Universidade Federal de Mato Grosso
Instituto de Ciências Exatas e da Terra
Departamento de Matemática

Douglas Vinicius Antunes Rodrigues

Uma abordagem da criptografia com tecnologia para o aprimoramento da aprendizagem de funções polinomiais

Cuiabá - MT

2024

Douglas Vinicius Antunes Rodrigues

Uma abordagem da criptografia com tecnologia para o aprimoramento da aprendizagem de funções polinomiais

Dissertação apresentada ao curso de Mestrado Profissional em Matemática – Profmat, como requisito para obtenção do título de **Mestre em Matemática** pela Universidade Federal de Mato Grosso.

Área de Concentração: Educação Matemática.
Linha de Pesquisa: Tecnologia e Educação Matemática.

Prof.ª Dr.ª Anna Ligia Oenning Soares
Orientadora

Cuiabá - MT

2024

Dados Internacionais de Catalogação na Fonte.

R696a Rodrigues, Douglas Vinicius Antunes.
Uma abordagem da criptografia com tecnologia para o aprimoramento da aprendizagem de funções polinomiais [recurso eletrônico] / Douglas Vinicius Antunes Rodrigues. -- Dados eletrônicos (1 arquivo : 116 f., il. color., pdf). -- 2024.

Orientadora: Anna Ligia Oenning Soares.
Dissertação (mestrado profissional) – Universidade Federal de Mato Grosso, Instituto de Ciências Exatas e da Terra, Programa de Pós-Graduação Profissional em Matemática, Cuiabá, 2024.
Modo de acesso: World Wide Web: https://ri.ufmt.br.
Inclui bibliografia.

1. Ensino de matemática. 2. Pensamento computacional. 3. Função polinomial. 4. Automação. I. Soares, Anna Ligia Oenning, *orientador.* II. Título.

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE MATO GROSSO
PRÓ-REITORIA DE ENSINO DE PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM [NOME DO PPG]

FOLHA DE APROVAÇÃO

TÍTULO: **UMA ABORDAGEM DA CRIPTOGRAFIA COM TECNOLOGIA PARA O APRIMORAMENTO DA APRENDIZAGEM DE FUNÇÕES POLINOMIAIS**

AUTOR: **MESTRANDO DOUGLAS VINICIUS ANTUNES RODRIGUES**

Dissertação defendida e aprovada em 5 de setembro de 2024.

COMPOSIÇÃO DA BANCA EXAMINADORA

1. **Profª. Drª. Anna Lígia Oenning Soares** (Presidenta Banca/orientadora)

Instituição: Universidade Federal de Mato Grosso

2. **Prof. Dr. Reinaldo de Marchi** (Membro Interno)

Instituição: Universidade Federal de Mato Grosso

3. **Prof. Dr. Edgar Nascimento** (Membro externo)

Instituição: Instituto Federal de Mato Grosso

Cuiabá, 05/09/2024.

Documento assinado eletronicamente por **Edgar Nascimento**, **Usuário Externo**, em 06/09/2024, às 08:39, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **REINALDO DE MARCHI**, **Docente da Universidade Federal de Mato Grosso**, em 06/09/2024, às 09:06, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **ANNA LIGIA OENNING SOARES**, **Docente da Universidade Federal de Mato Grosso**, em 06/09/2024, às 09:53, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site http://sei.ufmt.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **7141726** e o código CRC **A37DFD81**.

Resumo

Esta dissertação propõe uma metodologia para o aprimoramento do conhecimento sobre funções matemáticas, utilizando a criptografia como ferramenta pedagógica e a linguagem de programação Python para automatizar o processo. A proposta, direcionada para estudantes do ensino médio, é baseada na BNCC e no Pensamento Computacional, explorando a codificação e decodificação de mensagens utilizando funções afim e quadrática, demonstrando a aplicação prática desses conceitos matemáticos e desenvolvendo habilidades de resolução de problemas. Teoricamente, o trabalho se apoia em WING (2006), BARICHELLO e MATHIAS (2021), LIMA et al. (2006) e LAMBERT (2019), entre outros. A dissertação apresenta uma sequência didática detalhada para o ensino fundamental e médio, discute as limitações do código e sugere soluções para aprimorá-lo. Adicionalmente, apresenta ideias para pesquisas futuras, como a busca por uma linguagem de programação em português e a exploração de outros tipos de funções.

Palavras chave: Ensino de matemática; Pensamento computacional; Função polinomial; Automação.

Abstract

This dissertation proposes a methodology for improving knowledge of mathematical functions, using cryptography as a pedagogical tool and the Python programming language to automate the process. The proposal, aimed at high school students, is based on the BNCC and Computational Thinking, exploring the encoding and decoding of messages using affine and quadratic functions, demonstrating the practical application of these mathematical concepts and developing problem-solving skills. The theoretical framework of this work includes contributions from WING (2006), BARICHELLO e MATHIAS (2021), LIMA et al. (2006) and LAMBERT (2019), among others. The dissertation presents a detailed didactic sequence for primary and secondary education, discusses the limitations of the code and suggests solutions to improve it. Additionally, it presents ideas for future research, such as the search for a programming language in portuguese and the exploration of other types of functions.

Keywords: Mathematics teaching; Computational thinking; Polynomial functions; Automation.

Lista de Figuras

1.1 Função Soma em Python 9
1.2 Função Soma em C 10
1.3 Interface do VS Code 11
1.4 Interface de um projeto aberto 12
1.5 Declaração de Variável 13
1.6 Redeclaração de Variável 14
1.7 Nova Declaração de Variável 14
1.8 Saída de dados 15
1.9 Saída de dados com *format* 15
1.10 Entrada de dados 16
1.11 Entrada de dados com tratamento da variável 16
1.12 Exemplo de ordenação dos operadores lógicos 20
1.13 Código que calcula o raio de um círculo dado a sua área 21
1.14 Código base de uma Estrutura de Decisão 22
1.15 Código base de uma Estrutura de Repetição 22
1.16 Códigos que capturam números 23
1.17 Exemplos de Listas 24
1.18 Exemplo de Dicionário 25
1.19 Acessando os elementos de um dicionário 26
1.20 Retângulo de lados x e $s - x$ 37

2.1 Translação da função afim 49
2.2 Translação da função quadrática 50
2.3 Inversa da função $f(x) = x - 2$ 51

3.1 Associação caracteres - números 58

3.2 Opção base de escrita 58
3.3 Entrada dos coeficientes da função afim e da mensagem 59
3.4 Separação dos caracteres da mensagem 60
3.5 Bloco de codificação com função afim 60
3.6 Saída da mensagem codificada 61
3.7 Executar o código 61
3.8 Entradas e saídas mostradas no terminal 62
3.9 Entrada da mensagem e dos coeficientes da função quadrática 62
3.10 Separação dos caracteres da mensagem 63
3.11 Bloco de codificação com função quadrática 63
3.12 Saída da mensagem codificada 63
3.13 Entradas e saídas mostradas no terminal 64
3.14 Associação números - caracteres 64
3.15 Entrada da mensagem e dos coeficientes da função afim 65
3.16 Exemplo de posicionamento ruim 66
3.17 Bloco de decodificação com função afim 66
3.18 Saída da mensagem decodificada 67
3.19 Executar o código 67
3.20 Entrada e saída de dados no terminal 68
3.21 Entrada e saída de dados no terminal 68
3.22 Entrada da mensagem e dos coeficientes da função quadrática 69
3.23 Bloco de decodificação com função quadrática 69
3.24 Saída da mensagem decodificada 70
3.25 Entrada e saída de dados no terminal 71
3.26 Entrada e saída de dados no terminal 71

4.1 Validador da função escolhida 80
4.2 Validador da mensagem 81
4.3 Validador da mensagem aprimorado 82
A.1 Preparação de ambiente 1 88
A.2 Preparação de ambiente 2 89
A.3 Preparação de ambiente 3 89
A.4 Preparação de ambiente 4 90

A.5 Preparação de ambiente 5 90
A.6 Preparação de ambiente 6 91
A.7 Preparação de ambiente 7 91
A.8 Preparação de ambiente 8 92
A.9 Preparação de ambiente 9 93
A.10 Preparação de ambiente 10 94
A.11 Preparação de ambiente 11 94
A.12 Preparação de ambiente 12 95
A.13 Preparação de ambiente 13 95
A.14 Preparação de ambiente 14 96
A.15 Preparação de ambiente 15 96
A.16 Preparação de ambiente 16 97
A.17 Preparação de ambiente 17 97
B.1 Mensagem informativa do *Whatsapp* 98
C.1 Interface do Codificador da Função Afim 99
C.2 Interface do Decodificador da Função Afim 100
C.3 Interfaces para Função Quadrática 101

Lista de Tabelas

1.1 Um exemplo de tabela. 17
1.2 Algumas expressões aritméticas 18
1.3 Algumas expressões aritméticas 18
1.4 Tabela Verdade (and) 19
1.5 Tabela Verdade (or) 20
1.6 Tabela Verdade (not) 20
1.7 Métodos de uma Lista 24

2.1 Estrutura de composição da chave criptográfica 42
2.2 Base de uma Tabela de Cifras 43
2.3 Tabela de Cifras 45

Sumário

Introdução **1**

1 Fundamentação Teórica **4**

1.1 A BNCC e o Pensamento Computacional 4

1.2 O Pensamento Computacional 6

1.3 Python 8

1.3.1 Tutorial Visual Studio Code 10

1.3.2 Tutorial Python 13

1.4 Criptografia 27

1.5 Fundamentos Matemáticos 30

1.5.1 Funções Polinomiais 31

1.5.2 Continuidade de uma Função 31

1.5.3 Função Afim 34

1.5.4 Função Quadrática 35

1.5.5 Função Inversa 40

2 Metodologia **42**

2.1 Criptografia com Funções 42

2.2 Criptografia com Função Afim 44

2.3 Criptografia com Função Quadrática 46

2.4 Considerações 47

3 Automatizando o Processo Criptográfico **56**

3.1 O Algoritmo Base 56

3.2 Codificando 57

3.2.1 Função Afim 57

3.2.2 Função Quadrática 62
3.3 Decodificando 64
3.3.1 Função Afim 64
3.3.2 Função Quadrática 68

4 Uma Proposta de Sequência Didática **73**

Considerações Finais **79**

Referências Bibliográficas **87**

Apêndice: Material Adicional **88**
A Primeira Seção do Apêndice 88
B Segunda Seção do Apêndice 98
C Terceira Seção do Apêndice 98
D Quarta Seção do Apêndice 101
D.1 Codificação Função Afim 101
D.2 Decodificação Função Afim 105
D.3 Codificação Função Quadrática 108
D.4 Decodificação Função Quadrática 113

Introdução

Em uma era definida pela onipresença da tecnologia e pela digitalização crescente em todas as esferas da vida, a segurança da informação emerge como um pilar fundamental para a sociedade. Vivemos em um mundo onde dados sensíveis, transações financeiras e comunicações pessoais trafegam constantemente no ciberespaço, demandando mecanismos robustos para garantir sua proteção. Nesse contexto, a criptografia age na proteção da privacidade e na prevenção de fraudes e ataques cibernéticos.

Consequentemente, a integração de tecnologias digitais no currículo escolar é um processo gradual e contínuo. O Artigo 1º da lei Nº 14.533, de 11 de janeiro de 2023 estabelece a Política Nacional de Educação Digital (PNED), que visa potencializar o acesso da população brasileira a recursos, ferramentas e práticas digitais. A PNED integra programas, projetos e ações de diferentes entes federados que abrangem inovação e tecnologia na educação, com apoio técnico ou financeiro do governo federal, e apresenta quatro eixos estruturantes: Inclusão Digital, Educação Digital Escolar, Capacitação e Especialização Digital e Pesquisa e Desenvolvimento em Tecnologias da Informação e Comunicação (TICs).

Metodologias ativas e projetos interdisciplinares têm sido adotados, envolvendo a integração de tecnologias em diferentes disciplinas. Na área da matemática, pesquisadores como José Armando Valente, que destaca o uso de tecnologias digitais e a aprendizagem colaborativa, e Seymour Papert, com sua teoria do construcionismo, incentivam a construção ativa do conhecimento e trazem possibilidades que visam criar um ambiente de aprendizagem mais dinâmico e centrado no estudante. Este trabalho tomará como base, diversos autores, com destaque para Jeannette Wing e Leonardo Barichello, que dissertam sobre o pensamento computacional no ensino e Elon Lages Lima, que trará a base matemática necessária.

Com esse intuito, a proposta é desenvolver uma abordagem que contribua para o

aprimoramento do conhecimento de funções matemáticas utilizando a criptografia como ferramenta pedagógica para estudantes do ensino médio. Além disso, busca a automação do processo criptográfico por meio de uma linguagem de programação. As funções afim e quadrática são os focos, pois além de serem objetos matemáticos familiares aos estudantes, possuem a capacidade de ilustrar, de forma prática e intuitiva, os princípios básicos da criptografia, como a codificação e decodificação de mensagens. A escolha dessas funções e da automação se justifica por sua presença no currículo do ensino médio, em consonância com a Base Nacional Comum Curricular (BNCC).

A metodologia proposta se baseia na construção de um sistema de criptografia simplificado, em que os coeficientes das funções polinomiais atuam como chaves criptográficas. A mensagem a ser codificada é inicialmente convertida em uma sequência numérica, utilizando uma tabela de associação entre caracteres e números. Em seguida, cada número da sequência é transformado pela função polinomial, gerando a mensagem codificada. Para decodificar a mensagem, o processo inverso é realizado, utilizando a função inversa ou resolvendo uma equação.

A aplicação prática da criptografia por meio de funções polinomiais pede oferecer aos estudantes a oportunidade de vivenciar, na prática, a conexão entre a matemática e a segurança da informação. Através da manipulação de funções, da resolução de equações e da análise de gráficos, os alunos visualizam os mecanismos por trás da criptografia, desenvolvendo, ao mesmo tempo, habilidades de raciocínio lógico, abstração e resolução de problemas.

A estrutura da dissertação foi elaborada para apresentar o tema de forma gradual e aprofundada. O Capítulo 1 explora o conceito de Pensamento Computacional e sua relação com a BNCC. Assim como, a linguagem de programação Python é introduzida junto a suas ferramentas e possibilidades de uso para o desenvolvimento de atividades práticas. Também, apresenta a definição de criptografia e uma revisão dos conceitos matemáticos. O Capítulo 2 se dedica à aplicação prática da criptografia com funções, demonstrando, passo a passo, o processo de codificação e decodificação de mensagens utilizando funções afim e quadrática. O Capítulo 3 apresenta um exemplo de algoritmo que automatiza o processo criptográfico, explicando a compreensão da lógica por trás da automação. No Capítulo 4, a proposta didática é concretizada em uma sequência de atividades para o ensino de funções com criptografia, com o intuito de promover os

momentos de construção, observação, correção e validação dos conhecimentos estudados.

Finalmente, as considerações finais refletem sobre o estudo. O capítulo analisa algumas limitações do código e propõe soluções para aprimorá-lo. Também, faz a relação dos códigos em Python, utilizados na automação, com os conceitos da matemática. Além disso, discute desafios como a barreira linguística do Python e sugere ideias para pesquisas futuras, como o uso de uma linguagem em português. A integração da programação contribui no desenvolvimento das habilidades de resolução de problemas.

Espera-se que este trabalho sirva como material de apoio para integrar os conceitos de funções matemáticas com o estudo sobre o pensamento computacional e a praticidade da automação.

Capítulo 1

Fundamentação Teórica

Para a dissertação do tema é de suma importância conceituar a relação entre a BNCC e o Pensamento Computacional, assim como aproximar as ferramentas tecnológicas com o conceito de criptografia. De início aborda-se a integração do Pensamento Computacional no currículo educacional brasileiro, destacando sua relevância no ensino de matemática e sua relação com o desenvolvimento de habilidades cognitivas. Em seguida, apresenta-se a linguagem de programação Python e seu papel na simplificação e versatilidade do desenvolvimento de software. Explora-se os fundamentos da criptografia, discutindo seus objetivos, técnicas e a importância das frentes matemáticas envolvidas em seu processo. Por fim, conceitua as ferramentas matemáticas que serão utilizadas no trabalho.

1.1 A BNCC e o Pensamento Computacional

A Base Nacional Comum Curricular (BNCC) foi prevista em 1988 no Art. 210 da Constituição da República Federativa do Brasil (BRASIL, 1988) estabelecendo as bases legais para a formulação de políticas educacionais no país. O documento traz o seguinte texto:

> Serão fixados conteúdos mínimos para o ensino fundamental, de maneira a assegurar formação básica comum e respeito aos valores culturais e artísticos, nacionais e regionais. (...)
>
> (...) § 2º O ensino fundamental regular será ministrado em língua portuguesa, assegurada às comunidades indígenas também a utilização de suas línguas maternas e processos próprios de aprendizagem. (BRASIL, 1988, Art. 210)

O documento visava uma forma de unificar, nacionalmente, o currículo educacional. Assim, em 1996 foi promulgada a Lei de Diretrizes e Bases da Educação (LDB 9396/96), uma legislação fundamental que regulamentou o sistema educacional brasileiro, estabelecendo princípios e normas para a educação em todos os níveis. A esse respeito, em seu Art. 26, é estabelecido que os currículos do ensino fundamental e médio devem ter uma base nacional comum, a ser complementada, em cada sistema de ensino e estabelecimento escolar, por uma parte diversificada, exigida pelas características regionais e locais da sociedade, da cultura, da economia e da clientela (BRASIL, 1996). Com o intuito de cumprir essa diretriz, foram consolidados os Parâmetros Curriculares Nacionais (PCN), que vieram para auxiliar as equipes escolares na execução de seus trabalhos, sobretudo no desenvolvimento do currículo.

Em 2015 a primeira versão da BNCC foi disponibilizada, estabelecendo os primeiros parâmetros para a definição dos conteúdos mínimos a serem ensinados nas escolas brasileiras. Após ser discutida por todo o país, a segunda versão é disponibilizada em 2016. Por fim, em 2018 foi homologado a versão final do documento.

Ressaltando a área de matemática e suas tecnologias dentro do documento, a BNCC traz o Pensamento Computacional como uma habilidade essencial para os estudantes.

> No Ensino Médio, na área de Matemática e suas Tecnologias, os estudantes devem utilizar conceitos, procedimentos e estratégias não apenas para resolver problemas, mas também para formulá-los, descrever dados, selecionar modelos matemáticos e desenvolver o pensamento computacional, por meio da utilização de diferentes recursos da área. (BRASIL, 2018, p. 470)

Nessa etapa de ensino da educação básica todo conteúdo do componente curricular é aprimorado, visando o alcance de estágios que proporcionam uma visão da matemática integrada à realidade do estudante. Ainda, no mesmo documento:

> (...) a BNCC propõe que os estudantes utilizem tecnologias, como calculadoras e planilhas eletrônicas, desde os anos iniciais do Ensino Fundamental. Tal valorização possibilita que, ao chegarem aos anos finais, eles possam ser estimulados a desenvolver o pensamento computacional, por meio da interpretação e da elaboração de fluxogramas e algoritmos. (BRASIL, 2018, p. 518)

A versão final do documento estabelece uma articulação entre o Ensino Fundamental e o Ensino médio trazendo o pensamento computacional como um dos meios para resolver problemas, desenvolvendo, assim, as competências e habilidades propostas. É justamente o pensamento computacional que será utilizado nesse trabalho como um dos fundamentos necessários para formulação das atividades a serem desenvolvidas.

1.2 O Pensamento Computacional

Quando nos deparamos com o termo **Pensamento Computacional**, é natural fazer associações diretas com o funcionamento dos computadores e suas ações. No entanto, segundo Blikstein (2008), essa associação não é exatamente precisa. Não se resume simplesmente a saber navegar na internet, enviar e-mails, publicar em blogs ou operar programas de processamento de texto. Da mesma forma, conforme Wing (2006, p.35) evidencia, o Pensamento Computacional não é uma habilidade mecânica, tampouco se trata de entender como os computadores pensam, ou mesmo sobre os artefatos que eles criam, como softwares e hardwares.

O que fica evidente é que o Pensamento Computacional não se refere apenas aos benefícios que os computadores nos proporcionam, como a rapidez nos cálculos, precisão nos resultados ou a beleza das altas resoluções visuais de tela. Segundo Wing (2006, p.33), o Pensamento Computacional é uma habilidade fundamental, algo que todos os seres humanos deveriam possuir para funcionar na sociedade moderna. É uma abordagem para resolver problemas, utilizando conceitos computacionais para gerenciar nossas vidas diárias, comunicar-nos e interagir com outras pessoas.

O Centro de Inovação para a Educação Brasileira (CIEB), é uma associação sem fins lucrativos, criada em 2016, com o intuito de promover a cultura de inovação na educação pública brasileira. Entre suas iniciativas, destaca-se o desenvolvimento de um Currículo de Referência para Computação e Tecnologia que, fundamentada pelo parecer CNE/CEB Nº: 2/2022, sugere a adoção de três eixos sendo um deles o Pensamento Computacional. Esse eixo é subdividido em quatro: Reconhecimento de Padrões, Decomposição, Abstração e Algoritmos.

O Reconhecimento de Padrões envolve identificar semelhanças ou tendências nos dados para prever comportamentos futuros e encontrar soluções baseadas em experiências

passadas. A Decomposição consiste em dividir um problema complexo em partes menores e mais gerenciáveis, facilitando sua compreensão e resolução. A Abstração filtra detalhes desnecessários para focar nos aspectos mais importantes de um problema, simplificando a resolução ao concentrar-se nas características essenciais. Os Algoritmos são sequências de passos bem definidos que resolvem um problema ou realizam uma tarefa, proporcionando soluções sistemáticas e repetíveis.

Os três primeiros são elementos já desenvolvidos no componente curricular de matemática, nesse sentido um foco sugerido no documento sobre o quarto, conforme apontado por Leonardo Barichello (2021, p.45), "pode ampliar o nosso olhar na direção desse novo objeto, sem implicar na exclusão dos outros conceitos, pois estes serão desenvolvidos como parte do processo de resolução de problemas matemáticos". Com base nisso, tem-se uma forte relação do Pensamento Computacional com a inserção de suas características no ensino de matemática.

Para promover o Pensamento Computacional em sala de aula, o professor de matemática pode se basear em dois movimentos propostos por Barichello (2021, p.45-46):

> O primeiro movimento propõe a discussão de problemas matemáticos sob o ponto de vista computacional. Isso significa colocar o processo de resolução de um problema matemático como objeto de interesse das discussões, salientando aspectos como clareza, corretude, eficiência, possibilidades para extensão para problemas correlatos, similaridade com outros processos de resolução, etc. (...)
>
> (...) O segundo movimento vai na outra direção, partindo de algoritmos de volta para problemas matemáticos. Nesse movimento, uma vez que os estudantes sejam capazes de criar algoritmos, estes serão usados como ferramenta para resolver e rediscutir problemas matemáticos de forma que talvez não fossem possíveis sem o auxílio de tais ferramentas.

A proposta visa uma discussão a respeito de momentos específicos durante o processo de resolução de um problema. Dado que um problema pode ser abordado de várias maneiras, abordar ambos os movimentos contribui para o desenvolvimento do raciocínio por meio de algoritmos.

Uma forma de representação desses algoritmos é utilizando uma linguagem de programação. Aliás, a BNCC em seu texto traz, no componente curricular de matemática, as seguintes habilidades:

> (EM13MAT315) Investigar e registrar, por meio de um fluxograma, quando possível, um algoritmo que resolve um problema.
>
> (EM13MAT405) Utilizar os conceitos básicos de uma linguagem de programação na implementação de algoritmos escritos em linguagem corrente e/ou matemática.
>
> (EF06MA04) Construir algoritmo em linguagem natural e representá-lo por fluxograma que indique a resolução de um problema simples (por exemplo, se um número natural qualquer é par). (BRASIL, 2018)

Essas habilidades enfatizam o uso do algoritmo como ferramenta de aprendizado, o que mostra que a BNCC entra em conjunto com o CIEB para prover um currículo buscando inovações para chegar no objetivo dentro da área da matemática, sendo ele, "(...) possibilitar que os estudantes construam uma visão mais integrada da Matemática, ainda na perspectiva de sua aplicação à realidade" (BRASIL, 2018).

Por fim, destacando a habilidade EM13MAT405, o uso de uma linguagem de programação é uma maneira de desenvolver o Pensamento Computacional. Daí, conclui-se que, é viável utilizar uma linguagem atrelada com um conteúdo para gerar uma discussão construtiva que busca estar alinhada com o objetivo central da base curricular.

1.3 Python

Com base em uma pesquisa realizada pelo site StackOverflow (2023), a linguagem de programação Python é a segunda mais popular entre a comunidade e a terceira entre os profissionais da área.

Python é reconhecida como uma linguagem interpretada e de alto nível, comumente empregada em uma variedade de domínios, incluindo desenvolvimento web, automação, testes de software, análise de dados, machine learning e criação de jogos (LAMBERT, 2019, p. 22).

Sendo uma linguagem interpretada, isso implica que cada linha de código é verificada e executada sequencialmente, permitindo ajustes rápidos conforme necessário. Por ser uma linguagem de alto nível, Python é próxima da linguagem humana, tornando-a relativamente fácil de ler e escrever. Essas características combinadas facilitam a modi-

ficação e testes rápidos do código, além de contribuir para uma melhor compreensão do mesmo.

Apresenta-se um exemplo em que se compara o Python com uma linguagem não interpretada e de baixo nível, como C ou Assembly. Enquanto em Python pode-se escrever um código mais legível e com menos detalhes técnicos, em linguagens como C ou Assembly, é necessário lidar com detalhes de memória e de hardware, tornando o código mais complexo e menos acessível para aqueles que não têm um conhecimento profundo do funcionamento interno do sistema.

Nos exemplos das Figuras 1.1 e 1.2, a função soma é definida para somar dois números em ambos os casos. Em Python, a sintaxe é mais simples e expressiva, enquanto em C, é necessário declarar explicitamente os tipos dos parâmetros da função, além de incluir os cabeçalhos de biblioteca (#include <stdio.h>) necessários e a função main para iniciar o programa.

Por meio dessas propriedades, Python se destaca como uma ferramenta versátil e acessível para uma ampla gama de aplicações de desenvolvimento de software e análise de dados, tornando-a uma boa escolha de linguagem de programação para o que é proposto neste trabalho.

```python
# Função para somar dois números
def soma(a, b):
    return a + b

# Testa a função e mostra o resultado da soma na tela
resultado = soma(3, 5)
print("Resultado da soma:", resultado)
```

Figura 1.1: Função Soma em Python

```c
// Cabeçalho de Biblioteca
#include <stdio.h>

// Função para somar dois números
int soma(int a, int b) {
    return a + b;
}

// Inicia o Programa
int main() {
    int resultado;

    //Testa a função e mostra o resultado da soma na tela
    resultado = soma(3, 5);
    printf("Resultado da soma: %d\n", resultado);

    //Indica se o código foi executado com sucesso
    return 0;
}

```

Figura 1.2: Função Soma em C

1.3.1 Tutorial Visual Studio Code

Uma opção para trabalhar com a linguagem de programação Python é utilizando o Visual Studio Code (VS Code). Essa ferramenta é um editor de código-fonte gratuito desenvolvido pela Microsoft (2024) e de código aberto que oferece recursos para desenvolver programas. É multiplataforma (Windowns, macOS e Linux), tem suporte para várias linguagens de programação e permite utilizar extensões que auxiliam na construção do código

O download pode ser realizado por meio do site[1] oficial da plataforma e a instalação pode ser acompanhada no apêndice A.

Tal como muitos outros editores de código, o VS Code adota uma interface de usuários comum.

[1]https://code.visualstudio.com/Download

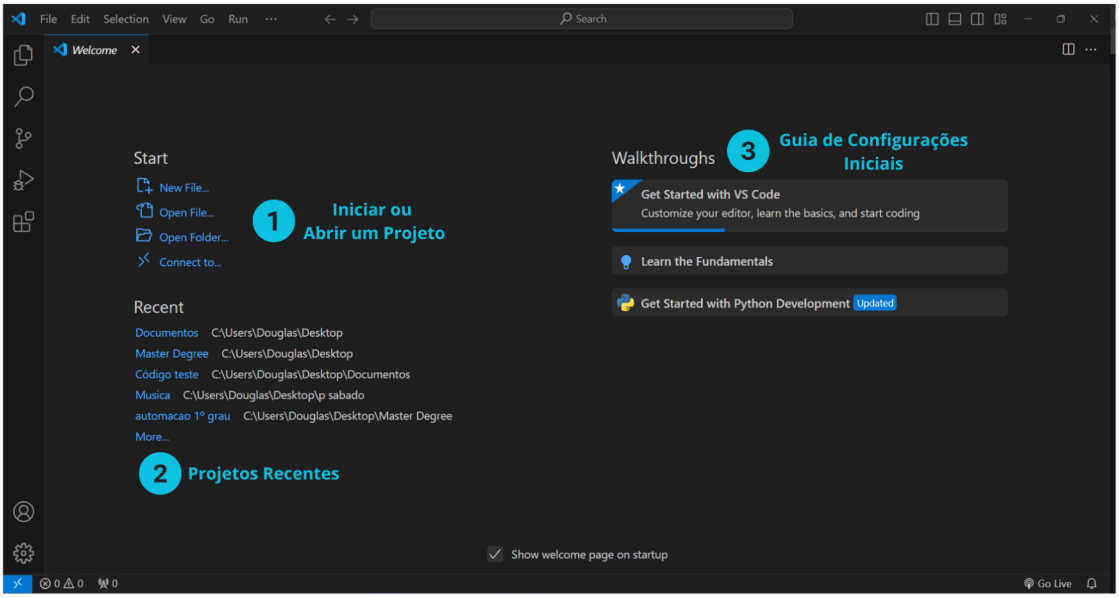


Figura 1.3: Interface do VS Code

A interface de abertura do software, Figura 1.3, apresenta alguns atalhos que guiam o início do projeto.

1. Iniciar ou Abrir um Projeto - Apresenta as opções de criação ou abertura de um arquivo ou pasta;

2. Projetos Recentes - Os últimos projetos que estiverem sendo editados, aparecerá nessa região. (Toda vez que abrir o software, será aberto o último projeto editado)

3. Guia de Configurações Iniciais - Nesse espaço, há alguns tutoriais para configurar o ambiente do software, assim como, alguns comandos básicos de acesso no software.

Recomenda-se inicialmente criar uma pasta no computador para organizar todo o projeto. Após, abrir pelo VS Code a pasta criada selecionando *Open Folder*. Em seguida, a interface que aparecerá será como na Figura 1.4. Nesse ambiente, cria-se o arquivo onde o código de programação será escrito, indo em *File* > *New File*.

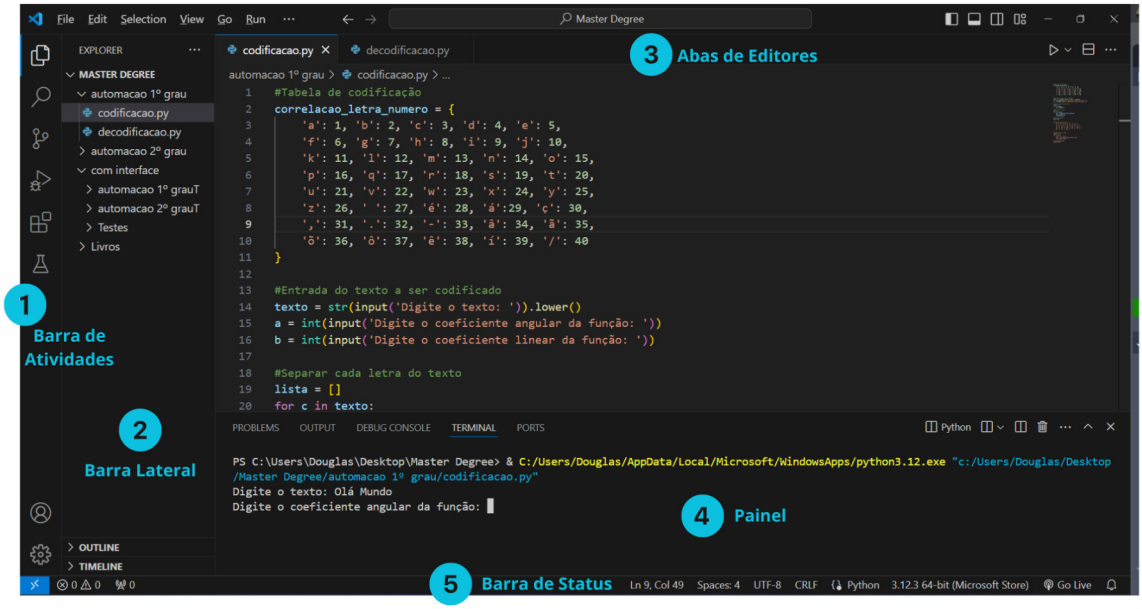


Figura 1.4: Interface de um projeto aberto

O VS Code é fornecido com um layout que maximiza o espaço fornecido para o editor, deixando um amplo espaço para navegar e aceder a todo o contexto da sua pasta ou projeto. A interface do utilizador está dividida em cinco áreas principais:

1. Barra de Atividades - Localizada no lado esquerdo, permite-lhe alternar entre os diferentes modos de exibições, como o número de alterações efetuadas quando o Git[2] está ativado, explorador de arquivos, barra de pesquisa, execução e depuração do código e acesso a extensões;

2. Barra Lateral - Apresenta o conteúdo da Barra de Atividades, ressaltando o explorador de arquivos que permite navegar, criar, excluir e renomear arquivos e pastas, e executar outras operações de gerenciamento de arquivos;

3. Abas de Editores: Abas que selecionam a área principal da interface para editar o código-fonte dos arquivos. O editor de texto oferece recursos avançados, como realce de sintaxe, formatação automática, sugestões de código, integração com Git, navegação por cursor, e diversas outras funções;

4. Painel - Um espaço adicional para visualizações abaixo da região do editor. Por predefinição, aloja a saída, informações de depuração, erros e avisos e um terminal

[2]Git é o sistema usado para controlar o histórico de alterações de arquivos e principalmente de projetos de desenvolvimento de software. Ele permite mais flexibilidade no fluxo de trabalho, segurança e desempenho.

integrado. O painel também pode ser movido para a esquerda ou para a direita para obter mais espaço vertical;

5. Barra de Status - Informações sobre o projeto aberto. Exibe informações como o estado do Git, o número da linha e a coluna do cursor, o tipo de final de linha do arquivo, o idioma do arquivo, entre outros.

