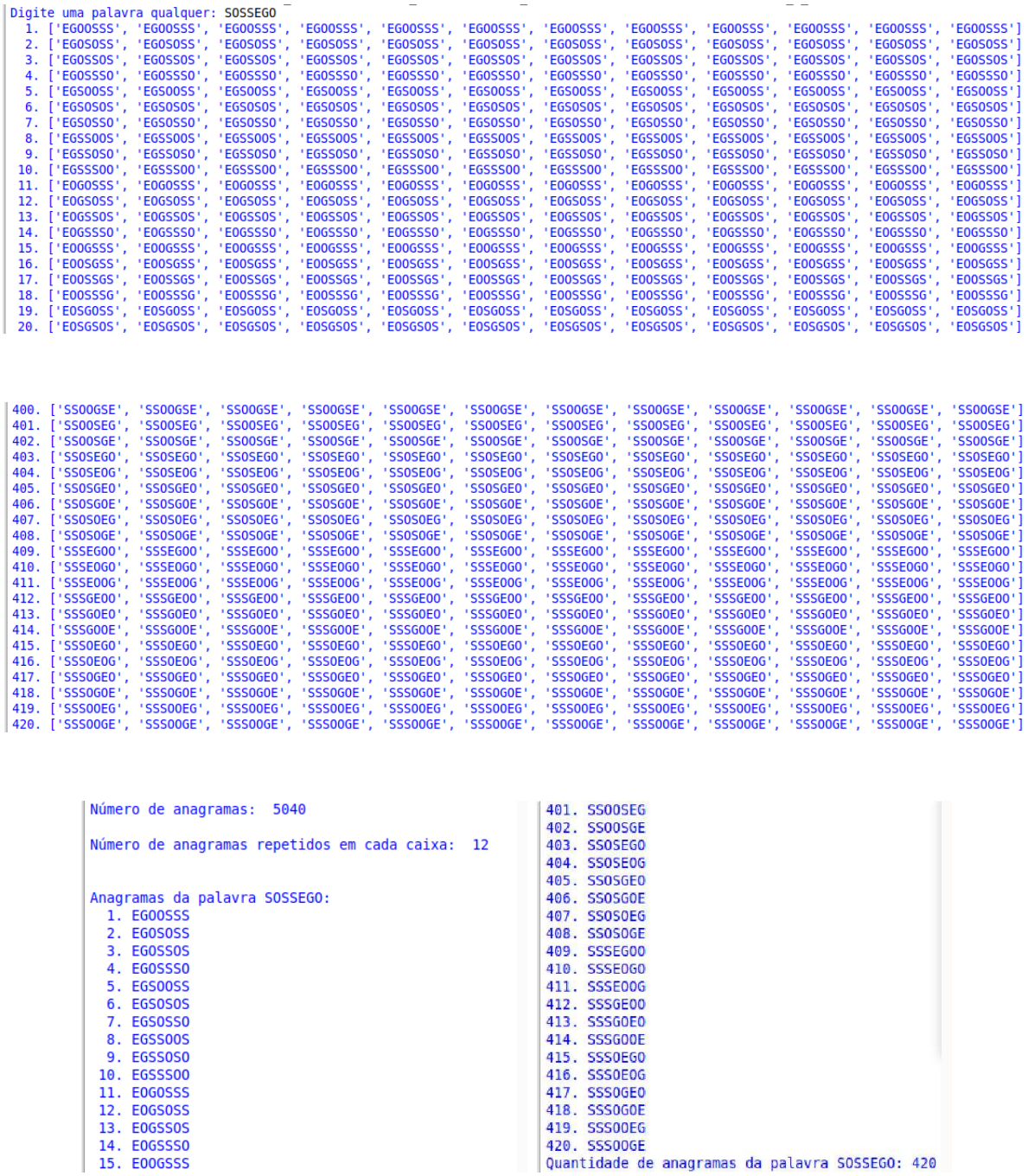


Figura 23: Resumo da Saída do Código 2.12 usando a palavra SOSSEGO.
Fonte: Autoria própria(2026).

f) Determine o número de anagramas das palavras MANIA, CORRER e PASSISTA, usando o Princípio Fundamental da Contagem e o Princípio k para 1.

