UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO CIÊNCIA EXATA E DA TERRA
DEPARTAMENTO DE MATEMÁTICA
MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL

Marcos César Cabral Galvão

Ensino e Aprendizagem da Matemática na Educação Básica Utilizando Tecnologias e Desenvolvendo Pensamento Computacional: Abordagem com Scratch, Portugol, Python e Geogebra

Natal-RN
2021

Marcos César Cabral Galvão

Ensino e Aprendizagem da Matemática na Educação Básica Utilizando Tecnologias e Desenvolvendo Pensamento Computacional: Abordagem com Scratch, Portugol, Python e Geogebra

Dissertação apresentada ao Corpo Docente do Mestrado Profissional em Matemática em Rede Nacional - PROFMAT - CCET - UFRN, como requisito parcial para obtenção do título de Mestre em Matemática

UFRN

Orientador: Prof. Dr. Ronaldo Freire de Lima

Natal-RN
2021

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET

Galvão, Marcos César Cabral.
Ensino e aprendizagem da matemática na educação básica utilizando tecnologias e desenvolvendo pensamento computacional: abordagem com Scratch, Portugol, Python e Geogebra / Marcos César Cabral Galvão. - 2021.
164f.: il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Departamento de Matemática, Mestrado Profissional em Matemática em Rede Nacional - PROFMAT. Natal, 2021.
Orientador: Prof. Dr. Ronaldo Freire de Lima.

1. Matemática - Dissertação. 2. Tecnologias para o ensino de matemática - Dissertação. 3. Pensamento computacional - Dissertação. 4. Algoritmos - Dissertação. 5. Programação de computadores - Dissertação. I. Lima, Ronaldo Freire de. II. Título.

RN/UF/CCET CDU 51

Elaborado por Joseneide Ferreira Dantas - CRB-15/324

MARCOS CÉSAR CABRAL GALVÃO

Ensino e Aprendizagem da Matemática na Educação Básica Utilizando Tecnologias e Desenvolvendo Pensamento Computacional: Abordagem com Scratch, Portugol, Python e Geogebra

Dissertação apresentada ao Corpo Docente do Mestrado Profissional em Matemática em Rede Nacional - PROFMAT - CCET - UFRN, como requisito parcial para obtenção do título de Mestre em Matemática

Natal-RN, 30 de Agosto de 2021:

Prof. Dr. Ronaldo Freire de Lima
PROFMAT-UFRN
Presidente da Banca

Prof. Dr. Carlos Alexandre Gomes da Silva
PROFMAT-UFRN

Prof. Dr. Fagner Lemos de Santana
PROFMAT-UFRN

Prof. Dr. Joaquim Elias de Freitas
convidado UFRN

Natal-RN
2021

Dedico este trabalho a DEUS, à minha família e a todos que buscam incessantemente por conhecimento.

AGRADECIMENTOS

Agradeço a Deus pelas graças em minha vida; À minha esposa, Gardênia, e aos meus filhos, Gabriel, Rafael e Camila, pela compreensão das horas de convivência que lhes faltaram durante meus trabalhos e estudos; À minha mãe, Regina, pelas horas gastas me ensinando em minha infância; Ao meu pai, Fred *(in memoriam)*, pelo exemplo de vida; Ao meu avô João Cabral *(in memoriam)* pelas lições de vida, ensinamentos filosóficos e pelo livro “O Homem que Calculava", Malba Tahan, que despertou em mim interesse pela matemática; Ao meu tio Gileno *(in memoriam)* pela amizade e alegria da convivência; Ao meu tio João Caldas *(in memoriam)* pelas poucas mas valorosas aulas de matemática em que me ensinou a “Tabuada do nove nos dedos"e narrou a história da “Soma de Gauss"; Ao cardiologista, Dr Paulo Bitencourt *(in memoriam)*, pelos problemas e indagações com abordagem matemática, notadamente o problema “um tijolo pesa um quilo e meio tijolo. Quanto pesa um tijolo e meio?” ficou marcado em minha memória.
Agradeço pelas oportunidades que tive: estudar no Colégio São Judas Tadeu - CSJT/SP (1976-1983), notadamente pelo Curso Técnico em Eletrônica (antigo 2° grau); Ensinar Eletricidade Básica e Matemática (1984) no curso profissionalizante do Serviço Nacional da Indústria - SENAI/CE, unidade Barra do Ceará - primorosa qualidade do material didático no formato apostilas; Cursar Bacharelado em Matemática na Universidade de Fortaleza - UNIFOR/CE (1984-1988) e ter tido excelentes professores como Ulisses, Américo, Sônia, Plácido, Raimundo Wilson, Noé e em especial ao Prof. Francisco Cavalcanti *(in memoriam)*; Ensinar diversas disciplinas no Curso de Bacharelado de Informática (1988-2002) - o magistério contribuiu muito para meu crescimento pessoal e profissional.
Agradeço às Instituições que viabilizam o funcionamento do Mestrado Profissionalizante em Matemática em Rede Nacional - PROFMAT: Coordenação de Aperfeiçoamento Pessoal de Nível Superior (CAPES) do Ministério da Educação, Sociedade Brasileira de Matemática (SBM), Instituto Nacional de Matemática Pura e Aplicada (IMPA) e Universidade Federal do Rio Grande do Norte (UFRN).
Agradeço: ao meu orientador Prof. Dr. Ronaldo Freire de Lima, pela aceitação do tema escolhido e paciência com meu ritmo de trabalho em tempos de pandemia; à Coordenadora Profa. Dra. Débora Borges Ferreira e aos substitutos Profa. Dra Viviane Simioli Medeiros Campos e Prof. Dr. Fagner Lemos de Santana pela condução do programa nos anos que estive como discente; aos professores Profa. Dra. Gabriela Lucheze de Oliveira Lopes, Profa. Dra. Débora Borges Ferreira, Prof. Dr. Edgar Silva Pereira, Prof. Dr. Fagner Lemos de Santana, Prof. Dr. Jaques Silveira Lopes, Prof. Dr. Carlos Alexandre Gomes da Silva e Prof. Dr. Paulo Roberto Ferreira dos Santos Silva que ministraram disciplinas com excelência para a Turma 2019.
Por fim, agradeço aos colegas discentes pela agradável convivência durante o período de curso: Renato, Mônica, Sabrina, Iago, João Victor, Cleiton, Rosângela, Lucylla, Alexandre, Gisalmir, Acácio e Adriano.

RESUMO

Este trabalho propõe formas de utilizar programas de computadores para ensino e aprendizagem da matemática desenvolvendo o pensamento computacional. As modalidades propostas são “Laboratório de Matemática”, “Jogos e Gamificação” e “Construção de Algoritmos e Programação”. As duas primeiras são conceituadas e têm seus usos justificados, tendo Laboratório de Matemática exemplos com o *software* Geogebra nas modalidades Geogebra Calculadora Gráfica para estudo do comportamento de uma função quadrática, Geogebra Geometria para estudo de polígonos regulares circuncêntricos e Geogebra CAS (*Computer Algebra System*) - para fatorar alguns Números de Fermat e Mersenne. A modalidade Construção de Algoritmos e Programação é explorada de forma mais exaustiva com o Scratch, Portugol e Python. Geometria é abordada com a proposta Construcionista de Papert, de forma similar à Geometria da Tartaruga da linguagem LOGO, em construções de triângulos, quadrados e polígonos regulares. A Lógica Matemática desenvolvida por Boole e De Morgan é abordada com Diagramas de Venn e tem importância destacada para construções de expressões das estruturas de controle condicionais e de repetições que controlam fluxos em algoritmos e programas. O Triângulo de Pascal é utilizado como elemento matemático motivador para exploração de sequências, dentre elas: soma dos naturais e de Fibonacci que são desenvolvidas computacionalmente nas formas iterativas e recursivas. Divisibilidade, números primos e compostos, Crivo de Eratóstenes, Algoritmo de Euclides para cálculo do Máximo Divisor Comum - MDC, Teorema Fundamental da Aritmética e Fatoração, Sistemas de Numeração nas Bases Binária, Decimal e Hexadecimal são alguns dos algoritmos discutidos e implementados em Scratch, Portugol e Python.

Palavras-chaves: Tecnologias para o Ensino de Matemática. Pensamento Computacional. Algoritmos.

ABSTRACT

This work proposes ways to use computer programs for teaching and learning mathematics, developing computational thinking. The proposed modalities are “Mathematics Laboratory”, “Games and Gamification” and “Algorithm Construction and Programming”. The first two are conceptualized and have their uses justified, with examples with the Mathematical Laboratory using Geogebra software in the modalities Geogebra Graphical Calculator to study the behavior of a quadratic function, Geogebra Geometry to study circumcentric regular polygons and Geogebra CAS (Computer Algebra System) - to factor out some Fermat and Mersenne Numbers. The Algorithm Construction and Programming modality is more exhaustively explored with Scratch, Portugol and Python. Geometry is approached with Papert’s Constructionist proposal, in a similar way to the Turtle Geometry of the LOGO language, in constructions of triangles, squares and regular polygons. The Mathematical Logic developed by Boole and De Morgan is approached with Venn Diagrams and has highlighted importance for expression constructions of the conditional and repetition control structures that control flows in algorithms and programs. The Pascal Triangle is used as a motivating mathematical element for exploring sequences, including: sum of natural and Fibonacci sequences that are computationally developed in iterative and recursive forms. Divisibility, prime and composite numbers, Sieve of Eratosthenes, Euclid’s Algorithm for calculating the Greatest Common Divisor - GCD, Fundamental Theorem of Arithmetic and Factorization, Numbering Systems in Binary, Decimal and Hexadecimal Bases are some of the algorithms discussed and implemented in Scratch, Portugol and Python.

Keywords: Technologies for Teaching Mathematics. Computational Thinking. Algorithms.

LISTA DE ALGORITMOS

1 Entrada e saída de dados . . . 35
2 Atribuição de valor a variável . . . 36
3 Estrutura condicional simples - Se/Entao . . . 39
4 Estrutura condicional composta - Se/Entao/Senao . . . 39
5 Estrutura Enquanto - exibição de 1 a 10 . . . 41
6 Estrutura Enquanto - exibição de 10 a 1 . . . 42
7 Estrutura Repita até - Par ou ímpar em loop . . . 42
8 Estrutura Para - exibição dos pares de 2 a 20 . . . 43
9 Estrutura Repita até que - exibição de 1 a 10 . . . 44
10 Estrutura Repita n vezes - exibição de 1 a 10 . . . 44
11 Procedimento exibe números pares até n . . . 45
12 Função calcula dobro . . . 45

13 Cálculo da soma Sn dos naturais utilizando iterações . . . 56
14 Cálculo do n−ésimo termo da sequência de Fibonacci . . . 59

15 Contagem recursiva 0 até n progressivamente . . . 64
16 Contagem recursiva 0 até n regressivamente . . . 64
17 Cálculo recursivo de Fatorial de n - $n! = n * (n-1)!$ . . . 67
18 Cálculo recursivo da Soma dos n primeiros naturais - $S_n = n + S_{n-1}$ . . . 68
19 Cálculo recursivo do n−ésimo termo sequência Fibonacci . . . 69
20 Torre de Hanoi . . . 71

21 Exibe os divisores de um número . . . 73
22 Calcula MDC de dois valores - Algoritmo de Euclides . . . 75
23 Classifica um número como Primo ou Composto . . . 78
24 Números primos até 100 - Crivo de Eratóstenes . . . 80
25 Fatoração de um número n $(n > 1)$ . . . 82

26 Conversão de número na base decimal para base binária . . . 91
27 Conversão de número na base binária para base decimal . . . 92
28 Conversão de número na base decimal para base hexadecimal . . . 93
29 Conversão de número na base hexadecimal para base decimal . . . 95

LISTA DE ILUSTRAÇÕES

Figura 1 – OBI2020 - Corrida na Floresta 9
Figura 2 – OBI2019 - Parcelamento Sem Juros 9
Figura 3 – Trilhas para aprendizagem de programação de computadores 10
Figura 4 – Aplicativos Geogebra 12
Figura 5 – Geogebra Materiais 12
Figura 6 – Geogebra Calculadora Grafica - Equação 2^o grau 13
Figura 7 – Algoritmo de Euclides com Voz em Scratch 17
Figura 8 – Ambiente Scratch 18
Figura 9 – Palco e Categorias para Desenho em Scratch 22
Figura 10 – Construção de Quadrado em Scratch 22
Figura 11 – Construção de Triangulo em Scratch 23
Figura 12 – Quadrado e Triângulo com Repita n Vezes 24
Figura 13 – Bloco Triangulo em Scratch 24
Figura 14 – Bloco Quadrado em Scratch 25
Figura 15 – Quadrados Aleatórios em Scratch 25
Figura 16 – Bloco Polígono Regular em Scratch 26
Figura 17 – Polígonos Regulares Circunscritos em Scratch 27
Figura 18 – Polígonos Regulares Circunscritos em Geogebra 28
Figura 19 – Tabela parcial ASCII - *American Standard Code for Information Interchange* 32
Figura 20 – Blocos das Categorias Variáveis e Operadores do Scratch 33
Figura 21 – Blocos das Categorias Aparência e Sensores do Scratch 36
Figura 22 – Codificação do Algoritmo 1 em Scratch 37
Figura 23 – Primeiro Projeto Scratch 37
Figura 24 – Fluxos das Estruturas de Controle Condicional 38
Figura 25 – Exemplos de Controle Condicional em Scratch 40
Figura 26 – Fluxos das Estruturas de Controle de Repetição 41
Figura 27 – Fluxos das Estruturas de Repetição do Scratch 43
Figura 28 – Diagramas Venn: Disjunção, Conjunção e Complemento 46
Figura 29 – Lógica em termos de álgebra dos conjuntos 47
Figura 30 – Diagrama de Venn - Exemplo operador OU: Aprovação 48
Figura 31 – Diagrama de Venn - Exemplo operador E: Conceito B 49
Figura 32 – Diagrama de Venn - Exemplo Negação B 50
Figura 33 – Diagrama De Morgan 01 51
Figura 34 – Diagrama De Morgan 02 51
Figura 35 – Triângulo de Pascal 52
Figura 36 – Triângulo de Pascal na Forma Matricial 53