Uma sugestão alternativa de editor de código-fonte é utilizar o Google Colab, ferramenta desenvolvida pela Google para escrever e executar código Python diretamente no navegador, sem a necessidade de instalar nada no computador, ou seja, é totalmente online.

1.3.2 Tutorial Python

1.3.2.1 Tipos de Dados

Na programação, os tipos de dados consistem em um conjunto de valores e um conjunto de operações que podem ser realizados sobre esses valores. A maioria dos dados em Python são escritos como: Inteiros, Pontos Flutuantes e "Strings".

Os dados do tipo Inteiro são escritos como *int* e são aqueles dados numéricos representados pelos números inteiros.

Os do tipo Pontos Flutuantes, escritos como *float*, são os dados numéricos representados por números reais.

As "Strings", escritos como *str*, são tipos de dados em formato de texto e sempre sevem ser escritos dentro e aspas simples ou duplas.

1.3.2.2 Variáveis e Declaração de Atribuição

Chama-se de variável a associação de um nome a um valor, simplificando a lembrança e utilização do valor em um programa. Por exemplo:

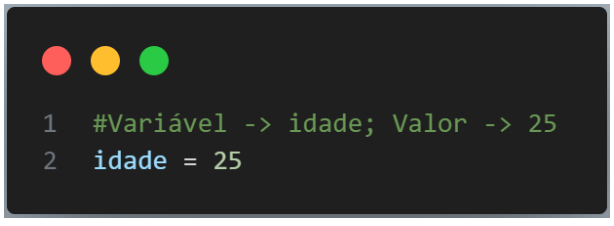


Figura 1.5: Declaração de Variável

Na Figura 1.5, idade é o nome dado para a variável, enquanto 25 é o valor atribuído a ela. Uma variável em Python é um nome dado a um valor específico armazenado na memória do computador. É importante seguir algumas regras ao escolher nomes para variáveis. Por exemplo, certos nomes, como **if**, **def** e **import**, são conhecidos como palavras-chave, sendo reservados para outros fins e não podem ser usados como nomes de variáveis. Em geral, um nome de variável deve começar com uma letra ou um sublinhado, e pode conter qualquer número de letras, dígitos ou outros sublinhados. Além disso, é fundamental lembrar que os nomes de variáveis em Python são sensíveis a maiúsculas e minúsculas, o que significa que *HEIGHT* é considerado um nome diferente de *height*.

As variáveis podem ter seus valores alterados mesmo após já terem sido atribuídos, conforme apresenta-se na Figura 1.6.

Figura 1.6: Redeclaração de Variável

Ao executar o código dessa forma, a variável `idade` receberá o valor 25 e logo na sequência receberá o valor 26. Ao pedir para mostrar o valor da variável, será apresentado o valor 26. Uma outra possibilidade pode ser vista na Figura 1.7.

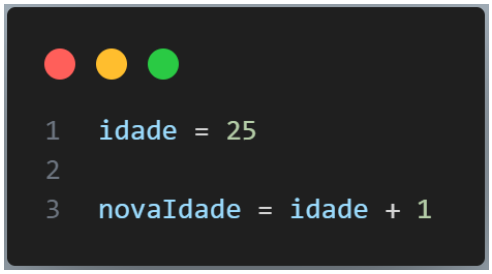


Figura 1.7: Nova Declaração de Variável

Nessa situação, a variável `idade` receberá o valor 25 e na sequência, uma nova variável nomeada como `novaIdade`, receberá o valor de `idade` somado com 1. A execução desse código resultará em 26.

1.3.2.3 Saída de Dados

A saída de dados é uma parte crucial da programação, pois essa ação permite comunicar informações para o usuário ou para outros programas. Por meio dela tem-se o feedback sobre o resultado da execução do código.

Apresentado na Figura 1.8, a forma mais comum de realizar essa ação é utilizando o comando `print( )`.

```
1  #Variável -> nome; Valor/string -> Douglas
2  nome = "Douglas"
3
4  #Variável -> idade; Valor/inteiro -> 30
5  idade = 30
6
7  #Saída de dados
8  print("Olá,", nome, "! Você tem", idade, "anos.")
```

Figura 1.8: Saída de dados

Da estrutura do comando `print`, entre os parênteses vai o conteúdo que será apresentado na saída de dados. Entre aspas duplas (podendo ser aspas simples) estão escritos as palavras que irão compor uma mensagem. A vírgula, fora das aspas, separa o que é palavra escrita e o que é variável. Dessa forma, a saída será mostrada da seguinte maneira:

Olá, Douglas! Você tem 30 anos.

Uma outra forma para o mesmo exemplo, que resultará na mesma saída, é colocar dentro do parênteses a letra `f`, assim como na Figura 1.9, que significa *format*, permitindo que a frase seja escrita de uma vez só e fazendo com que os valores editáveis fiquem entre chaves.

```
1  print(f"Olá, {nome}! Você tem {idade} anos.")
```

Figura 1.9: Saída de dados com *format*

Caso o programador queira, pode-se apenas colocar a variável dentro dos parênteses para obter uma saída.

1.3.2.4 Entrada de Dados

Pode ser que o usuário de um software queira acrescentar um dado ao programa em execução para poder obter um resultado de saída. Em Python, utiliza-se o comando `input( )`, conforme as Figuras 1.10 e 1.11.

```
1  #Entrada de dados
2  nome = input("Digite seu nome: ")
3
4  #Saída de dados
5  print("Olá,", nome,"! Bem-vindo ao programa.")
```

Figura 1.10: Entrada de dados

Neste exemplo, o programa solicita ao usuário que digite seu nome. O que for digitado é armazenado na variável `nome` e, em seguida, é apresentado ao usuário. O texto dentro dos parênteses do `input` aparecerá, também, na tela e o código só continuará executando depois que receber um valor.

É importante notar que tudo o que é recebido pela função `input( )` é tratado como uma string. Se você deseja trabalhar com outros tipos de dados, como inteiros ou pontos flutuantes, precisará converter a entrada usando funções como `int( )` ou `float( )`.

```
1  idade = int(input("Digite sua idade: "))
```

Figura 1.11: Entrada de dados com tratamento da variável

Dessa forma, quando o usuário fornece o dado de entrada, ele é, automaticamente, reconhecido como um valor inteiro.

1.3.2.5 Operadores Matemáticos

Uma expressão aritmética é constituída por operandos e operadores combinados de uma forma que é familiar na álgebra. A Tabela 1.1 mostra vários operadores aritméticos relacionados com sua sintaxe para uso no código Python e a Tabela 1.2 traz exemplos de usos desses operadores.

Tabela 1.1: Um exemplo de tabela.

Operador	Significado	Sintaxe
`-`	Negação	`- a`
`**`	Exponenciação	`a**b`
`*`	Multiplicação	`a*b`
`/`	Divisão	`a / b`
`//`	Quociente	`a // b`
`%`	Resto da divisão	`a % b`
`+`	Adição	`a + b`
`-`	Subtração	`a - b`

Fonte: LAMBERT (2019).

Na álgebra, a multiplicação pode ser indicada da seguinte maneira: `ab`. Em Python, as expressões devem ser escritas explicitamente usando os operadores, como na tabela: `a*b`.

As regras de ordenação das operações matemáticas se mantém na linguagem Python:

- A operação de exponenciação tem a maior prioridade entre os operadores;
- O sinal de negação é verificado na sequência, antes da multiplicação, divisão e resto;
- A multiplicação, os dois tipos de divisão e o resto são verificados antes da adição e da subtração;
- A adição e a subtração são verificados antes da atribuição;
- Pode-se utilizar parênteses para alterar a ordem de operação.

Com exceção da divisão exata, quando ambos os operandos de uma expressão aritmética são do mesmo tipo numérico (*int* ou *float*), o valor resultante também é desse

Tabela 1.2: Algumas expressões aritméticas

Expressão	Desenvolvimento	Resultado
5 + 3 * 2	5 + 6	11
(5 + 3) * 2	8 * 2	16
6 % 2	0	0
2 * 3 ** 2	2 * 9	18
- 3 ** 2	- (3 ** 2)	- 9
(3) ** 2	9	9
2 ** 3 ** 2	2 ** 9	512
(2 ** 3) ** 2	8 ** 2	64

Fonte: LAMBERT (2019).

tipo. Quando cada operando é de um tipo diferente, o valor resultante é do tipo mais geral. Note que o tipo *float* é mais geral que o tipo *int*. O operador de quociente (`//`) produz um quociente inteiro, enquanto que o operador de divisão exata (`/`) produz sempre um *float*. Assim, `3 // 4` produz `0`, enquanto `3 / 4` produz `0.75`.

1.3.2.6 Operadores Relacionais

Complementando os operadores matemáticos, existem os operadores relacionais que são utilizados para comparar dois valores. Esses operadores são essenciais para realizar comparações em expressões lógicas e controlar o fluxo de execução em programas. Para escrever expressões que incluem comparações, utiliza-se os operadores listados na Tabela 1.3.

Tabela 1.3: Algumas expressões aritméticas

Operador	Significado
==	Igual
! =	Diferente
<	Menor que
>	Maior que
<=	Menor ou igual
>=	Maior ou igual

Fonte: LAMBERT (2019).

Ao executar uma expressão utilizando operadores relacionais, o resultado da expressão será um valor booleano, que por sua vez, é um tipo de dado em programação que pode ter um entre dois valores: **True** (verdadeiro) ou **False** (falso). Por exemplo, considere as variáveis `a` e `b`, sendo `a` maior que `b`. Fazendo a comparação com o operador, o resultado de `a` $>$ `b` será True e `a` $<$ `b` será False.

É importante ressaltar que `==` significa igual, enquanto `=` significa atribuição.

1.3.2.7 Operadores Lógicos

O Python inclui três operadores lógicos, **and**, **or** e **not**. Tanto o operador **and** quanto o operador **or** esperam dois operandos. O operador **and** retorna True se e somente se ambos os seus operandos forem verdadeiros, e retorna False caso contrário. O operador **or** retorna False se e somente se ambos os seus operandos forem falsos, e retorna True caso contrário. O operador **not** espera um único operando e retorna sua negação lógica, True, se for falso, e False, se for verdadeiro.

Uma tabela verdade é uma tabela que mostra todas as combinações possíveis de valores para os operandos de um operador, juntamente com o resultado da aplicação desse operador a esses valores. Cada linha representa uma dessas combinações, e a primeira linha geralmente contém os rótulos dos operandos e a expressão que está sendo avaliada, como vê-se nas Tabelas 1.4 até 1.6.

Tabela 1.4: Tabela Verdade (and)

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

Fonte: LAMBERT (2019).

Tabela 1.5: Tabela Verdade (or)

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Fonte: LAMBERT (2019).

Tabela 1.6: Tabela Verdade (not)

A	not A
True	False
False	True

Fonte: LAMBERT (2019).

Na subseção 1.3.2.5, foi apresentado regras de ordenação dos operadores matemáticos, como a multiplicação e a divisão ter maior prioridade do que a adição e a subtração. Isso significa que os operadores com prioridade mais alta são verificados primeiro, mesmo que apareçam à direita dos operadores de prioridade mais baixa. A mesma ideia se aplica aos operadores relacionais, lógicos e de atribuição. Os operadores lógicos são verificados após as comparações, mas antes do operador de atribuição. O operador **not** tem uma prioridade maior do que o operador **and**, que tem prioridade maior do que o operador **or**, verificados nas Figuras 1.12(a) e 1.12(b).

```
1  a = True
2  b = False
3
4  print(a or b and b)
```

(a) A saída será True

```
1  a = True
2  b = False
3
4  print((a or b) and b)
```

(b) A saída será False

Figura 1.12: Exemplo de ordenação dos operadores lógicos

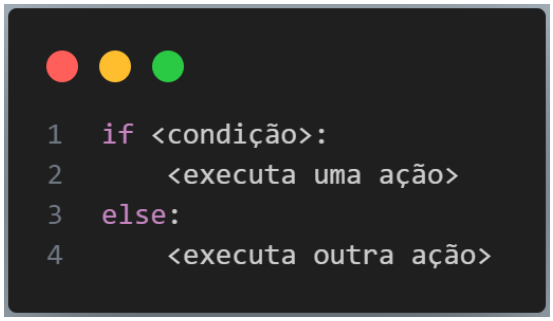
1.3.2.8 Estrutura de Decisão

A estrutura de decisão *if-else* é amplamente utilizada para verificar a validade das entradas e para lidar com erros, se houver, fornecendo mensagens de erro apropriadas. Se as entradas forem consideradas válidas, a estrutura *if-else* permite que o programa execute cálculos ou ações específicas. Ela também é chamada de estrutura de seleção bidirecional, pois orienta o computador a fazer uma escolha entre dois cursos de ação alternativos. Por exemplo, suponha que um programa permita que um usuário insira o valor da área de um círculo para calcular seu raio. As entradas legítimas para esse programa seriam números positivos. Mas, por engano, o usuário ainda poderia inserir um zero ou um número negativo. Como o programa não tem escolha a não ser usar esse valor para calcular o raio, ele pode travar (parar de funcionar) ou produzir um resultado sem sentido.

Na Figura 1.13, o código mostra como esse programa funciona sem apresentar um possível resultado imprevisível. A estrutura de decisão permite conferir se a área do círculo é positiva, prosseguindo com o cálculo do raio ou pedindo para que indique um valor válido. Enquanto a Figura 1.14, traz duas possibilidades de sintaxe para essa estrutura de decisão.

```python
pi = 3.141592653589793

#Aqui o programa recebe um valor
areaDoCirculo = float(input("Digite a área do círculo: "))

#Estrutura de Decisão
if areaDoCirculo > 0:
    raioDoCirculo = (areaDoCirculo / pi)**(1/2)
    print(raioDoCirculo)
else:
    print("Digite um valor positivo para a área do círculo")
```

Figura 1.13: Código que calcula o raio de um círculo dado a sua área



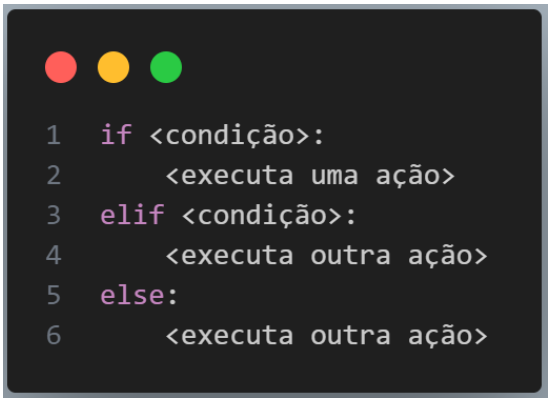
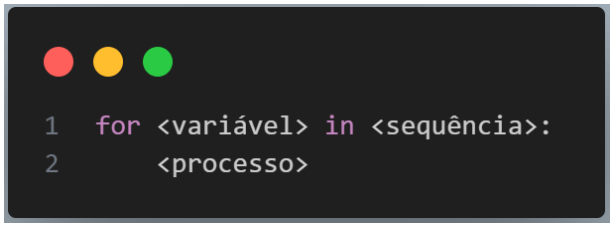


Figura 1.14: Código base de uma Estrutura de Decisão

Além de *if-else*, existe o *elif* que é uma contração para *else if*. Essa palavra-chave serve para quando tem-se mais de duas condições a serem consideradas em um problema.

1.3.2.9 Estrutura de Repetição

Também conhecido como **loop** (ou **laço**), há dois tipos de estruturas de repetição: o *for* e o *while*. O primeiro repete uma ação uma quantidade pré-definida de vezes, enquanto o segundo repete uma ação até o programa determinar quando precisa parar. Pode-se visualizar as estruturas base dos códigos na Figura 1.15.



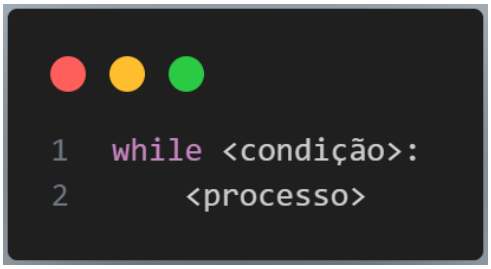


Figura 1.15: Código base de uma Estrutura de Repetição

Como exemplo, considere escolher 6 números para jogar na loteria. O seguinte programa pergunta quais números serão escolhidos:

```python
#Para cada número no "range", faça a pergunta
for numero in range(1, 7):
    pergunta = int(input(f"Digite o {numero}º número: "))
```

(a) Utilizando for

```python
#Começando do 1, enquanto "contador" for menor que 7, faça a pergunta
contador = 1
while contador < 7:
    pergunta = int(input(f"Digite o {contador}º número: "))
    contador = contador + 1
```

(b) Utilizando while

Figura 1.16: Códigos que capturam números

Na Figura 1.16(a), é utilizado a estrutura de repetição *for* para perguntar quais números o usuário quer escolher. A linha 2 pode ser lida da seguinte forma: Para cada número no intervalo de 1 a 6, realize a pergunta. É importante ressaltar que a palavra-chave *range* é uma sequência que aceita dois parâmetros. O primeiro é o valor inicial e o segundo é o valor que procede o último valor da sequência, ou seja, se for 7, a sequência irá até 6.

Na Figura 1.16(b), é utilizado a estrutura de repetição *while* para executar a mesma ação que o primeiro exemplo. Na linha 2, cria-se uma variável que receberá o valor 1 indicando ser o primeiro elemento da sequência. Na linha 3, inicia-se o loop com a condição de que ele será executado enquanto a variável `contador` for menor que 7. A cada vez que o usuário escolher um número será acrescentado uma unidade ao contador, até que o loop chegue na sua condição de parada.

1.3.2.10 Listas

Uma lista é uma sequência de valores de dados chamados de itens ou elementos. Esses elementos podem ser de qualquer tipo, inclusive do tipo lista. Fazendo uma relação do conceito de lista em Python com uma fila de pessoas em uma loja, em ambos os casos pode-se adicionar alguém ao final da fila, chamar alguém pelo nome para sair da fila ou,

até mesmo, trocar uma pessoa por outra, ressaltando que a ordem das pessoas na fila importa.

A lista é escrita como uma sequência de elementos separados por vírgula, sendo que toda a sequência é fechada por colchetes. Alguns exemplos são apresentador na Figura 1.17.

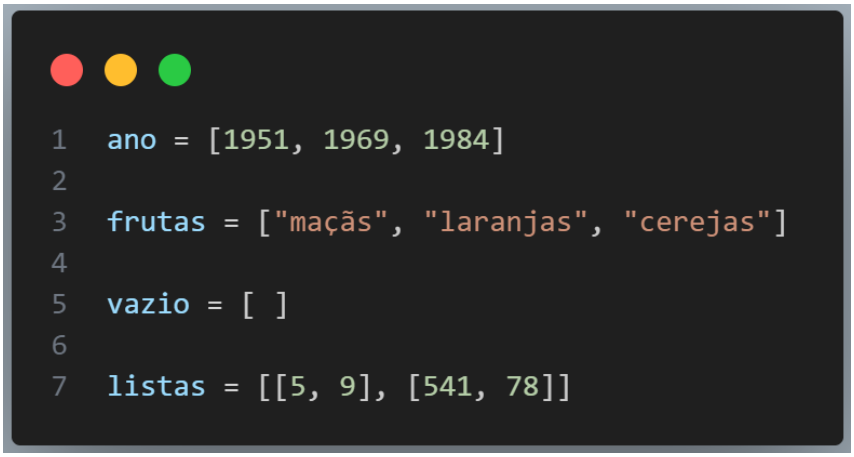


Figura 1.17: Exemplos de Listas

Ao escrever o código, a lista pode ser atribuída diretamente à uma variável já com todos os seus elementos ou pode recebê-los conforme o código vai sendo executado. Como a ordem de cada elemento importa, a posição de cada um é numerada iniciando de 0. Logo, tomando as listas do exemplo acima, frutas é uma lista de strings onde o elemento `"maçãs"` ocupa a posição 0, `"laranjas"` ocupa a posição 1 e `"cerejas"` ocupa a posição 2, totalizando 3 elementos na lista.

Segue na Tabela 1.7 como manipular as listas no Python.

Tabela 1.7: Métodos de uma Lista

Métodos de uma Lista	O que faz
.append(elemento)	Adiciona o elemento no final da lista
.insert(indice, elemento)	Insere o elemento na posição do indice
.pop( )	Remove e retorna o elemento do final da lista
.pop(indice)	Remove e retorna o elemento na posição do indice

Fonte: LAMBERT (2019).

1.3.2.11 Dicionários

Um dicionário organiza as informações por associação, não por posição como feito pelas listas. Por exemplo, quando um dicionário é usado para procurar a definição de "mamífero", a busca não inicia na página 1; em vez disso, vai diretamente para as palavras que começam com "M". Listas telefônicas, catálogos de endereços, enciclopédias e outras fontes de referência também organizam as informações por associação.

Em Python, um dicionário associa um conjunto de chaves a valores. Por exemplo, as chaves de uma agenda telefônica compreendem o conjunto de nomes, enquanto os valores de dados associados são seus respectivos números.

A sintaxe de um dicionário é uma sequência, delimitada por chaves, contendo as associações separadas por vírgula, como vê-se nas Figuras 1.18(a) e 1.18(b).

(a) Estrutura base

(b) Exemplo Agenda Telefônica

Figura 1.18: Exemplo de Dicionário

Para acessar os elementos de um dicionário em Python, pode-se usar a chave correspondente ao valor que deseja acessar. A sintaxe necessária é usar colchetes após o nome do dicionário, e dentro dos colchetes, coloca-se a chave do elemento que deseja acessar.

Figura 1.19: Acessando os elementos de um dicionário

Exemplificando, a variável `numero`, na Figura 1.19, receberá o valor 12345678.

1.3.2.12 Funções

Em Python, uma função é um bloco de código reutilizável que realiza uma tarefa específica quando chamado. Ela é uma maneira de organizar e modularizar o código, permitindo que seja reutilizado em diferentes partes de um programa. Por exemplo, a Figura 1.1 mostra uma função que realiza a soma de dois números.

Uma função se forma com os seguintes elementos:

- Nome da Função: Uma função é definida usando a palavra-chave *def*, seguida pelo nome da função. O nome da função deve seguir as mesmas regras de nomenclatura que se aplicam às variáveis em Python. No exemplo da figura o nome é `soma`.

- Parâmetros (Opcionais): Uma função pode aceitar zero ou mais parâmetros, que são valores que podem ser passados para a função quando ela é chamada. Os parâmetros são colocados entre parênteses após o nome da função. Eles servem como entradas para a função e podem ser usados dentro do bloco de código da função para realizar operações. No exemplo, os parâmetros são `a` e `b`.

- Corpo da Função: O corpo da função contém o código que define o comportamento da função. Ele é indentado (geralmente usando espaços em branco ou tabulação) e pode conter qualquer número de instruções Python válidas. Este é o bloco de código que é executado quando a função é chamada.

- Instrução *return* (Opcional): Uma função pode ou não retornar um valor de volta para o local onde foi chamada. Se uma função não tiver uma instrução *return*, ela retornará *None*. A instrução *return* pode ser usada para especificar explicitamente o valor a ser retornado pela função. No exemplo, a função retorna o resultado da conta `a + b`.

Chamar uma função significa utilizar o bloco definido para execução da tarefa. Quando a palavra-chave *def* é usada para definir uma função, não é dentro desse bloco que as tarefas específicas são realizadas, ele apenas armazena o que aquela função faz. Para executá-la é necessário alocar a função em uma variável, por exemplo:

```
resultado = soma(3, 5)
```

Ao chamar a função *soma* com os parâmetros 3 e 5, a variável *resultado* armazenará o resultado da soma retornada pela função (Figura 1.1).

1.4 Criptografia

Em um meio em que as tecnologias da informação e comunicação crescem em relação aos anos anteriores, a utilização de sistemas que empregam criptografia para manter a segurança da informação e dados do usuário é essencial. Conforme a Pesquisa Nacional por Amostra de Domicílios Contínua (IBGE, 2024), entre os anos 2016 e 2022, houve um aumento em 9% na presença de televisores, em 16% no uso de telefone celular e em 44% ao acesso à internet, quando se refere ao ambiente domiciliar. Esses números destacam a crescente dependência das pessoas em relação às tecnologias de comunicação e informação, evidenciando a importância de proteger essas comunicações por meio da criptografia.

Derivada de dois vocábulos gregos: kryptos (oculto) e graphien (escrever), a criptografia é uma técnica que consiste em ocultar o significado de uma mensagem. Se há necessidade de ocultar o que está nessa mensagem, então há um sujeito que não pode saber o conteúdo dela (FIARRESGA, 2010, p. 3). Sendo assim, os objetivos da criptografia são:

> Confidencialidade – mantém o conteúdo da informação secreto para todos excepto para as pessoas que tenham acesso à mesma.
>
> Integridade da informação – assegura que não há alteração, intencional ou não, da informação por pessoas não autorizadas.
>
> Autenticação de informação – serve para identificar pessoas ou processos com quem se estabelece comunicação.
>
> Não repudiação – evita que qualquer das partes envolvidas na comunicação negue o envio ou a recepção de uma informação. (FIARRESGA, 2010, p. 3)

Podemos observar que o celular, junto da internet, está bem presente nas tarefas diárias e, pelo celular carregar uma grande quantidade de informações, sejam elas, bancárias, textuais ou visuais, precisa-se de certo cuidado ao utilizar suas ferramentas e possibilidades. As características buscadas ao se aplicar a criptografia em sistemas servem para manter o sigilo desses tipos de informações.

No que se refere ao processo de criptografar um conteúdo, o mecanismo básico é chamado de cifra (SHOUP e BONEH, 2023, p.4). O conteúdo a ser criptografado passa por transformações por meio das cifras, que por sua vez é dividido em dois tipos: cifras de transposição e cifras de substituição. Enquanto na cifra de transposição cada letra conserva a sua identidade, mas muda de posição dentro da mensagem; na cifra de substituição, cada letra conserva a sua posição, mas é substituída por uma outra letra ou símbolo (KAHN, 1968, p.xiii).

A maioria das cifras utiliza uma chave, que determina aspectos como a disposição das letras em um alfabeto cifrado ou o padrão de embaralhamento em uma transposição. Essa chave pode ser uma palavra, frase ou número, e é denominada palavra-chave, frase-chave ou número-chave, conforme apropriado (KAHN, 1968, p. xv). Pode-se entender que uma chave é uma regra pela qual o conteúdo irá passar para haver a transformação em um conteúdo criptografado. Essa regra precisa ser de conhecimento do destinatário para que possa realizar o processo de reversão da transformação e recuperar o conteúdo original.

Consequentemente, algumas frentes matemáticas são utilizadas para fazer o processo criptográfico. Entre essas frentes, destacam-se a Aritmética Modular e a Fatoração.

Além disso, Tamarozzi (2001, p. 41) enfatiza que, o processo criptográfico utilizando funções e matrizes constitui um material útil para elaborar e desenvolver exercícios para fixação do conhecimento matemático associado a esses conteúdos.

Iniciando com a Aritmética Modular, ela é utilizada em uma variedade de algoritmos de cifração e troca de chaves. Ela envolve operações como adição, subtração e multiplicação, onde o resultado é tomado como o resto da divisão por um número inteiro, chamado de módulo. Esse conceito aparece em sistemas como o RSA e o Diffie-Hellman, onde a segurança é baseada na dificuldade computacional de resolver problemas matemáticos relacionados à aritmética modular.

A Fatoração desempenha um papel central em sistemas de criptografia de chave pública, como o algoritmo RSA. Nesse método, a chave pública é derivada da multiplicação de dois números primos grandes, enquanto a chave privada é obtida a partir da fatoração desses números. A dificuldade de fatoração de números grandes em seus primos constituintes é a base da segurança do algoritmo, tornando a fatoração um componente crítico da criptografia assimétrica.

Matrizes também são frequentemente empregadas em criptografia, principalmente em algoritmos de cifração simétrica, como o AES (Advanced Encryption Standard). Nesses sistemas, a matriz de chave é combinada com os dados de entrada por meio de operações matriciais, como multiplicação ou adição modular, para produzir a saída cifrada. A estrutura matricial permite a aplicação eficiente de transformações complexas nos dados, garantindo a segurança do processo de cifragem.

Por fim, as funções, desempenham um papel vital na criptografia moderna, especialmente em algoritmos de hash criptográficos. Uma função de hash recebe uma entrada de tamanho variável e produz uma saída de tamanho fixo, chamada de resumo hash. Essa saída é única para cada entrada específica e é praticamente impossível reverter o processo para recuperar a entrada original, tornando as funções de hash essenciais para garantir a integridade dos dados e a autenticação das mensagens.

Contudo, abordar as funções matemáticas por meio de algoritmos hash tem alto nível de complexidade. Buscando justamente uma forma mais didática em que possa trabalhar o conceito de função matemática por meio da criptografia, esse trabalho focará na utilização das funções polinomiais de primeiro e segundo grau.

1.5 Fundamentos Matemáticos

Esta subseção apresenta as ferramentas matemáticas utilizadas para o desenvolvimento das atividades propostas nesse trabalho. As fontes originam dos autores LIMA et al. (2006), LIMA (2011), MUNIZ (2022) e IEZZI e MURAKAMI (2013).

Definição 1 *Dados conjuntos não vazios X e Y, uma relação de X em Y é uma* ***função*** *se a seguinte condição for satisfeita:*

$$\forall x \in X, \exists \text{ um único } y \in Y, \text{ tal que}, y = f(x)$$

O conjunto X chama-se domínio e o conjunto Y contra-domínio da função f ($f : X \to Y$). Para cada elemento $x \in X$, o elemento $f(x) \in Y$ chama-se a imagem de x pela função f, ou o valor assumido pela função f no ponto $x \in X$. Uma forma de visualizar o domínio e a imagem é chamando-os de conjunto de partida e subconjunto do contradomínio, respectivamente.

A **restrição** de uma função $f : X \to Y$ a um subconjunto $A \subset X$ é a função $f|A : A \to Y$, definida por $(f|A)(x) = f(x)$ para todo $x \in A$.

Definição 2 *(Injetividade) Uma função $f : X \to Y$ chama-se* ***injetiva*** *quando elementos diferentes em X são transformados por f em elementos diferentes em Y. Ou seja, f é injetiva quando, para $x, x' \in X$, tem-se $x \neq x' \Rightarrow f(x) \neq f(x')$.*

Definição 3 *(Sobrejetividade) Diz-se que uma função $f : X \to Y$ é* ***sobrejetiva*** *quando, para qualquer elemento $y \in Y$, pode-se encontrar (pelo menos) um elemento $x \in X$ tal que $f(x) = y$.*

Utilizando-se das duas definições anteriores, uma função $f : X \to Y$ é **bijetiva**, ou uma correspondência biunívoca entre X e Y, quando é ao mesmo tempo injetiva e sobrejetiva.