Para cada palavra, temos:

- **Objetos básicos:** As letras que compõem cada palavra, considerando-as distintas, por exemplo: para a palavra MANIA, temos $letras = [M,A,N,I,A]$.
- **Configurações:** Cada configuração é um anagrama da palavra considerada.
- **Restrições:** Temos uma única restrição, cada letra só pode ser usada uma única vez.
- **Decisões:** A quantidade de decisões é igual à quantidade de elementos do conjunto formado pelos objetos básicos: $1^{\underline{a}}$) Escolher a primeira letra do anagrama; $2^{\underline{a}}$) Escolher a segunda letra do anagrama; $3^{\underline{a}}$) Escolher a terceira letra do anagrama; $4^{\underline{a}}$) Escolher a quarta letra do anagrama; e, assim por diante.
- **MANIA**
  - Supondo as letras distintas, pelo Princípio Fundamental da Contagem, há $5 \times 4 \times 3 \times 2 \times 1 = 5! = 120$ anagramas;
  - Como há duas letras "A", contamos cada anagrama $2! = 2$ vezes;
  - Logo, pelo Princípio k para 1 (nesse caso $k = 2!$), a palavra MANIA possui $\frac{120}{2!} = \frac{120}{2} = 60$ anagramas.
- **CORRER**
  - Supondo as letras distintas, pelo Princípio Fundamental da Contagem, há $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 6! = 720$ anagramas;
  - Como há três letras "R", contamos cada anagrama $3! = 6$ vezes;
  - Logo, pelo Princípio k para 1(nesse caso $k = 3!$), a palavra CORRER possui $\frac{720}{3!} = \frac{720}{6} = 120$ anagramas.

- **PASSISTA**
  - Supondo as letras distintas, pelo Princípio Fundamental da Contagem, há $8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 8! = 40320$ anagramas;
  - Como há três letras "S", contamos cada anagrama $3! = 6$ vezes e, há, também duas letras "A", contamos cada anagrama $2! = 2$ vezes, logo, estamos contando cada anagrama $2! \times 3! = 2 \times 6 = 12$ vezes;
  - Logo, pelo Princípio k para 1(nesse caso $k = 2! \times 3!$), a palavra CORRER possui $\frac{40320}{2! \times 3!} = \frac{40320}{2 \times 6} = \frac{40320}{12} = 3360$ anagramas.

***Atividade* 14:** Vamos ao seguinte problema: "Considere 5 pessoas A, B, C, D e E. Quantas filas com 3 dessas 5 pessoas podemos formar?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: São as 5 pessoas: $\{A, B, C, D, E\}$.
- Configurações: Cada configuração é uma fila com três dessas cinco pessoas. Matematicamente, é uma tripla ordenada $(pessoa\_1, pessoa\_2, pessoa\_3)$, onde $pessoa\_1$ é a pessoa no começo da fila, $pessoa\_2$ é a pessoa no meio da fila e $pessoa\_3$ é a última pessoa da fila.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Temos uma restrição implícita no problema, uma lugar na fila para cada pessoa.
Precisamos tomar 3 decisões: 1ª) Escolher a pessoa do início da fila($pessoa\_1$); 2ª) Escolher a pessoa do meio da fila($pessoa\_2$); 3ª) Escolher a pessoa do final da fila($pessoa\_3$).
Temos, então:
1ª decisão: 5 maneiras;
2ª decisão: 4 maneiras;
3ª decisão: 3 maneiras.

c) Execute o seguinte código 2.13.

I. **Linha 1 à Linha 21:** *def formar_filas(pessoas)*Define uma função que forma filas.

II. **Linha 10:** *for pessoa_1 in pessoas:* → Representa a 1ª Decisão. O computador "escolhe" uma pessoa para ser a primeira da fila e "fixa" ela.

IV. **Linha 11:** *for pessoa_2 in pessoas:* → Representa a 2ª Decisão. O computador "escolhe" uma pessoa para ser a segunda da fila e "fixa" ela.

V. **Linha 12:** *if pessoa_2 != pessoa_1 :* → Esta linha é a materialização da restrição de que cada pessoa ocupa um único lugar na fila. Ela diz: "Eu estou prestes a gerar a tripla $(A,A,?)$, mas a restrição do problema diz que ela não serve. Descarte-a."