Figura 37 – Triângulo de Pascal construído como relação de conjuntos 54
Figura 38 – Iterações no cálculo da Soma S_n dos Naturais 56
Figura 39 – Tela da Soma S_n dos Naturais em Scratch 57
Figura 40 – Sequências dos Números Ímpares, Soma de Termos e Ternos Pitagóricos . . 58
Figura 41 – Iterações no cálculo do n-ésimo termo de Fibonacci 60
Figura 42 – Cálculo do n-ésimo termo de Fibonacci em Scratch 60
Figura 43 – Sequência de Fibonacci no Triângulo de Pascal 61
Figura 44 – Funcionamento do Algoritmo 15 contagem recursiva 65
Figura 45 – Contagem Recursiva em Scratch 66
Figura 46 – Cálculo recursivo de Fatorial de n em Scratch 67
Figura 47 – Cálculo recursivo de Fatorial de n em Scratch 68
Figura 48 – Sequência de Chamadas Recursivas para $Fibo(4)$ 69
Figura 49 – Cálculo recursivo do n-ésimo termo de Fibonacci em Scratch 70
Figura 50 – Sequência de Cálculo Recursivo para Fibo(7) 70
Figura 51 – Torre de Hanoi 71
Figura 52 – Divisores de um Número em Scratch 73
Figura 53 – Cálculo do MDC entre 36 e 15 75
Figura 54 – Cálculo do Máximo Divisor Comum (MDC) em Scratch 76
Figura 55 – Primo ou Composto em Scratch 78
Figura 56 – Crivo de Eratóstenes em Scratch 81
Figura 57 – Telas do Crivo de Eratóstenes em Scratch 81
Figura 58 – Fatoração de um Número em Scratch 83
Figura 59 – Triângulo de Pascal com destaque para abordagem do PTF 84
Figura 60 – Cálculos com Geogebra CAS 87
Figura 61 – Valores possíveis para um byte nos sistemas Decimal, Binário e Hexadecimal 89
Figura 62 – Conversão de n para base b 90
Figura 63 – Conversão de um valor Decimal para Binário em Scratch 92
Figura 64 – Conversão de um valor Binário para Decimal em Scratch 93
Figura 65 – Conversão valor Decimal para Hexadecimal em Scratch 94
Figura 66 – Conversão valor Hexadecimal para Decimal em Scratch 95
Figura 67 – Portugol Studio Principal 103
Figura 68 – Portugol Studio Edição 104
Figura 69 – Entrada e Saida em Portugol 108
Figura 70 – Preco Com Desconto em Portugol 108
Figura 71 – Controle Condicional Simples em Portugol 109
Figura 72 – Controle Condicional Composta em Portugol 110
Figura 73 – Exibe 1 a 10 com Enquanto em Portugol 111
Figura 74 – Exibe 10 a 1 com Enquanto em Portugol 111
Figura 75 – Par ou Impar com repetição até 0 em Portugol 112
Figura 76 – Pares de 2 a 20 com Para em Portugol 113

Figura 77 – Exibição de 1 a 10 em Portugol 113
Figura 78 – Repita 10 Vezes em Portugol 114
Figura 79 – Procedimento Exibe Pares em Portugol 115
Figura 80 – Função Dobro em Portugol 116
Figura 81 – Soma iterativa dos Naturais em Portugol 117
Figura 82 – Fibonacci Iterativo em Portugol 118
Figura 83 – Procedimento Contagem01 em Portugol 119
Figura 84 – Procedimento Contagem02 em Portugol 120
Figura 85 – Fatorial Recursivo em Portugol 121
Figura 86 – Soma dos Naturais Recursiva em Portugol 122
Figura 87 – Fibonacci Recursivo em Portugol 123
Figura 88 – Hanoi em Portugol 124
Figura 89 – Divisores de n em Portugol 125
Figura 90 – MDC em Portugol 126
Figura 91 – Primo ou Composto em Portugol 127
Figura 92 – Crivo de Eratostenes em Portugol 128
Figura 93 – Fatoração de n em Portugol 129
Figura 94 – Conversão base decimal para base binário em Portugol 130
Figura 95 – Conversão base binário para base decimal em Portugol 131
Figura 96 – Conversão base decimal para base hexadecimal em Portugol 132
Figura 97 – Conversão base hexadecimal para base decimal em Portugol 133
Figura 98 – Ambiente OnlineGDB 134
Figura 99 – Ambiente Google Colab 135
Figura 100 – Entrada e Saída em Python 138
Figura 101 – Preco com Desconto em Python 139
Figura 102 – Se/Entao em Python 140
Figura 103 – Se/Entao/Senao em Python 141
Figura 104 – Exibição de 1 a 10 com *while* em Python 142
Figura 105 – Exibição de 10 a 1 com *while* em Python 143
Figura 106 – Par ou Impar com repetição até 0 em Python 144
Figura 107 – Pares de 2 a 20 com *for* e *range* em Python 145
Figura 108 – Repita até que em Python 146
Figura 109 – Repita 10 Vezes com *for* e *range* em Python 147
Figura 110 – Procedimento Exibe Pares Ate N em Python 148
Figura 111 – Funcao Dobro em Python 148
Figura 112 – Soma Iterativa dos Naturais em Python 149
Figura 113 – Fibonacci iterativo em Python 150
Figura 114 – Contagem Progressiva Recursiva em Python 151
Figura 115 – Contagem Regressiva Recursiva em Python 152
Figura 116 – Fatorial Recursivo em Python 153

Figura 117 – Soma Recursiva dos Naturais em Python 154
Figura 118 – Fibonacci Recursivo em Python 155
Figura 119 – Torre de Hanoi em Python 155
Figura 120 – Divisores de N em Python 156
Figura 121 – MDC em Python 157
Figura 122 – Primo ou Composto em Python 158
Figura 123 – Crivo de Eratostenes em Python 159
Figura 124 – Fatora N em Python 160
Figura 125 – Conversão decimal para binário em Python 161
Figura 126 – Conversão binário para decimal em Python 162
Figura 127 – Conversão decimal para hexadecimal em Python 163
Figura 128 – Conversão hexadecimal para decimal em Python 164

SUMÁRIO

	INTRODUÇÃO	1
1	USO DO COMPUTADOR NA EDUCAÇÃO BÁSICA	4
1.1	Laboratório de Matemática	5
1.2	Jogos e Gamificação	6
1.3	Algoritmos e Programação	8
1.4	Principais barreiras e como mitigá-las	11
1.5	Geogebra	11
1.6	Scratch	16
1.7	Portugol	19
1.8	Python	19
2	GEOMETRIA E CONSTRUCIONISMO DE PAPERT	21
2.1	Construção de um Quadrado	22
2.2	Construção de um Triângulo Equilátero	23
2.3	Blocos: Triângulo, Quadrado e Polígono	24
2.4	Polígonos Regulares Circunscritos	26
2.5	Geogebra - Polígonos Regulares Circunscritos	27
3	ALGORITMOS E PROGRAMAÇÃO	29
3.1	Tipos de Dados, Variáveis e Operações	30
3.2	Interação e Manipulação de Dados	35
3.3	Estrutura de Controle Condicional	38
3.4	Estruturas de Controle de Repetições	40
3.5	Construindo Abstrações	44
4	LÓGICA E ÁLGEBRA DE BOOLE	46
4.1	Disjunção Inclusiva - operador ou	47
4.2	Conjunção - operador e	48
4.3	Negação ou Complemento - operador não	50
4.4	As Leis de De Morgan	51
5	SEQUÊNCIAS E SOLUÇÕES COMPUTACIONAIS ITERATIVAS	52
5.1	Triângulo de Pascal	52
5.2	Naturais, Axiomas de Peano e Indução Finita	55
5.3	Soma dos Naturais e Gauss	55
5.4	Sequência de Fibonacci	58

6 RECORRÊNCIAS E SOLUÇÕES COMPUTACIONAIS RECURSIVAS 62
6.1 Recorrências Lineares de Primeira . . . 62
6.2 Recorrências Lineares de Segunda Ordem . . . 63
6.3 Soluções Computacionais Recursivas . . . 64

7 DIVISIBILIDADE E NÚMEROS PRIMOS . . . 72
7.1 Divisores de um Número . . . 73
7.2 Máximo Divisor Comum (MDC) - Algoritmo de Euclides . . . 74
7.3 Número Primo . . . 76
7.4 Crivo de Eratóstenes . . . 79
7.5 Fatoração - Teorema da Fatoração Única . . . 82
7.6 Geogebra CAS . . . 87

8 SISTEMAS DE NUMERAÇÃO . . . 88
8.1 Conversão Inteiro Base Decimal $\rightarrow$ Binária . . . 91
8.2 Conversão Inteiro Base Binária $\rightarrow$ Decimal . . . 92
8.3 Conversão Inteiro Base Decimal $\rightarrow$ Hexadecimal . . . 93
8.4 Conversão Inteiro Base Hexadecimal $\rightarrow$ Decimal . . . 94

CONSIDERAÇÕES FINAIS . . . 97

ÍNDICE REMISSIVO . . . 98

REFERÊNCIAS . . . 100

A APÊNDICE PORTUGOL . . . 103
A.1 Ambiente . . . 103
A.2 Tipos de Dados, Variáveis e Operações . . . 104
A.3 Interação e Manipulação de Dados . . . 105
A.4 Estrutura de Controle Condicional . . . 105
A.5 Estruturas de Controle de Repetições . . . 106
A.6 Algoritmo 1 em Portugol - Entrada e Saída . . . 108
A.7 Algoritmo 2 em Portugol - Atribuição a variável . . . 108
A.8 Algoritmo 3 em Portugol - Se/Entao . . . 109
A.9 Algoritmo 4 em Portugol - Se/Entao/Senao . . . 110
A.10 Algoritmo 5 em Portugol - Enquanto: 1 a 10 . . . 111
A.11 Algoritmo 6 em Portugol - Enquanto: 10 a 1 . . . 111
A.12 Algoritmo 7 em Portugol - Repita até 0 (Par ou ímpar) . . . 112
A.13 Algoritmo 8 em Portugol - Para: Pares de 2 a 20 . . . 113
A.14 Algoritmo 9 em Portugol - Repita até que: 1 a 10 . . . 113
A.15 Algoritmo 10 em Portugol - Repita n vezes: 1 a 10 . . . 114
A.16 Algoritmo 11 em Portugol: Procedimento exibe pares até n . . . 115

A.17 Algoritmo 12 em Portugol: Função calcula dobro 116
A.18 Algoritmo 13 em Portugol - Soma iterativa dos naturais 117
A.19 Algoritmo 14 em Portugol - Termo de Fibonacci iterativo 118
A.20 Algoritmo 15 em Portugol - Contagem recursiva progressiva . . . 119
A.21 Algoritmo 16 - em Portugol - Contagem recursiva regressiva . . . 120
A.22 Algoritmo 17 em Portugol - Fatorial recursivo 121
A.23 Algoritmo 18 em Portugol - Soma recursiva dos naturais 122
A.24 Algoritmo 19 em Portugol - Termo de Fibonacci recursivo 123
A.25 Algoritmo 20 em Portugol: Torre de Hanoi 124
A.26 Algoritmo 21 em Portugol - Divisores de um número 125
A.27 Algoritmo 22 em Portugol - MDC: Algoritmo de Euclides 126
A.28 Algoritmo 23 em Portugol - Primo ou Composto 127
A.29 Algoritmo 24 em Portugol - Crivo de Eratóstenes: primos até 100 128
A.30 Algoritmo 25 em Portugol - Fatoração 129
A.31 Algoritmo 26 em Portugol - Base decimal para binária 130
A.32 Algoritmo 27 em Portugol: Conversão binário para decimal 131
A.33 Algoritmo 28 em Portugol: Base decimal para hexadecimal 132
A.34 Algoritmo 29 em Portugol: Conversão hexadecimal para decimal 133