A respeito do comportamento de uma função quanto ao seu crescimento ou decrescimento ao longo do seu domínio, define-se:

Definição 4 *Uma função $f : X \to \mathbb{R}$ é dita* ***crescente*** *quando para $x, y \in X \subset \mathbb{R}$, $x < y \Rightarrow f(x) \leq f(y)$. Se $x < y \Rightarrow f(x) \geq f(y)$, f diz-se* ***decrescente****. Se vale a implicação mais estrita $x < y \Rightarrow f(x) < f(y)$ dizemos que a função f é* ***estritamente crescente****. Finalmente, se $x < y \Rightarrow f(x) > f(y)$, dizemos que f é uma função* ***estritamente decrescente****.*

1.5.1 Funções Polinomiais

Definição 5 *Diz-se que $p : \mathbb{R} \to \mathbb{R}$ é uma **função polinomial** quando existem números reais $a_0, a_1, ..., a_n$ tais que, para todo $x \in \mathbb{R}$, tem-se*

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0. \tag{1.1}$$

Se $a_n \neq 0$, diz-se que p tem grau n. Os números $a_0, a_1, ..., a_n$ são denominados **coeficientes** e as parcelas $a_0, a_1 x, ..., a_{n-1} x^{n-1}, a_n x^n$ são chamados **termos** do polinômio p. De acordo com a quantidade de termos, não nulos, pode-se classificar os polinômios da seguinte forma:

- monômio $\Rightarrow$ $q(x) = a_0$
- binômio $\Rightarrow$ $h(x) = a_1 x + a_0$
- trinômio $\Rightarrow$ $w(x) = a_2 x^2 + a_1 x + a_0$
- acima de três termos, chama-se polinômio $\Rightarrow$ função $p(x)$ (1.1)

Os binômios e os trinômios são funções polinomiais bem conhecidas por modelar diversas situações, como pode-se verificar nas subseções seguintes (1.5.3 e 1.5.4).

1.5.2 Continuidade de uma Função

Para utilizarmos a continuidade das funções e algumas de suas propriedades, comecemos com a definição formal e alguns exemplos.

Definição 6 *Uma função $f : X \to \mathbb{R}$ é contínua em um ponto $x_0 \in X$ se a seguinte condição for satisfeita: dado $\epsilon > 0$, existe $\delta > 0$ tal que*

$$x \in X,\ |x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \epsilon.$$

A função f é dita contínua se o for em todo $x_0 \in X$.

Exemplo 1 Toda função constante é contínua.

Solução. Sejam c um real dado e $f : \mathbb{R} \to \mathbb{R}$ função constante e igual a c. Para mostrar que f é contínua em $x_0 \in \mathbb{R}$ a definição de continuidade pede que, dado $\epsilon > 0$,

encontremos $\delta > 0$ tal que, para $x \in \mathbb{R}$, a validade da desigualdade $|x - x_0| < \delta$ implique a validade de $|f(x) - f(x_0)| < \epsilon$. Mas, como $|f(x) - f(x_0)| = |c - c| = 0$, a desigualdade $|f(x) - f(x_0)| < \epsilon$ é sempre válida, independentemente de qualquer restrição sob $|x - x_0|$. De outra forma, tomando δ igual a qualquer real positivo, sempre teremos que $|x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \epsilon$, uma vez que a desigualdade $|f(x) - f(x_0)| < \epsilon$ não tem como ser falsa.

Exemplo 2 Sejam a e b números reais dados, sendo $a \neq 0$. Se $f : \mathbb{R} \to \mathbb{R}$ é dada por $f(x) = ax + b$, então f é contínua.

Solução. Novamente, fixado $x_0 \in \mathbb{R}$ se $|x - x_0| < \delta$, temos

$$|f(x) - f(x_0)| = |(ax + b) - (ax_0 + b)| = |a||x - x_0| < |a|\delta.$$

Portanto, se $|a|\delta \leq \epsilon$ (ou, equivalentemente, $\delta < \dfrac{\epsilon}{|a|}$), teremos $|f(x) - f(x_0)| < \epsilon$ para $|x - x_0| < \delta$.

Proposição 1 *Se $f, g : I \to \mathbb{R}$ são funções contínuas em $x_0 \in I$, então as funções $f \pm g, f \cdot g : I \to \mathbb{R}$ também são contínuas em x_0.*

Prova. Seja $(a_n)_{n\geq 1}$ uma sequência em I, tal que $\lim\limits_{x\to+\infty} a_n = x_0$. A continuidade das funções f e g em x_0 garante que

$$\lim_{n\to+\infty} f(a_n) = f(x_0) \text{ e } \lim_{n\to+\infty} g(a_n) = g(x_0).$$

Tomando $\oplus =$ operações básicas (adição, subtração e multiplicação), segue que

$$\begin{aligned}
\lim_{n\to+\infty}(f \oplus g)(a_n) &= \lim_{n\to+\infty}(f(a_n) \oplus g(a_n)) \\
&= \lim_{n\to+\infty} f(a_n) \oplus \lim_{n\to+\infty} g(a_n) \\
&= f(x_0) \oplus g(x_0) \\
&= (f \oplus g)(x_0).
\end{aligned}$$

Portanto, as funções $f \pm g$ e $f \cdot g$ são contínuas em x_0.

Exemplo 3 Dado $a \in \mathbb{R}$ e $k \in \mathbb{N}$ a função $f : \mathbb{R} \to \mathbb{R}$ dada por $f(x) = ax^k$ é contínua.

Solução. Podemos ver a função f como o produto da função constante $g(x) = a$ por k cópias da função identidade $h(x) = x$. Pelos Exemplos 1 e 2, sabemos que g e h são contínuas em x_0, logo podemos afirmar que f é contínua em x_0, assegurada pela Proposição 1.

Teorema 1 Toda função polinomial, isto é, toda função $f : \mathbb{R} \to \mathbb{R}$ tipo

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0,$$

para certos $n \in \mathbb{N}$ e $a_0, a_1, ..., a_n \in \mathbb{R}$, com $a_n \neq 0$, é contínua.

Prova. Vimos, pelo Exemplo 3, que cada termo $a_n x^n, a_{n-1} x^{n-1}, ..., a_1 x, a_0$ são funções contínuas em x_0. Logo, ao somarmos esses termos, pela Proposição 1, teremos uma função contínua. Portanto, a função f é contínua em x_0.

Teorema 2 *Seja $I \subset \mathbb{R}$ um intervalo. Toda função contínua injetiva $f : I \to \mathbb{R}$ é estritamente crescente ou estritamente decrescente e sua inversa $g : J \to I$, definida no intervalo $J = f(I)$, é contínua.*

Prova. Disponível na página 80 do livro Análise Real.

Teorema 3 *(Weierstrass) Se $f : [a, b] \to \mathbb{R}$ é uma função contínua, então existem $x_1, x_2 \in [a, b]$ tais que*

$$f(x_1) \leq f(x) \leq f(x_2)$$

Prova. Disponível na página 83 do livro Análise Real.

OBS.: No Teorema 3, se adicionarmos a hipótese de que f seja estritamente crescente, teremos que $x_1 = a$ e $x_2 = b$. De forma análoga, se f for estritamente decrescente, teremos que $x_1 = b$ e $x_2 = a$.

Proposição 2 *Toda função estritamente crescente ou estritamente decrescente é injetiva.*

Prova. Suponha que $f : X \to Y$ é estritamente crescente. Para mostrar que f é injetiva, precisamos mostrar que $x \neq x' \Rightarrow f(x) \neq f(x')$. Tomando $x, x' \in X$, sendo $x \neq x'$, sem perda de generalidade, podemos assumir $x < x'$. Como f é estritamente crescente, pela Definição 4 teremos $f(x) < f(x') \Rightarrow f(x) \neq f(x')$. Portanto, toda função estritamente crescente é injetiva. (O processo é análogo para uma função estritamente decrescente)

1.5.3 Função Afim

Definição 7 *Uma função $f : \mathbb{R} \to \mathbb{R}$ chama-se **afim** quando existem constantes $a, b \in \mathbb{R}$, com $a \neq 0$, tais que $f(x) = ax + b$ para todo $x \in \mathbb{R}$.*

Exemplo 4 Considere que em um posto de combustível o valor por litro de gasolina é de $R\$5,46$ e a lavagem do carro é um valor fixo de $R\$35,00$. Seja x a quantidade de litros de gasolina, assim o valor a ser pago por abastecer x litros de gasolina é dada pela função $h(x) = 5,46x$ e o valor a ser pago por abastecer x litros e lavar o carro é dada pela função $g(x) = 5,46x + 35$.

Uma vez que x representa a quantidade de litros de gasolina, podemos restringir o domínio da função g para o intervalo $[0, x_{max}]$ em que x_{max} representa a quantidade em litros do tanque com a maior capacidade entre todos os veículos existentes.

Neste exemplo podemos tomar $D(g) = \{x \in \mathbb{R} \mid 0 \leq x \leq x_{max}\}$ e dessa forma, determinada pela observação do Teorema 3, $I_m(g) = \{y \in \mathbb{R} \mid 35 \leq y \leq g(x_{max})\}$ que representa o conjunto dos preços pago pelo combustível abastecido e pela lavagem.

Note ainda que o domínio da função h é igual ao domínio da função g, porém a imagem de h é dada pelo intervalo $[0, h(x_{max})]$ que representa o conjunto dos preços pago pelo combustível abastecido.

Uma característica da função afim, $f : \mathbb{R} \to \mathbb{R}$, é sua bijetividade, inclusive ela é uma função polinomial de grau 1. Considerando a função $f(x) = ax + b$ e tomando dois valores $f(x_1) = f(x_2)$, por contrapositiva teremos,

$$\begin{aligned} & f(x_1) = f(x_2) \\ \Rightarrow\ & ax_1 + b = ax_2 + b \\ \Rightarrow\ & ax_1 = ax_2 \\ \Rightarrow\ & x_1 = x_2 \end{aligned}$$

Logo, a função afim é injetiva. Ainda, considerando um valor $c \in \mathbb{R}$ tal que

$f(x) = c$ teremos,

$$\begin{aligned} & f(x) = c \\ \Rightarrow\ & ax + b = c \\ \Rightarrow\ & ax = c - b \\ \Rightarrow\ & x = \frac{c-b}{a} \end{aligned}$$

Logo, o ponto $x = \dfrac{c-b}{a}$ terá uma imagem c, mostrando que a função é sobrejetiva. Portando, tem-se que a função afim é bijetiva.

1.5.4 Função Quadrática

Definição 8 *Um função $f : \mathbb{R} \to \mathbb{R}$ chama-se **quadrática** quando existem números reais a, b, c, com $a \neq 0$, tais que $f(x) = ax^2 + bx + c$ para todo $x \in \mathbb{R}$.*

Exemplo 5 Quer-se encontrar a maior área retangular cercada por arames, conhecendo seu perímetro e mantendo-o fixo. Considerando w e l, respectivamente, como largura e comprimento da região, e perímetro fixo igual a 20, tem-se a equação $2w+2l = 20$. Como o objetivo é encontrar a maior área, isola-se a variável l na equação do perímetro e substitui na função da área, $A = wl$, resultando em, $A = w(10 - w) \Rightarrow A = -w^2 + 10w$. Essa é a função quadrática (polinomial de grau 2) que modela essa situação, com $a = -1, b = 10$ e $c = 0$. Para solucionar esse exemplo, é necessário encontrar o maior valor que essa função assume.

Quando se trata de calcular os valores de uma equação ou função quadrática, pode-se utilizar a forma canônica do trinômio $y = ax^2 + bx + c$.

$$ax^2 + bx + c = a\left[x^2 + \frac{b}{a}x + \frac{c}{a}\right] = a\left[x^2 + \frac{b}{a}x + \frac{b^2}{4a^2} - \frac{b^2}{4a^2} + \frac{c}{a}\right] =$$

$$= a\left[\left(x^2 + \frac{b}{a}x + \frac{b^2}{4a^2}\right) - \left(\frac{b^2}{4a^2} - \frac{c}{a}\right)\right] = a\left[\left(x + \frac{b}{2a}\right)^2 - \frac{b^2 - 4ac}{4a^2}\right]$$

Representando $b^2 - 4ac$ por Δ, temos a forma canônica do trinômio.

$$y = a\left[\left(x + \frac{b}{2a}\right)^2 - \frac{\Delta}{4a^2}\right] \tag{1.2}$$

Se $a < 0$, o valor de y será tanto maior quanto menor for o valor da diferença

$$\left(x + \frac{b}{2a}\right)^2 - \frac{\Delta}{4a^2}.$$

Nessa diferença, $-\frac{\Delta}{4a^2}$ é constante, porque não depende de x, e $\left(x + \frac{b}{2a}\right)^2 \geq 0$ para todo x real. Então a diferença assume o menor valor possível quando $\left(x + \frac{b}{2a}\right)^2 = 0$, ou seja, quando $x = -\frac{b}{2a}$.

Para $x = -\frac{b}{2a}$, temos na forma canônica:

$$y = a\left[\left(-\frac{b}{2a} + \frac{b}{2a}\right)^2 - \frac{\Delta}{4a^2}\right] = a\left[0^2 - \frac{\Delta}{4a^2}\right] = -\frac{\Delta}{4a}.$$

Logo, para $a < 0$, a função tem um valor **máximo**. Para $a > 0$, os passos são análogos e resulta em uma função com um valor **mínimo**.

Agora, sabendo como encontrar máximo ou mínimo de uma função quadrática, pode-se resolver o Exemplo 4. Faz-se:

$$w = -\frac{b}{2a} \quad \Rightarrow \quad w = -\frac{10}{2.(-1)} \quad \Rightarrow \quad w = 5$$

Portanto, a função que modela a área atingirá o seu valor máximo quando $w = 5$. Quanto a variável l, o valor também terá que ser 5, obtendo área máxima de 25 unidade de área. Consequentemente, tem-se que a imagem dessa função é $I_m(A) = \{y \in \mathbb{R} \mid 0 < y \leq 25\}$.

Outros valores que podem ser obtidos na função quadrática são os pontos onde o valor da função é igual a zero. Esses pontos são conhecidos como **raízes da função**.

Tomando um retângulo qualquer, é possível determinar quais são seus lados conhecendo seu semi-perímetro s e sua área p.

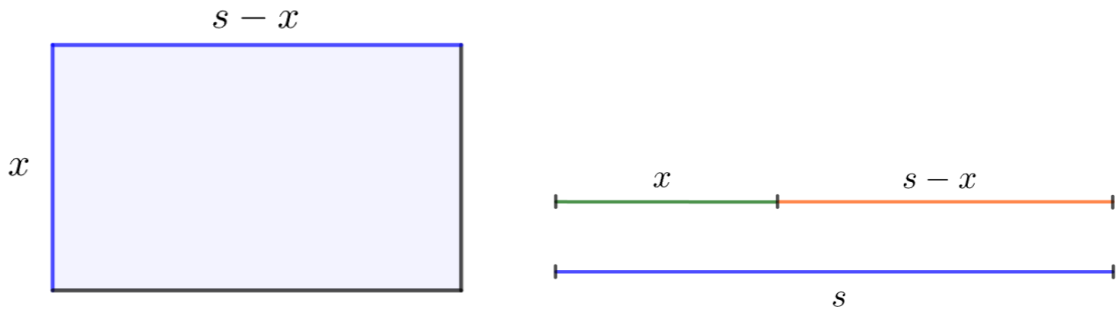


Figura 1.20: Retângulo de lados x e $s - x$

Se um dos lados é x, o outro é $s - x$. Como sua área é conhecida, pode-se formar uma equação com somente uma incógnita.

$$p = x(s - x) = sx - x^2 \qquad \Rightarrow \qquad x^2 - sx + p = 0 \tag{1.3}$$

Achar as raízes (ou zeros) da equação (1.3), significa encontrar os valores de x reais tais que o resultado da equação seja igual a 0. Além disso, encontrar as raízes, nesse contexto, significa encontrar os valores dos lados do retângulo.

Novamente pelo trinômio $ax^2 + bx + c = 0$, sendo $a \neq 0$, temos as seguintes equivalências:

$$\begin{aligned} ax^2 + bx + c = 0 &\Leftrightarrow \left(x + \frac{b}{2a}\right)^2 - \frac{\Delta}{4a^2} = 0 \\ &\Leftrightarrow \left(x + \frac{b}{2a}\right)^2 = \frac{\Delta}{4a^2} \qquad (1.4) \\ &\Leftrightarrow x + \frac{b}{2a} = \pm\frac{\sqrt{\Delta}}{2a} \qquad (1.5) \\ &\Leftrightarrow x = \frac{-b \pm \sqrt{\Delta}}{2a} \end{aligned}$$

A passagem da linha (1.4) para a (1.5) só tem sentido quando $\Delta \geq 0$. Caso $\Delta < 0$ a equação dada não possui solução real.

Portanto, na equação (1.3), encontram-se as raízes quando $x = \dfrac{s \pm \sqrt{s^2 - 4p}}{2}$.

Diferente da função afim, a função quadrática não apresenta bijetividade. Considerando a função $f(x) = ax^2 + bx + c$ e tomando pontos diferentes em x e $r \neq 0$, verifica-se

a seguinte igualdade $f\left(-\frac{b}{2a}+r\right)=f\left(-\frac{b}{2a}-r\right)$, por contrapositiva teremos,

$$f\left(-\frac{b}{2a}+r\right)=f\left(-\frac{b}{2a}-r\right)$$

$$a\left(-\frac{b}{2a}+r\right)^2+b\left(-\frac{b}{2a}+r\right)+c=a\left(-\frac{b}{2a}-r\right)^2+b\left(-\frac{b}{2a}-r\right)+c$$

$$a\left(\frac{b^2}{4a^2}-\frac{br}{a}+r^2\right)-\frac{b^2}{2a}+br=a\left(\frac{b^2}{4a^2}+\frac{br}{a}+r^2\right)-\frac{b^2}{2a}-br$$

$$\frac{b^2}{4a}-br+ar^2+br=\frac{b^2}{4a}+br+ar^2-br$$

$$ar^2=ar^2$$

Logo, como dois pontos diferentes geraram uma mesma imagem, a função quadrática não é injetiva. A respeito de sua sobrejetividade, suponha que existe um $x\in\mathbb{R}$ tal que, $f(x)=-\frac{b^2-4ac}{4a}-1$, com $a>0$. Teremos,

$$f(x)=-\frac{b^2-4ac}{4a}-1$$

$$\Rightarrow \quad ax^2+bx+c=\frac{-b^2+4ac-4a}{4a}$$

$$\Rightarrow 4a^2x^2+4abx+4ac=-b^2+4ac-4a$$

$$\Rightarrow \quad (2ax+b)^2=-4a \tag{1.6}$$

A igualdade (1.6) não é válida, pois o quadrado de qualquer número real é sempre positivo, enquanto o lado direito da equação é um número negativo. Portanto, para $a>0$, a equação resulta em uma contradição. Analogamente, para $a<0$, toma-se o ponto $f(x)=-\frac{b^2-4ac}{4a}+1$. Desenvolvendo de forma similar, chega-se novamente a uma contradição. Logo, conclui-se que a função quadrática não é, também, sobrejetiva.

Apesar da função quadrática não ser bijetiva, com domínio e imagem nos reais, ela pode se tornar injetiva ao restringirmos seu domínio.

Para $a>0$, consideremos os pontos $x_1=-\frac{b}{2a}+\delta$ e $x_2=-\frac{b}{2a}+\epsilon$, com $\delta,\epsilon>0$

e $\delta < \epsilon$. Tendo, $x_1 < x_2$, queremos mostrar que $f(x_1) < f(x_2)$. De (1.2), façamos:

$$f(x_1) = a\left[\left(-\frac{b}{2a} + \delta + \frac{b}{2a}\right)^2 - \frac{\Delta}{4a^2}\right] \quad \Rightarrow \quad f(x_1) = a\left[\delta^2 - \frac{\Delta}{4a^2}\right]$$

$$f(x_2) = a\left[\left(-\frac{b}{2a} + \epsilon + \frac{b}{2a}\right)^2 - \frac{\Delta}{4a^2}\right] \quad \Rightarrow \quad f(x_2) = a\left[\epsilon^2 - \frac{\Delta}{4a^2}\right]$$

Uma vez que,

$$\begin{aligned} & \delta < \epsilon \\ \Rightarrow \quad & \delta^2 < \epsilon^2 \\ \Rightarrow \quad & \delta^2 - \frac{\Delta}{4a^2} < \epsilon^2 - \frac{\Delta}{4a^2} \\ \Rightarrow \quad & a\left[\delta^2 - \frac{\Delta}{4a^2}\right] < a\left[\epsilon^2 - \frac{\Delta}{4a^2}\right] \\ \Rightarrow \quad & f(x_1) < f(x_2) \end{aligned}$$

Obtemos que $f : \left[-\frac{b}{2a}, +\infty\right) \to \mathbb{R}$ é estritamente crescente, portanto, por consequência da Proposição 2, f é injetiva no intervalo $\left[-\frac{b}{2a}, +\infty\right)$.

Tomando, ainda, $a > 0$ e considerando os pontos $x_1 = -\frac{b}{2a} - \epsilon$ e $x_2 = -\frac{b}{2a} - \delta$, com $\delta, \epsilon > 0$ e $\delta < \epsilon$. Tendo, $x_1 < x_2$, queremos mostrar que $f(x_1) > f(x_2)$.

$$f(x_1) = a\left[\left(-\frac{b}{2a} - \epsilon + \frac{b}{2a}\right)^2 - \frac{\Delta}{4a^2}\right] \quad \Rightarrow \quad f(x_1) = a\left[(-\epsilon)^2 - \frac{\Delta}{4a^2}\right]$$

$$f(x_2) = a\left[\left(-\frac{b}{2a} - \delta + \frac{b}{2a}\right)^2 - \frac{\Delta}{4a^2}\right] \quad \Rightarrow \quad f(x_2) = a\left[(-\delta)^2 - \frac{\Delta}{4a^2}\right]$$

Uma vez que,

$$\begin{aligned}
& \delta < \epsilon \\
\Rightarrow \quad & \delta^2 < \epsilon^2 \\
\Rightarrow \quad & (-\delta)^2 < (-\epsilon)^2 \\
\Rightarrow \quad & (-\delta)^2 - \frac{\Delta}{4a^2} < (-\epsilon)^2 - \frac{\Delta}{4a^2} \\
\Rightarrow \quad & a\left[(-\delta)^2 - \frac{\Delta}{4a^2}\right] < a\left[(-\epsilon)^2 - \frac{\Delta}{4a^2}\right] \\
\Rightarrow \quad & f(x_2) < f(x_1)
\end{aligned}$$

Verificamos que $f : \left(-\infty, -\frac{b}{2a}\right] \to \mathbb{R}$ é estritamente decrescente, portanto, f é injetiva no intervalo $\left(-\infty, -\frac{b}{2a}\right]$.

Assim, podemos concluir que a restrição do domínio da função quadrática para o intervalo $\left(-\infty, -\frac{b}{2a}\right]^{(*)}$ ou para o intervalo $\left[-\frac{b}{2a}, +\infty\right)^{(**)}$, torna a função injetiva. De forma análoga, se $a < 0$, sob as restrições $(*)$ e $(**)$, a função quadrática torna-se injetiva.

Exemplo 6 A função $f(x) = x^2 + x - 1$ não é injetiva, entretanto aplicando uma restrição $f|X_1 : \left(-\infty, -\frac{b}{2a}\right] \to \mathbb{R}$, ou $f|X_2 : \left[-\frac{b}{2a}, +\infty\right) \to \mathbb{R}$, ela passa a ser injetiva naquele intervalo.

1.5.5 Função Inversa

Definição 9 *Diz-se que a função $g : Y \to X$ é a **inversa** da função $f : X \to Y$ quando se tem $g(f(x)) = x$ e $f(g(y)) = y$ para quaisquer $x \in X$ e $y \in Y$. Evidentemente, g é inversa de f se, e somente se, f é inversa de g.*

Quando g é a inversa de f, tem-se $g(y) = x$ se, e somente se, $f(x) = y$. Se $g(f(x)) = x$ para todo $x \in X$ então a função f é injetiva, pois

$$f(x_1) = f(x_2) \quad \Rightarrow \quad g(f(x_1)) = g(f(x_2)) \quad \Rightarrow \quad x_1 = x_2$$

Por sua vez, a igualdade $f(g(y)) = y$, valendo para todo $y \in Y$, implica que f é sobrejetiva pois, dado $y \in Y$ arbitrário, tomamos $x = g(y) \in X$ e temos $f(x) = y$.

Portanto, se a função $f : X \Rightarrow Y$ possui inversa então f é injetiva e sobrejetiva, ou seja, é uma correspondência biunívoca entre X e Y.

Reciprocamente, se $f : X \Rightarrow Y$ é uma correspondência biunívoca entre X e Y então f possui uma inversa $g : Y \Rightarrow X$. Para definir g notamos que, sendo f sobrejetiva, para todo $y \in Y$ existe algum $x \in X$ tal que $f(x) = y$. Além disso, como f é injetiva, este x é único. Pomos então $g(y) = x$. Assim, $g : Y \Rightarrow X$ é a função que associa a cada $y \in Y$ o único $x \in X$ tal que $f(x) = y$. É imediato que $g(f(x)) = x$ e $f(g(y)) = y$ para $x \in X$ e $y \in Y$ quaisquer.

Exemplo 7 As conversões de temperaturas podem ser feitas por meio de algumas funções. Dado a função que converte graus Celsius para graus Fahrenheit $f(C) = \frac{9}{5}C + 32$, sendo f bijetiva, podemos fazer a conversão inversa utilizando a função $c(F) = \frac{5}{9}(F - 32)$, pois

$$f(c(F)) = \frac{9}{5}\left(\frac{5}{9}(F - 32)\right) + 32 = F$$

$$c(f(C)) = \frac{5}{9}\left(\left(\frac{9}{5}C + 32\right) - 32\right) = C$$

Capítulo 2

Metodologia

Este capítulo explora o uso de funções polinomiais como chaves criptográficas para codificação e decodificação de mensagens. A abordagem proposta envolve a definição de uma chave criptográfica que é uma sequência de termos representando os coeficientes de uma função polinomial de grau n. A tabela de conversão utilizada associa valores de funções polinomiais a letras do alfabeto romano, criando um sistema de criptografia baseado em funções matemáticas.

2.1 Criptografia com Funções

Definiremos a chave criptográfica como uma sequência de $n+2$ termos, para todo $n \geq 1$, de tal forma que os primeiros $n+1$ números representam os coeficientes da função polinomial de grau n, e o último dígito m representa a quantidade de caracteres de cada letra a ser decodificada (modelo na Tabela 2.1). A Tabela 2.2, apresenta a função injetiva que associa cada $p(x_i)$ a um caractere, em que $x_i, a_i \in \mathbb{Z}$ e $i \in \mathbb{N}$.

Tabela 2.1: Estrutura de composição da chave criptográfica

Chave	Função
$(a_1,\ a_0,\ m)$	$p(x) = a_1x + a_0$
$(a_2,\ a_1,\ a_0,\ m)$	$p(x) = a_2x^2 + a_1x + a_0$
$(a_3,\ a_2,\ a_1,\ a_0,\ m)$	$p(x) = a_3x^3 + a_2x^2 + a_1x + a_0$
$\vdots$	$\vdots$
$(a_n,\ a_{n-1}, ...,\ a_1,\ a_0,\ m)$	$p(x) = a_nx^n + a_{n-1}x^{n-1} + ... + a_1x + a_0$

Fonte: PONTES et al. (2022).

Tabela 2.2: Base de uma Tabela de Cifras

$p(x)$	Caractere	$p(x)$	Caractere
$p(x_1)$	A	$p(x_{10})$	J
$p(x_2)$	B	$p(x_{11})$	K
$p(x_3)$	C	$p(x_{12})$	L
$p(x_4)$	D	$p(x_{13})$	M
$p(x_5)$	E	$p(x_{14})$	N
$p(x_6)$	F	$p(x_{15})$	O
$p(x_7)$	G	$p(x_{16})$	P
$p(x_8)$	H	$\vdots$	$\vdots$
$p(x_9)$	I	$p(x_r)$	

Fonte: Próprio autor.

Com base na regra definida para a chave e da tabela de conversão, podemos utilizar o algoritmo abaixo para codificar e decodificar uma mensagem:

1. Defina a função polinomial de grau n, $p(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$, injetiva no intervalo $[x_1, x_r]$ para codificar cada caractere da mensagem;

2. Determine o parâmetro m. O valor de m deve ser o valor máximo de caracteres de $p(x)$ no intervalo $[x_1, x_r]$. Dessa forma, considere c_1 a quantidade de caracteres de $p(x_1)$ e c_r a quantidade de caracteres de $p(x_r)$, então o parâmetro m é definido por

$$m = max\{c_1, c_r\}$$

3. Escolha a mensagem para ser codificada. Suponhamos a frase "SEJA BEM-VINDO";

4. Calcule as cifras da mensagem. Para a mensagem "SEJA BEM-VINDO", calcule $p(x_{19}), p(x_5), ..., p(x_{15})$ e suponha que,

$$\begin{aligned} p(x_{19}) &= q_1 \\ p(x_5) &= q_2 \\ \vdots \quad &\quad \vdots \\ p(x_{15}) &= q_{14} \end{aligned}$$

em que, a quantidade de caracteres de q_j é igual a m, para $j = 1, ..., 14$, por exemplo, se $m = 2$ e $p(x_{19}) = 8$, então $q_1 = 08$.

Assim o vetor $[q_1 q_2 ... q_{14}]$ é a mensagem codificada;

5. Decodifique a mensagem. Este processo é realizado resolvendo as equações

$$p(x) = q_j, \quad \text{para} \quad 1 \leq j \leq 14$$

em que $x_1 \leq x \leq x_r$.

A respeito do segundo passo do algoritmo descrito acima, calcular o parâmetro m é necessário para o momento de decodificação. Quando obtemos uma mensagem já codificada, precisamos saber quantos algarismos serão selecionados por vez para decodifica-la, ou seja, identificar quantos algarismos representam um caractere da mensagem.

Durante o cálculo do m, utilizamos a imagem da função polinomial aplicada somente no menor (x_1) e no maior (x_r) ponto do domínio da função. Uma vez que a função polinomial é contínua e injetiva no intervalo $[x_1, x_r]$, segue que a função polinomial é estritamente crescente ou estritamente decrescente (conforme o Teorema 2). Portanto, pelo Teorema 3, o valor máximo e mínimo da função está nos valores extremos do intervalo.

As seções seguintes ilustram o algoritmo utilizando função afim e quadrática.

2.2 Criptografia com Função Afim

Nesta seção apresentamos o processo de criptografia de mensagens utilizando função afim e utilizamos a Tabela 2.3 como referência da regra que associa a imagem da função com os caracteres.

Tabela 2.3: Tabela de Cifras

$p(x)$	Caractere	$p(x)$	Caractere	$p(x)$	Caractere
$p(1)$	A	$p(11)$	K	$p(21)$	U
$p(2)$	B	$p(12)$	L	$p(22)$	V
$p(3)$	C	$p(13)$	M	$p(23)$	X
$p(4)$	D	$p(14)$	N	$p(24)$	W
$p(5)$	E	$p(15)$	O	$p(25)$	Y
$p(6)$	F	$p(16)$	P	$p(26)$	Z
$p(7)$	G	$p(17)$	Q	$p(27)$	-
$p(8)$	H	$p(18)$	R	$p(28)$	
$p(9)$	I	$p(19)$	S	$p(29)$	Õ
$p(10)$	J	$p(20)$	T	$p(30)$	Ç

Fonte: Próprio autor.

Com base na tabela de conversão, realizamos os passos do algoritmo descritos na seção 2.1:

1. A função injetiva $p(x) = x - 2$ foi definida para compor a chave;

2. Determinando o valor do parâmetro m. Temos que $p(1) = -1$ e $p(30) = 28$, uma vez que $c_1 = c_2 = 2$, temos que $m = max\{c_1, c_2\} = 2$. Desta forma, a chave criptográfica é formada por $(1, -2, 2)$;

3. Mensagem escolhida: "SEJA BEM-VINDO";

4. Codificando a mensagem calculando os valores de cada cifra, como segue, $p(19) = 17$, $p(5) = 03$, $p(10) = 08$, $p(1) = -1$, $p(28) = 26$, $p(2) = 00$, $p(5) = 03$, $p(13) = 11$, $p(27) = 25$, $p(22) = 20$, $p(9) = 07$, $p(14) = 12$, $p(4) = 02$ e $p(15) = 13$, assim, a mensagem codificada é dada pelo vetor [170308-126000311252007120213]

Com a mensagem codificada, realizamos o passo 5 da seguinte forma:

- Sabendo que $m = 2$, identificamos as duplas de algarismos que representam uma letra da mensagem, [17, 03, 08, -1, 26, 00, 03, 11, 25, 20, 07, 12, 02, 13];

- Utilizamos a função, conhecida por meio das informações trazidas pela chave de criptografia, $p(x) = x - 2$ para fazermos os cálculos:

1^{o} Letra: $17 \Rightarrow\ x-2=17\ \Rightarrow\ x-2-17=0\ \Rightarrow\ x-19=0 \Rightarrow\ x=19$.

Logo, 19 é correspondente da letra S.

2^{o} Letra: $03 \Rightarrow\ x-2=3\ \Rightarrow\ x-2-3=0\ \Rightarrow\ x-5=0\ \Rightarrow\ x=5$.

Logo, 5 é correspondente da letra E.

[...]

4^{o} Letra: -1 $\Rightarrow\ x-2=-1\ \Rightarrow\ x-2+1=0\ \Rightarrow\ x-1=0\ \Rightarrow\ x=1$.

Logo, 1 é correspondente da letra A.