VI. **Linha 13:** *for pessoa_3 in pessoas:* → Representa a 3ª Decisão. Para cada pessoa que está "fixada" na 1ª e 2ª decisões, o computador testa todas as outras pessoas possíveis para ser a última da fila.

VII. **Linha 14:** *if pessoa_3 != pessoa_2 and pessoa_3 != pessoa_1:* → Esta linha é a materialização da restrição de que que cada pessoa ocupa um único lugar na fila. Ela diz: "Eu estou prestes a gerar a tripla (A,B,A), por exemplo, mas a restrição do problema diz que ela não serve. Descarte-a."E, "Eu estou prestes a gerar a tripla (A,B,B), mas a restrição do problema diz que ela não serve. Descarte-a."

VIII. **Linhas 15 e 16:** *contador_filas += 1* → Conta a fila formada; $fila\_formada$ = *pessoa_1,pessoa_2,pessoa_3* → Materializa uma configuração (a tripla ordenada).

IX. **Linha 17:** *resultado.append(fila_formada)* → Realiza a inserção da tripla formada na lista *resultado*, que conterá todas as filas formadas.

X. **Resultado da execução do código:** O programa imprimirá as filas formadas.

E ao final: *Total de filas diferentes: 60.* Para a lista *pessoas* = [*A*,*B*,*C*,*D*,*E*].

Na Figura 24 temos a saída após a execução do código.

```
============================================================
PASSO 1: Forma as filas.
============================================================
Fila   1: ['A', 'B', 'C']
Fila   2: ['A', 'B', 'D']
Fila   3: ['A', 'B', 'E']
Fila   4: ['A', 'C', 'B']
Fila   5: ['A', 'C', 'D']
Fila   6: ['A', 'C', 'E']
Fila   7: ['A', 'D', 'B']
Fila   8: ['A', 'D', 'C']
Fila   9: ['A', 'D', 'E']
Fila  10: ['A', 'E', 'B']
Fila  11: ['A', 'E', 'C']
Fila  12: ['A', 'E', 'D']
Fila  13: ['B', 'A', 'C']
Fila  14: ['B', 'A', 'D']
Fila  15: ['B', 'A', 'E']
Fila  16: ['B', 'C', 'A']
Fila  17: ['B', 'C', 'D']
Fila  18: ['B', 'C', 'E']
Fila  19: ['B', 'D', 'A']
Fila  20: ['B', 'D', 'C']
Fila  21: ['B', 'D', 'E']
Fila  22: ['B', 'E', 'A']
Fila  23: ['B', 'E', 'C']
Fila  24: ['B', 'E', 'D']
Fila  25: ['C', 'A', 'B']
Fila  26: ['C', 'A', 'D']
Fila  27: ['C', 'A', 'E']
Fila  28: ['C', 'B', 'A']
Fila  29: ['C', 'B', 'D']
Fila  30: ['C', 'B', 'E']
Fila  31: ['C', 'D', 'A']
Fila  32: ['C', 'D', 'B']
Fila  33: ['C', 'D', 'E']
Fila  34: ['C', 'E', 'A']
Fila  35: ['C', 'E', 'B']
Fila  36: ['C', 'E', 'D']
Fila  37: ['D', 'A', 'B']
Fila  38: ['D', 'A', 'C']
Fila  39: ['D', 'A', 'E']
Fila  40: ['D', 'B', 'A']
Fila  41: ['D', 'B', 'C']
Fila  42: ['D', 'B', 'E']
Fila  43: ['D', 'C', 'A']
Fila  44: ['D', 'C', 'B']
Fila  45: ['D', 'C', 'E']
Fila  46: ['D', 'E', 'A']
Fila  47: ['D', 'E', 'B']
Fila  48: ['D', 'E', 'C']
Fila  49: ['E', 'A', 'B']
Fila  50: ['E', 'A', 'C']
Fila  51: ['E', 'A', 'D']
Fila  52: ['E', 'B', 'A']
Fila  53: ['E', 'B', 'C']
Fila  54: ['E', 'B', 'D']
Fila  55: ['E', 'C', 'A']
Fila  56: ['E', 'C', 'B']
Fila  57: ['E', 'C', 'D']
Fila  58: ['E', 'D', 'A']
Fila  59: ['E', 'D', 'B']
Fila  60: ['E', 'D', 'C']

Total de filas diferentes: 60
```

Figura 24: Saída do Código 2.13.
Fonte: Autoria própria(2026).

d) As configurações obtidas satisfazem as condições do problema? Quantas configurações o código forneceu?