B APÊNDICE PYTHON 134
B.1 Ambiente 134
B.2 Tipos de Dados, Variáveis e Operações 135
B.3 Interação e Manipulação de Dados 136
B.4 Estrutura de Controle Condicional 136
B.5 Estruturas de Controle de Repetições 137
B.6 Algoritmo 1 em Python - Entrada e Saída 138
B.7 Algoritmo 2 em Python - Atribuição a variável 139
B.8 Algoritmo 3 em Python - Se/Entao 140
B.9 Algoritmo 4 em Python - Se/Entao/Senao 141
B.10 Algoritmo 5 em Python - Enquanto: 1 a 10 142
B.11 Algoritmo 6 em Python - Enquanto: 10 a 1 143
B.12 Algoritmo 7 em Python - Repita até 0 (Par ou ímpar) 144
B.13 Algoritmo 8 em Python - Para: pares de 2 a 20 145
B.14 Algoritmo 9 em Python - Repita até que: 1 a 10 146
B.15 Algoritmo 10 em Python - Repita n vezes: 1 a 10 147
B.16 Algoritmo 11 em Python: Procedimento exibe pares até n 148
B.17 Algoritmo 12 em Python: Função calcula dobro 148
B.18 Algoritmo 13 em Python - Soma iterativa dos naturais 149
B.19 Algoritmo 14 em Python - Termo de Fibonacci iterativo 150
B.20 Algoritmo 15 em Python - Contagem recursiva progressiva 151

B.21 Algoritmo 16 em Python - Contagem recursiva regressiva 152
B.22 Algoritmo 17 em Python - Fatorial recursivo 153
B.23 Algoritmo 18 em Python - Soma recursiva dos naturais 154
B.24 Algoritmo 19 em Python - Termo de Fibonacci recursivo 155
B.25 Algoritmo 20 em Python: Torre de Hanoi 155
B.26 Algoritmo 21 em Python - Divisores de um número 156
B.27 Algoritmo 22 em Python - MDC: Algoritmo de Euclides 157
B.28 Algoritmo 23 em Python - Primo ou Composto 158
B.29 Algoritmo 24 e Python - Crivo de Eratóstenes: primos até 100 . . 159
B.30 Algoritmo 25 em Python - Fatoração 160
B.31 Algoritmo 26 em Python - Base decimal para binária 161
B.32 Algoritmo 27 em Python: Conversão binário para decimal 162
B.33 Algoritmo 28 em Python: Base decimal para hexadecimal 163
B.34 Algoritmo 29 em Python: Conversão hexadecimal para decimal . 164

INTRODUÇÃO

Este trabalho apresenta as seguintes características: **Problema de pesquisa -** "No ensino da matemática na educação básica é possível utilizar *softwares* para apresentar conceitos e estimular o aprendizado, assim como desenvolver no aluno o pensamento computacional ?". **Objetivo Geral -** Apresentar estratégias de ensino e aprendizagem da matemática utilizando o computador como ferramenta, assim como desenvolver o pensamento computacional para elaboração de algoritmos e programação de computadores. **Objetivos Específicos -** Justificar o uso de tecnologias no ensino da matemática; Propor formas de uso de tecnologias no ensino e aprendizagem da matemática na educação básica; Enumerar as principais barreiras para o uso de tecnologias no ensino da matemática e propor soluções para mitigá-las; Apresentar problemas matemáticos e formas de abordagem com o uso de tecnologias; Elaborar algoritmos e programas de computador com abordagem centrada na matemática. **Metodologia** - A metodologia adotada foi a pesquisa bibliográfica com o levantamento de conhecimentos enunciados por diferentes autores. Os conceitos e fatos referenciados foram encadeados e articulados formando uma sequência coerente na abordagem dos diversos conteúdos. **Revisão bibliográfica** - Para abordagem de conceitos matemáticos e contextualizações históricas foram utilizadas as obras Boyer (2012), Garbi (2010), Coutinho (2011), Andrade (RPM83), Coelho (2014), Euclides (2009), Hefez (2016), Moreira (2010), Morgado (2015), Singh (2019), Neto (2013). BRASIL (2018) e BRASIL (1996) foram utilizados como bases normativa e de legislação brasileira. Utilização de laboratórios matemática no ensino foi fundamentada com os trabalhos Abreu (1997), Borba (2019), Lucena (2017), UNESP (2021), Geogebra (2021). Para Jogos e Gameficação foram trazidos argumentos dos trabalhos Alves (2015), Huizinga (2014), McGonigal (2012), Prensky (2012). Construções de Algoritmos e Programação foram respaldadas com os trabalhos de Papert (1988), Campos (2013), Cormen (2002), SBC (2021), UNIVALI (2021), Marji (2014), Esteves (2019).

O capítulo 1 aborda o uso do tecnologias para o ensino e aprendizagem da matemática na educação básica. Primeiramente, o uso de tecnologias no ensino da matemática é fundamentado com base nas recomendações apresentadas na Base Nacional Comum Curricular (BNCC). Em seguida, são propostas e fundamentadas as modalidades: "Laboratório de Matemática", "Jogos e Gamificação" e "Construção de Algoritmos e Programação", como formas de utilização de tecnologias no ensino e aprendizagem da matemática. Após, algumas das principais barreiras para o uso de tecnologias são discutidas, junto a formas de mitigá-las. Por fim, são apresentadas e justificadas as escolhas do Geogebra, Scratch, Portugol e Python para abordagem neste trabalho.

O capítulo 2 aborda Geometria: construções de polígonos regulares com Scratch e com o Geogebra Geometria. Em Scratch foi adotada a abordagem Construcionista proposta por Papert, estilo geometria da Tartaruga do LOGO. Inicialmente são trabalhadas as construções

geométricas do quadrado e do triângulo equilátero, passo a passo. Após, a construção é simplificada com um bloco de estrutura de repetição. Em seguida, são criados blocos para construções de quadrados, de triângulos equiláteros e de polígonos regulares com lado de qualquer dimensão. Seguindo, ainda utilizando Scratch, a abordagem é a construção de polígonos regulares circunscritos. Encerrando o capítulo, a construção de polígonos regulares circunscritos com o Geogebra Geometria aborda listas e estruturas de repetições que possibilitam construções geométricas em um paradigma diferente.

O capítulo 3 trata conceitos de algoritmos e programação. Primeiramente, são vistos como dados podem ser representados no computador e quais as operações possíveis de serem realizadas com esses dados. Após a abordagem dos dados, são apresentados comandos de atribuição e comandos de interação. Os comandos de atribuição servem para armazenar e manipular dados na memória do computador. Já os comandos de interação servem para troca de dados e informações entre o computador e o usuário. Após a abordagem dos comandos de atribuição e interação são vistas as estruturas de controle condicional e de controle de repetições. Essas estruturas permitem alterar a forma natural de processamento que é sequêncial. O controle condicional condiciona a execução de um grupo de instruções ao teste de uma condição resultar em verdadeiro ou falso. Já as estruturas de controle de repetições permitem que um conjunto de instruções seja repetido de acordo com algum critério. Por fim, são abordados conceitos de procedimentos e funções, ou seja, formas de construções de novas abstrações (novos de comandos).

O capítulo 4 trata da lógica utilizada na programação, notadamente da Álgebra de Boole que trabalha num contexto binário (falso ou verdadeiro, 0 ou 1). São abordados com exemplos e Diagramas de Venn, os operadores lógicos ou booleanos da disjunção inclusiva ou união - “ou”, da conjunção ou interseção - “e”, da negação ou complemento - “não”. Ao final, são apresentadas as Leis de De Morgan.

O capítulo 5 tem como foco o desenvolvimento de soluções computacionais iterativas. Soluções com iterações [1] utilizam estruturas de repetições. A abordagem é focada na matemática que envolve o triângulo de Pascal. Primeiramente, o triângulo de Pascal é definido e contextualizado. Em seguida, as sequências dos números naturais, da soma dos n primeiros números naturais e de Fibonacci são caracterizadas, contextualizadas na história e abordadas em projetos.

O capítulo 6 aborda recorrências tanto sob a ótica da matemática quanto da computação. No contexto da matemática são abordados os conceitos de recorrências lineares de primeira e segunda ordem. Os exemplos abordados para recorrências de primeira ordem são a soma dos n primeiros números naturais e fatorial de um número. O n-ésimo termo da sequência de Fibonacci e a fórmula de Binet são abordados na contextualização de recorrência de segunda ordem. No contexto da computação é apresentado o conceito de recursividade, uma forma alternativa de desenvolver o pensamento computacional, e são apresentados projetos em Scratch

[1] Iterações e interações são coisas distintas. Iterações são repetições e interações são diálogos (verbo interagir)

para calcular a a soma dos n primeiros números naturais, fatorial de um número e o n-ésimo termo da sequência de Fibonacci. Para o cálculo do n-ésimo termo de Fibonacci é discutida a ineficiência da solução recursiva e introduzido o conceito de complexidade computacional de um algoritmo. Por fim, o jogo Torre de Hanói e solução por recorrência é abordada.

O capítulo 7 têm como foco a divisibilidade de números inteiros e números primos. Inicialmente, o conceito de divisibilidade é abordado. Em seguida, são apresentados e discutidos Projetos Scratch para exibir os divisores de um número e para exibir o maior número que divide dois números - Máximo Divisor Comum (MDC), utilizando o algoritmo de Euclides. Dando sequência, são apresentados os conceitos de primalidade e coprimalidade. Dois Projetos Scratch abordam primalidade: um Projeto faz a classificação de um número como primo ou composto, baseado na lógica do projeto que exibe divisores de um número; outro projeto gera e exibe os números primos até 100 utilizando a técnica do Crivo de Eratóstenes. Após essas abordagens, o Teorema da Fatoração Única ou Teorema Fundamental da Aritmética é apresentado e seu respectivo Projeto Scratch discutido. Após, conceitos da Teoria dos Números relacionados a primalidade são apresentados, tais como: Pequeno Teorema de Fermat, Pseudoprimos, Números de Carmichael, Números de Mersenne, Números de Fermat e Primos Gêmeos. Ao final, o Geogebra CAS é utilizado em alguns cálculos de grandes números para comprovar alguns resultados apresentados.

O capítulo 8 aborda representações de números na base decimal (10), amplamente utilizada, e nas bases binária (2) e hexadecimal (16), utilizadas no mundo digital. Fazem parte dessa abordagem projetos Scratch para conversões das bases decimal em binário, binário em decimal, decimal em hexadecimal e hexadecimal em decimal. Ao final, são apresentadas formas de conversões de representações entre os sistemas binário e hexadecimal.

O apêndice A apresenta conceitos da linguagem de programação Portugol e 29 (vinte e nove) programas Portugol referentes aos algorítmos abordados neste trabalho, cada um com *link* de acesso para teste da execução no Portugol Studio.

O apêndice B apresenta conceitos da linguagem de programação Python e 29 (vinte e nove) programas Python referentes aos algorítmos abordados neste trabalho, cada um com *link* de acesso para teste da execução no OnlineGDB.

1 USO DO COMPUTADOR NA EDUCAÇÃO BÁSICA

Este capítulo aborda o uso do tecnologias para o ensino e aprendizagem da matemática na educação básica. Primeiramente, o uso de tecnologias no ensino da matemática é fundamentado com base nas recomendações apresentadas na Base Nacional Comum Curricular (BNCC). Em seguida, são propostas e fundamentadas as modalidades: "Laboratório de Matemática", "Jogos e Gamificação" e "Construção de Algoritmos e Programação", como formas de utilização de tecnologias no ensino e aprendizagem da matemática. Após, algumas das principais barreiras para o uso de tecnologias são discutidas, junto a formas de mitigá-las. Por fim, são apresentadas e justificadas as escolhas do Geogebra, Scratch, Portugol e Python para abordagem neste trabalho.

A lei 9.394/1996 e alterações estabelece diretrizes e bases da educação no Brasil. O art. 26 define que os currículos da educação infantil, do ensino fundamental e do ensino médio devem ter uma base nacional comum que é definida pelo documento denominado Base Nacional Comum Curricular - BNCC. (BRASIL, 2018)

> "Conforme definido na Lei de Diretrizes e Bases da Educação Nacional (LDB, Lei n° 9.394/1996) , a Base deve nortear os currículos dos sistemas e redes de ensino das Unidades Federativas, como também as propostas pedagógicas de todas as escolas públicas e privadas de Educação Infantil, Ensino Fundamental e Ensino Médio, em todo o Brasil." (BRASIL, 1996)

O documento BNCC[1] afirma que o conhecimento matemático é necessário a todos os alunos da Educacão Básica. A matemática cria sistemas abstratos com ideias que são fundamentais para a compreensão de fenômenos e argumentações consistentes nos mais variados contextos. Apesar da matemática ser uma ciência hipotético-dedutiva, é de fundamental importância o papel heurístico das experimentações em sua aprendizagem, notadamente nos diversos campos - Aritmética, Álgebra, Geometria, Estatística e Probabilidade. **A proposta é que os estudantes** do Ensino Fundamental **utilizem tecnologias (calculadoras, planilhas)** para quando chegarem aos anos finais, possam ser **estimulados a desenvolver o pensamento computacional, por meio da interpretação e elaboração de algoritmos**. Com isso, no Ensino Médio, focar na construção de uma visão integrada da Matemática, aplicada à realidade, em diversos contextos (BRASIL, 1996, grifos do autor).

Tecnologias digitais no ensino da matemática podem ser utilizadas em diversas vertentes. Neste trabalho serão caracterizados os usos de tecnologias no ensino da matemática nas formas de: 1) Laboratório de Matemática; 2) Jogos e Gamificação; 3) Construção de Algoritmos e Programação. As duas primeiras conceituadas e exemplificadas de modo superficial, tendo abordagem mais exaustiva a Construção de Algoritmos e Programação.