[...]

14^{o} Letra: $13 \Rightarrow\ x-2=13\ \Rightarrow\ x-2-13=0\ \Rightarrow\ x-15=0\ \Rightarrow x=15$.

Logo, 15 é correspondente da letra O.

- Após todos os cálculos, os quais rementem o processo de decodificação, é obtido a mensagem original "SEJA BEM-VINDO".

2.3 Criptografia com Função Quadrática

Nesta seção abordamos o processo de criptografia, como na seção anterior, utilizando função quadrática.

1. A função injetiva $p(x) = x^2 + x - 1$ foi definida para compor a chave;

2. Determinando o valor do parâmetro m. Temos que $p(1) = 1$ e $p(30) = 929$, uma vez que $c_1 = 1$ e $c_2 = 3$, temos que $m = max\{c_1, c_2\} = 3$. Dessa forma, a chave criptográfica é formada por $(1, 1, -1, 3)$;

3. Mensagem escolhida: "SEJA BEM-VINDO";

4. Codificando a mensagem calculando os valores de cada cifra, como segue, $p(19) = 379$, $p(5) = 029$, $p(10) = 109$, $p(1) = 001$, $p(28) = 811$, $p(2) = 005$, $p(5) = 029$, $p(13) = 181$, $p(27) = 755$, $p(22) = 505$, $p(9) = 089$, $p(14) = 209$, $p(4) = 019$, $p(15) = 239$, assim, a mensagem codificada é dada pelo vetor

$$[37902910900181100502918175550508920901923 9]$$

Com a mensagem codificada, realizamos o passo 5:

- Sabendo que $m = 3$, identificamos as triplas de algarismos que representam uma letra da mensagem, [379, 029, 109, 001, 811, 005, 029, 181, 755, 505, 089, 209, 019, 239];

- Utilizando a função, conhecida por meio das informações trazidas pela chave de criptografia, $p(x) = x^2 + x - 1$, fazer os cálculos:

  1º Letra: $379 \Rightarrow\ x^2 + x - 1 = 379\ \Rightarrow\ x^2 + x - 380 = 0\ \Rightarrow\ x^{'} = 19$ e $x^{''} = -20$.
  Logo, escolhemos o número 19 que é correspondente da letra S.

  2º Letra: $029 \Rightarrow\ x^2 + x - 1 = 29\ \Rightarrow\ x^2 + x - 30 = 0\ \Rightarrow\ x^{'} = 5$ e $x^{''} = -6$.
  Logo, escolhemos o número 5 que é correspondente da letra E.

  [...]

  8º Letra: $755 \Rightarrow\ x^2 + x - 1 = 755\ \Rightarrow\ x^2 + x - 756 = 0\ \Rightarrow x^{'} = 27$ e $x^{''} = -28$.
  Logo, escolhemos o número 27 que é correspondente ao símbolo $-$.

  [...]

  14º Letra: $239 \Rightarrow\ x^2 + x - 1 = 239\ \Rightarrow\ x^2 + x - 240 = 0\ \Rightarrow x^{'} = 15$ e $x^{''} = -16$.
  Logo, escolhemos o número 15 que é correspondente da letra O.

- Após os cálculos, é obtido a mensagem decodificada: "SEJA BEM-VINDO".

Observe que para cada caractere decodificado há dois possíveis valores para x, que são as raízes da equação do segundo grau, entretanto, há somente um valor de x que pertence ao intervalo $[1, 30]$, onde a função quadrática é injetora. Portanto, não há dubiedade na decodificação, tornando sempre possível a obtenção da mensagem original.

2.4 Considerações

Primeira observação: A escolha das cifras

A escolha das cifras descritas na Tabela 2.2 pode ser feita de maneira arbitrária, adicionando ou retirando quantas cifras desejar, diferenciando letras maiúsculas de minúsculas, inserindo espaço em branco, considerando acentuação, entre outras possibilidades. Além disso, definimos que $x_i \in \mathbb{Z}$ com o objetivo de facilitar o cálculo do processo de codificação e decodificação da mensagem, entretanto, é possível estender o domínio da função para o conjunto dos números racionais.

Segunda observação: Processo de decodificação

Há duas formas de se obter a mensagem original pelo processo de decodificação que são pela extração de raízes ou pela função inversa.

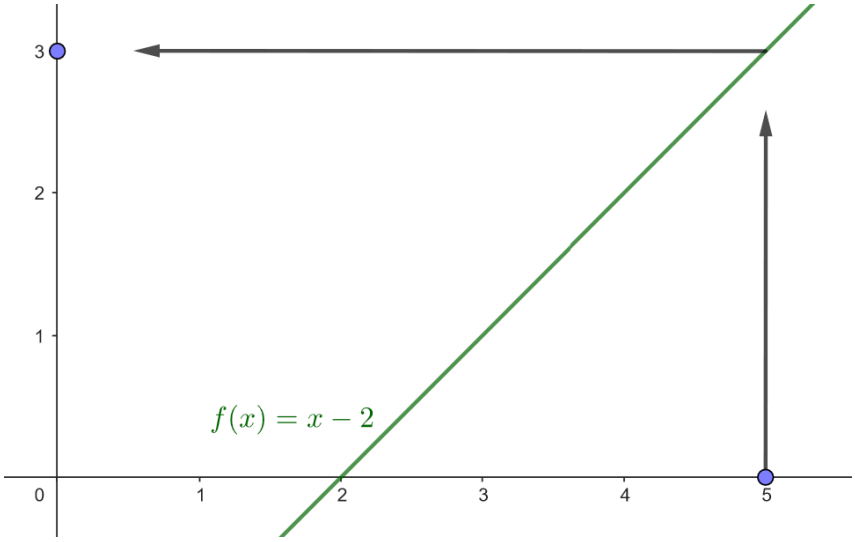
Para a decodificação de cada caractere, resolvemos a equação $p(x) - q_j = 0$ para cada $j \in J$ em que $J = \{1, 2, ..., l\}$ e l é a quantidade de caracteres da mensagem.

Considerando $g_j(x) = p(x) - q_j$, a decodificação de cada caractere é a obtenção da raiz da função g_j em que $x \in [x_1, x_r]$. Neste sentido, o processo de decodificação, pode ser relacionado com a extração de raízes da função que são transladadas da função utilizada na chave criptográfica.

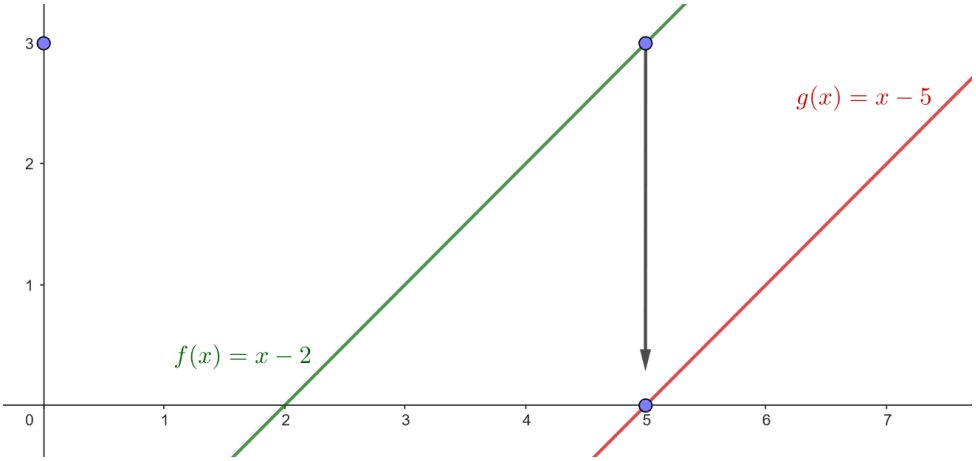
No exemplo da seção 2.2, na decodificação do segundo caractere temos $q_2 = 03$ e da equação

$$x - 2 = 3 \quad \Rightarrow \quad x - 5 = 0$$

obtemos a função $g_2(x) = x - 5$, desta forma a decodificação do segundo caractere representa a obtenção da raiz da função g_2. Este procedimento é ilustrado visualizando o gráfico dessas funções.



(a) Codificação



(b) Decodificação

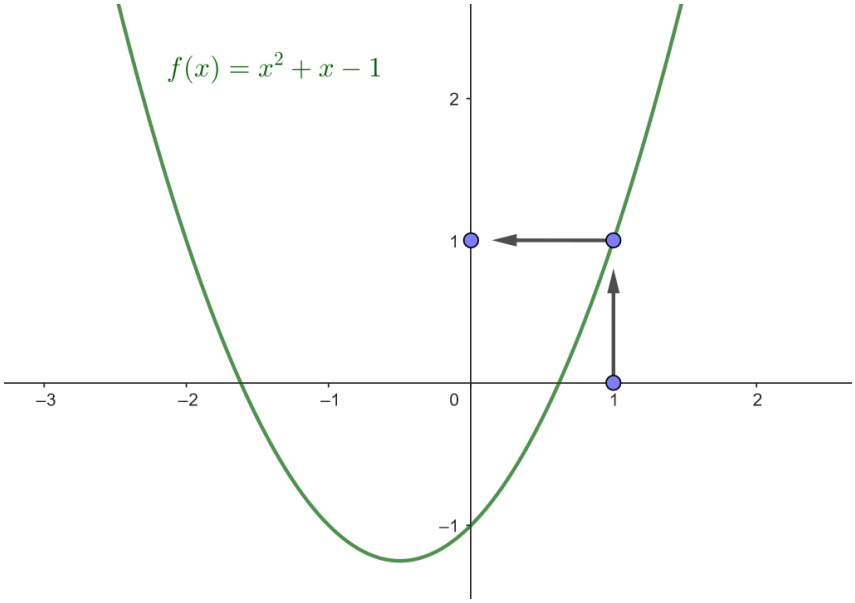
Figura 2.1: Translação da função afim

Como podemos observar na Figura 2.1, ocorre uma translação vertical de forma que o valor que procuramos passa a ser a raiz da função transladada e cada cifra gera uma função transladada em relação à função da chave criptográfica.

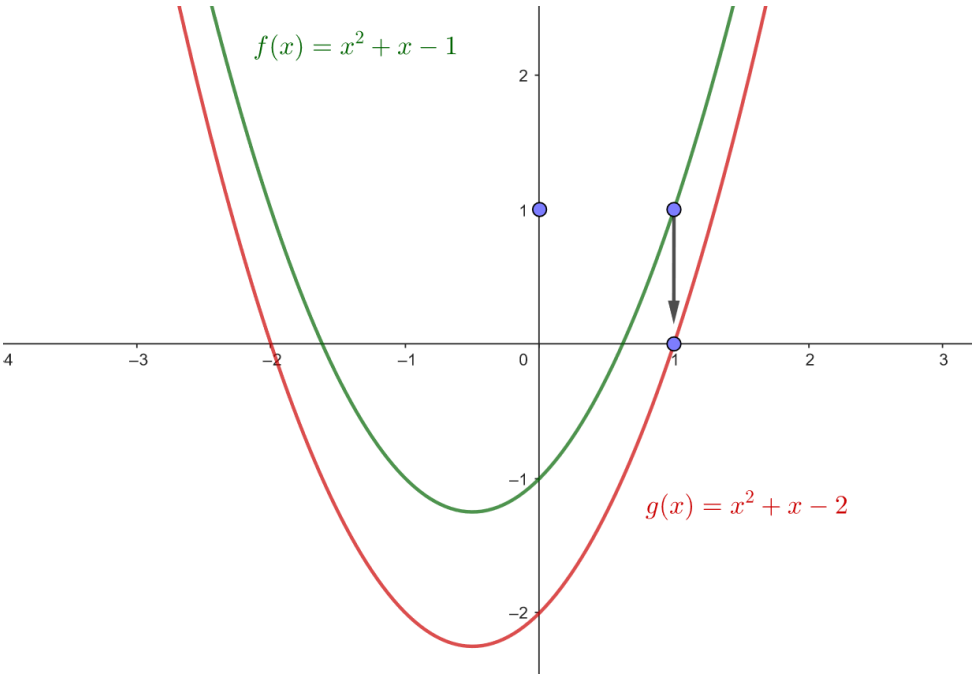
De forma semelhante, no exemplo da seção 2.3, na decodificação do quarto caractere temos que $q_4 = 001$ e da equação

$$x^2 + x - 1 = 1 \quad \Rightarrow \quad x^2 + x - 2 = 0$$

obtemos a função $g_4 = x^2 + x - 2$, desta forma a decodificação do quarto caractere representa a raiz da função g_4 que pertence ao intervalo $[1, 30]$. Vejamos a representação gráfica nas Figuras 2.2(a) e 2.2(b).



(a) Codificação



(b) Decodificação

Figura 2.2: Translação da função quadrática

Outra maneira de fazer a decodificação da mensagem é explicitar a função inversa da função da chave criptográfica.

No caso da chave criptográfica da seção 2.2, temos que $f(x) = x - 2$, então a função $g(y) = y + 2$ (Figura 2.3) é a função inversa da função f. De fato,

$$f(g(y)) = f(y + 2) = y + 2 - 2 = y$$
$$g(f(x)) = g(x - 2) = x - 2 + 2 = x$$

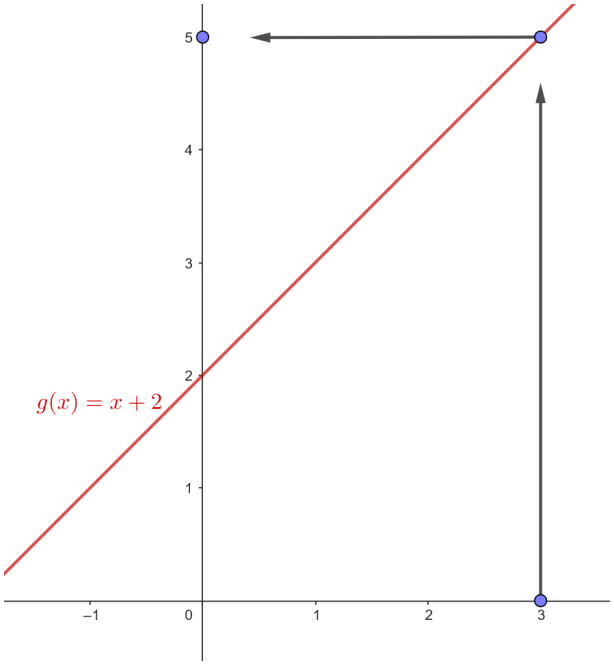


Figura 2.3: Inversa da função $f(x) = x - 2$

Assim, para a decodificação do segundo caractere $q_2 = 03$, basta calcular

$$g(3) = 3 + 2 = 5$$

que corresponde a letra E.

De forma análoga, é possível obter a função inversa da função $f : [1, 30] \to [1, 929]$ em que $f(x) = x^2 + x - 1$, como no exemplo da seção 2.3.

Temos que $g : [1, 929] \to [1, 30]$ dada por $g(y) = -\frac{1}{2} + \frac{1}{2}\sqrt{4y + 5}$ é a função

inversa de f. De fato,

$$\begin{aligned} f(g(y)) &= f\left(-\frac{1}{2}+\frac{1}{2}\sqrt{4y+5}\right) \\ &= \left(-\frac{1}{2}+\frac{1}{2}\sqrt{4y+5}\right)^2 - \frac{1}{2}+\frac{1}{2}\sqrt{4y+5} - 1 \\ &= \frac{1}{4} - \frac{1}{2}\sqrt{4y+5} + \frac{1}{4}(4y+5) - \frac{1}{2} + \frac{1}{2}\sqrt{4y+5} - 1 \\ &= \frac{1}{4} - \frac{3}{2} + y + \frac{5}{4} \\ &= \frac{1-6+5}{4} + y \\ &= y \end{aligned}$$

e

$$\begin{aligned} g(f(x)) &= g(x^2+x-1) \\ &= \left(-\frac{1}{2}+\frac{1}{2}\sqrt{4(x^2+x-1)+5}\right) \\ &= \left(-\frac{1}{2}+\frac{1}{2}\sqrt{4x^2+4x-4+5}\right) \\ &= \left(-\frac{1}{2}+\frac{1}{2}\sqrt{(2x+1)^2}\right) \\ &= \left(-\frac{1}{2}+\frac{1}{2}(2x+1)\right) \\ &= -\frac{1}{2}+x+\frac{1}{2} \\ &= x \end{aligned}$$

Desta forma, a decodificação do quarto caractere $q_4 = 001$ pode ser obtido calculando

$$g(1) = -\frac{1}{2}+\frac{1}{2}\sqrt{4\cdot 1+5} = -\frac{1}{2}+\frac{1}{2}\cdot 3 = 1$$

que corresponde a letra A.

Obtendo a inversa da função f

A inversa da função quadrática foi obtida partindo da equação (1.2).

$$y = a\left[\left(x + \frac{b}{2a}\right)^2 - \frac{b^2 - 4ac}{4a^2}\right]$$

$$\frac{y}{a} + \frac{b^2 - 4ac}{4a^2} = \left(x + \frac{b}{2a}\right)^2$$

$$\pm\sqrt{\frac{y}{a} + \frac{b^2 - 4ac}{4a^2}} = x + \frac{b}{2a}$$

$$x = -\frac{b}{2a} \pm \sqrt{\frac{y}{a} + \frac{b^2 - 4ac}{4a^2}}$$

Para a função $y = x^2 + x - 1$ em que $x \in [1, 30]$, tem-se:

$$x = -\frac{1}{2} + \sqrt{\frac{y}{1} + \frac{1 - 4(-1)}{4}}$$

$$x = -\frac{1}{2} + \sqrt{\frac{4y + 5}{4}}$$

$$x = -\frac{1}{2} + \frac{1}{2}\sqrt{4y + 5}$$

Terceira observação: Injetividade da função

Considere a Tabela 2.3 e a função $f(x) = x^2 - 10x + 2$, temos que $x_v = \dfrac{10}{2} = 5$, logo $f(4) = f(6)$, $f(3) = f(7)$, $f(2) = f(8)$ e $f(1) = f(9)$, e portanto para qualquer mensagem que contenha as letras A, B, C, D, E, F, G, H e I, terá dubiedade na decodificação. Por exemplo, a decodificação da palavra CABIDE terá 2^6 combinações de agrupamentos das letras.

Este contexto justifica, de forma clara, a necessidade da função ser injetiva no intervalo de correspondência com as cifras.

Uma vez estabelecida a tabela de correspondência entre a f e os caracteres, o objetivo é sempre fazer a escolha correta da função para que sempre seja possível decodificar a mensagem original com segurança.

Desta forma, há condições suficientes que garantem a decodificação sem que haja dubiedade quando utilizando a função quadrática.

Tomando a Tabela 2.2 como referencia, a condição

$$x_v = -\frac{b}{2a} \leq x_1 \tag{2.1}$$

ou a condição

$$x_r \leq x_v = -\frac{b}{2a} \tag{2.2}$$

são suficientes para garantir que a função seja injetiva no intervalo de correspondência com as cifras.

Na subseção 1.5.4, mostramos que dada a função quadrática, $f(x) = ax^2 + bx + c$ e $X_1 = (-\infty, x_v]$ e $X_2 = [x_v, +\infty)$, as funções $f|X_1$ e $f|X_2$ são injetivas.

Assim, as condições $x_v \leq x_1$ ou $x_r \leq x_v$ são suficientes para garantir a injetividade da função no intervalo de interesse.

Note que para $f(x) = x^2 - 10x + 20$, nenhuma das condições é satisfeita, pois

$$x_v = 5 \geq x_1 = 1 \quad \text{e} \quad x_r = 30 \geq 5 = x_v$$

Note, ainda, que a função $f(x) = x^2 + x - 1$, utilizada no exemplo da seção 2.3, satisfaz a condição (2.1), como pode ser visto abaixo,

$$x_v = 1 \leq x_1 = 1$$

Logo, como a função $f(x) = x^2 + x - 1$, satisfaz uma das condições (2.1) ou (2.2), podemos utilizá-la no processo de criptografia com segurança.

Quarta observação: Coeficientes da função polinomial

Este bloco de observação, analisa lado teórico e não computacional do processo criptográfico. Dada uma função polinomial injetiva qualquer $p(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$, para utilização do processo criptográfico é razoável restringir a escolha dos coeficientes do polinômio a números racionais, isto é, $a_i \in \mathbb{Q}$, para $i = 0, 1, ..., n$, visto a impossibilidade de determinar m, caso os coeficientes sejam irracionais.

Vejamos um exemplo, suponha que $p(x) = \pi x + e$. Temos que $p(1) = \pi + e, p(2) = 2\pi + e, ..., p(n) = n\pi + e$, neste caso, não fica claro como descrever a codificação de qualquer mensagem e nem como definir o parâmetro m.

No caso em que os coeficientes são racionais, por exemplo, $f(x) = \frac{1}{2}x + 1$, então $f(1) = \frac{3}{2}, f(2) = 2, f(3) = \frac{5}{2}, ..., f(n) = \frac{n}{2} + 1 = \frac{n+2}{2}$, como $f(n) \in \mathbb{Q}$, podemos escrever qualquer valor da imagem como uma fração $f(n) = \frac{a}{b}$. Dessa forma, será possível descrever a codificação de uma mensagem e determinar o parâmetro m, que definiremos da seguinte forma: se a fração for exata, o número de caracteres c_i, será igual a quantidade de algarismos do número; se não for exata, o número de caracteres será igual a 1.

Fundamentado nos passos descritos na subseção 2.1, tomamos a Tabela 2.2 com $x_i \in \mathbb{Q}$ e $i \in \mathbb{N}$. Suponha uma função polinomial de grau n, $p(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$, injetiva no intervalo $[x_1, x_r]$. Considere o exemplo em que, $p(1) = -1$ e $p\left(\frac{41}{2}\right) = \frac{37}{2}$. Nessa situação tem-se $m = 2$, pois -1 tem 2 caracteres e $\frac{37}{2}$ tem 1. Logo, a mensagem codificada $[0801\frac{7}{2}12]$ é "lida"da forma $[08, 01, \frac{7}{2}, 12]$. Não utilizaremos, portanto, a escrita em decimal de uma fração racional não exata.

Concluímos que, além da possibilidade dos coeficientes da função serem racionais, podemos utilizar valores do domínio pertencentes ao conjunto dos racionais. Contudo, haverão alguns passos que não poderemos utilizar da mesma maneira como feito nas seções anteriores.

Capítulo 3

Automatizando o Processo Criptográfico

Os capítulos anteriores trouxeram os meios da utilização das funções polinomiais no processo criptográfico. Neste capítulo, apresenta-se uma maneira de automatizar esse processo.

3.1 O Algoritmo Base

De acordo com Lambert (2019, p. 4), um algoritmo consiste em um número finito de instruções bem definidas que descrevem, passo a passo, um processo a ser seguido para resolver um problema específico ou executar uma tarefa. Informalmente, é como uma receita de bolo. Os passos descrevem desde a quantidade de ovos que devem ser usados no início da receita, até o tempo que a massa deve ficar no forno antes de tirá-lo.

Analisando e colocando os passos que utilizamos para codificar e decodificar uma mensagem, pontuaremo-os em uma lista como um algoritmo:

- Escolha as cifras que serão utilizadas. Consequentemente, é definido o domínio da função;
- Defina uma função injetiva, afim ou quadrática, que servirá como chave do processo;
- Escolha a mensagem;
- Calcule os valores das imagens que formarão a mensagem codificada;

- Ao receber a mensagem codificada, identifique qual a função e o agrupamento para o cálculo;
- Calcule as raízes das funções;
- Use a tabela de cifras para decifrar a mensagem.

A lista acima é o algoritmo que utilizamos nos exemplos do capítulo 2. Faremos o mesmo, porém pensando nos passos de fazê-lo por meio da linguagem de programação Python.

- Adicione, ao código, a tabela de cifras;
- Solicite ao usuário os coeficientes da função e a mensagem;
- Separe cada caractere da mensagem;
- Realize o processo de codificação;
- Mostre a mensagem codificada na tela;
- Para decodificar: Solicite ao usuário a mensagem codificada;
- Realize o processo de decodificação;
- Mostre a mensagem decodificada na tela.

Após construirmos um passo a passo para orientação, é necessário detalhar como queremos realizar cada etapa. Nas próximas seções, apresentaremos cada etapa do algoritmo. Ressaltamos que não utilizaremos o cálculo do parâmetro m no código, pois incluí-lo aumentaria a complexidade. Neste momento, queremos simplificar o máximo possível para alcançar um nível mais didático.

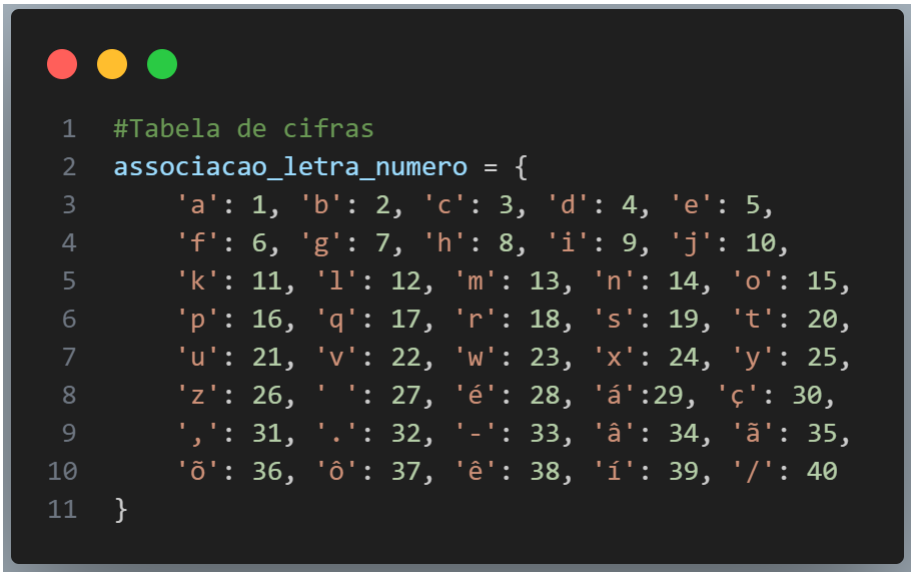
3.2 Codificando

3.2.1 Função Afim

Assim como na Tabela 2.3, definiremos quais são as cifras, ou seja, quais associações serão utilizadas durante o processo criptográfico. Para isso, criaremos uma estrutura que mapeie cada caractere escolhido a um número correspondente. Isso será útil

para converter os caracteres em números durante o processo de codificação e os números em caracteres no processo de decodificação.

A Figura 3.1 apresenta como fica a tabela de cifras em código Python, utilizando os conceitos de atribuição e dicionário vistos nas subseções 1.3.2.2 e 1.3.2.11. Os caracteres escolhidos para compor a tabela são as chaves do dicionário, enquanto os números associados a cada caractere são os valores.

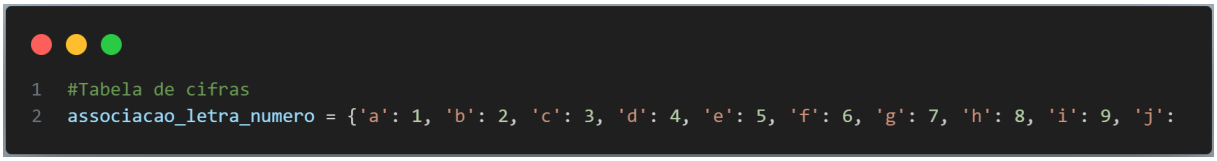


```
1  #Tabela de cifras
2  associacao_letra_numero = {
3      'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5,
4      'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10,
5      'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15,
6      'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20,
7      'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25,
8      'z': 26, ' ': 27, 'é': 28, 'á':29, 'ç': 30,
9      ',': 31, '.': 32, '-': 33, 'â': 34, 'ã': 35,
10     'õ': 36, 'ô': 37, 'ê': 38, 'í': 39, '/': 40
11 }
```

Figura 3.1: Associação caracteres - números

Na linha 1 da imagem acima, há o texto `#Tabela de cifras`. O computador não irá considerar essa linha como código, devido ao caractere especial "`#`"adicionado antes da frase. Quando colocamos esse símbolo dessa maneira, chamados a linha de **comentário**. Sua função é nomear ou orientar sobre os blocos subsequentes.

Nas linhas 2 a 11, construímos o bloco com as cifras. Escolhemos o nome `associacao_letra_numero` para o dicionário e optamos por utilizar 40 associações contendo letras, acentos e caracteres especiais. Lembrando que podemos escolher, também, letras maiúsculas, números e outros tipos de símbolos.



```
1  #Tabela de cifras
2  associacao_letra_numero = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j':
```

Figura 3.2: Opção base de escrita

A Figura 3.2, mostra uma forma de alocar os elementos no dicionário, porém

quando o dicionário tem uma quantidade grande de elementos, é mais claro para visualização que seja organizado como na Figura 3.1.

Em seguida, precisamos saber qual será a função afim e a mensagem a ser codificada. Então, solicitaremos a entrada no código dos coeficientes da função e, em sequência, da mensagem. Utilizaremos os conceitos apresentados nas subseções 1.3.2.1, 1.3.2.2 e 1.3.2.4.

Criaremos as seguintes variáveis: `texto`, que receberá a mensagem a ser codificada; e `a` e `b` que receberão os valores dos coeficientes da função.

```
1  #Entrada dos coeficientes da função
2  a = int(input('Digite o coeficiente angular da função: '))
3  b = int(input('Digite o coeficiente linear da função: '))
4
5  #Entrada do texto a ser codificado
6  texto = str(input('Digite o texto: ')).lower()
```

Figura 3.3: Entrada dos coeficientes da função afim e da mensagem

Repare o comando `.lower()` na Figura 3.3. Ele é uma ação que podemos adicionar em strings para que todo o texto fique com caracteres minúsculas. Utilizá-la também nos permite simplificar a tabela de cifras ao não necessitar adicionar letras maiúsculas.

Esses tipos de ações são chamadas de **método**, ou seja, a ação `.lower()` é um método de strings.

Outros métodos que podem ser utilizadas são `.upper()`, que converte todos os caracteres da string para maiúsculas, e `.capitalize()`, que converte o primeiro caractere da string para maíuscula e os demais para minúsculas.

Uma vez que o programa sabe quais são os coeficientes da função e a mensagem, seguimos ao próximo passo antes da codificação.

O computador precisa cifrar um caractere por vez, logo é necessário separar cada caractere da mensagem e adicioná-los em uma lista (visto na subseção 1.3.2.10). É como se estivéssemos soletrando a mensagem. Por exemplo, se recebermos a mensagem **bolacha > biscoito**, ao separar, aparecerá da seguinte forma na lista:

[‘b’, ‘o’, ‘l’, ‘a’, ‘c’, ‘h’, ‘a’, ‘ ’, ‘>’, ‘ ’, ‘b’, ‘i’, ‘s’, ‘c’, ‘o’, ‘i’, ‘t’, ‘o’]

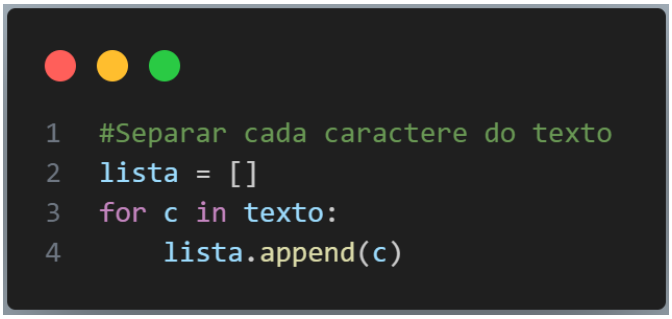


Figura 3.4: Separação dos caracteres da mensagem

Na subseção 1.3.2.9, vimos sobre as estruturas de repetição. Precisaremos desse conceito para adicionar caractere por caractere em uma lista. Na linha 2, da Figura 3.4, nomeamos a lista como `lista`. Nas linhas 3 e 4 a estrutura de repetição *for* pode ser lida da seguinte forma: Para cada caractere na mensagem, adicione-os na lista. Então, a letra `c` na figura representa cada caractere digitado na variável `texto`. Em seguida, esse caractere `c` é adicionado à `lista` por meio do método `.append()`.

Agora que já separamos a mensagem, podemos codificá-la.

Criamos um loop *for* que irá, a cada iteração[1], pegar um elemento de `lista`, realizar a associação necessária por meio da tabela de cifras, adicionar em uma variável, calcular a imagem da função e adicionar a imagem em uma lista.