As configurações satisfazem as condições do problema.
Obtemos 60 configurações.

e) Utilizando o Princípio Fundamental da Contagem, resolva o problema sugerido.

Pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher uma pessoa para ser a primeira da fila e, feito isso, há quatro maneiras de escolher uma pessoa para ser a segunda da fila e, feito isso, há três maneiras de escolher uma pessoa para ser a última da fila, temos que, podemos formar $5 \times 4 \times 3 = 60$ filas com três pessoas. Resultado também encontrado ao executar o Código 2.13.

f) Esse problema é parecido com algum problema que já foi resolvido? Qual?

Esse problema é idêntico ao problema da Atividade 7. É um problema onde as configurações são arranjos simples de cinco pessoas tomadas três a três.

g) E quantas filas com 4 pessoas podemos formar?

Agora, teremos quatro decisões a tomar.
Então, pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher uma pessoa para ser a primeira da fila e, feito isso, há quatro maneiras de escolher uma pessoa para ser a segunda da fila e, feito isso, há três maneiras de escolher uma pessoa para ser a terceira da fila e, feito isso, há duas maneiras de escolher uma pessoa para ser a última da fila, temos que, podemos formar $5 \times 4 \times 3 \times 2 = 120$ filas com quatro pessoas. Ou seja, temos 120 arranjos de cinco pessoas tomadas quatro a quatro.

***Atividade* 15:** Agora, vamos a esse problema: "Considere 5 pessoas A, B, C, D e E. Quantos grupos com 3 dessas 5 pessoas podemos formar?".

a) Quais são os objetos básicos do problema? E o que são as configurações desejadas?

- Objetos básicos: São as 5 pessoas: $\{A, B, C, D, E\}$.
- Configurações: Cada configuração é um grupo com três dessas cinco pessoas. Matematicamente, é um conjunto $\{pessoa\_1, pessoa\_2, pessoa\_3\}$, formado por três das cinco pessoas consideradas.

b) Quantas decisões precisamos tomar para resolver o problema? Quais são essas decisões? Há alguma restrição na tomada de decisões? Qual? De quantas maneiras cada decisão pode ser tomada?

Temos, então uma restrição implícita no problema, uma lugar no grupo para cada pessoa.
Precisamos tomar 3 decisões: 1ª) Escolher a primeira pessoa do grupo(*pessoa*_1); 2ª) Escolher a segunda pessoa do grupo(*pessoa*_2); 3ª) Escolher a terceira pessoa do grupo(*pessoa*_3).
Temos, então:
1ª decisão: 5 maneiras;
2ª decisão: 4 maneiras;
3ª decisão: 3 maneiras.

c) A solução desse problema é igual à solução do problema da Atividade 14? Ou seja, a quantidade de filas é igual à quantidade de grupos? Justifique.

Na maioria das vezes, a resposta dada pelos alunos a essa pergunta é SIM. Vejamos:
Pelo Princípio Fundamental da Contagem, como há cinco maneiras de escolher uma pessoa para ser o primeira do grupo e, feito isso, há quatro maneiras de escolher uma pessoa para ser a segunda do grupo e, feito isso, há três maneiras de escolher uma pessoa para ser a terceira do grupo, temos que, podemos formar $5 \times 4 \times 3 = 60$ grupos com três pessoas.
Mas, essa resposta está equivocada, a configuração que desejamos formar é um conjunto(grupo) com 3 pessoas, isto é, se os objetos de cada configuração devem ser escolhidos como um grupo, então a escolha de uma configuração deve ser feita de maneira *simultânea*. Então, não apenas um objeto, mas um grupo inteiro de objetos é escolhido ao mesmo tempo. Isso quer dizer que, por exemplo, ao montar o grupo $\{A,B,C\}$ tanto faz se, organizando cada uma pessoa sucessivamente, obtemos, por exemplo, A, e depois B, e depois C; ou se obtemos primeiro, por exemplo, C, e depois B, e depois A. Isto é, todas as filas: (A,B,C),(A,C,B),(B,A,C),(B,C,A),(C,A,B), (C,B,A) representam o mesmo grupo $\{A,B,C\}$.