[1] Histórico da Base Nacional Comum Curricular em http://basenacionalcomum.mec.gov.br/historico/

1.1 Laboratório de Matemática

Laboratórios de Matemática ou de Ensino da Matemática já são realidades em diversas escolas e universidades. Assim como os laboratórios de física ou de química, laboratórios de matemática são locais para realizações de atividades práticas focadas na exploração de conceitos matemáticos com utilizações de materiais recicláveis, jogos, materias didáticos ou artefatos apropriados para as abordagens.

Abreu (1997) afirma que Laboratório de Matemática é o espaço onde o aluno vai criar novas soluções para os problemas apresentados, trabalhar com atividades lúdicas e refletir sobre ideias matemáticas. Um Laboratório de Matemática deve ser reconhecido como necessário para: oferecer aos alunos um local adequado com materiais variados que possam ser utilizados para o desenvolvimento de conceitos matemáticos fundamentais; realizar atividades e recuperação de alunos que apresentem dificuldades no raciocínio e na construção de conhecimentos matemáticos; fazer o aluno pensar produtivamente; desenvolver o raciocínio lógico; envolver o aluno com aplicações da matemática; tornar o estudo da matemática mais interessante e desafiador; equipar o aluno com estratégias para resolver problemas, individualmente ou em grupo; dentre outras características.

As tecnologias digitais (Tecnologias da Informação e Comunicação - TICs) podem estar presentes em um laboratório de matemática nas formas de calculadoras ou *softwares* específicos. Borba (2019) e Lucena (2017) abordam o uso de tecnologias no ensino da matemática.

Borba (2019) relata o trabalho do Grupo de Pesquisa em Informática, outras Mídias e Educação Matemática - GPIMEM da Universidade Estadual Paulista - UNESP foi pioneiro na pesquisa de utilização de tecnologias em salas de aulas, tendo iniciado pesquisas de uso de calculadoras gráficas desde o ano 1993 no primeiro ano de graduação em Biologia da UNESP, em escolas públicas de ensino fundamental e médio, e ainda em outros ambientes. Em seu trabalho ressalta que ao final do século passado a ênfase para o ensino de funções se dava via álgebra, com grande destaque para a expressão analítica de uma função e quase nada para aspectos gráficos ou tabulares. Uma epistemologia das representações múltiplas seria a proposta de novo caminho para conhecimento de funções com abordagens da expressão algébrica, gráfico e tabela.

> As novas mídias, como os computadores com softwares gráficos e as calculadoras gráficas, permitem que o aluno experimente bastante, de modo semelhante ao que faz em aulas experimentais de biologia ou de física. Podem experimentar com gráficos de funções quadráticas do tipo $y = ax^2 + bx + c$, por exemplo, antes de conhecerem uma sistematização de função quadrática.
> ... alunos têm investigado como os diferentes coeficientes de polinômios do tipo acima influenciam os gráficos de funções e tentam coordenar ambas as representações: que alteração ocorre no gráfico quando um determinado coeficiente é alterado.
> ... nesse processo os alunos experimentam, geram conjecturas por escrito e oralmente e as debatem. (BORBA, 2019, p.36-37)

> O GPIMEM já investigou a relevância das calculadoras gráficas e dos sensores associados à Educação Matemática e hoje realiza a difusão do conheci-

> mento utilizando-se dessas mídias. Atualmente, analisa as possibilidades propiciadas por *softwares*, abordando diversos temas da Matemática. Pesquisa, ainda, questões relacionadas à formação de professores; modelagem matemática; educação à distância; o uso de tecnologias da informação nas aulas de Matemática; geometria nos livros didáticos e a integração das tecnologias digitais; performance matemática digital envolvendo Arte e Matemática, baseando-se em diferentes abordagens teóricas. (UNESP, 2021)

Para Lucena (2017) "O professor, ao realizar a escolha de uma tecnologia como ferramenta para o ensino da matemática, deve considerar quais objetivos de ensino deseja alcançar e planejar a condução pedagógica". Justifica, como exemplo, que o uso da calculadora como tecnologia permitirá ao aluno reservar mais tempo para leitura, compreensão e interpretação do problema já que o tempo com cálculos será abreviado com o uso da tecnologia.

Os aplicativos Scratch e Geogebra podem ser utilizados na modalidade laboratório de matemática com a utilização de algum projeto previamente desenvolvido (pelo professor ou por terceiros) ou com a atividade realizada na modalidade *playground*[1].

1.2 Jogos e Gamificação

Atividades lúdicas são uma maneira alternativa e atrativa de abordagem de conteúdos nas diversas áreas do conhecimento e, portanto, para a matemática não seria diferente.

Para Huizinga (2014) e Alves (2015) jogo é mais antigo que a cultura, pois a cultura pressupõe a existência da sociedade. Os animais brincam como os homens. O jogo é mais que um fenômeno fisiológico ou um reflexo psicológico. Tanto a psicologia como a filosofia procuram investigar e entender a natureza do jogo e situá-lo no sistema da vida. Uma das teorias atribui ao jogo o papel de preparar o jovem para tarefas que terá que realizar mais tarde e assim contribuem para o processo de aprendizagem.

> *Gamefication* [2] não é a transformação de qualquer atividade em um *game*[3]. *Gamefication* é aprender a partir dos *games*, encontrar elementos dos *games* que podem melhorar uma experiência sem desprezar o mundo real. Encontrar o conceito central de uma experiência e torná-la mais divertida e engajadora. (ALVES, 2015, p.30)

Prensky (2012) aponta razões que justificam a aprendizagem baseada em jogos : **envolvimento** - colocar o aprendizado no contexto de um jogo e mudar o foco da atividade de aprendizagem que para alguns pode ser entediante; **processo interativo de aprendizagem** - as diferentes formas que o processo pode assumir; **maneira como os dois são unidos** - são os modos como o envolvimento e a interação são usados para se obter uma melhor solução,

[1] denominação dada a área de trabalho
[2] Gameficação
[3] Jogo

ou seja, a contextualização dependendo dos objetivos de aprendizagem e do envolvimento do jogador.

McGonigal (2012) afirma que quatro características definem um jogo: **meta** - propicia um senso de objetivo. É o resultado que jogadores buscam; **regras** - impõem limitações aos jogadores, mas liberam a criatividade e estimulam o pensamento estratégico ; ***feedback*(retorno)** - fornecem a motivação para continuar jogando. Informam aos jogadores quão perto estão de atingir a meta; **Participação voluntária** - é a liberdade de entrar ou sair de um jogo que assegura que um trabalho desafiador seja vivenciado como uma atividade segura e prazerosa. Exige que cada jogador aceite, consciente e voluntariamente, a meta, as regras e o *feedback*.

Na aprendizagem baseada em jogos digitais já foram utilizadas as seguintes técnicas de aprendizagem interativa: **Aprender na Prática** - a prática repetitiva é uma maneira de aprender coisas. O computador é muito bom nisso; **Prática e *feedback* (retorno)** - combina a prática com retorno baseado no acompanhamento, de acordo com a maneira como as pessoas respondem. Técnicas de programação adaptáveis que alteram o grau de dificuldade das tarefas ou problemas, dependendo, estatisticamente, do quanto correto as coisas forem feitas; **Aprender com os erros** - uma das principais maneiras de aprender é a tentativa e o erro. Os jogos digitais são bons na tentativa e erro porque dão aos jogadores motivação para continuarem tentando; **Aprendizagem guiada por metas** - a aprendizagem pode ser guiada pelos fatos (aprender sobre algo) ou guiada por metas (aprender a fazer algo). O objetivo de um jogo é seu elemento chave; **Aprendizagem pela descoberta e "descobertas guiadas"** - a ideia chave dessa técnica é a de que algo é mais bem aprendido se a pessoa aprender por si próprio; **Aprendizagem baseada em tarefas** - É uma variação do aprender fazendo. Consiste numa série de tarefas ou problemas que se baseiam uns nos outros e aumentam de dificuldade gradativamente. Um possível problema com essa metodologia é que os usuários podem aprender menos a teoria. Então é importante pensar em maneiras criativas de implementar; **Aprendizagem guiada por perguntas** - tradicionalmente associado a um jogo de perguntas e respostas, desperta o pensamento e reflexão sobre as resposta; **Aprendizagem contextualizada** - o ambiente de aprendizagem simula o ambiente real; ***Role-playing*** - estratégia de aprendizagem em treinamentos interativos, onde o aprendiz assume diferentes papeis de atuação; **Treinamento** - É a incorporação elementos de instrução em um contexto de um jogo, como parte integrante dele; **Aprendizagem construtivista** - Baseada nos trabalhos de Seymour Papert e Jean Piaget, a aprendizagem chamada de construtivista tem como premissa"as pessoas aprendem melhor quando constroem ativamente ideias e relações na mente com base em experimentos que faz"; **Aprendizagem "acelerada"(múltiplos sentidos)** - aprendizagem que envolve experiências multissensoriais; **Selecionar a partir dos objetos de aprendizagem** - o conceito é "se alguém pode desenvolver partes do conteúdo e talvez certas interações que são independentes, então elas podem ser ligadas". Seria algo como dividir o problema para conquistar. **Instrução inteligente** - a instrução inteligente avalia a resposta do aprendiz e procura identificar por que ele cometeu o erro, ou seja, busca entender se a concepção do aprendiz está equivocada e apontar como corrigir. (PRENSKY, 2012)

Os aplicativos Scratch e Geogebra podem ser utilizados na modalidade jogos com a uti-

lização de algum projeto de jogo desenvolvido (pelo professor ou por terceiros), como será demonstrado nas seções 1.5 e 1.6. O Scratch por si já tem uma característica própria de jogo que é a montagem de peças, tipo LEGO[1], para construções dos programas.

1.3 Algoritmos e Programação

Os algoritmos[2] fazem parte da matemática, muito antes da computação digital. Processos de adições, subtrações, multiplicações e divisões entre números com dois ou mais dígitos são ensinados em forma de algoritmos, mesmo sem a associação com a nomenclatura do termo. Uma infinidade de conceitos matemáticos podem ser traduzidos em algoritmos, tais como: operações aritméticas com frações, cálculo de máximo divisor comum (MDC), cálculo de mínimo múltiplo comum (MMC), fatoração de um número, cálculo de raízes de uma equação do 2^o grau, construções geométricas, etc.

O ensino da construção de algoritmos e programação de computadores já é uma realidade em muitas escolas brasileiras e a Sociedade Brasileira de Computação - SBC fomenta essa prática com a promoção anual da Olimpíada Brasileira de Computação - OBI.

> A Olimpíada Brasileira de Informática (OBI) é uma iniciativa da Sociedade Brasileira de Computação e tem por objetivos: Estimular o interesse pela Computação e por Ciências em geral; Promover a introdução de disciplinas de raciocíno computacional e técnicas de programação de computadores nas escolas de ensino médio e fundamental; Proporcionar novos desafios aos estudantes; Identificar talentos e vocações em Ciência da Computação de forma a melhor instruí-los e incentivá-los a seguir carreiras nas áreas de ciência e tecnologia. (SBC, 2021)

A Olimpíada Brasileira de Computação do ano 2021 - OBI2021 será a vigésima terceira edição do evento, ofertada em duas modalidades e sete níveis. A modalidade iniciação tem provas de lógica realizadas com lápis e papel. Possui três níveis: Nível Júnior, para alunos do quarto e quinto anos do Ensino Fundamental; Nível 1, para alunos do sexto e sétimo anos do Ensino Fundamental e Nível 2, para alunos do oitavo e o nono anos do Ensino Fundamental. A modalidade programação tem provas realizadas no computador, podendo ser resolvidas em linguagem de programação Python, C, C++, Java, Javascript e Pascal. Os níveis são: Nível Júnior, para alunos até o nono ano do ensino fundamental; Nível 1, para alunos até o primeiro ano do Ensino Médio; e Nível 2, para alunos até o terceiro ano do ensino médio; Nível Sênior, para alunos do quarto ano do Ensino Técnico e alunos cursando pela primeira vez o primeiro ano de um curso de graduação. O Instituto de Computação da UNICAMP é responsável pela

[1] LEGO é abreviação de duas palavras dinamarquêsas *leg godt* que significa jogar bem

[2] sequência finita de regras, raciocínios ou operações que, aplicada a um número finito de dados, permite solucionar classes semelhantes de problemas

organização. A figura 1 mostra a questão Corrida na Floresta, aplicada na edição do ano 2020 ao Nível Júnior - Modalidade Iniciação, e a figura 2 mostra a questão Parcelamento sem Juros, aplicada ao ao Nível Júnior - Modalidade Programação na edição do ano 2019. (SBC, 2021)

Tarefas das modalidades Iniciação e Programação da OBI, de diferentes níveis de dificuldade, estão disponíveis em: https://olimpiada.ic.unicamp.br/pratique/.