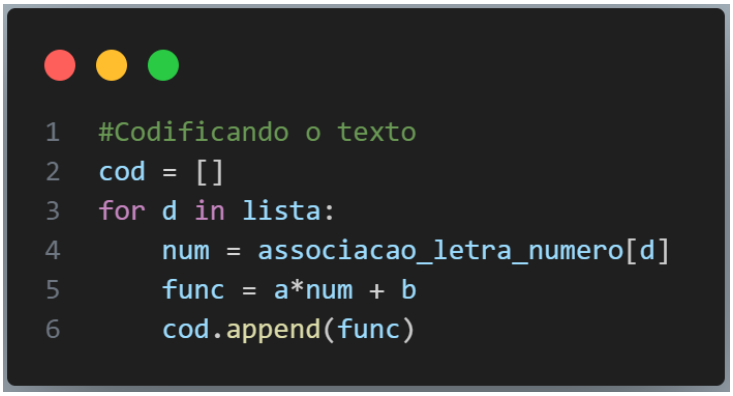


Figura 3.5: Bloco de codificação com função afim

Acompanhando pela Figura 3.5, iniciamos esse bloco criando uma nova lista chamada `cod`. Após, por meio da estrutura de repetição *for*, pegamos cada elemento de `lista` e calculamos a cifra. Na linha 4, adicionamos na variável `num` o elemento do dicionário associado ao caractere `d` da `lista` (vimos na Figura 1.19). Após, na linha 5, ciframos o valor `num` utilizando a expressão `a*num + b`, onde `a` e `b` foram definidos anteriormente,

[1]Iteração é a repetição de um processo ou uma sequência de instruções várias vezes

e atribuímos na variável `func`. O caractere cifrado é, então, adicionado na lista `cod` utilizando o método `.append()`.

Por fim, precisamos mostrar na tela como ficou a mensagem após a codificação. Usaremos aqui o conceito visto na seção 1.3.2.3.

Como na Figura 3.6, adicionamos o comando `print()` que mostrará no terminal, localizado no painel do VS Code (Figura 1.4), o que estiver entre parênteses. Nesse caso colocamos a lista `cod` entre parênteses, o que indica que será mostrado na tela o que estiver armazenado nessa lista.

```
1  #Saída do texto codificado
2  print(cod)
```

Figura 3.6: Saída da mensagem codificada

Com o código de codificação construído, podemos executá-lo clicando na opção Run Python File, localizada no canto superior direito da interface do VS Code (Figura 3.7).

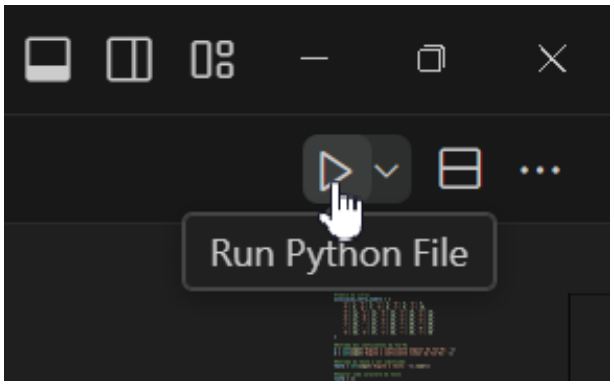


Figura 3.7: Executar o código

Ao executar, as entradas dos coeficiente da função e da mensagem são realizadas no terminal, assim como, a saída da mensagem codificada (Figura 3.8).

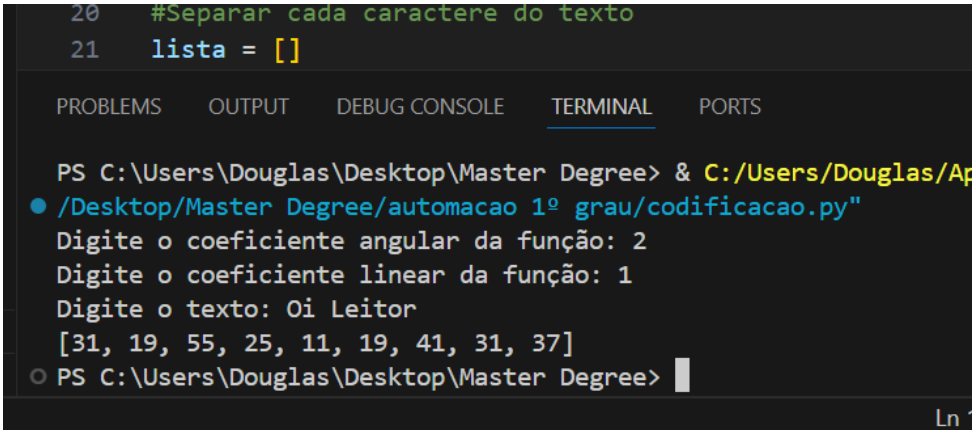


Figura 3.8: Entradas e saídas mostradas no terminal

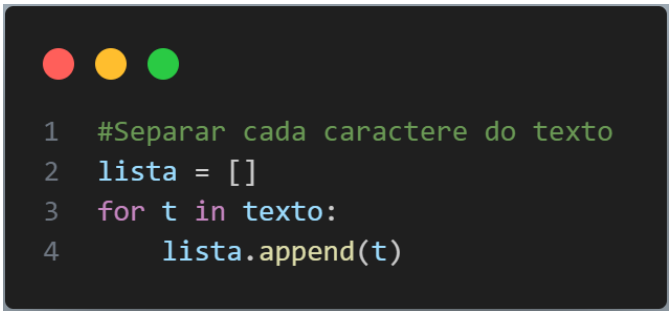
3.2.2 Função Quadrática

Seguindo os passos do processo criptográfico com função afim, primeiro coloque no código a mesma tabela de cifras da Figura 3.1 (alteramos apenas o nome da variável para `correlacao_letra_numero`). Em seguida, criaremos as seguintes variáveis: `texto`, que receberá a mensagem a ser codificada; e `a`, `b` e `c` que receberão os valores dos coeficientes da função, conforme Figura 3.9.

```
1 #Entrada dos coeficientes da função
2 a = int(input('Digite o coeficiente a da função: '))
3 b = int(input('Digite o coeficiente b da função: '))
4 c = int(input('Digite o coeficiente c da função: '))
5
6 #Entrada do texto a ser codificado
7 texto = str(input('Digite o texto: ')).lower()
```

Figura 3.9: Entrada da mensagem e dos coeficientes da função quadrática

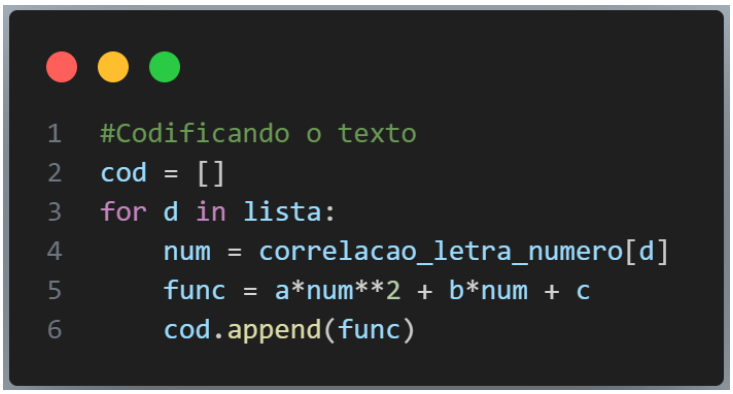
Utilizando os conceitos de lista, subseção 1.3.2.10, e estruturas de repetição, subseção 1.3.2.9, separamos capa caractere da mensagem atribuída na variável `texto` (Figura 3.10).



```
1  #Separar cada caractere do texto
2  lista = []
3  for t in texto:
4      lista.append(t)
```

Figura 3.10: Separação dos caracteres da mensagem

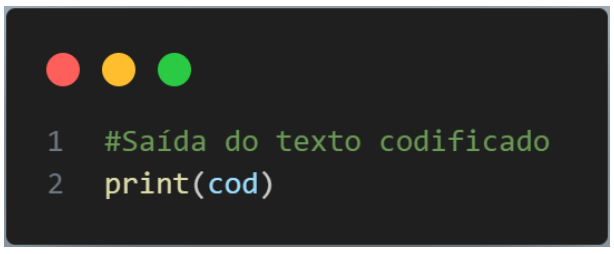
Após, usamos as estruturas de repetição e listas junto com dicionários, vistos na subseção 1.3.2.11, para cifrar cada caractere da mensagem.



```
1  #Codificando o texto
2  cod = []
3  for d in lista:
4      num = correlacao_letra_numero[d]
5      func = a*num**2 + b*num + c
6      cod.append(func)
```

Figura 3.11: Bloco de codificação com função quadrática

A Figura 3.11, segue a mesma construção da Figura 3.5. Na linha 4, adicionamos na variável `num` o elemento do dicionário associado ao caractere `d` da `lista` (vimos na Figura 1.19). Após, na linha 5, ciframos o valor `num` utilizando a expressão `a*num**2 +b*num + c` ($ax^2 + bx + c$), onde `a`, `b` e `c` foram definidos anteriormente, e atribuímos na variável `func`. O caractere cifrado é, então, adicionado na lista `cod` utilizando o método `.append()`.



```
1  #Saída do texto codificado
2  print(cod)
```

Figura 3.12: Saída da mensagem codificada

Enfim, apresentaremos a mensagem codificada, por meio do `print(cod)` (Figura

3.12).

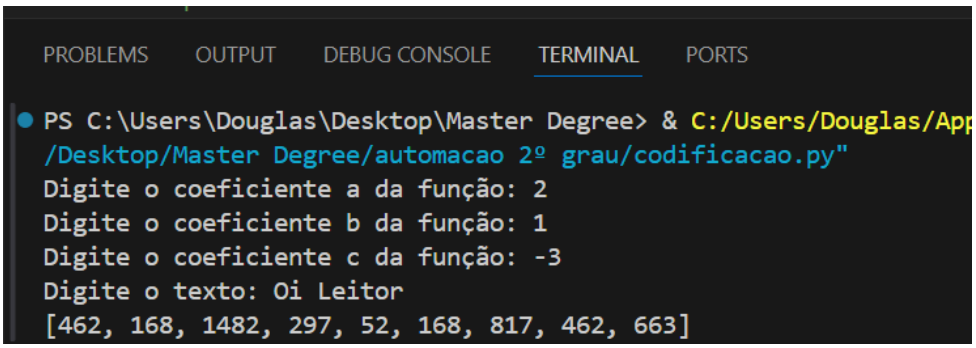


Figura 3.13: Entradas e saídas mostradas no terminal

Quando o código é executado, clicando na opção mostrada na Figura 3.7, aparecerá as entradas e as saídas mostradas no exemplo da Figura 3.13.

3.3 Decodificando

3.3.1 Função Afim

Uma vez que construímos um código em Python para codificar uma mensagem usando função afim, iremos construir um para decodificar.

Iniciaremos adicionando uma tabela de cifras ao código. Essa tabela é semelhante a mostrada na Figura 3.1, porém têm seus elementos invertidos, ou seja, o que era chave agora é valor. A Figura 3.14 apresenta como ficará o dicionário.

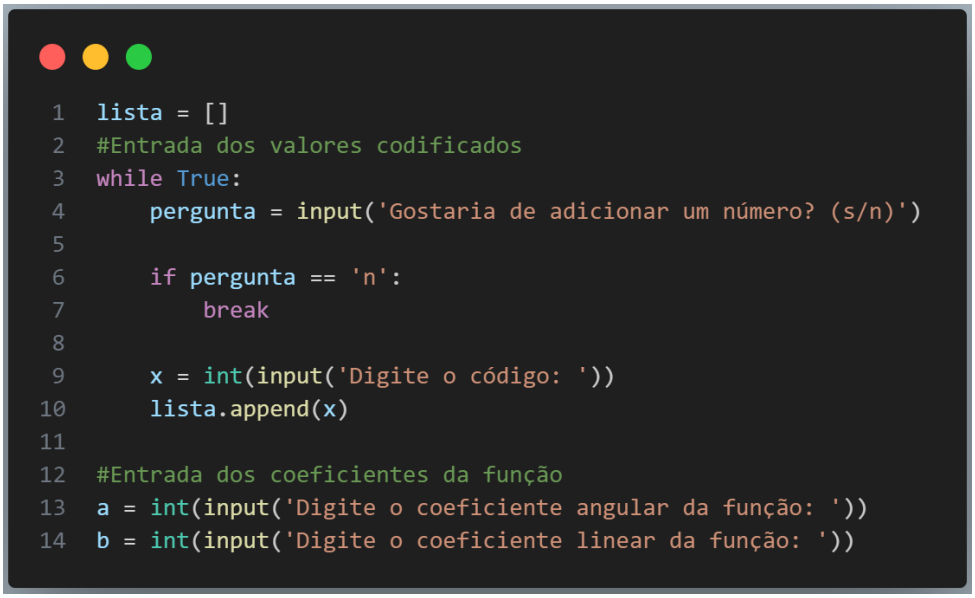
```
#Tabela de cifras
associacao_numero_letra = {
    1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e',
    6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j',
    11: 'k', 12: 'l', 13: 'm', 14: 'n', 15: 'o',
    16: 'p', 17: 'q', 18: 'r', 19: 's', 20: 't',
    21: 'u', 22: 'v', 23: 'w', 24: 'x', 25: 'y',
    26: 'z', 27: ' ', 28: 'é', 29: 'á', 30: 'ç',
    31: ',', 32: '.', 33: '-', 34: 'â', 35: 'ã',
    36: 'õ', 37: 'ô', 38: 'ê', 39: 'í', 40: '/'
}
```

Figura 3.14: Associação números - caracteres

O próximo passo do algoritmo é receber as informações da mensagem e dos coeficientes da função. Utilizaremos os conceitos vistos nas subseções 1.3.2.1, 1.3.2.4, 1.3.2.6, 1.3.2.8, 1.3.2.9 e 1.3.2.10.

Visto que a mensagem a ser decodificada está em formato numérico, criaremos uma lista e um loop *while* para receber cada um dos números que compõem a mensagem. Dentro do loop, é perguntado ao usuário se ele gostaria de adicionar um número, respondendo com "s"para sim ou "n"para não. Caso responda "n", o loop é interrompido. Caso responda "s", é pedido que ele insira o número que será adicionado na lista.

Em seguida, pedimos ao usuário que adicione os coeficientes da função.



```
1  lista = []
2  #Entrada dos valores codificados
3  while True:
4      pergunta = input('Gostaria de adicionar um número? (s/n)')
5
6      if pergunta == 'n':
7          break
8
9      x = int(input('Digite o código: '))
10     lista.append(x)
11
12 #Entrada dos coeficientes da função
13 a = int(input('Digite o coeficiente angular da função: '))
14 b = int(input('Digite o coeficiente linear da função: '))
```

Figura 3.15: Entrada da mensagem e dos coeficientes da função afim

Acompanhando a Figura 3.15, na linha 1, criamos uma lista que nomeamos de `lista`. Na linha 3 adicionamos uma estrutura de repetição escrevendo `while True`, indicando ao computador que todo o bloco deve ser repetido enquanto for verdade (`True`). Só irá parar de repetir caso a instrução `break`, dentro do bloco, seja executada. Na linha 4, inserimos uma variável `pergunta` a qual atribuímos a decisão de adicionar ou não um número na lista, por meio de um `input`. A estrutura de decisão *if*, linha 6, permite que o loop *while* seja interrompido caso a condição, `pergunta == 'n'`, seja atendida. A interrupção é realizada pela instrução `break`. Caso a variável `pergunta` receba "s", o loop não irá parar e ocorrerá, nas linhas 9 e 10, a adição de um dos números da mensagem codificada na `lista`. Após, o processo se repete.

As linhas 13 e 14, trazem duas variáveis, `a` e `b`, que receberão os coeficientes da

função afim.

Outro detalhe essencial é no posicionamento da lista que criamos para armazenar a mensagem separada. Na Figura 3.15, nomeamos a variável que será uma lista como `lista`. Repare que coloca-se a lista fora do loop *while*, pois caso esteja dentro, tem-se um problema.

```
1  while True:
2      lista = []
3      pergunta = input('Gostaria de adicionar um número? (s/n)')
4
5      if pergunta == 'n':
6          break
7
8      x = int(input('Digite o código: '))
9      lista.append(x)
```

Figura 3.16: Exemplo de posicionamento ruim

Na situação da Figura 3.16, sempre que ocorrer uma iteração, a lista irá ser recriada do zero, o que significa que os elementos anteriores são perdidos. Assim, colocando a lista fora do loop, garantimos que ela persista ao longo das iterações e mantenha seus elementos anteriores.

Até o momento, organizamos o código para receber as informações necessárias para decodificar a mensagem. Então, construiremos o bloco que irá realizar esse passo.

Utilizando os conceitos das subseções 1.3.2.2, 1.3.2.5, 1.3.2.9 e 1.3.2.10, na linha 2, criaremos uma lista chamada `decod`, conforme Figura 3.17.

```
1  #Decodificando o texto
2  decod = []
3  for c in lista:
4      inver = (c - b)/a
5      letra = associacao_numero_letra[inver]
6      decod.append(letra)
```

Figura 3.17: Bloco de decodificação com função afim

Necessitaremos de uma estrutura de repetição *for* para decifrar a mensagem re-

cebida. A linha 3 traz, `for c in lista` o que podemos ler como: para cada numero na lista.

Na linha 4, criamos a variável `inver` e atribuímos a ela a expressão `(c - b)/a`. Nesse momento estamos calculando a inversa da função afim, ou seja, $x = \frac{(y-b)}{a}$ para após buscar, por meio do dicionário na linha 5, a associação de `inver` com a letra correspondente. Então, na linha 6, adicionamos na lista `decod` a `letra` decifrada.

Ao final adicionamos o comando `print(decod)` (Figura 3.18) para mostrar na tela o resultado da decodificação.

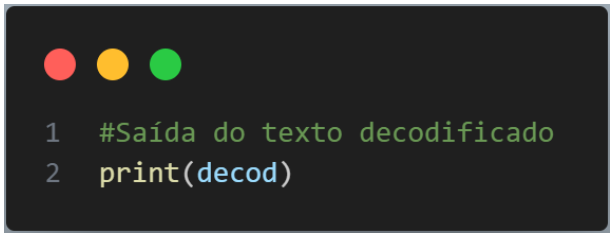


Figura 3.18: Saída da mensagem decodificada

Agora, podemos executar o código. Lembrando de inicia-lo em Run Python File no canto superior direito da tela (Figura 3.19).

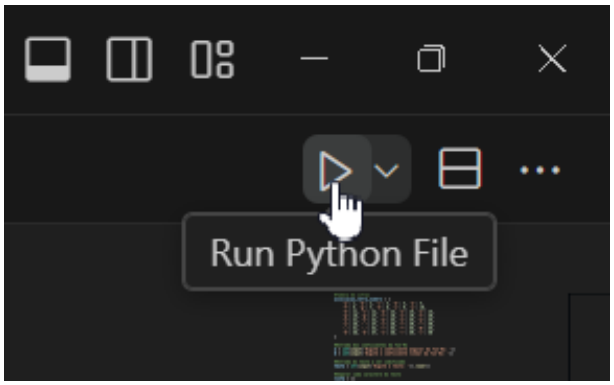


Figura 3.19: Executar o código

Ao executar, inicialmente é solicitado as entradas de cada número da mensagem codificada e, em seguida, das entradas dos coeficiente da função afim. Podemos acompanhar nas Figura 3.20 e 3.21.

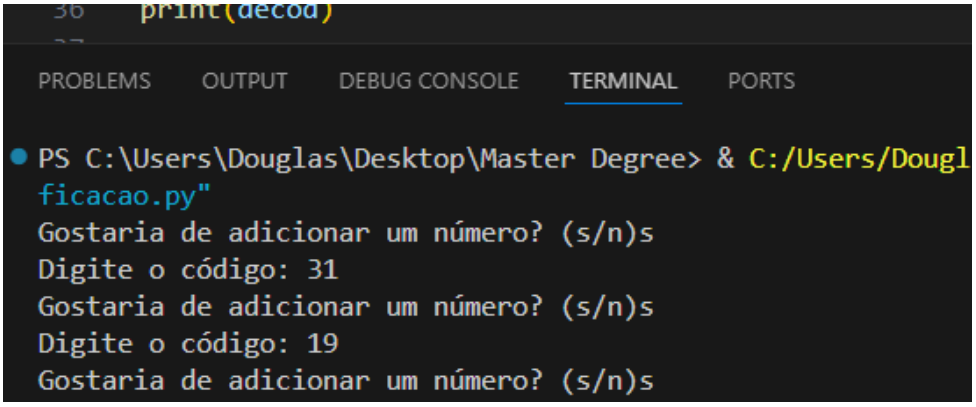


Figura 3.20: Entrada e saída de dados no terminal

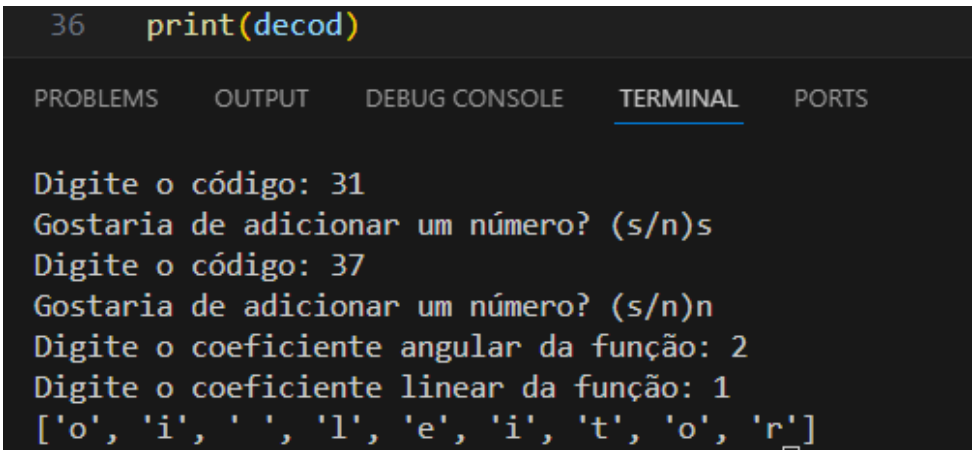


Figura 3.21: Entrada e saída de dados no terminal

Construímos, assim, os códigos que codificam e decodificam mensagens usando função afim como parte da chave criptográfica.

3.3.2 Função Quadrática

Uma vez que construímos um código em Python para codificar uma mensagem usando função quadrática, iremos construir um para decodificar.

Adicionamos uma tabela de cifras ao código. Será o mesmo dicionário apresentado na Figura 3.14 alterando apenas o nome da variável para `correlacao_numero_letra`. Precisamos, então, construir um bloco de código para receber cada caractere.

Utilizaremos os conceitos de: tipos de dados, entrada de dados, operadores relacionais, estrutura de decisão, estrutura de repetição e listas. Conceitos, estes, vistos nas subseções 1.3.2.1, 1.3.2.4, 1.3.2.6, 1.3.2.8, 1.3.2.9 e 1.3.2.10.

```python
#Entrada dos valores codificados
lista = []
while True:
    pergunta = input('Você gostaria de adicionar um número?(s/n)')

    if pergunta == 'n':
        break

    num = int(input('Adicione o número: '))
    lista.append(num)

#Entrada dos coeficientes da chave
a = int(input('Digite o coeficiente a da função: '))
b = int(input('Digite o coeficiente b da função: '))
c = int(input('Digite o coeficiente c da função: '))
```

Figura 3.22: Entrada da mensagem e dos coeficientes da função quadrática

A estrutura que construímos na Figura 3.22 é muito próxima da construída na Figura 3.15, com adição de mais uma variável para entrada do coeficiente `c` da função quadrática.

O próximo bloco é onde decifraremos cada caractere numérico da mensagem codificada.

```python
#Decodificando o texto
decod = []

for e in lista:
    c2 = (c - e)
    delta = b**2 - 4*a*c2
    raiz1 = (-b + (delta)**(1/2))/(2*a)
    raiz2 = (-b - (delta)**(1/2))/(2*a)

    if raiz1 in correlacao_numero_letra:
        num2 = correlacao_numero_letra[raiz1]
    elif raiz2 in correlacao_numero_letra:
        num2 = correlacao_numero_letra[raiz2]

    decod.append(num2)
```

Figura 3.23: Bloco de decodificação com função quadrática

Na linha 5 da Figura 3.23, a atribuição `c2 = (c - e)` se refere ao momento ocorrido na passagem,

$$x^2 + x - 1 = 1 \quad \Rightarrow \quad x^2 + x - 2 = 0$$

já vista na seção 2.4. Repare que o coeficiente `c` da função, é o único que é alterado e os cálculos de decodificação são realizados com essa nova função.

Em seguida, linhas 6, 7 e 8, apresenta-se o método para o cálculo das raízes de funções quadráticas.

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

com $\Delta = b^2 - 4ac$.

Nas linhas 10 a 13, está a escolha da raiz da função que irá compor a mensagem decodificada (visto na seção 2.3).

É nessa etapa que a mensagem é decodificada por meio de uma função.

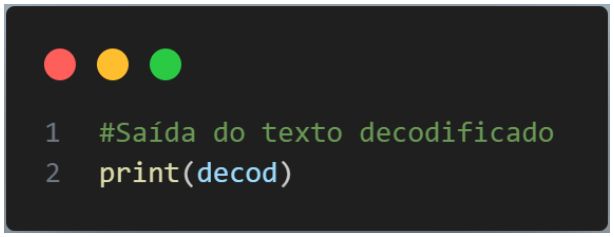


Figura 3.24: Saída da mensagem decodificada

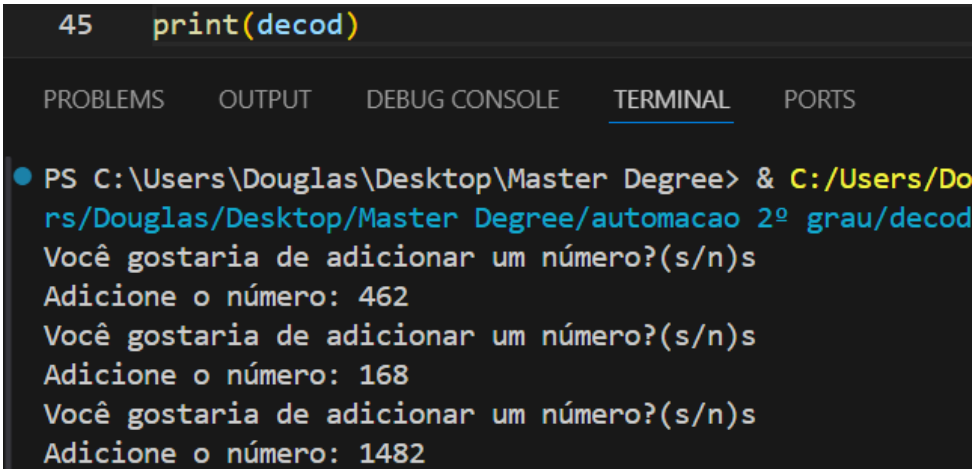


Figura 3.25: Entrada e saída de dados no terminal

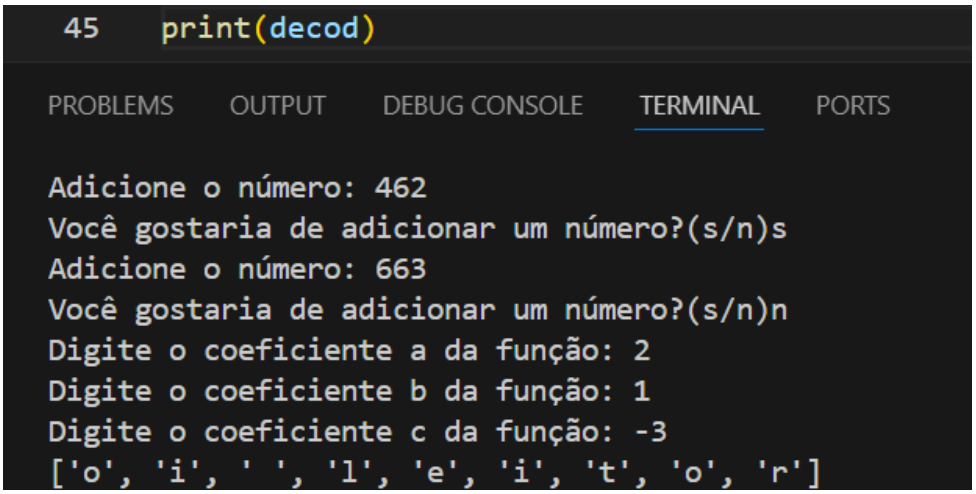


Figura 3.26: Entrada e saída de dados no terminal

Quando o código é executado, clicando na opção mostrada na Figura 3.7, aparecerá as entradas e as saídas mostradas no exemplo das Figuras 3.25 e 3.26.

Observação

A função quadrática utilizada para o exemplo desta subseção faz com que ao decodificar a mensagem codificada, não haja dubiedade para o receptor. Por isso, ressaltamos a importância de escolher funções quadráticas que sejam injetivas no domínio escolhido.

Se tratando do código, na linha 10 da Figura 3.23 inicia-se uma estrutura de decisão em que o valor de uma das variáveis, `raiz1` ou `raiz2`, é utilizada para compor a mensagem decifrada. Porém, caso a função quadrática escolhida não seja injetiva dentro do domínio (definido por meio da tabela de cifras), ambas as variáveis pertencerão ao dicionário `correlacao_numero_letra` e, devido a ordem de escrita do código, sempre a variável `raiz1` será escolhida. Consequentemente, há grande chance da mensagem decodificada ficar destorcida ou não identificável.

Capítulo 4

Uma Proposta de Sequência Didática

Conteúdo

Funções afim e quadrática.

Habilidades (BNCC)

(EF09MA06) Compreender as funções como relações de dependência unívoca entre duas variáveis e suas representações numérica, algébrica e gráfica e utilizar esse conceito para analisar situações que envolvam relações funcionais entre duas variáveis.

(EM13MAT302) Construir modelos empregando as funções polinomiais de 1º ou 2º graus, para resolver problemas em contextos diversos, com ou sem apoio de tecnologias digitais.

(EM13MAT401) Converter representações algébricas de funções polinomiais de 1º grau em representações geométricas no plano cartesiano, distinguindo os casos nos quais o comportamento é proporcional, recorrendo ou não a softwares ou aplicativos de álgebra e geometria dinâmica.

(EM13MAT402) Converter representações algébricas de funções polinomiais de 2º grau em representações geométricas no plano cartesiano, distinguindo os casos nos quais uma variável for diretamente proporcional ao quadrado da outra, recorrendo ou não a softwares ou aplicativos de álgebra e geometria dinâmica, entre outros materiais.

(EM13MAT503) Investigar pontos de máximo ou de mínimo de funções quadráticas em contextos envolvendo superfícies, Matemática Financeira ou Cinemática, entre outros, com apoio de tecnologias digitais.

Objetivos

O objetivo principal é estimular a compreensão dos conceitos matemáticos por meio de atividades contextualizadas. Mediante a criptografia, os estudantes vivenciam a aplicação prática das funções, explorando relações de dependência entre variáveis, construindo modelos matemáticos e interpretando seus gráficos.

Anos: Ensino Médio.

Duração: Embora a sequência tenha cinco etapas, não estipulamos a quantidade de aulas, pois a construção dos conhecimentos pedidos em cada atividade podem exigir tempos diferentes para diferentes turmas.

Desenvolvimento

1ª etapa

Nessa etapa inicial, apresente aos alunos os problemas abaixo e peça que resolvam individualmente:

- Suponha que você está administrando uma empresa de entrega de alimentos e está analisando o desempenho de um dos seus entregadores. Você observa que, após 3 pedidos entregues, o número médio de pedidos entregues por hora por esse entregador é proporcional ao tempo de trabalho, além disso, você descobre que para cada hora trabalhada, o entregador entrega 2 pedidos. Considerando que das 8h às 8h40 foram entregues os 3 primeiros pedidos, qual o número de pedidos entregues até as 12h40?

- Considere uma função afim $f(x) = ax + b$. Se o coeficiente angular a é igual a 2 e o coeficiente linear b é igual a 3, determine o valor de $f(4)$.

- (UEL-PR) Um grupo de amigos alugou um ônibus com 40 lugares para uma excursão. Foi combinado com o dono do ônibus que cada participante pagaria R$60,00 pelo seu lugar e mais uma taxa de R$3,00 para cada lugar não ocupado. Qual o valor máximo que o dono do ônibus receberá?

- Considere uma função quadrática $f(x) = ax^2 + bx + c$. Se o coeficiente a é igual a -3, o coeficiente b é igual a 180 e o coeficiente c é igual a 0, determine o valor máximo da função.

Sondagem: essa primeira atividade serve como uma sondagem inicial. Ela é interessante porque põe os estudantes em contato com uma situação mais próxima da realidade em que precisam mostrar de que forma solucionariam os problemas apresentados.

Organização da turma: ao optar pelo trabalho individual, a intenção é fazer com que cada um acesse os conhecimentos que possui e busque solucionar a questão sozinho.

2ª etapa

O desenvolvimento do processo de criptografia utilizando funções envolve diversos conteúdos matemáticos, como funções afins e quadráticas, funções inversas, manipulação algébrica e resolução de sistemas de equações. Além disso, raciocínio lógico e habilidades de resolução de problemas são cruciais para seguir e implementar corretamente os passos da criptografia.