Uma ação que funciona, para convencer os alunos, é o professor escolher três alunos da sala e dizer que aquele é o grupo escolhido. Depois disso, trocando os integrantes do grupo de lugar entre eles, perguntar se o grupo é o mesmo. Logo, a quantidade de filas com três das cinco pessoas NÃO é igual a quantidade de grupos com três das cinco pessoas, estamos contando grupos demais.

d) Se a resposta do item anterior for NÃO, corrija a resposta, obtendo, assim, a resposta correta.

Vamos usar a mesma estratégia que utilizamos no problema da Atividade 12. Comecemos colocando os grupos iguais em uma mesma "caixa":
[[A,B,C],[A,C,B],[B,A,C],[B,C,A],[C,A,B],[C,B,A]]
[[A,B,D],[A,D,B],[B,A,D],[B,D,A],[D,A,B],[D,B,A]]
[[A,B,E],[A,E,B],[B,A,E],[B,E,A],[E,A,B],[E,B,A]]
[[A,C,D],[A,D,C],[C,A,D],[C,D,A],[D,A,C],[D,C,A]]
[[A,C,E],[A,E,C],[C,A,E],[C,E,A],[E,A,C],[E,C,A]]
[[A,D,E],[A,E,D],[D,A,E],[D,E,A],[E,A,D],[E,D,A]]
[[B,C,D],[B,D,C],[C,B,D],[C,D,B],[D,B,C],[D,C,B]]
[[B,C,E],[B,E,C],[C,B,E],[C,E,B],[E,B,C],[E,C,B]]
[[B,D,E],[B,E,D],[D,B,E],[D,E,B],[E,B,D],[E,D,B]]
[[C,D,E],[C,E,D],[D,C,E],[D,E,C],[E,C,D],[E,D,C]]
Observemos que cada "caixa" possui seis arranjos(filas), ou seja, para cada grupo produzido com três pessoas dentre cinco pessoas consideradas, corresponde seis filas com três pessoas dentre cinco pessoas consideradas. Com isso, podemos concluir, usando o Princípio k para 1(onde $k = 6$, nesse caso), que a quantidade de grupos com três pessoas dentre as pessoas A,B,C,D,E é igual a $\frac{60}{6} = 10$.
Importante notar que as seis filas colocadas em cada "caixa" são as permutações simples das 3 pessoas que compõem um grupo.
Grupos com três pessoas dentre A,B,C,D,E:

$$[\{A,B,C\}, \{A,B,D\}, \{A,B,E\}, \{A,C,D\}, \{A,C,E\},$$

$$\{A,D,E\}, \{B,C,D\}, \{B,C,E\}, \{B,D,E\}, \{C,D,E\}]$$

.

O código da Atividade 16 tem como objetivo corrigir o excesso encontrado na Atividade 15, isto é, calcular a quantidade de grupos com três pessoas dentre as cinco pessoas A,B,C,D,E dadas.

***Atividade* 16:** Considere o código 2.14 em Python que resolve um determinado problema de contagem:

a) Execute o código acima.

Analisando o código:
A ideia por trás desse Código é a mesma que utilizamos no item d) da Atividade 15.

- **Linhas 1 a 21:** *def formar_filas(pessoas):* $\rightarrow$ O código gera todos as filas formadas por três pessoas dentre cinco pessoas dadas;
- **Linhas 23 a 40:** *def ordenar_e_imprimir_filas(filas):* $\rightarrow$ O código ordena cada uma das filas formadas por três pessoas dentre cinco pessoas dadas;
- **Linhas 42 a 74:** *def remover_duplicatas_sort(lista):* $\rightarrow$ Neste bloco, o código agrupa as filas ordenadas e iguais em uma mesma "caixa" e, depois, remove as duplicatas.

b) O que esse código fornece como resultado?

Ao executar o código, teremos:

- A função *formar_filas* gera as 60 filas formadas por três pessoas, dadas cinco pessoas;
- A função *ordenar_e_imprimir_filas* ordena as filas.
- A função *remover_duplicatas_sort* agrupa as filas iguais em "caixas"(vemos que cada "caixa" possui seis filas iguais), filtra as filas(lista os $\frac{60}{6} = 10$ grupos com três pessoas, dadas cinco pessoas).