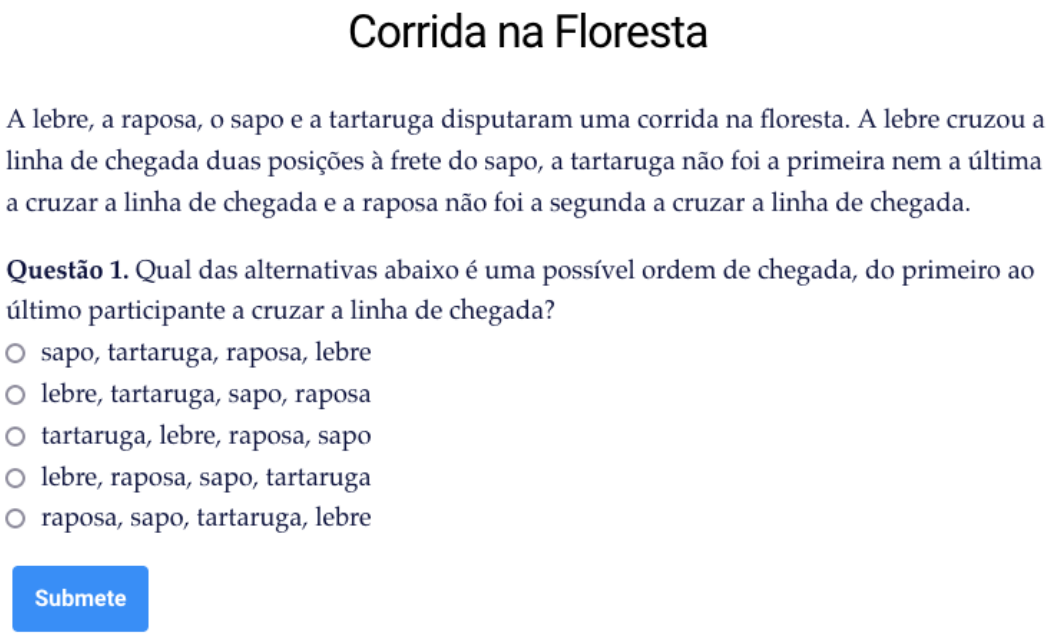

Figura 1 – OBI2020 - Corrida na Floresta (SBC, 2021)

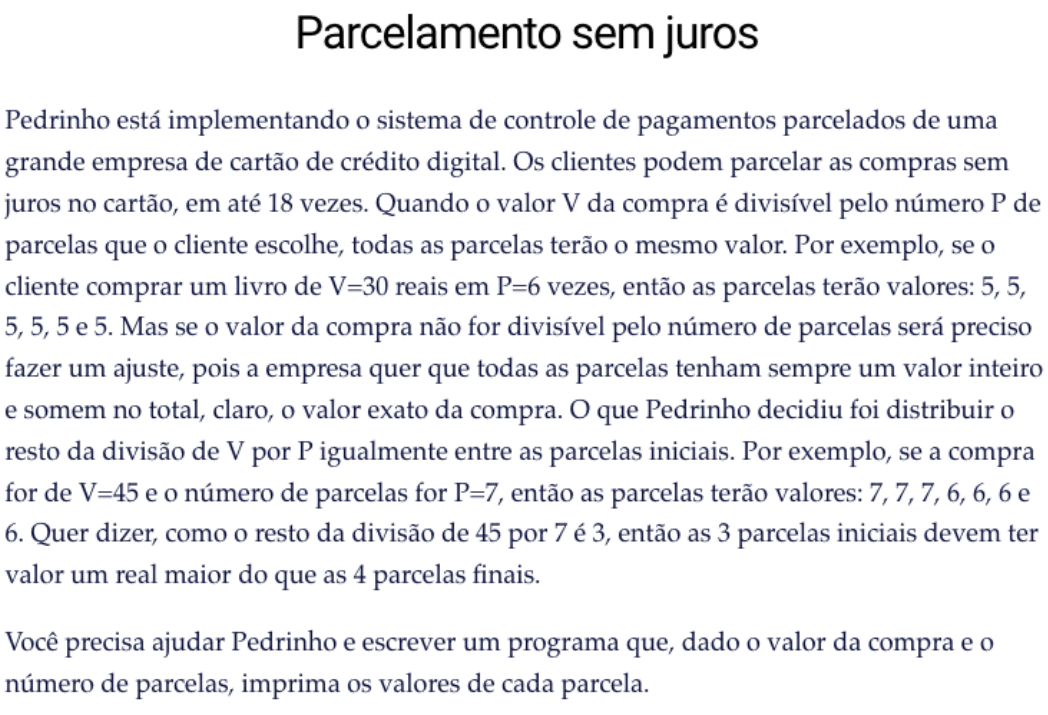

Figura 2 – OBI019 - Parcelamento Sem Juros(SBC, 2021)

Um dos precursores do ensino da computação para crianças e jovens foi Seymour Papert (1928-2016). Papert trabalhou em Genebra com Jean Piaget no período de 1958-1963. Em 1964 iniciou trabalhos no *Massachusetts Institute of Technology* - MIT, convidado por Martin Minsky, um dos principais nomes da Inteligência Artificial. No trabalho conjunto iniciaram diversos projetos de pesquisas nas áreas de teoria da computação, robótica, percepção humana e psicologia da criança. Na década de 60, Papert e outros pesquisadores desenvolveram a linguagem LOGO[1], uma linguagem de computador para crianças adotada em todo o mundo. Na décadade 80, Papert define a Teoria Construcionista de Aprendizagem, valendo-se de sua experiência e estudos com pesquisadores como Piaget, Dewey, Montessori e Paulo Freire. A teoria constucionista determina uma nova abordagem educativa com uso do computador no processo de aprendizagem. (CAMPOS, 2013)

Papert (1988) se contrapõe a ideia de Piaget de que o pensamento formal não se desenvolve antes dos 12 anos. A suposição de Papert é que o computador pode concretizar o formal. Quando a criança aprende a programar, o processo de aprendizagem é transformado. O conhecimento é adquirido para um propósito pessoal reconhecível. Por ter sido experimentado, se consolida na mente da criança. A metáfora do computador como uma entidade que "fala" uma linguagem matemática coloca o aprendiz numa nova qualidade de relacionamento com um importante domínio do conhecimento.

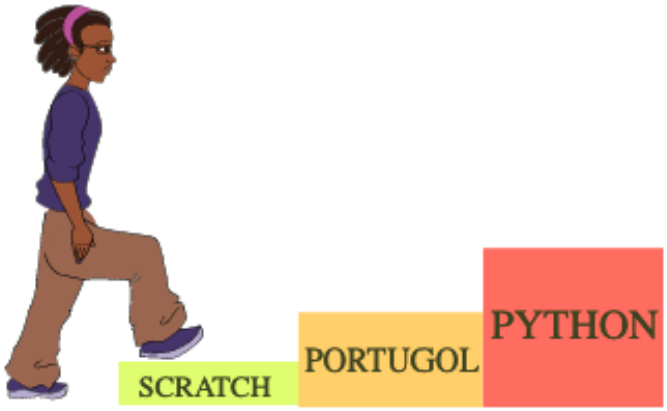

Figura 3 – Trilhas para aprendizagem de programação de computadores (Elaborado pelo autor)

Os algoritmos deste trabalho estão codificados em Scratch, em Portugol[2] e em Python[3]. Os códigos em Scratch estão no próprio corpo do trabalho fazendo parte do contexto das explicações dos conteúdos da matemática e computação. Já os códigos em Portugol e Python estão nos apêndices. Assim, a abordagem da construção de programas poderá ser de acordo com a conveniência: Scratch, Portugol ou Python. Um caminho progressivo que pode ser seguido é a trilha da figura 3. Primeiramente, aprende-se Scratch com a facilidade da construção por

[1] referência a um termo grego que significa: pensamento, razão, raciocínio
[2] O Portugol é uma linguagem de programação para fins didáticos cujas instruções estão em português
[3] Linguagem de Programação amplamente utilizada, notadamente na área de ciência de dados

blocos. Depois, a facilidade da construção por blocos é suprimida e a programação passa a ser em Portugol. Por fim, quebra-se a barreira do idioma e o trabalho passa a ser em Python, uma linguagem de programação amplamente utilizada nos meios acadêmicos e comerciais. Outra abordagem que pode ser realizada seria as comparações entre os programas gerados nos ambientes Scratch, Portugol e Python.

1.4 Principais barreiras e como mitigá-las

Sistemas de computação são compostos por equipamentos(*hardware*), programas(*software*) e usuários. Cada um desses elementos pode apresentar entraves para o funcionamento ou utilização do sistema. Existem diversos tipos e tecnologias de equipamentos: celular, tablet, computador. Programas são desenvolvidos para plataformas específicas e para utilização é preciso ter licença de uso que pode custar caro. Pessoas (professores e alunos) necessitam de treinamento.

Borba (2019) destaca alguns aspectos que impactam no uso da informática na educação pública: o interesse e envolvimento de diretores e coordenadores; apoio do governo na implantação da informática educativa e sua continuidade - mudanças políticas implicam em diminuição ou cancelamento de verbas; burocracia ou subutilização das salas de informática; quantidade de computadores e espaços insuficientes na sala de informática para comportar uma turma completa; necessidade de apoio de um técnico em informática para corrigir problemas, instalar e configurar *softwares*, dentre outras demandas.

A escolha do Scratch e Geogebra para o ensino da matemática possibilita mitigar algumas das barreiras elencadas pelas razões que seguem:

- **Uso *on-line* ou instalado, multiplataformas** - ambos podem ser utilizado diretamente da internet, via navegador, sem a necessidade de instalação no computador. Caso seja do interesse podem ser instalados e executados no equipamento do usuário sem conexão com a internet pois possuem versões para sistemas operacionais Windows, macOS, ChromeOS e Android.
- **Idioma Português**- ambos possuem tradução para o português do Brasil
- **Licença de uso gratuito** - ambos possuem licença para uso não comercial gratuito

1.5 Geogebra

> O GeoGebra é um software de matemática dinâmica para todos os níveis de ensino que reúne Geometria, Álgebra, Planilha de Cálculo, Gráficos, Probabilidade, Estatística e Cálculos Simbólicos em um único pacote fácil de se usar. O GeoGebra possui uma comunidade de milhões de usuários em praticamente todos os países. O GeoGebra se tornou um líder na área de softwares de matemática dinâmica, apoiando o ensino e a aprendizagem em Ciência, Tecnologia, Engenharia e Matemática. (GEOGEBRA, 2021)

A figura 4 mostra as diversas versões do geogebra e os recursos disponíveis em cada uma delas.

apps / features	Scientific	Graphing	Geometry	Suite	3D	CAS	Classic
Numeric calculations	✓	✓	✓	✓	✓	✓	✓
Function operations	✓	✓	✓	✓	✓	✓	✓
Fraction operations	✓	✓	✓	✓	✓	✓	✓
Graphing		✓	✓	✓	✓	✓	✓
Sliders		✓	✓	✓	✓	✓	✓
Vectors & matrices		✓	✓	✓	✓	✓	✓
Table of values		✓		✓		✓	✓
Geometric constructions			✓	✓	✓	✓	✓
3D graphing				✓	✓		✓
Probability Calculator				✓*			✓
Derivatives & integrals				✓	✓	✓	✓
Equation solving				✓	✓	✓	✓
Symbolic calculations				✓	✓	✓	✓
Spreadsheet							✓

*coming soon

Figura 4 – Aplicativos Geogebra (Fonte: geogebra.org)

Um grande acervo de materiais geogebra já prontos para uso está disponível no portal do geogebra[1]. São jogos ou aplicações para abordagens de conteúdos. Esses materiais estão agrupados por áreas e subáreas da matemática. A figura 5 exibe alguns desses agrupamentos.

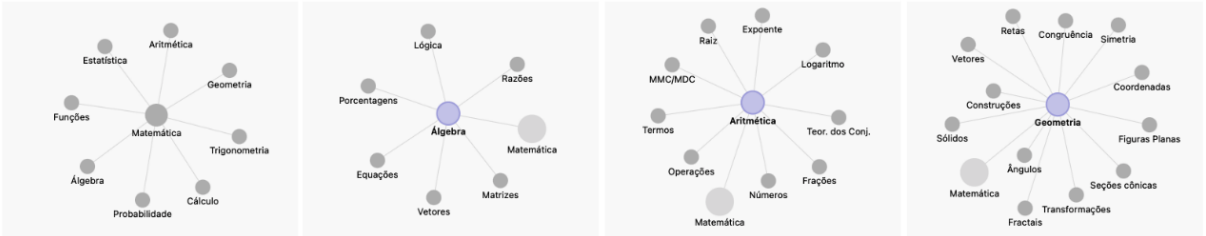

Figura 5 – Geogebra Materiais(Fonte: geogebra.org)

Neste trabalho constam 03 (três) aplicações do Geogebra nas modalidades: **Geogebra Calculadora Gráfica** para estudo do comportamento de uma função quadrática, **Geogebra Geo-**

[1] https://www.geogebra.org/materials

metria para estudo de polígonos regulares circuncêntricos e **Geogebra CAS** (*Computer Algebra System*) - para fatorar alguns Números de Fermat e Mersenne.

Geogebra Calculadora Gráfica - Função Quadrática

O experimento descrito por Borba (2019) realizado pelo GPIMEM com alunos para investigar diferentes coeficientes em funções quadráticas do tipo $f(x) = ax^2 + bx + c$, abordado na seção 1.1, pode ser facilmente realizado com Geogebra Calculadora Gráfica disponível em: https://www.geogebra.org/graphing, simplesmente digitando **f(x)=a*x^2+b*x+c** no campo Entrada da Janela de Álgebra.

A figura 6 exibe a tela do Geogebra Calculadora Gráfica após a entrada da função quadrática. Os elementos assinalados são: (1) endereço da página; (2) Menu Principal: possibilita Limpar, Abrir, Gravar, Compartilhar, Exportar Imagem, dentre outras opções; (3) Janela de Álgebra em que foi entrada a função quadrática - note que após a entrada da função foram criados 3 (três) controles deslizantes (*sliders - 3.1*), um para cada um dos coefientes: a, b e c; (4) Janela de visualização do gráfico.