Traga os conceitos de função matemática utilizando os exemplos da 1ª etapa como base referencial para o assunto. Enfatize o conceito de **injetividade** de uma função, pois será uma propriedade essencial para a função que irá compor a chave criptográfica. Por fim, construa e analise os gráficos das funções vistas nos exemplos da 1ª etapa.

3ª etapa

Fazer uma breve apresentação sobre o conceito de criptografia. Após, apresentar o processo de codificar e decodificar uma mensagem. Uma opção aqui é realizar uma atividade em sala em que, em duplas, grupos interagem por meio do processo criptográfico.

Primeiramente, converse com a turma a respeito da escolha das correlações da tabela de cifras. Consequentemente, é determinado qual será o domínio em que a função estará. Em seguida, organize a turma em grupos. Por exemplo: em uma turma com 20 estudantes, poderiam-se formar quatro grupos com cinco integrantes cada. Dois desses grupos serão responsáveis por codificar e enviar uma mensagem, enquanto os outros dois grupos serão encarregados de decifrar a mensagem recebida.

Para ter-se um maior controle da atividade, algumas condição são necessárias como, por exemplo, um estudante escolher uma mensagem que contenha 8 palavras. Essa situação tornaria o processo muito extenso não sendo apropriado para uma atividade dentro do tempo de aula. Então, combine com os estudantes as seguintes condições:

- Jogar um dado de 6 faces duas vezes. O primeiro resultado é o valor do coeficiente a da função e o segundo o valor do coeficiente b. (de início estamos tratando da

função afim)

- Jogar novamente um dado de 6 faces duas vezes. Tanto o primeiro quanto o segundo resultado se tratam do sinal dos coeficientes. Caso o resultado seja par, o sinal é positivo; se for ímpar, o sinal é negativo.

- Escolher uma frase com até 10 caracteres.

Até aqui, o primeiro grupo obterá a função polinomial e escolherá a mensagem. Na sequência, deve determinar o parâmetro m e cifrar a mensagem. Já o segundo grupo aguardará para receber, decifrar e, posteriormente, verificar com o primeiro grupo se a mensagem está correta.

Ao término da atividade reflita com os estudantes que encerraram a atividade quais as observações que fizeram sobre o procedimento.

Sondagem: Espera-se que o estudante compreenda bem os conceitos de domínio e imagem de uma função ao observar a mensagem original se tornando a mensagem codificada e, após, retornando à mensagem original por meio da decodificação.

Complemento: A criptografia pode ser apresentada com exemplos práticos, como o *WhatsApp* e o *D4sign*[1]. O *WhatsApp* informa[2] aos usuários sobre a segurança das mensagens criptografadas. Já o *D4sign* garante a legitimidade das assinaturas eletrônicas.

No *D4sign*, o processo é o seguinte: o assinante recebe um e-mail com o documento a ser assinado. Este documento passa por uma função hash, transformando seu conteúdo em uma saída de tamanho fixo. A hash é então criptografada com a chave privada do assinante, criando a assinatura. Como a chave privada é exclusiva do assinante, apenas ele pode utilizá-la. Por fim, o documento assinado é enviado ao destinatário.

4ª etapa

Nesta etapa, selecione e distribua para os grupos responsáveis pela codificação, uma função quadrática que seja injetiva dentro de um domínio determinado (pode ser utilizada a mesma tabela de cifras já construída na etapa anterior). A partir da distribuição, os estudantes podem iniciar com o processo criptográfico. Diferente da etapa anterior, eles não precisarão jogar dados, pois a função já estará pronta. O que precisam

[1] https://d4sign.com.br/
[2] Apêndice B

fazer é escolher uma frase com até 10 caracteres. Não há alterações para os grupos que decodificarão a mensagem.

Assim como na etapa anterior, pontuem as observações lembrando-se sempre de relacionar com os conhecimentos de funções apresentadas desde o início da sequência didática.

Após desenvolverem toda a atividade manual, inclua o produto educacional para trabalhar casos particulares. Nesse momento, peça para os estudantes realizarem o processo criptográfico de forma automatizada. A intenção é observar os casos em que a função quadrática não é injetiva no domínio e, então, ver e analisar os problemas que podem ocorrer.

No Apêndice C, há um guia de como utilizar o produto educacional para fazer os testes com as funções e analisar os resultados.

Sondagem: Esse momento é uma ótima oportunidade para apresentar de que forma cada coeficiente da função quadrática influencia no formato do gráfico. Aproveite e peça para os estudantes observarem e fazerem o mesmo com a função afim, caso não tenham percebido na etapa anterior.

5ª etapa

Este é o momento em que entramos em contato com a automação. Apresente aos estudantes a linguagem de programação Python fazendo alguns comandos simples e curtos, como adicionar um valor a uma variável ou realizar a soma de alguns números e mostrar na tela, conforme vistos na subseção 1.3.2.

Orientações: O papel do estudante nessa etapa é fazer a "tradução"dos procedimentos feitos a mão para a linguagem de programação Python. A cada fase de cálculo realizado nas etapas anteriores, há uma maneira de se realizar computacionalmente e essa atividade é o foco nessa quinta etapa da sequência didática. É esperado que o estudante reconheça o que é necessário em cada fase de cálculo e utilize as ferramentas da linguagem de programação necessárias para que a automatização funcione.

Avaliação

A avaliação se dará de forma processual durante o decorrer das cinco etapas. Alguns pontos principais a serem desenvolvidos são:

1. Participação e Atitude: Observar o engajamento dos alunos nas atividades

em grupo, a participação nas discussões, a colaboração e a criatividade na resolução de problemas; Avaliar a capacidade de os alunos formular perguntas relevantes, apresentar ideias e justificar suas respostas; Avaliar o interesse dos alunos pelos temas abordados, a curiosidade e a disposição para aprender.

2. Trabalho em Grupo: Avaliar a capacidade dos alunos de aplicar corretamente as funções afim e quadrática para codificar e decodificar mensagens, utilizando os coeficientes e sinais de forma precisa; Avaliar a capacidade dos alunos de analisar os resultados da criptografia, identificando a influência dos coeficientes nas funções e a relação entre domínio e imagem; Avaliar a colaboração e a comunicação entre os membros do grupo durante a realização da atividade.

3. Programação em Python: Avaliar a capacidade dos alunos de traduzir o processo de criptografia manual para o código Python, utilizando corretamente as estruturas de dados, variáveis, operadores e comandos de entrada e saída; Avaliar a capacidade de os alunos em testar o código, identificar e corrigir erros (debugging) e garantir que o programa funcione corretamente.

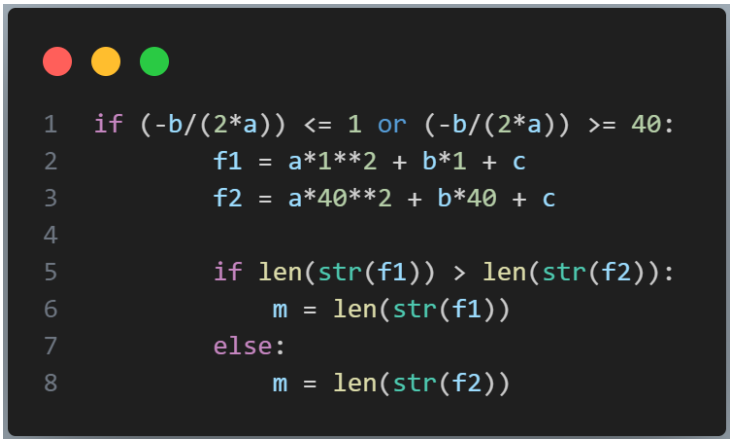
4. Teste Individual: Avaliar a compreensão dos alunos sobre os conceitos de função, domínio, imagem, coeficientes angular e linear, e a relação entre a representação algébrica, gráfica e tabular de funções; Avaliar a capacidade dos alunos de resolver problemas que envolvam funções afim e quadrática em situações reais, como cálculo de custos, análise de gráficos e interpretação de dados.

Considerações Finais

Este trabalho propôs automatizar o processo criptográfico e estudá-lo de forma a possibilitar o desenvolvimento de algumas funções matemáticas, com foco nos estudantes do 9º ano do ensino fundamental até o 3º ano do ensino médio.

O ensino de programação nas escolas é incentivado por diversas organizações ao redor do mundo. Um exemplo é o projeto "Hora do Código"oferecido pela organização Code.org, que é uma introdução gratuita à ciência da computação com atividades e vídeos para alunos de todos os níveis de habilidade. A Base Nacional Comum Curricular (BNCC) também oportuniza, em seu documento, a possibilidade de aprimorar os conhecimentos em matemática por meio de ações tecnológicas digitais. A automação do processo criptográfico conversa diretamente com essas fontes e permite criar situações enfáticas que desenvolvem o pensamento computacional.

Observando alguns possíveis momentos durante a escrita do código em linguagem Python, verificamos que apesar do código apresentado realizar o processo criptográfico conforme esperado, nem todas as situações são apropriadas para esse processo como foi visto no capítulo 2. Então, ao escolher a função que irá compor a chave, é melhor selecionar uma que satisfaça a condição: $\frac{-b}{2a} \leq x_0 \quad \text{ou} \quad x_n \leq \frac{-b}{2a}$. Dessa forma evita-se dubiedade na mensagem decodificada e uma possível troca excessiva do domínio. Em linhas de código podemos corrigir isso adicionando a estrutura de decisão *if* com a condição de execução do bloco somente se uma das desigualdades são satisfeitas.



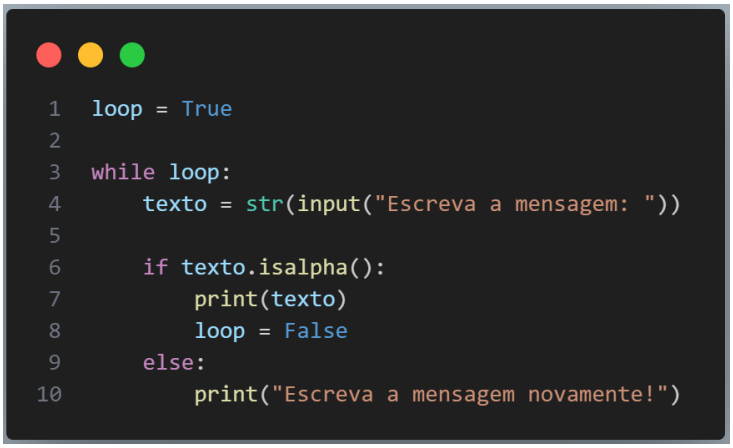
```
if (-b/(2*a)) <= 1 or (-b/(2*a)) >= 40:
        f1 = a*1**2 + b*1 + c
        f2 = a*40**2 + b*40 + c

        if len(str(f1)) > len(str(f2)):
            m = len(str(f1))
        else:
            m = len(str(f2))
```

Figura 4.1: Validador da função escolhida

Na Figura 4.1, podemos verificar que o valor do parâmetro m será atribuído somente se a condição dos dois blocos forem satisfeitas.

Por outro lado, o código visto no capítulo 3, não apresenta a necessidade de calcular o parâmetro m, pois a mensagem codificada é mostrada na tela em formato de lista, por exemplo: [2, 3, 7]. Um bom questionamento seria como modificar o código para que a mensagem codificada fosse exibida fora de uma lista, por exemplo, como uma sequência contínua: 237. Esses momentos são o que fundamentam o pensamento computacional. O foco não está no início e nem no final do processo, mas sim nos possíveis questionamentos que surgem durante a construção do mesmo.

Outro aspecto ocorre ao pedir para o usuário inserir uma mensagem. A inserção de um caractere numérico causa problemas na execução do código, considerando a sua estrutura. Uma solução seria implementar uma verificação da entrada da mensagem para garantir que apenas caracteres válidos sejam aceitos, prevenindo assim possíveis erros. Analisemos a Figura 4.2.



```python
loop = True

while loop:
    texto = str(input("Escreva a mensagem: "))

    if texto.isalpha():
        print(texto)
        loop = False
    else:
        print("Escreva a mensagem novamente!")
```

Figura 4.2: Validador da mensagem

A variável loop serve como um freio, interrompendo a estrutura de repetição *while* quando uma condição é satisfeita. Enquanto loop for verdadeiro, o bloco será executado. O primeiro requerimento é a entrada da mensagem pelo usuário. Em seguida, a estrutura de decisão *if* verifica se a variável texto está escrita inteiramente com letras do alfabeto. Caso esteja, a mensagem atribuída a variável texto é mostrada na tela e a variável loop altera para falso, fazendo com que o bloco *while* seja interrompido. Caso não esteja, é mostrado na tela uma mensagem pedindo para escrever novamente a mensagem e repete-se o procedimento. Essa lógica soluciona parte do problema, pois ainda se escrevermos uma mensagem com caracteres do alfabeto contendo caracteres especiais, ainda dará erro. Então, devemos acrescentar mais algumas linhas.

```python
def validador(text):
    # Define os caracteres especiais permitidos
    special_characters = "!@#$%^&*(),.?\":{}|<> "

    # Itera sobre cada caractere na string
    for char in text:
        # Verifica se o caractere é uma letra ou um caractere especial permitido
        if not (char.isalpha() or char in special_characters):
            return False
    return True

loop = True

while loop:
    texto = input("Escreva a mensagem: ")

    # Verifica se a string contém apenas letras e caracteres especiais
    if validador(texto):
        print(texto)
        loop = False
    else:
        print("Escreva a mensagem novamente!")
```

Figura 4.3: Validador da mensagem aprimorado

Na Figura 4.3, definimos uma função chamada *validador*, que receberá como variável a mensagem que será digitada pelo usuário. Essa função armazena todos os caracteres especiais que serão permitidos e verifica, por meio de uma estrutura de repetição *for*, caractere por caractere se há número na mensagem. Se houver a função retornará falso e irá ser pedido para escrever uma nova mensagem. Caso não haja números, a mensagem é mostrada na tela.

Em suma, esses detalhes fazem a diferença ao analisar o processo de automação. O pensamento computacional se manifesta na capacidade de questionar uma linha de código ou um problema matemático, buscando explorar todas as possibilidades de ação, mesmo que a solução atual esteja funcionando. Perguntas como "E se houver um número no meio da frase?", "E se também houver caracteres especiais?"e "E se houver letras de um alfabeto de outro país?"são fundamentais para construir esse pensamento.

Como visto, no capítulo 1, há diversos comandos (ou palavras-chave) para utilizar na linguagem de programação Python. Cada comando pode ser relacionado com os conceitos de matemática, estabelecendo paralelos para a compreensão das funções e

estruturas dessa linguagem de programação.

Quando utilizamos entrada e saída de dados, estamos atribuindo um valor a uma variável. É o mesmo que definir um valor para uma variável na matemática ou exibir o valor dessa variável como o resultado de uma operação. Por exemplo, a entrada de dados em Python, como em

```
x = int(input("Digite um número: ")),
```

equivale a receber o valor de uma variável na matemática, como $x \in \mathbb{R}$. A saída de dados, com

```
print("O valor de x é:", x),
```

corresponde a exibir o valor de uma variável ou o resultado de uma operação, como $(x = 5)$ ou $(f(x) = 5)$.

Estruturas de repetição, como

```
soma = 0
for i in range(1, 11):
    soma += i
print(soma),
```

se assemelham a somatórios ou produtórios na matemática, por exemplo, $\sum_{i=1}^{10} i$. Além disso, podemos relacionar com o princípio multiplicativo. Por exemplo, considere 3 opções de camisetas (verde, azul e amarelo) e 2 opções de calças (vermelha e branca). Para se alcançar todas as possibilidades de escolher uma camiseta e uma calça, podemos realizar a multiplicação 3x2, encontrando 6 possibilidades. Essa multiplicação ocorre pois, para cada uma das cores de camisetas, há duas possibilidades de cores de calças. Podemos fazer essa relação no código.

```
camisas = ['verde', 'azul', 'amarelo']
calcas = ['vermelha', 'branca']
possibilidades = []
for camisa in camisas:
    for calca in calcas:
        vestimenta = []
        vestimenta.append(camisa)
        vestimenta.append(calca)
        possibilidades.append(vestimenta)
print(possibilidades)
```

A saída será uma lista composta de outras listas com o seguinte conteúdo: `[['verde', 'vermelha'], ['verde', 'branca'], ['azul', 'vermelha'], ['azul', 'branca'], ['amarelo', 'vermelha'], ['amarelo', 'branca']]`.

As estruturas de decisão, como

```
if x > 0:
    print("x é positivo")
```

ou

```
if x > 0:
    print("x é positivo")
elif x == 0:
    print("x é zero")
else:
    print("x é negativo"),
```

são análogas a funções por partes, como a definição do valor absoluto $|x|$.

$$|x| = \begin{cases} x, & \text{se } x \geq 0 \\ -x, & \text{se } x < 0 \end{cases}$$

Listas em Python, definidas como `lista = [1, 2, 3, 4, 5]`, equivalem a vetores ou sequências, como $\{1, 2, 3, 4, 5\}$. Já os dicionários, como `dicionario = {"a": 1,`

`"b": 2, "c": 3`}, representam funções ou mapeamentos, por exemplo, $f : \{a, b, c\} \rightarrow \{1, 2, 3\}$. Outra possibilidade é ver as chaves como elementos no domínio enquanto os valores são as imagens. Pode ser trazido, também, as propriedades de um conjunto: finito; infinito; aberto; fechado; união; intersecção.

Compreender esses paralelos facilita o aprendizado e a aplicação de Python em contextos matemáticos.

No que se refere as limitações do trabalho, a principal é a barreira linguística. A linguagem de programação Python faz uso de palavras-chave em inglês, consequentemente, utiliza estruturas de repetição ou decisão que precisam ser escritas ou lidas em inglês. Assim como, os comandos de entrada e saída de dados. Embora o Python apresente essa característica, o que pode representar um desafio adicional para estudantes que não dominam esse idioma, existem alternativas como PORTUGOL e o SCRATCH. No entanto, essas linguagens possuem menos recursos em comparação com o Python.

Uma alternativa para pesquisas futuras seria a busca por uma linguagem de programação em português que consiga realizar a atividade proposta com a mesma eficiência ou utilizar o interpretador VisuAlg para escrever os códigos. Outra sugestão seria a adição de outros tipos de funções que possam compor a chave de criptografia ou a utilização de funções que não funcionam nesse processo para introduzir outros conteúdos matemáticos, como a função seno. Esse tipo de função poderia gerar múltiplas possibilidades ao decodificar a mensagem, permitindo a introdução do conceito de análise combinatória para determinar as possíveis mensagens resultantes.

Dessa forma, a integração da programação com a matemática não apenas enriquece o currículo escolar, mas também prepara os estudantes para um futuro onde o pensamento computacional e a habilidade de resolver problemas complexos serão cada vez mais valorizados. As propostas de melhorias e alternativas mencionadas oferecem um caminho para futuras pesquisas e implementação prática no ambiente educacional.

Referências Bibliográficas

BARICHELLO, L. e MATHIAS, C. (2021). Novos conteúdos e novas habilidades para a área de matemática e suas tecnologias. *Revista Internacional de Pesquisa em Educação Matemática*.

BLIKSTEIN, P. (2008). O pensamento computacional e a reivenção do computador na educação. Disponível em `http://blikstein.com/paulo/documents/online/ol_pensamento_computacional.html` Acesso em: 10/02/2024.

BRASIL (1996). Lei de diretrizes e bases da educação. Disponível em `http://portal.mec.gov.br/arquivos/pdf/lei%209394.pdf` Acesso em: 24/02/2024.

BRASIL (2018). Base nacional comum curricular. Disponível em `http://http://basenacionalcomum.mec.gov.br/images/BNCC_EI_EF_110518_versaofinal_site.pdf` Acesso em: 15/12/2023.

BRASIL (2022). Normas sobre computação na educação básica – complemento à base nacional comum curricular (bncc). Disponível em `https://www.gov.br/mec/pt-br/cne/parecer-ceb-2022` Acesso em: 28/02/2024.

CIEB (2016). Currículo de referência para computação e tecnologia. Disponível em `https://curriculo.cieb.net.br/` Acesso em: 20/02/2024.

FIARRESGA, V. M. C. (2010). Criptografia e matemática. Dissertação de Mestrado, Universidade de Lisboa (Portugal).

IBGE (2024). Pesquisa nacional por amostra de domicílios contínua. Disponível em `https://painel.ibge.gov.br/pnadc/` Acesso em: 06/03/2024.

IEZZI, G. e MURAKAMI, C. (2013). Fundamentos de matemática elementar volume 1, conjuntos e funções. *Editora Atual, São Paulo*.

KAHN, D. (1968). *The Codebreakers*. The Macmillan Company, Nova Yorque.

LAMBERT, K. A. (2019). *Fundamentals of Python First Programs*. Cengage.

LIMA, E. L. (2011). *Análise Real: Funções de Uma Variável*, volume 1. SBM, Rio de Janeiro.

LIMA, E. L., Carvalho, P. C. P., Wagner, E., e Morgado, A. C. (2006). *A Matemática do Ensino Médio*, volume 1. SBM, Rio de Janeiro.

MICROSOFT (2024). Visual studio code. Disponível em `https://code.visualstudio.com/Docs` Acesso em: 13/04/2024.

MUNIZ, A. C. (2022). *Fundamentos de Cálculo*. SBM, Rio de Janeiro.

PONTES, E. A. S., SILVA, B. H. M. d. S., OLIVEIRA, E. G., LEITE, L., et al. (2022). Criptografia em funções polinomiais: Um processo de ensino e aprendizagem de matemática na educação básica. *The Journal of Engineering and Exact Sciences*, 8(6):14609–01e.

SHOUP, V. e BONEH, D. (2023). A graduate course in applied cryptography. Disponível em `http://toc.cryptobook.us/` Acesso em: 19/03/2024.

STACKOVERFLOW (2023). 2023 developer survey. Disponível em `https://survey.stackoverflow.co/2023/` Acesso em: 03/03/2024.

TAMAROZZI, A. C. (2001). Codificando e decifrando mensagens. *Revista do Professor de Matemática*, 45:41–43.

WING, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.

Apêndice: Material Adicional

A Primeira Seção do Apêndice

Detalhando o processo de instalação, assim como o acesso ao ambiente para escrever os códigos, segue-se os passos das Figuras A.1 até A.17:

Após download, instalação e execução do software no computador, a tela inicial do VS Code trará algumas opções para configurações iniciais. Não é obrigatório fazer as escolhas de configurações nesse momento, então recomenda-se seguir acessando o *Welcome* no canto superior esquerdo da tela.

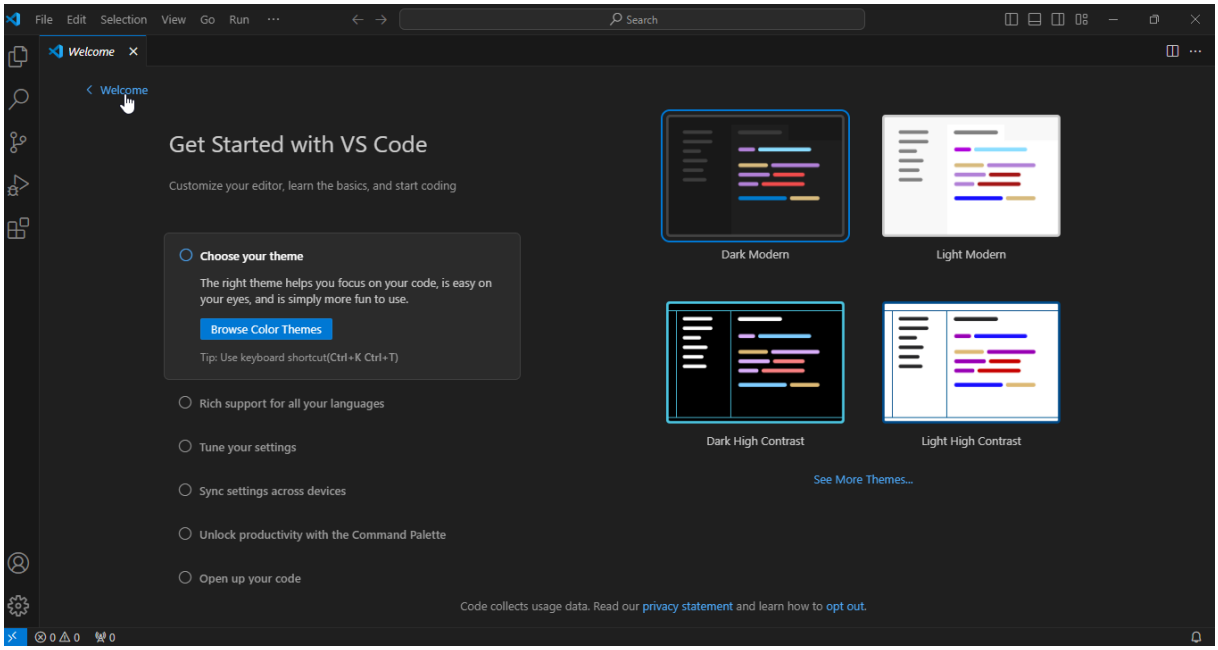


Figura A.1: Preparação de ambiente 1

Aparecerão algumas opções como *New File* (Novo arquivo), *Open File* (Abrir arquivo), *Open Folder* (Abrir pasta) e *Conect to* (Conectar com). Pode-se selecionar diretamente a opção *Open Folder*.

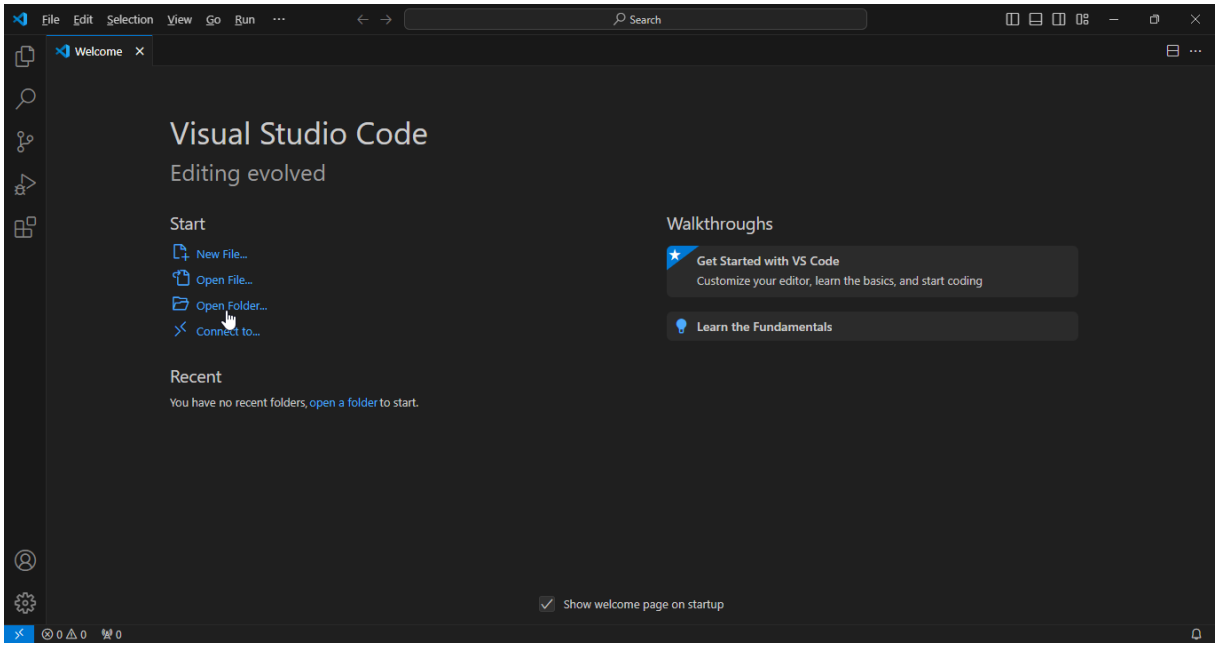


Figura A.2: Preparação de ambiente 2

Aparecerá uma janela para indicar onde está a pasta a ser aberta ou para criar uma nova pasta. Clique na opção Nova Pasta, atribua um nome a ela e clique em Selecionar Pasta.

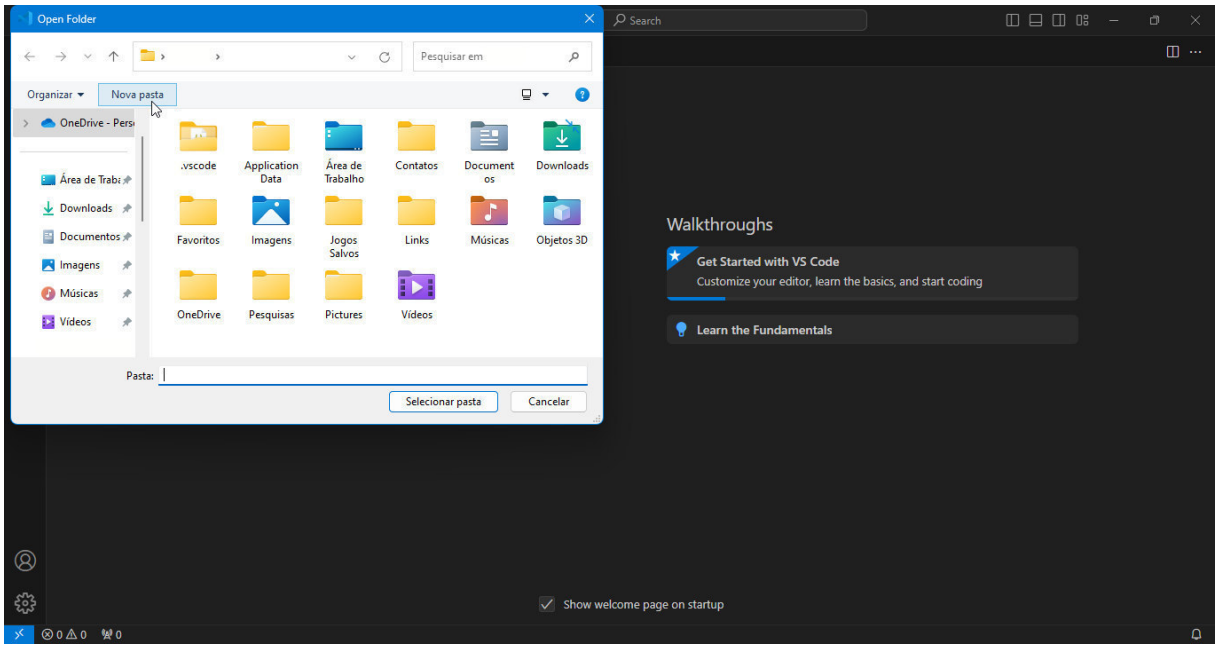


Figura A.3: Preparação de ambiente 3

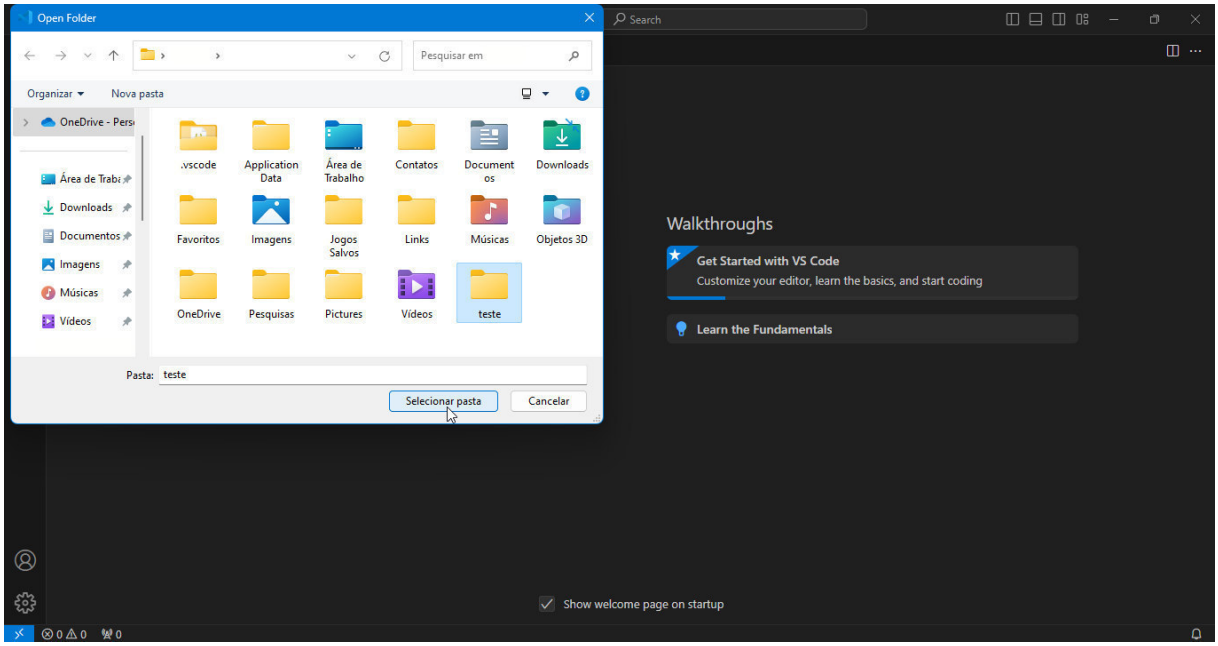


Figura A.4: Preparação de ambiente 4

Uma mensagem deve aparecer pedindo a ciência de que há confiança no autor dos arquivos na pasta selecionada. Como foi criada uma pasta nova, pode-se selecionar a opção *Yes, I trust the authors* (Sim, eu confio nos autores).