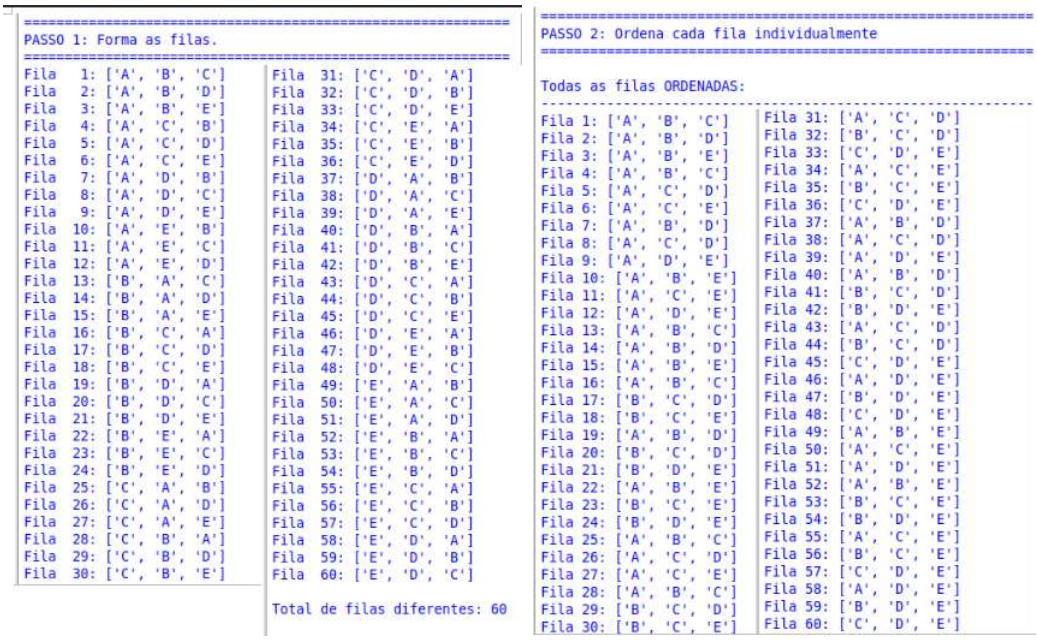


Figura 25: Saída do Código 2.14.
Fonte: Autoria própria(2026).

c) Compare-o com o Código 2.13. Qual é a diferença?

> A diferença está na presença das funções *ordenar_e_imprimir_filas* e *remover_duplicatas_sort* presentes no Código 2.14 que, a partir das filas formadas pelo Código 2.13, fornece os grupos com três pessoas escolhidas dentre cinco pessoas.

d) Esse código fornece a solução do problema da Atividade 15?

Sim. As configurações satisfazem as condições do problema.
Obtemos 10 configurações (10 grupos formados por três pessoas escolhidas dentre cinco pessoas).

e) Usando a ideia do código acima, dadas cinco pessoas quantos grupos com quatro pessoas podemos formar? E, dadas dez pessoas quantos grupos com quatro pessoas podemos formar?

- **Dadas cinco pessoas quantos grupos com quatro pessoas podemos formar?**
  - **Dadas cinco pessoas quantos filas com quatro pessoas podemos formar?**