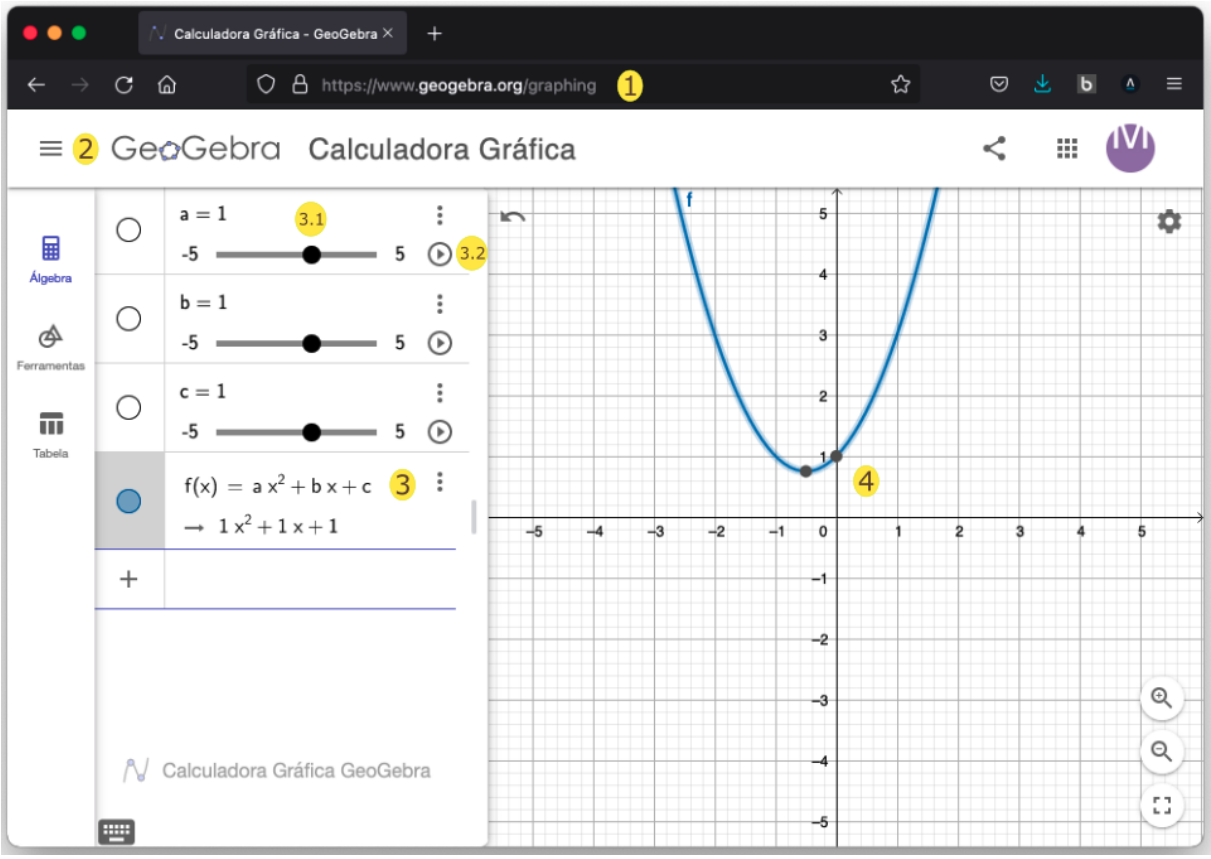

Figura 6 – Geogebra Calculadora Grafica - Equação 2^o grau (Elaborado pelo autor)

Controles deslizantes (*sliders - 3.1*) potencializam a característica de Laboratório de Matemática do Geogebra. Deslizando o controle associado a uma variável muda-se seu valor e o gráfico é atualizado instantaneamente. A mudança também pode ser feita automaticamente, em modo animação utilizando o botão tocar (3.2).

A seguir serão enumerados os tipos de objetos do Geogebra e os comandos por agrupamento afim. Isto possibilita ter-se noção, mesmo que parcial, da abrangência do *software* para abordagens nos diversos ramos da matemática.

Objetos Geogebra

- **Objetos Geométricos:** Pontos e Vetores, Retas e Eixos, Seções Cônicas, Funções, Curvas, Desigualdades, Intervalos
- **Objetos Gerais:** Números e Ângulos, Números Complexos, Listas, Matrizes, Textos, Imagens

Comandos Geogebra

- **Comandos do tipo Álgebra [Ferramentas]** - Expandir; Fatorar; FatoresPrimos; MDC; MMC; Máximo; Mínimo; Quociente; Resto; Simplificar
- **Comandos do tipo Diagramas** - BoxPlot; DiagramaDeBarras; DiagramaDePontos; DiagramaDeStem; DiagramaQuantilNormal; DiagramaResidual; HistogramaDireita; Histograma; PolígonoDeFrequências
- **Comandos do tipo Cônicas** - Assíntota; Centro; ComprimentoDoSemieixoMaior; ComprimentoDoSemieixoMenor; CírculoInscrito; Círculo; Cônica; Diretriz; DiâmetroConjugado; EixoMaior; EixoMenor; Eixos; Elipse; ExcentricidadeLinear; Excentricidade; Foco; Hipérbole; Parábola; Parâmetro; Polar; Semicírculo
- **Comandos do tipo Matemática Discreta** - CaminhoMínimo; DiagramaDeVoronoi; FechoConvexo; Fecho; ProblemaDoCaixeiroViajante; TriangulaçãoDeDelaunay; ÁrvoreGeradoraMínima
- **Comandos do tipo Funções e Cálculo** - Assíntota; Coeficientes; CompletarQuadrados; ComplexRoot; CurvaImplícita; Curvatura; Curva; CírculoOsculador; Denominador; Derivada; Extremo; Fatores; FraçõesParciais; Função; Grau; IntegralEntre; Integral; Interseção; Iteração; LimiteInferior; LimiteSuperior; Limite; ListaDeIteração; ListaParaPontosEixoX; Numerador; ParâmetroDoPontoSobreCaminho; PolinômioDeTaylor; Polinômio; PontoDeInflexão; Raiz; Raízes; ResolverEDO; SomaDeRiemannInferior; SomaDeRiemannSuperior; SomaDeRiemanÀEsquerda; SomaRetângulos; SomaTrapezoidal; VetorCurvatura
- **Comandos do tipo Geometria** - ArcoCircular; ArcoCircuncircular; Arco; Baricentro; Bissetriz; CaminhoPoligonal; Comprimento; Direção; Distância; Inclinação; InterseçãoDeRegiões; Interseção; LugarGeométrico; Mediatriz; Perpendicular; Perímetro; Polígo-

noRígido; Polígono; PontoEm; PontoMaisPróximo; PontoMédio; Ponto; Raio; RazãoAfim; RazãoDupla; Reta; Segmento; Semirreta; SetorCircular; SetorCircuncircular; Setor; Tangente; Vértice; Área; Ângulo

- **Comandos do tipo GeoGebra** - AnimarConstruçãoDoGráfico; Canto; CoordenadasDinâmicas; ImagemDaFerramenta; Nome; Objeto; PassoDaConstrução; PassoDoEixoX; PassoDoEixoY; PontoMaisPróximo

- **Comandos do tipo Listas** - Anexar; Classes; Concatenar; CriarPontosDeLista; ElementoSelecionado; ElementosÚnicos; Elemento; Empacotar; EscolherElementoAleatoriamente; Frequência; Inserir; InterseçãoDeListas; Interseção; ListaDeIteração; ListaParaPontosEixoX; Ordenar; ParteDaLista; PosiçõesMédias; Posições; Primeiros; Produto; RemoverIndefinidos; Reverter; Sequência; União; ÍndiceDe; ÍndiceSelecionado; Últimos

- **Comandos do tipo Lógica** - ContarSe; EstáDefinido; ManterSe; PertenceARegião; Relação; Se; ÉInteiro

- **Comandos do tipo Otimização** - Maximizar; Minimizar

- **Comandos do tipo Probabilidade** - CoeficienteBinomial; DistribuiçãoBinomialInversa; DistribuiçãoBinomial; DistribuiçãoChiQuadradoInversa; DistribuiçãoChiQuadrado; DistribuiçãoDeBernoulli; DistribuiçãoDeCauchyInversa; DistribuiçãoDeCauchy; DistribuiçãoDeErlang; DistribuiçãoDePascalInversa; DistribuiçãoDePascal; DistribuiçãoDePoissonInversa; DistribuiçãoDePoisson; DistribuiçãoDeWeibullInversa; DistribuiçãoDeWeibull; DistribuiçãoDeZipfInversa; DistribuiçãoDeZipf; DistribuiçãoExponencialInversa; DistribuiçãoExponencial; DistribuiçãoFInversa; DistribuiçãoF; DistribuiçãoGamaInversa; DistribuiçãoGama; DistribuiçãoHipergeométricaInversa; DistribuiçãoHipergeométrica; DistribuiçãoLogNormal; DistribuiçãoLogística; DistribuiçãoNormalInversa; DistribuiçãoNormal; DistribuiçãoTInversa; DistribuiçãoTriangular; DistribuiçãoT; DistribuiçãoUniforme; NúmeroAleatórioBinomial; NúmeroAleatórioDePoisson; NúmeroAleatórioNormal; NúmeroAleatórioUniforme; NúmeroAleatório

- **Comandos do tipo Programação** - Ampliar; Apagar; AtualizarConstrução; Botão; CaixaDeSeleção; CampoDeTexto; ControleDeslizante; CopiarObjetoLivre; DefinirCamada; DefinirCondiçãoParaExibirObjeto; DefinirCoordenadas; DefinirCorDeFundo; DefinirCorDinâmica; DefinirCor; DefinirEscalaDosEixos; DefinirEspessuraDaLinha; DefinirEstiloDaDescrição; DefinirEstiloDaLinha; DefinirEstiloDoPonto; DefinirEstiloDoR'otulo; DefinirLegenda; DefinirObjetoFixo; DefinirPreenchimento; DefinirTamanhoDoPonto; DefinirValor; DefinirVisibilidade; EsconderCamada; Executar; ExibirCamada; ExibirRótulo; IniciarAnimação; InterpretarTextoParaFunção; InterpretarTextoParaNúmero; Reduzir; Relógio; Renomear; SelecionarJanelaDeVisualizaçãoAtiva; SelecionarObjetos; TocarSom; TransladarJanelaDeVisualização

- **Comandos do tipo Planilha** - Bloco; Coluna; Célula; Linha; NomeDaColuna; PreencherColuna; PreencherCélulas; PreencherLinha
- **Comandos do tipo Estatística** - ANOVA; Amostra; Classes; CoeficienteDeCorrelação; CorrelaçãoDeSpearman; Covariância DesvioPadrãoAmostralX; DesvioPadrãoAmostralY; DesvioPadrãoAmostral; DesvioPadrãoX; DesvioPadrãoY; DesvioPadrão; Embaralhar; EstimarMédiaT2; EstimarMédiaT; Frequência; Mediana; Moda; MédiaGeométrica; MédiaHarmônica; MédiaQuadrática; MédiaX; MédiaY; Média; Percentil; Q1; Q3; RQuadrado; RegressãoDeCrescimento; RegressãoExponencial; RegressãoLinearX; RegressãoLinear; RegressãoLogarítmica; RegressãoLogística; RegressãoPolinomial; RegressãoPotência; RegressãoSenoidal; Regressão; SigmaXX; SigmaXY; SigmaYY; SomaDosErrosQuadrados; Soma; Sxx; Sxy; Syy; TabelaDeFrequências; TesteT2; TesteTEmparelhado; TesteT; VariânciaDaAmostra; Variância
- **Comandos do tipo Texto** - FraçãoEmTexto; GirarTexto; LaTeX; LetraParaUnicode; Ordinal; TabelaDeTexto; TextoParaUnicode; TextoVertical; Texto; UnicodeParaLetra; UnicodeParaTexto
- **Comandos do tipo Transformações** - Cisalhamento; Esticar; Girar; Homotetia; Reflexão; Transladar
- **Comandos do tipo Vetores e Matrizes** - AplicarMatriz; Determinante; MatrizEscalonada; MatrizIdentidade; MatrizInversa; MatrizTransposta; VetorCurvatura; VetorPerpendicularUnitário; VetorPerpendicular; VetorUnitário; Vetor

1.6 Scratch

> O Scratch é uma linguagem de programação visual que oferece um ambiente de aprendizado rico para pessoas de todas as idades. Ele permite a criação de projetos interativos, ricos em recursos de mídia, incluindo histórias animadas, avaliações de livros, projetos de ciências, jogos e simulações. O ambiente de programação visual do Scratch possibilita explorar áreas de conhecimentos que, de outro modo, seriam inacessíveis. O Scratch oferece um conjunto completo de ferramentas multimídia que pode ser utilizado para criar aplicações incríveis, o que pode ser feito muito mais facilmente se comparado à situação em que outras linguagens de programação são utilizadas. (MARJI, 2014, p.17)

O Scratch foi idealizado por Mitchel Resnick e é um projeto do grupo Lifelong Kindergarten no Media Lab do MIT (Instituto de Tecnologia de Massachusets) iniciado em 2003, tendo recebido o apoio da National Science Foundation, Fundação Intel, Microsoft, Fundação LEGO, Fundação Code-to-Learn, Google, Dell, entre outras. Scratch é utilizado em mais de 150 países e está disponível em mais de 40 idiomas.