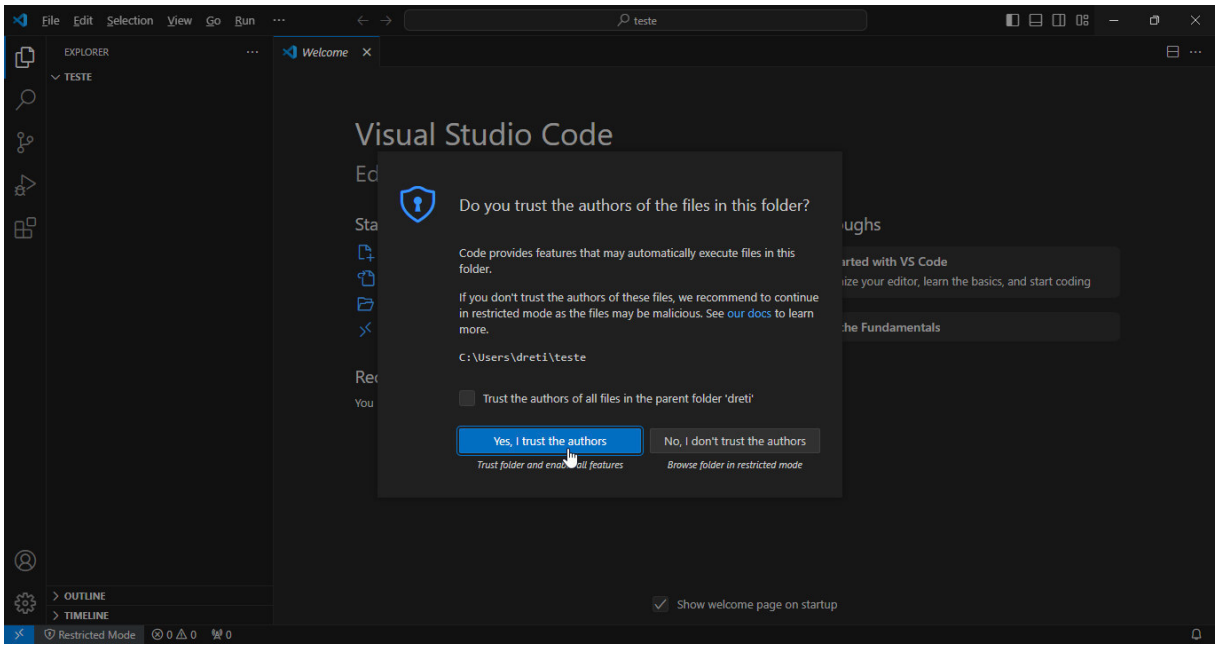


Figura A.5: Preparação de ambiente 5

A interface, como na Figura 1.4, aparecerá. Será onde os arquivos de código e o ambiente de programação são construídos.

No canto superior esquerdo da tela aparecerá o nome da pasta que foi selecionada e, ao lado direito, haverão quatro ícones. Acesse o primeiro, nomeado como *New File*.

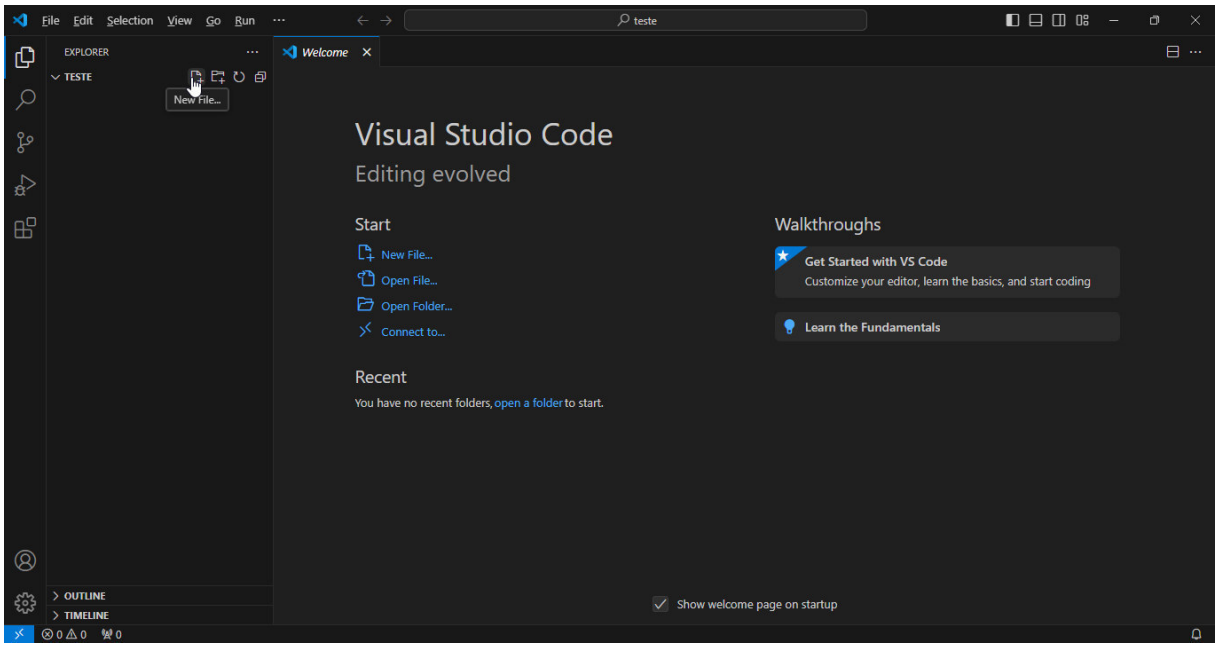


Figura A.6: Preparação de ambiente 6

Em seguida, nomeie o arquivo criado escrevendo *.py* ao final do nome. O *.py* é uma extensão de arquivo, que indica que é um arquivo de código fonte escrito na linguagem de programação Python, ou seja, é a identificação que indica que aquele arquivo é escrito em Python.

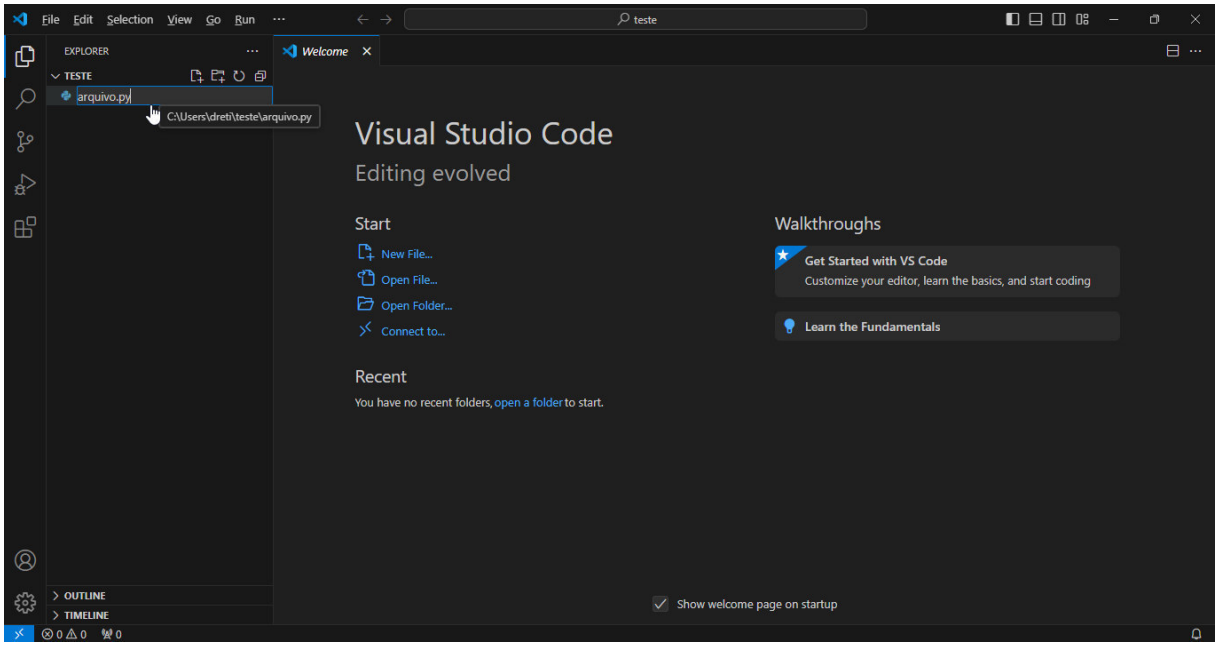


Figura A.7: Preparação de ambiente 7

Aparecerá uma Aba de Editores com o nome do arquivo e um espaço para edição do código.

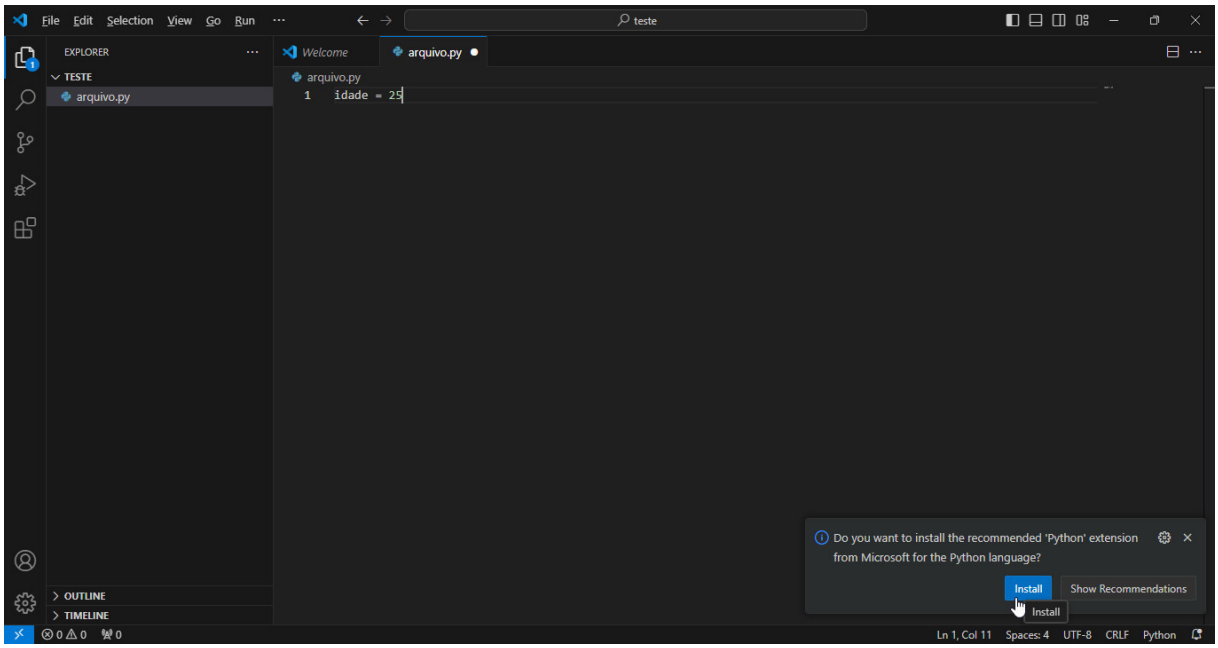


Figura A.8: Preparação de ambiente 8

A seguinte mensagem pode aparecer: *Do you want to install the recommended 'Python' extension from Microsoft for the Python language?* (Você gostaria de instalar a extensão 'Python' recomendada da Microsoft para a linguagem Python?). Caso apareça realize a instalação clicando em *Install*.

Uma nova Aba de Editores se abrirá com a extensão do Python para que o VS Code reconheça as palavras chave da linguagem. Instale essa extensão.

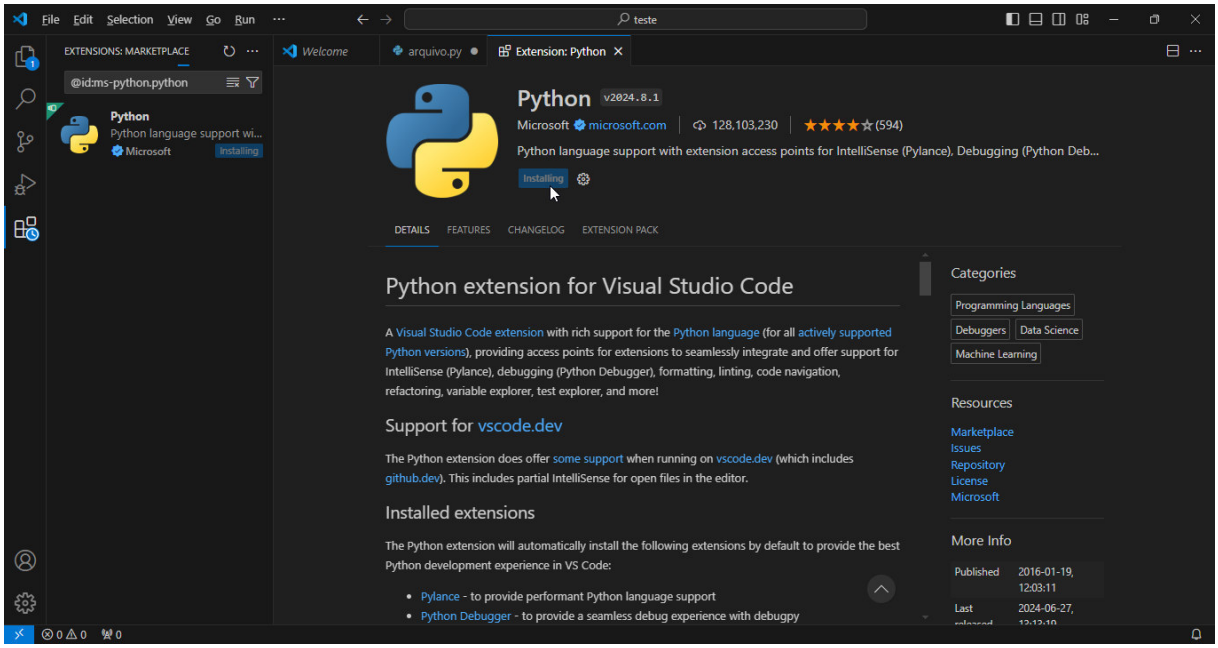


Figura A.9: Preparação de ambiente 9

No canto inferior direito da tela pode aparecer a mensagem *Select Interpreter* (Selecionar Interpretador).

Um interpretador de uma linguagem de programação é como um tradutor que pega o código escrito e o executa diretamente, linha por linha.

Clicando na mensagem aparecerá: *No Python interpreter is selected. Please select a Python interpreter to enable features such as IntelliSense*[3]*, linting*[4] *and debugging.* (Não está selecionado nenhum interpretador Python. Selecione um interpretador Python para ativar funcionalidades como IntelliSense, linting e depuração.).

[3]IntelliSense é uma tecnologia incorporada em muitos editores de código e IDEs (Ambientes de Desenvolvimento Integrado) que fornece sugestões e informações em tempo real enquanto você digita o código.

[4]Linting é o processo de analisar o código fonte de um programa para identificar e sinalizar possíveis erros, problemas de estilo e padrões que podem causar bugs ou comportamento inesperado.

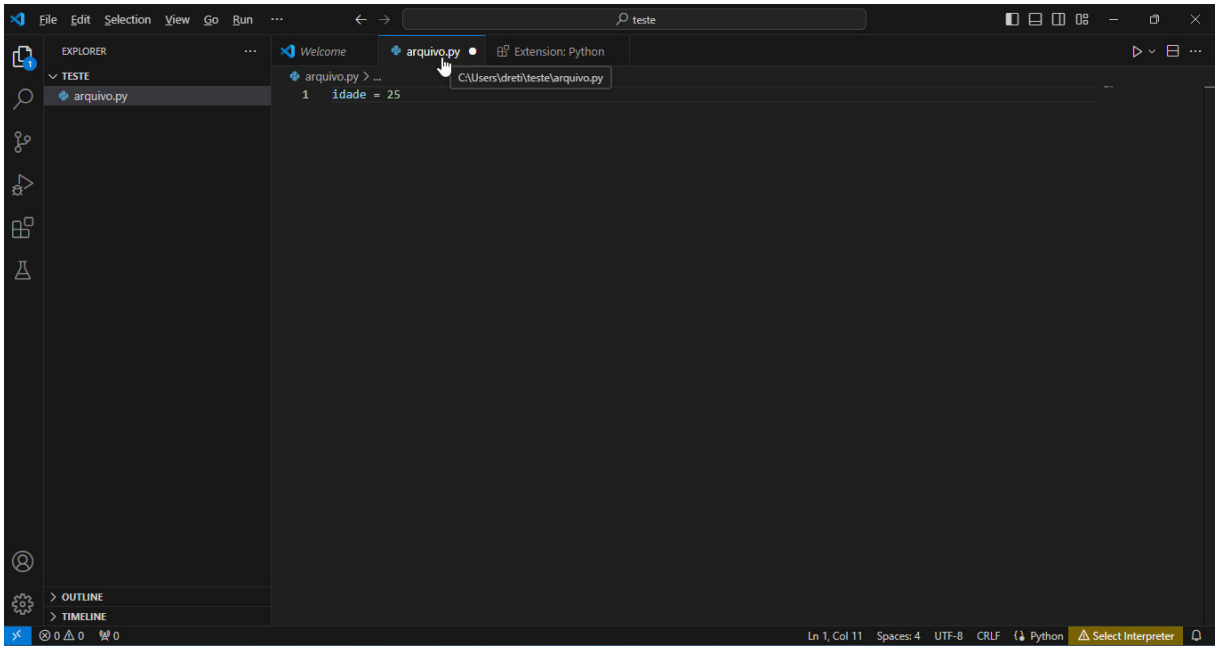


Figura A.10: Preparação de ambiente 10

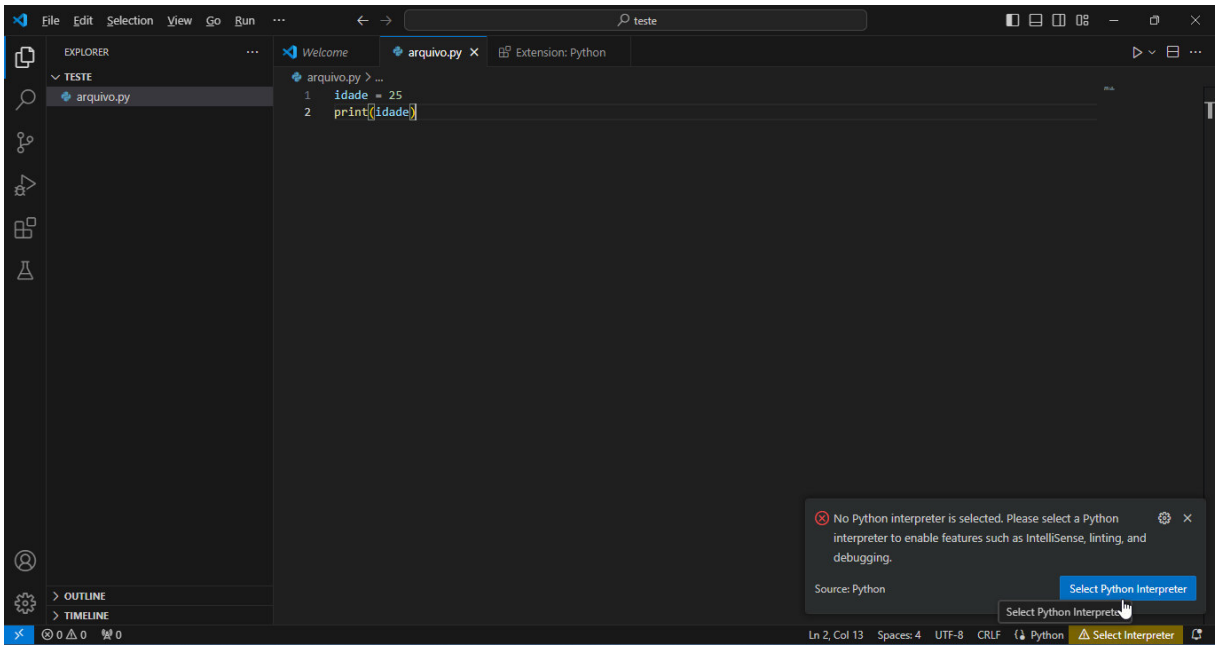


Figura A.11: Preparação de ambiente 11

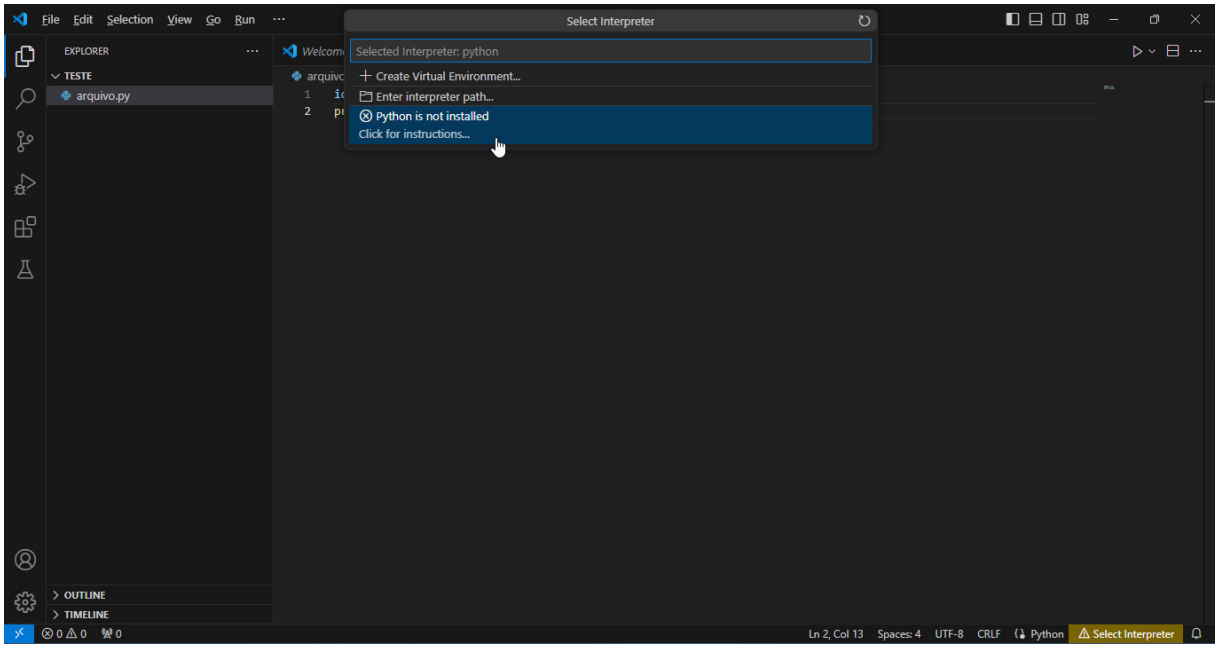


Figura A.12: Preparação de ambiente 12

Acesse o site python.org e faça o download da versão mais recente do Python.

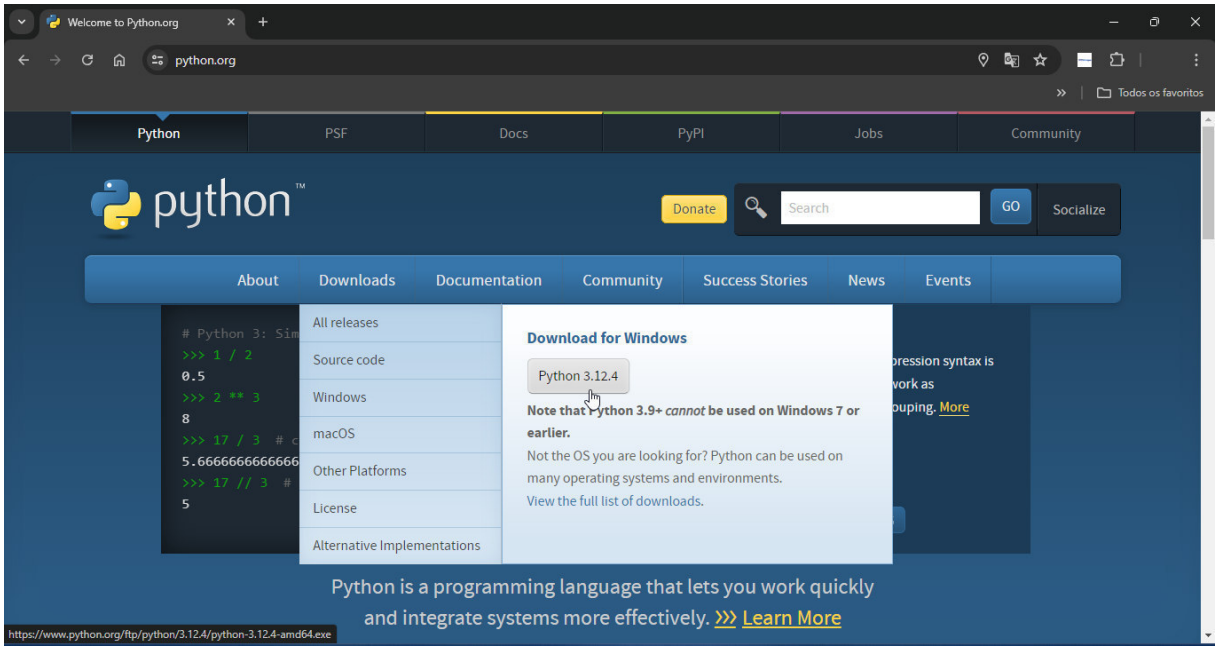


Figura A.13: Preparação de ambiente 13

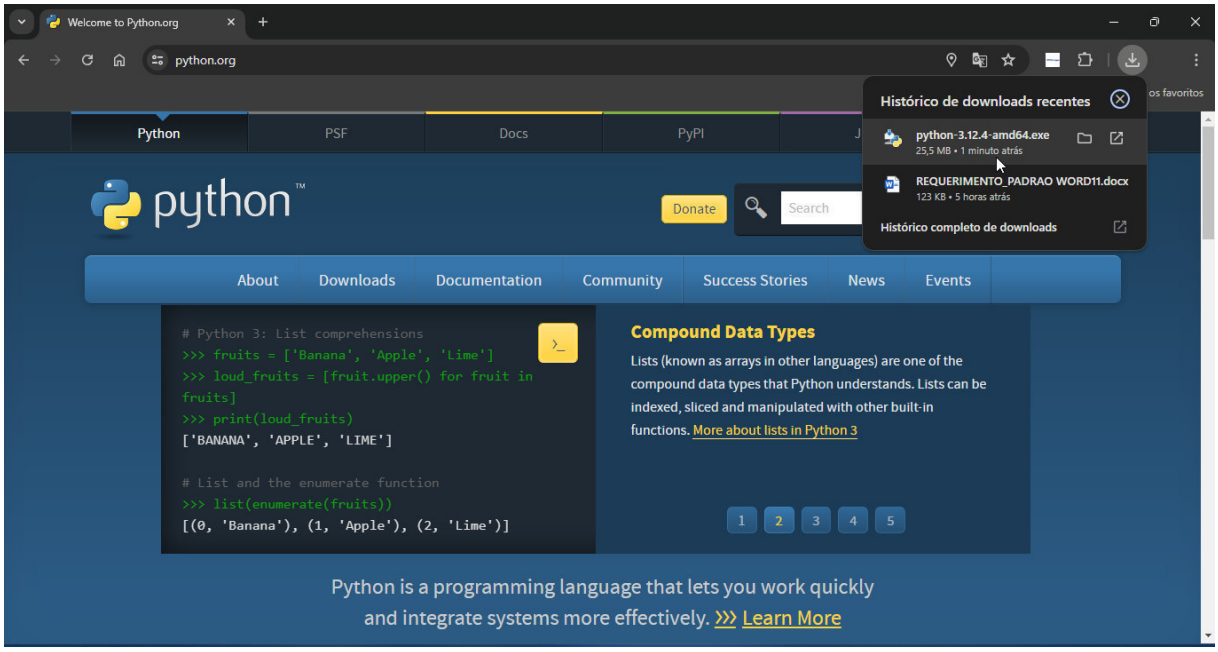


Figura A.14: Preparação de ambiente 14

Figura A.15: Preparação de ambiente 15

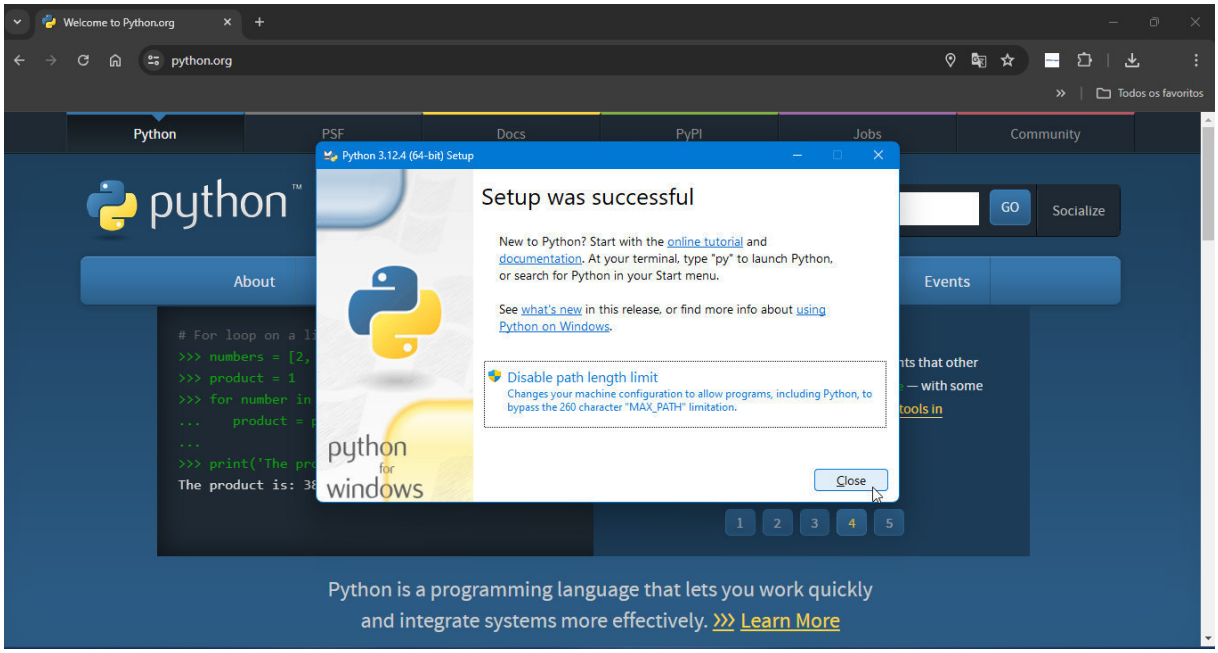


Figura A.16: Preparação de ambiente 16

Feche o VS Code e abra-o novamente para que as alterações sejam efetivadas. O software irá retornar na pasta e arquivo que estava antes de ser fechado.