    Os objetos básicos: São as 5 pessoas: $\{P1, P2, P3, P4, P5\}$.
    Cada configuração é uma fila com quatro dessas cinco pessoas.
    Temos uma restrição implícita no problema, uma lugar na fila para cada pessoa.
    Precisamos tomar 4 decisões: $1^{\underline{a}}$) Escolher a pessoa do início da fila; $2^{\underline{a}}$) Escolher a segunda pessoa da fila; $3^{\underline{a}}$) Escolher a terceira pessoa da fila; $4^{\underline{a}}$) Escolher a pessoa do final da fila.
    Temos, então:
    $1^{\underline{a}}$ decisão: 5 maneiras, pois antes de iniciarmos a formação da fila, temos 5 pessoas disponíveis;
    $2^{\underline{a}}$ decisão: 4 maneiras, pois após a $1^{\underline{a}}$ decisão ter sido realizada, para qualquer pessoa escolhida na $1^{\underline{a}}$ decisão , temos 4 pessoas disponíveis;
    $3^{\underline{a}}$ decisão: 3 maneiras, pois após a $1^{\underline{a}}$ e $2^{\underline{a}}$ decisões terem sido feitas, para quaisquer pessoas escolhidas nas $1^{\underline{a}}$ e $2^{\underline{a}}$ decisões, temos 3 pessoas disponíveis;
    $4^{\underline{a}}$ decisão: 2 maneiras, pois após a $1^{\underline{a}}$, $2^{\underline{a}}$ e $3^{\underline{a}}$ decisões terem sido feitas, para quaisquer pessoas escolhidas nas $1^{\underline{a}}$, $2^{\underline{a}}$ e $3^{\underline{a}}$ decisões, temos 2 pessoas disponíveis.
    Pelo Princípio Fundamental da Contagem, como há 5 maneiras de escolher uma pessoa para ser $1^{\underline{a}}$ pessoa da fila e, feito isso, há 4 maneiras de escolher uma pessoa para ser a $2^{\underline{a}}$ pessoa da fila e, feito isso, há 3 maneiras de escolher uma pessoa para ser a $3^{\underline{a}}$ pessoa da fila e, feito isso, há 2 maneiras de escolher uma pessoa para ser a $4^{\underline{a}}$ pessoa da fila, temos que, podemos

formar $5 \times 4 \times 3 \times 2 = 120$ filas.

– **Quantas filas "iguais" serão colocadas em cada caixa?**

Para resolver esse problema notemos, por exemplo, que 24 permutações simples das quatro pessoas $P1$,$P2$,$P3$,$P4$:

$$[P1, P2, P3, P4], [P1, P2, P4, P3], [P1, P3, P2, P4], [P1, P3, P4, P2],$$

$$[P1, P4, P2, P3], [P1, P4, P3, P2], [P2, P1, P3, P4], [P2, P1, P4, P3],$$

$$[P2, P3, P1, P4], [P2, P3, P4, P1], [P2, P4, P1, P3], [P2, P4, P3, P1],$$

$$[P3, P1, P2, P4], [P3, P1, P4, P2], [P3, P2, P1, P4], [P3, P2, P4, P1],$$

$$[P3, P4, P1, P2], [P3, P4, P2, P1], [P4, P1, P2, P3], [P4, P1, P3, P2],$$

$$[P4, P2, P1, P3], [P4, P2, P3, P1], [P4, P3, P1, P2], [P4, P3, P2, P1],$$

correspondem ao único grupo $\{P1$,$P2$,$P3$,$P4\}$.

Então, em cada caixa teremos 24 filas iguais, ou seja, a cada grupo formado por 4 pessoas dentre as 5 pessoas dadas, corresponde 24 permutações simples das mesmas 4 pessoas$(4! = 24)$.

Logo, pelo Princípio k para 1(nesse caso, $k = 4! = 24$), podemos formar $\frac{120}{24} = 5$ grupos com 4 pessoas escolhidas dentre 5 pessoas dadas.

- **Dadas dez pessoas quantos grupos com quatro pessoas podemos formar?**

  – **Dadas dez pessoas quantos filas com quatro pessoas podemos formar?**

  Os objetos básicos: São as 10 pessoas:

  $\{P1, P2, P3, P4, P5, P6, P7$,$P8, P9$,$P10\}$.

Cada configuração é uma fila com quatro dessas dez pessoas.

Temos uma restrição implícita no problema, uma lugar na fila para cada pessoa.

Precisamos tomar 4 decisões: 1ª) Escolher a pessoa do início da fila; 2ª) Escolher a segunda pessoa da fila; 3ª) Escolher a terceira pessoa da fila; 4ª) Escolher a pessoa do final da fila.

Temos, então:

1ª decisão: 10 maneiras, pois antes de iniciarmos a formação da fila, temos 5 pessoas disponíveis;

2ª decisão: 9 maneiras, pois após a 1ª decisão ter sido realizada, para qualquer pessoa escolhida na 1ª decisão , temos 9 pessoas disponíveis;

3ª decisão: 8 maneiras, pois após a 1ª e 2ª decisões terem sido feitas, para quaisquer pessoas escolhidas nas 1ª e 2ª decisões, temos 8 pessoas disponíveis;

4ª decisão: 7 maneiras, pois após a 1ª, 2ª e 3ª decisões terem sido feitas, para quaisquer pessoas escolhidas nas 1ª, 2ª e 3ª decisões, temos 7 pessoas disponíveis.