Figura 7 – Algoritmo de Euclides com Voz em Scratch

Um projeto Scratch pode ser um jogo, uma animação, um tutorial, uma arte ou música, uma história, ou seja, pode ter uma finalidade específica dentre muitas. Existem diversos projetos e tutorias[1] prontos na página do Scratch que podem ser utilizados e explorados. Um exemplo de um projeto Tutorial é o algoritmo de Euclides desenvolvido pelo autor e disponível em https://scratch.mit.edu/projects/554063400/. Esse projeto implementa o algoritmo de Euclides para o cálculo do máximo divisor comum de dois valores detalhado no capítulo 7 com a incorporação de recursos de voz e uma abordagem voltada para o ensino do método. A figura 7 mostra a página do projeto.

[1] https://scratch.mit.edu/ideas

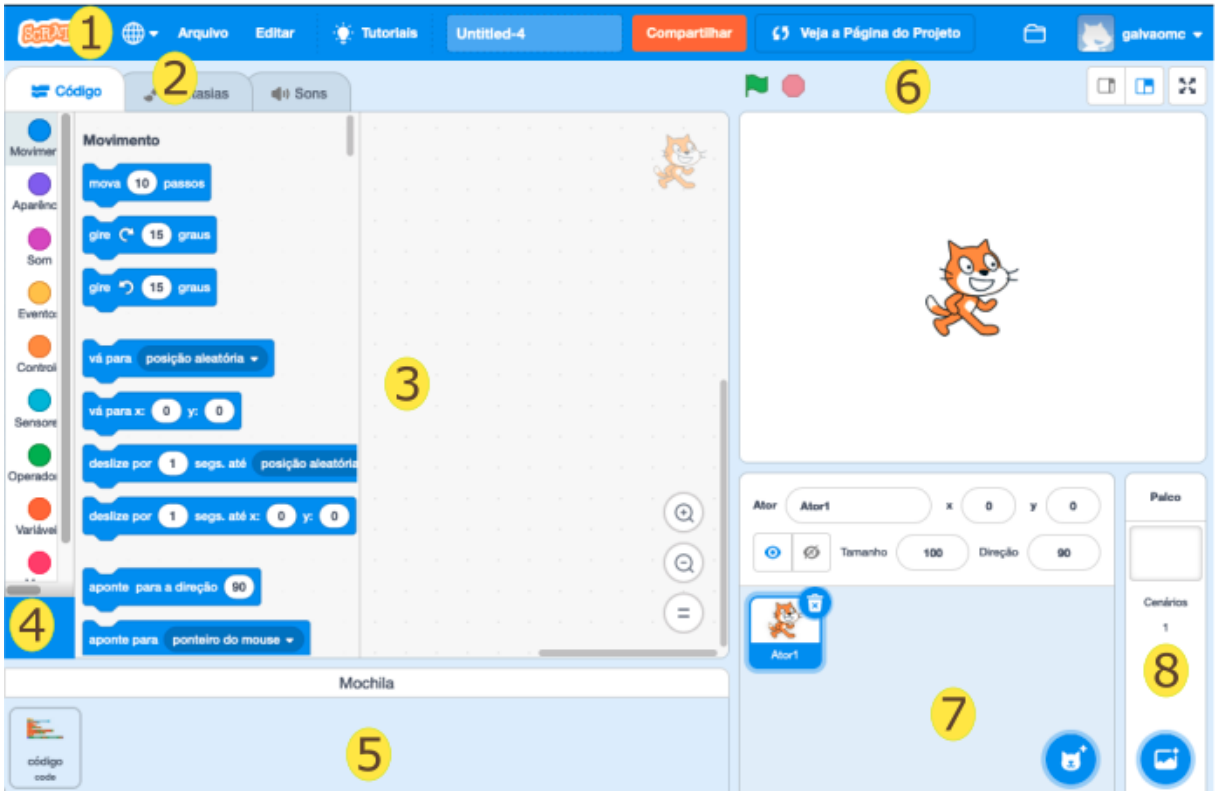

Figura 8 – Ambiente Scratch

A abordagem central deste trabalho é o desenvolvimento de programas em Scratch. O ambiente de programação do Scratch pode ser visualizado na figura 8. A programação é realizada em português[1] em um paradigma orientado a blocos, ou seja, os comandos e estruturas de controle estão em forma de blocos e as construções de programas são por encaixes desses blocos. Esse tipo de construção evita uma infinidade de erros que podem acontecer em uma programação convencional, tais como: erros na escrita de comandos (sintaxe), erros nas construções de expressões, erros por incompatibilidade de tipos de dados, erros na estruturação dos comandos, erros por falta de declaração de variável, etc. O Scratch pode ser utilizado diretamente da internet, via navegador, sem a necessidade de instalação no computador. Entretanto, caso seja do interesse, existem versões *Desktop* para os sistemas operacionais Windows, macOS, ChromeOS e Android, ou seja, versões para serem instaladas e executadas na máquina do usuário sem conexão com a internet.

Oito elementos da edição de um projeto estão destacados na figura 8. São eles: 1. Menu principal para realizar operações com o arquivo do projeto, tais como: abrir, salvar, compartilhar, renomear, dentre outras; 2. Abas de Código, Fantasias/Cenários e Sons: seleciona a edição de código vinculado a um elemento (ator ou cenário) ou Edição de imagens de Fantasias/Cenários ou Edição de Sons; 3. Área de Edição: edição de Código por montagem de blocos, edição de imagens de Fantasias/Cenários ou edição de Sons; 4. Categorias: blocos

[1] Scratch possui versões em diversos idiomas

para construções dos programas estão agrupados nas categorias: Movimento, Aparência, Som, Evento, Controle, Sensores, Operadores, Variáveis, Meus Blocos; 5. Mochila: para transferência de código entre elementos (copiar e colar); 6. Palco: palco do projeto; 7. Elenco: atores que participam do projeto; 8. Cenários - cenários do projeto.

1.7 Portugol

O Portugol é uma linguagem de programação para fins didáticos que se assemelha à linguagem C, tendo por premissa facilitar a construção de programas sem a barreira do idioma. Portugol começou a ser utilizada em 2011 na UNIVALI (Universidade do Vale do Itajaí), onde a plataforma foi criada por Luiz Fernando Noschang, graduando de ciência da computação à época.

O ambiente de programação Portugol é o Portugol Studio que pode ser utilizado diretamente da internet, via navegador, sem a necessidade de instalação no computador, pelo endereço: https://portugol-webstudio.cubos.io/. Entretanto, caso seja do interesse, existem versões *Desktop*[1] para os sistemas operacionais Windows, macOS e Linux, ou seja, versões para serem instaladas e executadas na máquina do usuário sem conexão com a internet. Por questões de simplicidade e praticidade neste trabalho foi utilizada a versão da internet (*on-line*).

Um resumo da linguagem Portugol e as implementações dos algoritmos discutidos neste trabalho estão no apêndice A. Os algoritmos codificados em Portugol possuem um *link* para acesso ao código, evitando a necessidade de digitá-lo.

> Portugol Studio é um ambiente de desenvolvimento que foi criado com o enfoque na facilitação da aprendizagem de programação para usuários que dominam o idioma da língua portuguesa. Esse ambiente foi desenvolvido com o propósito de reduzir as dificuldades que iniciantes em programação passam ao se depararem a primeira vez com as linguagens de programação e ambientes de desenvolvimento comerciais. (ESTEVES, 2019)

1.8 Python

O Python foi criado no início dos anos 1990 por Guido van Rossum na Stichting Mathematisch Centrum na Holanda como um sucessor de uma linguagem chamada ABC. Atualmente, Python e R são as linguagens de programação mais utilizadas na área de ciência de dados.

Nas implementações deste trabalho, por questões de praticidade e simplicidade, foi utilizado o ambiente OnlineGDB. GDB[2] online é uma ferramenta web[3] para compilação e depuração. Entretanto, caso seja do interesse, existem versões *Desktop*[4] para os sistemas operacionais

[1] http://lite.acad.univali.br/portugol/
[2] GNU Debugger
[3] Funciona no navegador (*browser*), sem necessidade de instalação e configuração
[4] https://www.python.org/downloads/

Windows, macOS e Linux, ou seja, versões para serem instaladas e executadas na máquina do usuário sem conexão com a internet.

Um resumo da linguagem Python e as implementações dos algoritmos discutidos neste trabaho estão no apêndice B. Os algoritmos codificados em Python possuem um *link* para acesso ao código, evitando a necessidade de digitá-lo.

2 GEOMETRIA E CONSTRUCIONISMO DE PAPERT

Este capítulo aborda Geometria: construções de polígonos regulares com Scratch e com o Geogebra Geometria. Em Scratch foi adotada a abordagem Construcionista proposta por Papert, estilo geometria da Tartaruga do LOGO. Inicialmente são trabalhadas as construções geométricas do quadrado e do triângulo equilátero, passo a passo. Após, a construção é simplificada com um bloco de estrutura de repetição. Em seguida, são criados blocos para construções de quadrados, de triângulos equiláteros e de polígonos regulares com lado de qualquer dimensão. Seguindo, ainda utilizando Scratch, a abordagem é a construção de polígonos regulares circunscritos. Encerrando o capítulo, a construção de polígonos regulares circunscritos com o Geogebra Geometria aborda listas e estruturas de repetições que possibilitam construções geométricas em um paradigma diferente.

> A geometria da Tartaruga[1] é um estilo diferente de geometria, da mesma forma que o estilo axiomático de Euclides é bem diferente do estilo analítico de Descartes. O de Euclides é lógico, o de Descartes é algébrico. A geometria da Tartaruga é um estilo computacional de geometria (PAPERT, 1988, p.77)

A figura 9 exibe os elementos necessários para execução da atividade. O ator Taylor[2] (1) será movimentado no palco do Scratch desenhando ou não, a depender da situação da caneta (em uso ou levantada), fazendo papel similar ao da Tartaruga do LOGO. A área de desenho é um espaço bidimensional com $-240 \leq x \leq 240$ e $-180 \leq y \leq 180$. Os blocos utilizados da **Categoria Movimento (2)** serão: aponte para a direção $angulo$ (3) - para mudar o ângulo de inclinação do ator; gire esquerda $angulo$ graus (4) - girar o ator para esquerda; gire direita $angulo$ graus (5) - girar o ator para direita; vá para x y (6) - mover para a coordenada (x, y); deslize por $segundos$ para x, y (7) - mover para a coordenada (x, y) deslizando no tempo definido; mova $passos$ (8) - mover em linha reta $passos$ à frente obedecendo a direção para qual o ator estiver posicionado. Os blocos da **Categoria Caneta**[3] (9) serão: apague tudo (10) - para limpar a área de desenho (palco) para elaboração de um novo desenho; use caneta (11) - ativa o uso da caneta, ou seja, os movimentos que ocorrerem após serão riscando o palco; levante a caneta(12) - desativa o uso da caneta para reposicionar o ator sem riscar o palco; mude o tamanho da caneta para $espessura$ (13) - altera a espessura do traço da caneta; mude a cor da caneta para cor (14) - muda a cor da caneta. As construções deste capítulo estão disponíveis em: https://scratch.mit.edu/projects/551430034/

[1] Referência à Linguagem LOGO cuja programação determina movimentos de uma tartaruga
[2] Gato Felix foi substituído por Taylor para dar mais realismo ao desenho com uma visão de cima
[3] Categoria Caneta é uma extensão que necessita ser incluída. Use botão **Adicionar uma extensão**

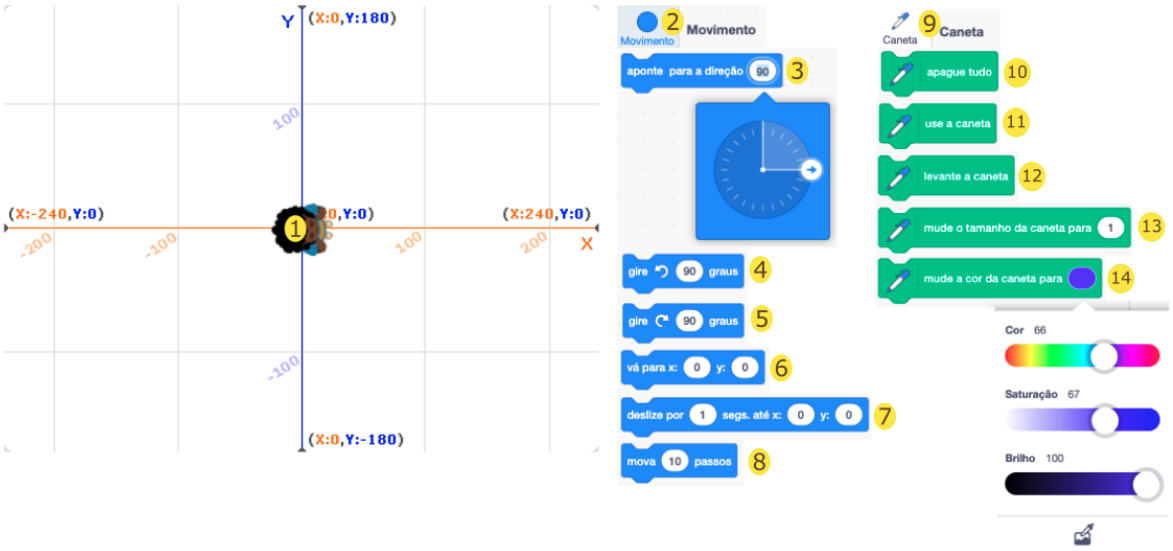

Figura 9 – Palco e Categorias para Desenho em Scratch (Elaborado pelo autor)