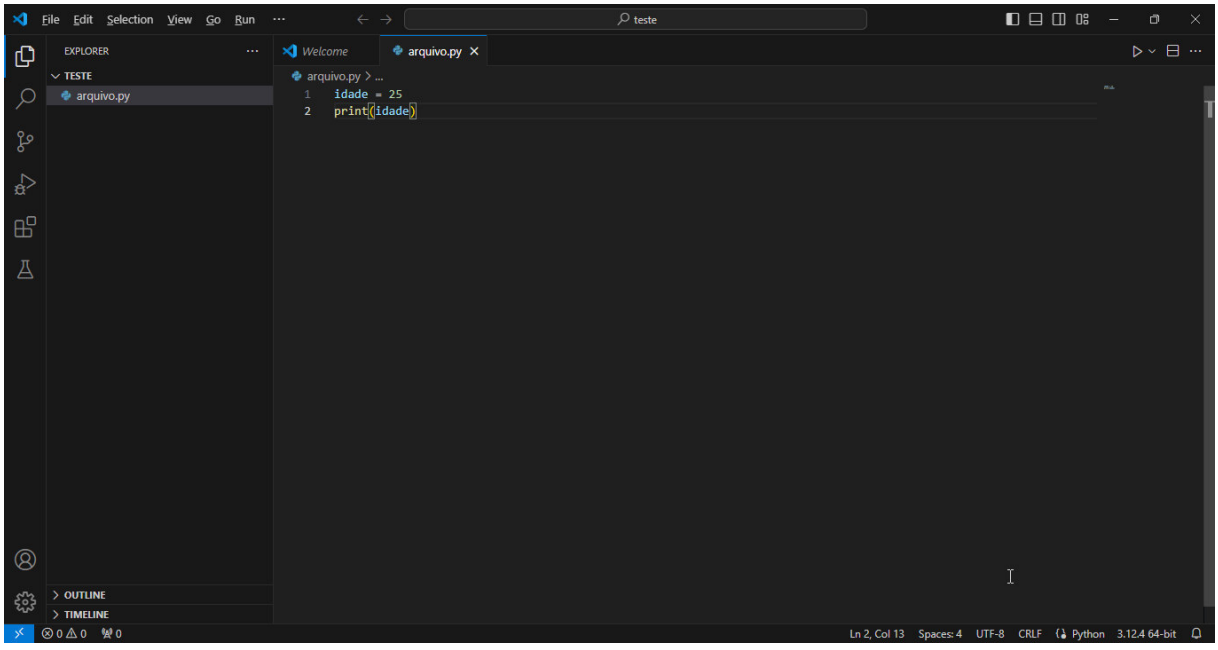


Figura A.17: Preparação de ambiente 17

B Segunda Seção do Apêndice

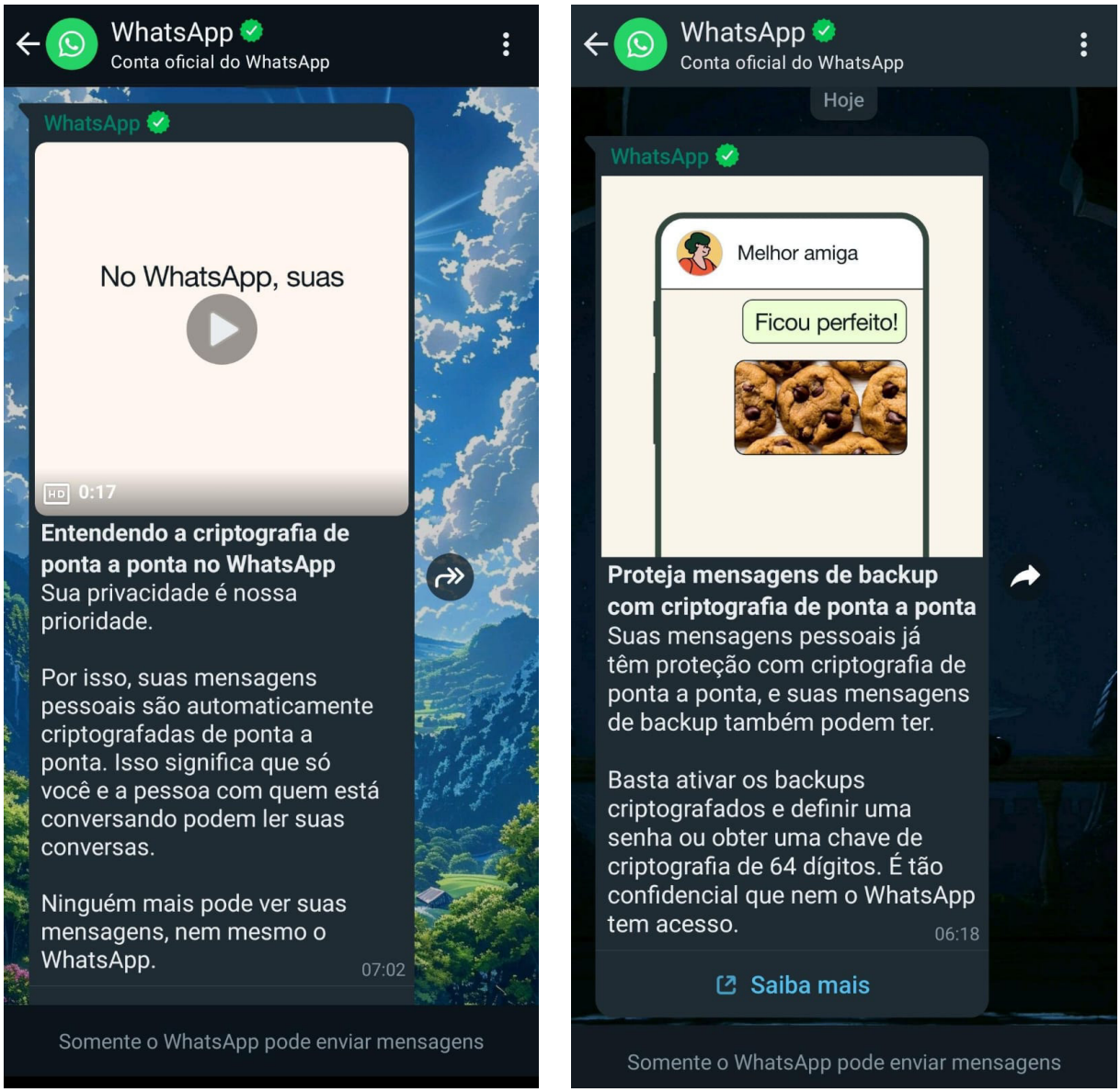


Figura B.1: Mensagem informativa do *Whatsapp*

C Terceira Seção do Apêndice

Há duas "caixinhas", uma para codificação e outra para decodificação. Na "caixinha"da codificação, haverão três espaços em branco com um rótulo em cima, sendo elas: Coeficiente Angular, Coeficiente Linear e Texto. Em cada um dos espaços em branco deve ser preenchido com valores correspondentes ao que o rótulo se refere. Ao completar com os dados, aperte o botão Codificar e a mensagem escolhida será codificada (Figura C.1).

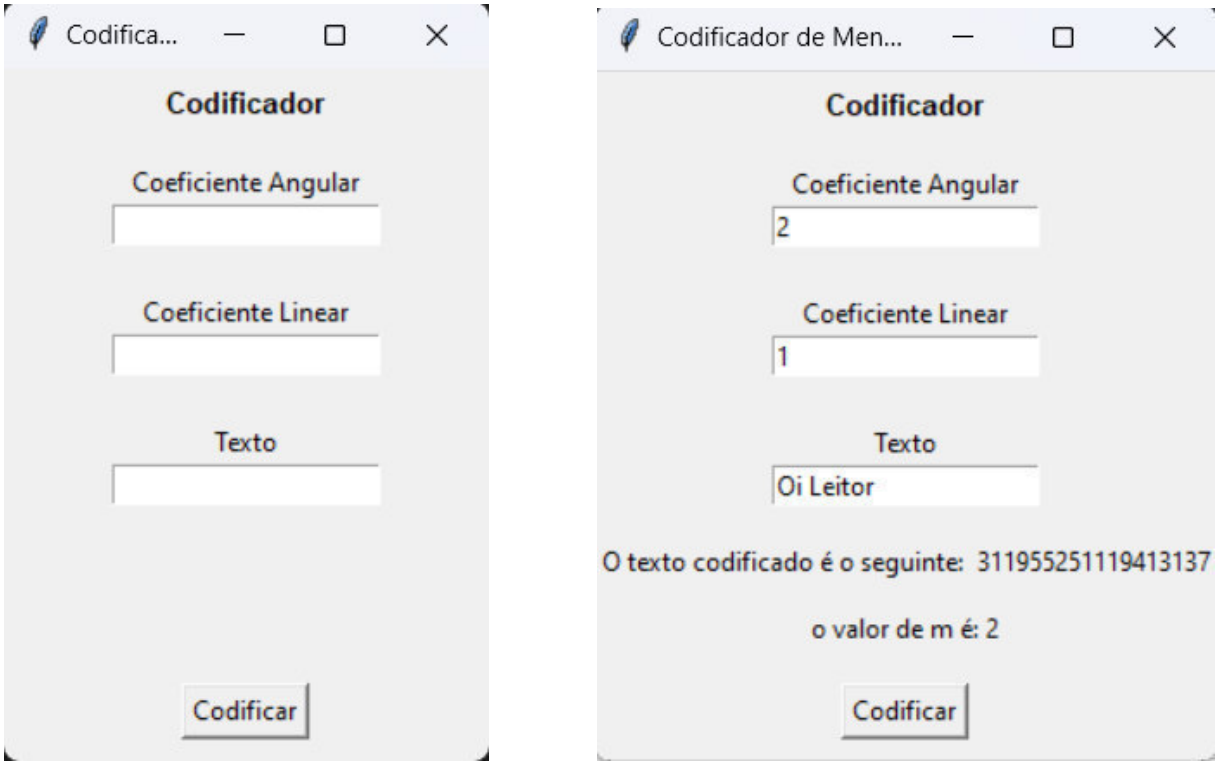


Figura C.1: Interface do Codificador da Função Afim

Semelhante a "caixinha" de codificação, a de Decodificação tem espaços em branco junto com rótulos que indicam qual dado deve ser inserido em cada espaço. Os dados inseridos são os que foram fornecidos e obtidos no passo de codificação. Após preenchimento, aperte o botão Decodificar e a mensagem será decodificada (Figura C.2).

A Figura C.3 apresenta o mesmo sistema que as figuras anteriores, porém servindo para codificar e decodificar mensagens usando funções quadráticas.

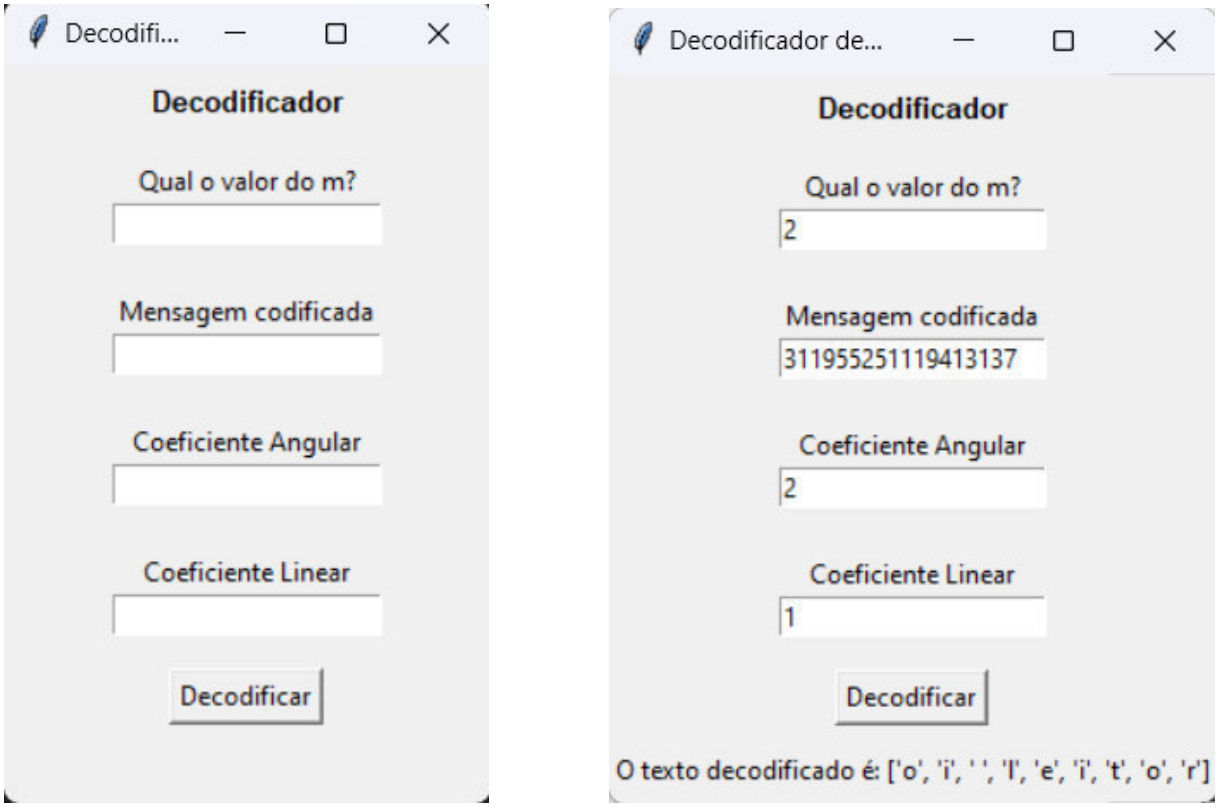


Figura C.2: Interface do Decodificador da Função Afim

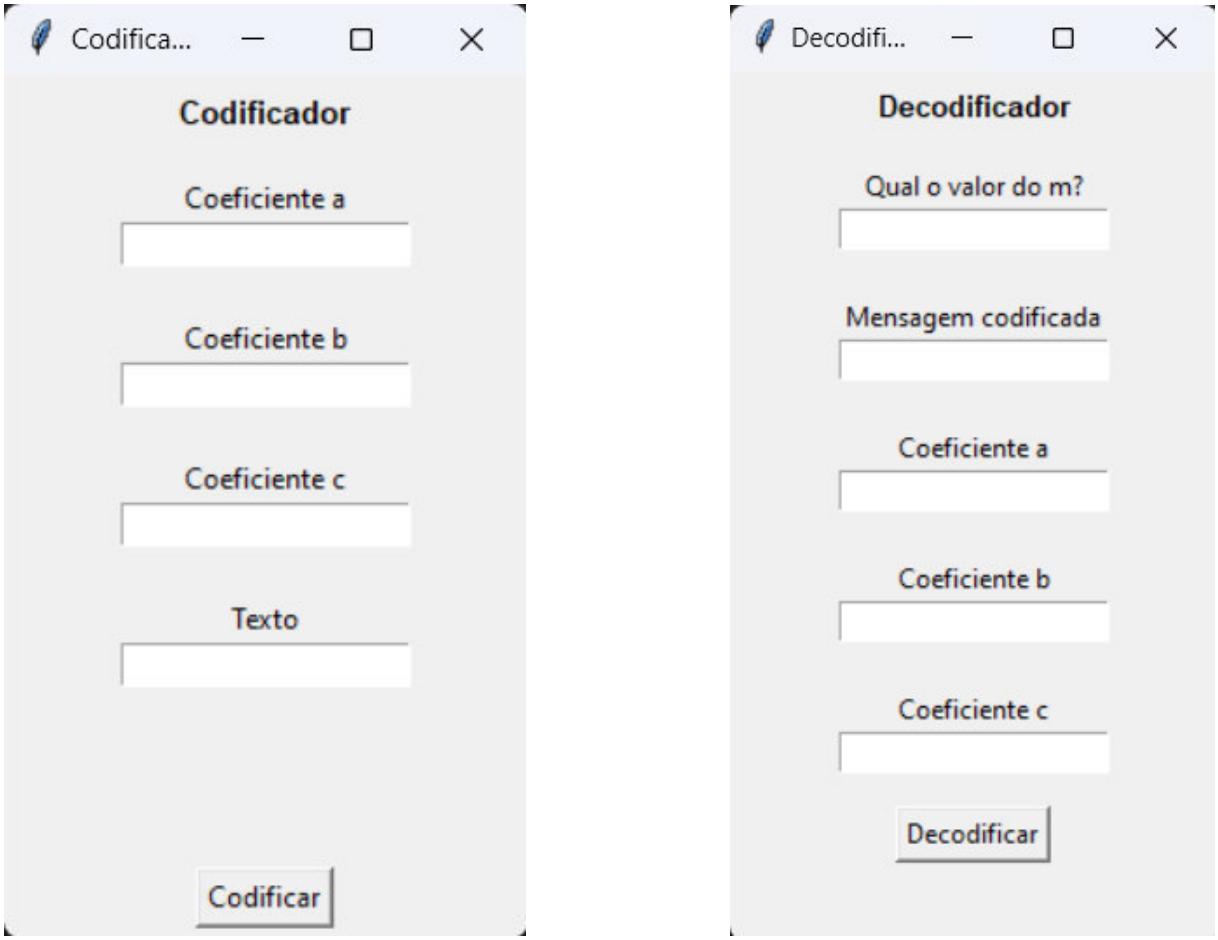


Figura C.3: Interfaces para Função Quadrática

D Quarta Seção do Apêndice

D.1 Codificação Função Afim

Código da ferramenta que codifica mensagens.

```
1 import tkinter as tk
2 from tkinter import messagebox
3
4 def validarTexto(texto):
5     caracteres_permitidos = "abcdefghijklmnopqrstuvwxyz éáç,.-âãõôêí/"
6
7     for char in texto:
8         if char not in caracteres_permitidos:
9             return False
```

```
10      return True
11
12  def codificar():
13      #Entrada do texto a ser codificado
14      texto = str(texto_entrada.get()).lower()
15
16      if not validarTexto(texto):
17          messagebox.showerror("Erro", "O texto contém caracteres especiais
            ↪  não permitidos. Por favor, digite novamente.")
18          return
19
20      #Tabela de codificação
21      correlacao_letra_numero = {
22      'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5,
23      'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10,
24      'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15,
25      'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20,
26      'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25,
27      'z': 26, ' ': 27, 'é': 28, 'á': 29, 'ç': 30,
28      ',': 31, '.': 32, '-': 33, 'â': 34, 'ã': 35,
29      'õ': 36, 'ô': 37, 'ê': 38, 'í': 39, '/': 40
30      }
31
32      #Entrada dos coeficientes
33      a = int(valor_entrada_angular.get())
34      b = int(valor_entrada_linear.get())
35
36      #Pensar no m (cálculo dos extremos)
37      f1 = a*1 + b
38      f2 = a*40 + b
39
40      if len(str(f1)) > len(str(f2)):
```

```
41          m = len(str(f1))
42      else:
43          m = len(str(f2))
44
45      #Separar cada letra do texto
46      lista = []
47      for c in texto:
48          lista.append(c)
49
50      #Codificando o texto
51      cod = []
52      for d in lista:
53          num = correlacao_letra_numero[d]
54          func = a*num + b
55          cod.append(func)
56
57      # transformando o vetor cod em uma palavra
58      codmsg = ' '
59      for f in cod:
60          codmsg = codmsg + str(f).zfill(m)
61
62      #Mensagem codificada
63      label_saida.config(text= f'O texto codificado é o seguinte:
        ↪ {codmsg}')
64      label_m.config(text=f'o valor de m é: {m}')
65
66  #Criar janela principal
67  root = tk.Tk()
68  root.title('Codificador de Mensagens')
69
70  #Widgets
```

```
71  label_título = tk.Label(root, text='Codificador', font=('Arial', '10',
    ↪  'bold'))
72  label_título.pack(pady=5)
73
74  #-------------Container 1-----------------#
75  primeiroContainer = tk.Frame(root)
76  primeiroContainer.pack(pady=10, padx=20)
77
78  label_text = tk.Label(primeiroContainer, text='Texto')
79  label_text.grid(row=0, column=0)
80
81  texto_entrada = tk.Entry(primeiroContainer, fg='black')
82  texto_entrada.grid(row=1, column=0)
83  #---------------------------------------#
84  #-------------Container 2-----------------#
85  segundoContainer = tk.Frame(root)
86  segundoContainer.pack(pady=10, padx=50)
87
88  label_entrada_angular = tk.Label(segundoContainer, text=('Coeficiente
    ↪  Angular'))
89  label_entrada_angular.grid(row=0, column=0)
90
91  valor_entrada_angular = tk.Entry(segundoContainer, fg='black')
92  valor_entrada_angular.grid(row=1, column=0)
93  #---------------------------------------#
94  #-------------Container 3-----------------#
95  terceiroContainer = tk.Frame(root)
96  terceiroContainer.pack(pady=10, padx=20)
97
98  label_entrada_linear = tk.Label(terceiroContainer, text='Coeficiente
    ↪  Linear')
99  label_entrada_linear.grid(row=0, column=0)
```

```
100
101 valor_entrada_linear = tk.Entry(terceiroContainer, fg='black')
102 valor_entrada_linear.grid(row=1, column=0)
103 #----------------------------------------#
104
105 label_saida = tk.Label(root)
106 label_saida.pack(pady=5)
107
108 label_m = tk.Label(root)
109 label_m.pack(pady=5)
110
111 botao_codificar = tk.Button(root, text='Codificar', command=codificar)
112 botao_codificar.pack(pady=10)
113
114 root.mainloop()
```

D.2 Decodificação Função Afim

```
1  import tkinter as tk
2
3  def decodificar():
4      #Tabela de decodificação
5      correlacao_numero_letra = {
6      1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e',
7      6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j',
8      11: 'k', 12: 'l', 13: 'm', 14: 'n', 15: 'o',
9      16: 'p', 17: 'q', 18: 'r', 19: 's', 20: 't',
10     21: 'u', 22: 'v', 23: 'w', 24: 'x', 25: 'y',
11     26: 'z', 27: ' ', 28: 'é', 29: 'á', 30: 'ç',
12     31: ',', 32: '.', 33: '-', 34: 'â', 35: 'ã',
13     36: 'õ', 37: 'ô', 38: 'ê', 39: 'í', 40: '/'
14     }
15
```

```
16      #Entrada dos coeficientes da função
17      lista = []
18      a = int(valor_entrada_angular.get())
19      b = int(valor_entrada_linear.get())
20
21      #Entrada dos valores codificados
22      valores = str(valor_entrada.get())
23
24      m = int(m_entrada.get())
25
26      while valores:
27          dupla = int(valores[:m])
28          lista.append(dupla)
29          valores = valores[m:]
30
31      print(lista)
32      #Decodificando o texto
33      decod = []
34      for c in lista:
35          inver = (c - b)/a
36          letra = correlacao_numero_letra[inver]
37          decod.append(letra)
38
39      #Mensagem decodificada
40      label_resultado.config(text=f'O texto decodificado é: {decod}')
41
42  #Criar janela principal
43  root = tk.Tk()
44  root.title('Decodificador de Mensagens')
45
46  #widgets
```

```
47 label_título = tk.Label(root, text='Decodificador', font=('Arial', '10',
    ↪ 'bold'))
48 label_título.pack(pady=5)
49
50 #-------------Container 1-----------------#
51 primeiroContainer = tk.Frame(root)
52 primeiroContainer.pack(pady=10, padx=50)
53
54 label_m_entrada = tk.Label(primeiroContainer, text=('Qual o valor do
    ↪ m?'))
55 label_m_entrada.grid(row=0, column=0)
56
57 m_entrada = tk.Entry(primeiroContainer, fg='black')
58 m_entrada.grid(row=1, column=0)
59
60 #-----------------------------------------#
61 #-------------Container 2-----------------#
62 segundoContainer = tk.Frame(root)
63 segundoContainer.pack(pady=10, padx=50)
64
65 label_entrada = tk.Label(segundoContainer, text=('Mensagem codificada'))
66 label_entrada.grid(row=0, column=0)
67
68 valor_entrada = tk.Entry(segundoContainer, fg='black')
69 valor_entrada.grid(row=1, column=0)
70 #-----------------------------------------#
71 #-------------Container 3-----------------#
72 terceiroContainer = tk.Frame(root)
73 terceiroContainer.pack(pady=10, padx=50)
74
75 label_entrada_angular = tk.Label(terceiroContainer, text=('Coeficiente
    ↪ Angular'))
```

```
76  label_entrada_angular.grid(row=0, column=0)
77
78  valor_entrada_angular = tk.Entry(terceiroContainer, fg='black')
79  valor_entrada_angular.grid(row=1, column=0)
80  #---------------------------------------#
81  #-------------Container 4----------------#
82  quartoContainer = tk.Frame(root)
83  quartoContainer.pack(pady=10, padx=50)
84
85  label_entrada_linear = tk.Label(quartoContainer, text=('Coeficiente
    ↪  Linear'))
86  label_entrada_linear.grid(row=0, column=0)
87
88  valor_entrada_linear = tk.Entry(quartoContainer)
89  valor_entrada_linear.grid(row=1, column=0)
90
91  #---------------------------------------#
92  botao_decodificador = tk.Button(root, text='Decodificar',
    ↪  command=decodificar)
93  botao_decodificador.pack(pady=5)
94
95  label_resultado = tk.Label(root)
96  label_resultado.pack(pady=5)
97
98  # Iniciar o loop principal
99  root.mainloop()
```

D.3 Codificação Função Quadrática

```
1  import tkinter as tk
2  from tkinter import messagebox
3
4  def validarTexto(texto):
```

```
5       caracteres_permitidos = "abcdefghijklmnopqrstuvwxyz éáç,.-âãõôêí/"
6
7       for char in texto:
8           if char not in caracteres_permitidos:
9               return False
10      return True
11
12  def codificar():
13      texto = str(entrada_texto.get()).lower()
14
15      if not validarTexto(texto):
16          messagebox.showerror("Erro", "O texto contém caracteres especiais
            ↪  não permitidos. Por favor, digite novamente.")
17          return
18
19      #Tabela de codificação
20      correlacao_letra_numero = {
21          'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5,
22          'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10,
23          'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15,
24          'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20,
25          'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25,
26          'z': 26, ' ': 27, 'é': 28, 'á': 29, 'ç': 30,
27          ',': 31, '.': 32, '-': 33, 'â': 34, 'ã': 35,
28          'õ': 36, 'ô': 37, 'ê': 38, 'í': 39, '/': 40
29      }
30
31      #Entrada do texto a ser codificado
32
33      a = int(entrada_a.get())
34      b = int(entrada_b.get())
35      c = int(entrada_c.get())
```

```
36
37      if (-b/(2*a)) <= 1 or (-b/(2*a)) >= 40:
38          #Pensar no m (cálculo dos extremos)
39          f1 = a*1**2 + b*1 + c
40          f2 = a*40**2 + b*40 + c
41
42          if len(str(f1)) > len(str(f2)):
43              m = len(str(f1))
44          else:
45              m = len(str(f2))
46
47          #Separar cada letra do texto
48          lista = []
49          for t in texto:
50              lista.append(t)
51
52          #Codificando o texto
53          cod = []
54          for d in lista:
55              num = correlacao_letra_numero[d]
56              func = a*num**2 + b*num + c
57              cod.append(func)
58
59          #Transformando o vetor cod em uma palavra
60          codmsg = ' '
61          for z in cod:
62              codmsg = codmsg + str(z).zfill(m)
63
64          #Texto Codificado
65          label_resultado.config(text= f'O texto codificado é: {codmsg}')
66          label_m.config(text=f'O valor de m é: {m}')
67          print(codmsg)
```

```
68      else:
69          label_resultado.config(text='Digite novamente')
70
71  #Interface
72  root = tk.Tk()
73  root.title('Codificador de Mensagens')
74
75  #Widgets
76  label_título = tk.Label(root, text='Codificador', font=('Arial', '10',
    ↪  'bold'))
77  label_título.pack(pady=5)
78
79  #-------------Container 1-----------------#
80  primeiroContainer = tk.Frame(root)
81  primeiroContainer.pack(pady=10, padx=50)
82
83  label_entrada_angular = tk.Label(primeiroContainer, text=('Coeficiente
    ↪  a'))
84  label_entrada_angular.grid(row=0, column=0)
85
86  entrada_a = tk.Entry(primeiroContainer, fg='black')
87  entrada_a.grid(row=1, column=0)
88  #---------------------------------------#
89  #-------------Container 2-----------------#
90  segundoContainer = tk.Frame(root)
91  segundoContainer.pack(pady=10, padx=50)
92
93  label_entrada_angular = tk.Label(segundoContainer, text=('Coeficiente
    ↪  b'))
94  label_entrada_angular.grid(row=0, column=0)
95
96  entrada_b = tk.Entry(segundoContainer, fg='black')
```

```
97  entrada_b.grid(row=1, column=0)
98  #----------------------------------------#
99  #-------------Container 3----------------#
100 terceiroContainer = tk.Frame(root)
101 terceiroContainer.pack(pady=10, padx=50)
102
103 label_entrada_angular = tk.Label(terceiroContainer, text=('Coeficiente
    ↪  c'))
104 label_entrada_angular.grid(row=0, column=0)
105
106 entrada_c = tk.Entry(terceiroContainer, fg='black')
107 entrada_c.grid(row=1, column=0)
108 #----------------------------------------#
109 #-------------Container 4----------------#
110 quartoContainer = tk.Frame(root)
111 quartoContainer.pack(pady=10, padx=50)
112
113 label_entrada_angular = tk.Label(quartoContainer, text=('Texto'))
114 label_entrada_angular.grid(row=0, column=0)
115
116 entrada_texto = tk.Entry(quartoContainer, fg='black')
117 entrada_texto.grid(row=1, column=0)
118 #----------------------------------------#
119 label_resultado = tk.Label(root)
120 label_resultado.pack(pady=5)
121
122 label_m = tk.Label(root)
123 label_m.pack(pady=5)
124
125 botao = tk.Button(root, text='Codificar', command=codificar)
126 botao.pack(pady=5)
127
```

```
128
129 #Iniciar o loop principal
130 root.mainloop()
```

D.4 Decodificação Função Quadrática

```
1  import tkinter as tk
2
3  def decodificar():
4      correlacao_numero_letra = {
5          1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e',
6          6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j',
7          11: 'k', 12: 'l', 13: 'm', 14: 'n', 15: 'o',
8          16: 'p', 17: 'q', 18: 'r', 19: 's', 20: 't',
9          21: 'u', 22: 'v', 23: 'w', 24: 'x', 25: 'y',
10         26: 'z', 27: ' ', 28: 'é', 29: 'á', 30: 'ç',
11         31: ',', 32: '.', 33: '-', 34: 'â', 35: 'ã',
12         36: 'õ', 37: 'ô', 38: 'ê', 39: 'í', 40: '/'
13     }
14
15     #Entrada do texto a ser decodificado
16     a = int(entrada_a.get())
17     b = int(entrada_b.get())
18     c = int(entrada_c.get())
19
20     #Entrada dos valores codificados
21     valores = str(entrada_valor.get())
22
23     m = int(m_entrada.get())
24
25     cod = []
26     while valores:
27
```

```
28          trio = int(valores[:m])
29          cod.append(trio)
30          valores = valores[m:]
31
32      #Decodificando o texto
33      decod = []
34
35      for e in cod:
36          c2 = (c - e)
37          delta = b**2 - 4*a*c2
38          raiz1 = (-b + (delta)**(1/2))/(2*a)
39          raiz2 = (-b - (delta)**(1/2))/(2*a)
40
41          if raiz1 in correlacao_numero_letra:
42              num2 = correlacao_numero_letra[raiz1]
43          elif raiz2 in correlacao_numero_letra:
44              num2 = correlacao_numero_letra[raiz2]
45
46          decod.append(num2)
47
48      #Texto Decodificado
49      label_resultado.config(text=f'A mensagem decodificada é: {decod}')
50
51  #Interface
52  root = tk.Tk()
53  root.title('Decodificador de Mensagem')
54
55  #Widgets
56  label_título = tk.Label(root, text='Decodificador', font=('Arial', '10',
    ↪  'bold'))
57  label_título.pack(pady=5)
58
```

```
59  #-------------Container 1-----------------#
60  primeiroContainer = tk.Frame(root)
61  primeiroContainer.pack(pady=10, padx=50)
62
63  label_m_entrada = tk.Label(primeiroContainer, text=('Qual o valor do
    ↪  m?'))
64  label_m_entrada.grid(row=0, column=0)
65
66  m_entrada = tk.Entry(primeiroContainer, fg='black')
67  m_entrada.grid(row=1, column=0)
68  #---------------------------------------#
69  #-------------Container 2-----------------#
70  segundoContainer = tk.Frame(root)
71  segundoContainer.pack(pady=10, padx=50)
72
73  label_entrada = tk.Label(segundoContainer, text=('Mensagem codificada'))
74  label_entrada.grid(row=0, column=0)
75
76  entrada_valor = tk.Entry(segundoContainer, fg='black')
77  entrada_valor.grid(row=1, column=0)
78  #---------------------------------------#
79  #-------------Container 3-----------------#
80  terceiroContainer = tk.Frame(root)
81  terceiroContainer.pack(pady=10, padx=50)
82
83  label_entrada_angular = tk.Label(terceiroContainer, text=('Coeficiente
    ↪  a'))
84  label_entrada_angular.grid(row=0, column=0)
85
86  entrada_a = tk.Entry(terceiroContainer, fg='black')
87  entrada_a.grid(row=1, column=0)
88  #---------------------------------------#
```

```
89  #-------------Container 4-----------------#
90  quartoContainer = tk.Frame(root)
91  quartoContainer.pack(pady=10, padx=50)
92
93  label_entrada_angular = tk.Label(quartoContainer, text=('Coeficiente b'))
94  label_entrada_angular.grid(row=0, column=0)
95
96  entrada_b = tk.Entry(quartoContainer, fg='black')
97  entrada_b.grid(row=1, column=0)
98  #-----------------------------------------#
99  #-------------Container 5-----------------#
100 quintoContainer = tk.Frame(root)
101 quintoContainer.pack(pady=10, padx=50)
102
103 label_entrada_angular = tk.Label(quintoContainer, text=('Coeficiente c'))
104 label_entrada_angular.grid(row=0, column=0)
105
106 entrada_c = tk.Entry(quintoContainer, fg='black')
107 entrada_c.grid(row=1, column=0)
108 #-----------------------------------------#
109 botao = tk.Button(root, text='Decodificar', command=decodificar)
110 botao.pack(pady=5)
111
112 label_resultado = tk.Label(root)
113 label_resultado.pack(pady=5)
114
115 #Iniciar o loop principal
116 root.mainloop()
```