Pelo Princípio Fundamental da Contagem, como há 10 maneiras de escolher uma pessoa para ser 1ª pessoa da fila e, feito isso, há 9 maneiras de escolher uma pessoa para ser a 2ª pessoa da fila e, feito isso, há 8 maneiras de escolher uma pessoa para ser a 3ª pessoa da fila e, feito isso, há 7 maneiras de escolher uma pessoa para ser a 4ª pessoa da fila, temos que, podemos formar $10 \times 9 \times 8 \times 7 = 5040$ filas.

– **Quantas filas "iguais" serão colocadas em cada caixa?**

Para resolver esse problema notemos, por exemplo, que 24 permutações simples das quatro pessoas $P1$,$P2$,$P3$,$P4$:

$$[P1, P2, P3, P4], [P1, P2, P4, P3], [P1, P3, P2, P4], [P1, P3, P4, P2],$$

$$[P1, P4, P2, P3], [P1, P4, P3, P2], [P2, P1, P3, P4], [P2, P1, P4, P3],$$

$$[P2, P3, P1, P4], [P2, P3, P4, P1], [P2, P4, P1, P3], [P2, P4, P3, P1],$$

$$[P3, P1, P2, P4], [P3, P1, P4, P2], [P3, P2, P1, P4], [P3, P2, P4, P1],$$

$$[P3, P4, P1, P2], [P3, P4, P2, P1], [P4, P1, P2, P3], [P4, P1, P3, P2],$$

$$[P4, P2, P1, P3], [P4, P2, P3, P1], [P4, P3, P1, P2], [P4, P3, P2, P1],$$

correspondem ao único grupo $\{P1, P2, P3, P4\}$.

Então, em cada caixa teremos 24 filas iguais, ou seja, a cada grupo formado por 4 pessoas dentre as 5 pessoas dadas, corresponde 24 permutações simples das mesmas 4 pessoas($4! = 24$).

Logo, pelo Princípio k para 1(nesse caso, $k = 4! = 24$), podemos formar $\frac{5040}{24} = 210$ grupos com 4 pessoas escolhidas dentre 10 pessoas dadas.

Referências

BRASIL. **Base Nacional Comum Curricular (BNCC)**. Brasília, 2018. Disponível em: https://basenacionalcomum.mec.gov.br.Acesso em: 16 jul. 2025.

CERIOLI, Márcia R.; VIANA, Petrucio. **Minicurso de Combinatória de Contagem**. II Colóquio de Matemática da Região Sul. Universidade Estadual de Londrina, PR, Abril 2012.

FERREIRA, Sidney da Silva. **A integração da Análise Combinatória e do Pensamento Computacional no Ensino de Matemática por meio da Linguagem Python**. 2026. Mestrado Profissional em Matemática em Rede Nacional – Universidade Federal Fluminense, Niterói.

LINS, Romulo Campos. O Modelo dos Campos Semânticos: Estabelecimentos e Notas de Teorizações. *In:* ANGELO, Claudia Laus *et al.* (Ed.). **Modelo dos Campos Semânticos e Educação Matemática: 20 anos de História**. São Paulo: Midiograf, 2012.

LOCKWOOD, Elise; GIBSON, Bryan R. Combinatorial Tasks and Outcome Listing: Examining Productive Listing among Undergraduate Students. **Educational Studies in Mathematics**, v. 91, p. 247–270, Fevereiro 2016.

LOCKWOOD, Elise Nicole. **Student Approaches to Combinatorial Enumeration**: The Role of Set-Oriented Thinking. Jan. 2011. Tese de Doutorado – Portland State University, United States of America. Disponível em https://pdxscholar.library.pdx.edu/open_access_etds.

MORGADO, Augusto César *et al.* **Análise Combinatória e Probabilidade**. Rio de Janeiro: SBM, 2006.

SPREAFICO, Elen Viviani Pereira; SILVA, Kenia Cristina Pereira. **Livro Aberto de Matemática**: Análise Combinatória. Rio de Janeiro: IMPA-OS, 2021. Disponível em https://umlivroaberto.impa.br/producao/analise-combinatoria/. Acesso em: 16 maio 2026.