2.1 Construção de um Quadrado

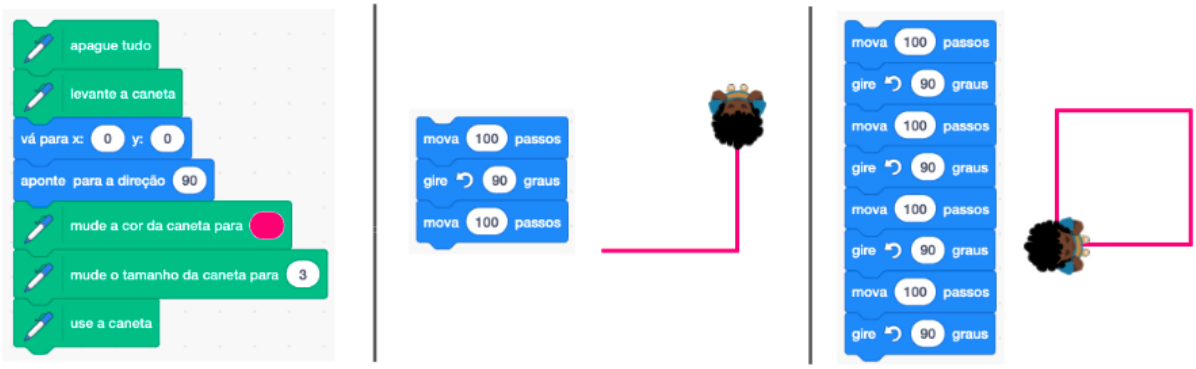

Figura 10 – Construção de Quadrado em Scratch (Elaborado pelo autor)

A figura 10 mostra à esquerda uma sequência de blocos para limpar o palco, posicionar o ator no centro com a orientação correta (posicionamento sem riscar, caneta levantada), configurar a cor e espessura do traço e abaixar a caneta. Esses passos deverão ser seguidos sempre que houver necessidade. Ao centro e a direita da figura estão passos para a construção de um quadrado de lado 100. O desenho consiste das instruções: mova 100 passos; gire para esquerda 90 graus; mova 100 passos; gire para esquerda 90 graus; mova 100 passos; gire para esquerda 90 graus; mova 100 passos; gire para esquerda 90 graus. Um clique nos blocos produzirá um quadrado de lado 100 na cor vermelha desenhado a partir da origem ($x = 0, y = 0$). O ator Taylor pode ser arrastado com o mouse para qualquer local do palco e um novo clique nos blocos montados para construção de um quadrado produzirá um novo desenho.

> A abordagem LOGO ou o construcionismo, embora originada na linguagem de programação LOGO, vai muito além. Trata-se de uma postura diante do ensinar e do aprender que difere da prática tradicional. No construcionismo

> o professor não caminha à frente do aluno, mas é seu parceiro no processo de aprendizagem. O professor propicia ao aluno condições de explorar o seu potencial intelectual no desenvolvimento de ideias e projetos que envolvem diferentes áreas de conhecimento, realizando sucessivas ações, reflexões e abstrações e criando seus próprios modelos intelectuais.(ALMEIDA, 2000, p.26)

Com essa primeira experiência de construção os alunos estarão aptos a fazerem suas próprias construções e artes. Quadrados com lados, espessuras e cores de traços diferentes podem ser produzidos em diversas localizações do palco. Ensaios e construções de outras formas geométricas e desenhos poderão ser realizados.

2.2 Construção de um Triângulo Equilátero

De maneira análoga à construção do quadrado, o ator Taylor poderá ser comandado com as instruções necessárias para produzir o desenho de um triângulo no palco. A figura 11 mostra os ângulos externos de um triângulo equilátero e a sequência de blocos de instruções para sua construção. O desenho consiste das instruções: mova 100 passos; gire para esquerda 120 graus; mova 100 passos; gire para esquerda 120 graus; mova 100 passos; gire para esquerda 120 graus. Novamente, o ator Taylor pode ser arrastado com o mouse para qualquer local do palco e um clique nos blocos que definem a construção de um triângulo produzirá um novo desenho.

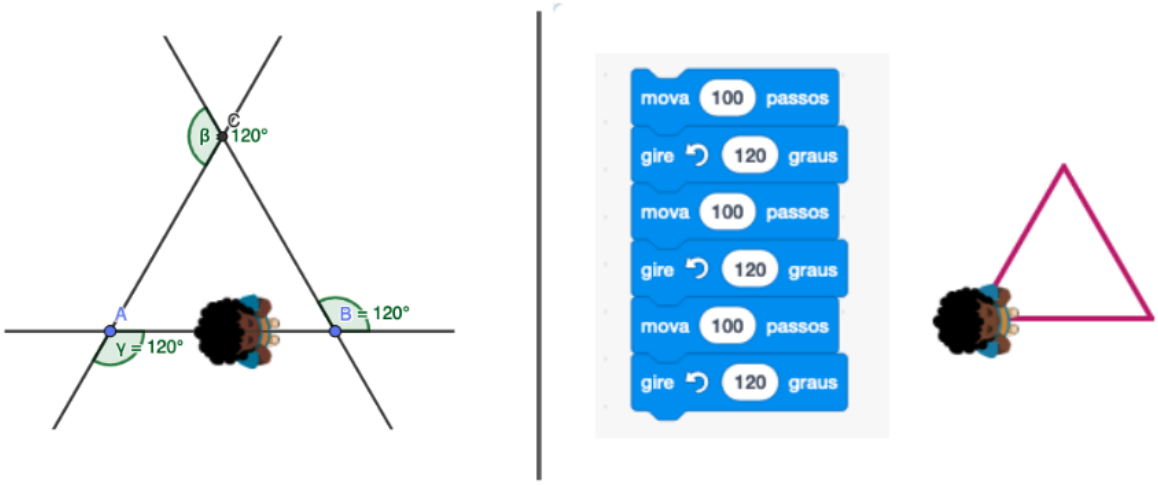

Figura 11 – Construção de Triangulo em Scratch (Elaborado pelo autor)

Simplificando Repetições

É perceptível que nas construções do quadrado e do triângulo existiram pares de comandos repetidos, quais sejam: no quadrado - 4 x [mova 100 passos; gire para esquerda 90 graus]; no triângulo - 3 x [mova 100 passos; gire para esquerda 120 graus]. A figura 12 mostra o Bloco de Controle de Repetição repita n vezes e sua utilizacão nas construções simplificadas do quadrado e do triângulo. O Bloco repita n vezes está na Categoria Controle.

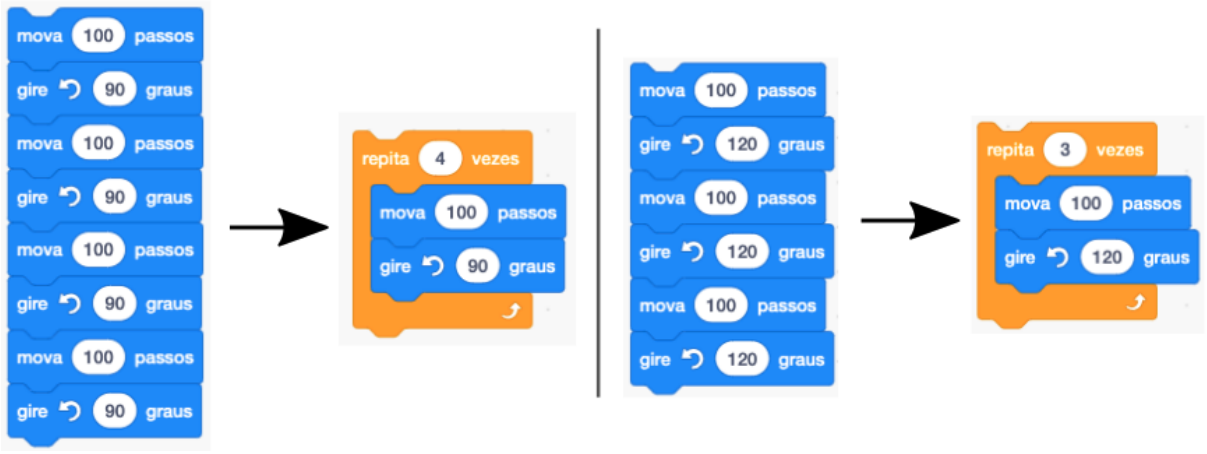

Figura 12 – Quadrado e Triângulo com Repita n Vezes (Elaborado pelo autor)

2.3 Blocos: Triângulo, Quadrado e Polígono

O Scratch permite aos usuários a criação de novos blocos com novas abstrações. As figuras 13 e 14 mostram as implementações para criações dos blocos Triângulo (equilátero) e Quadrado pelo usuário. A criação é simples e consiste: 1) Clicar no botão Criar um Bloco que está na Categoria Meus Blocos; 2) Na tela "Criar um Bloco" definir o nome do bloco (Quadrado ou Triangulo, conforme seja o caso); 3) Clicar em **Adicionar uma entrada** e definir *lado* como parâmetro de entrada. Um bloco defina será criado na área de código para definição da implementação, conforme figura. Também, na categoria Meus Blocos aparecerá o novo Bloco criado para ser utilizado nas construções com o valor do *lado* desejado.

Figura 13 – Bloco Triangulo em Scratch (Elaborado pelo autor)

Figura 14 – Bloco Quadrado em Scratch (Elaborado pelo autor)

Os blocos triângulo e quadrado podem ser utilizados livremente em construções. A figura 15 mostra um código que gera 50 quadrados de tamanhos aleatórios (valores entre 5 e 50), em cores diversas (o valor da cor é incrementado em 15 a cada desenho) e em posições aleatórias[1]. Uma arte aleatória que utiliza o bloco quadrado construído pelo usuário.

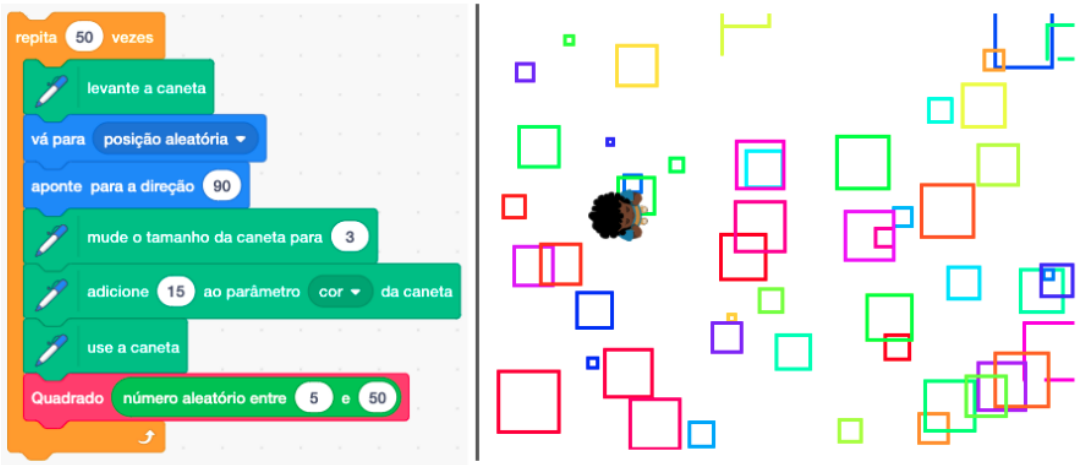

Figura 15 – Quadrados Aleatórios em Scratch (Elaborado pelo autor)

A figura 16 mostra o código de um bloco Polígono Regular com dois parâmetros: o número n de lados e o tamanho a do lado. O processo de construção é a generalização dos processos das construções do triângulo (3 lados) e quadrado (4 lados). O exemplo faz as construções de um triângulo, um quadrado, um pentágono e um hexágono nas cores vermelho, laranja, verde e azul, respectivamente. Como nas construções o ator sempre parte do mesmo ponto (origem), todas as figuras possuem um lado sobreposto. Experiências com construções de polígonos, com cores, quantidade de lados e tamanhos direntes podem ser realizadas com os alunos.

[1] As posições aleatórias deveríam ter restrições de localização em função do tamanho do quadrado a ser desenhado. Isso não foi tratado no exemplo para que seja percebido e gerado questionamento

Figura 16 – Bloco Polígono Regular em Scratch (Elaborado pelo autor)

Um polígono é dito **regular** se todos os seus lados e todos os seus ângulos internos tiverem medidas iguais. Em um polígono regular de n lados, cada ângulo interno[1] deve medir $\frac{180^o(n-2)}{n}$. Todo polígono regular é inscritível e circunscritível, e os círculos inscrito e circunscrito têm um mesmo centro Neto (2013). A seguir, será abordada a construção de um bloco que possibilita o desenho de um polígono regular circunscrito em uma circunferência tendo como parâmetros: número de lados do polígono, coordenadas x_c e y_c do centro e $raio$ da circunferência.

2.4 Polígonos Regulares Circunscritos

A figura 17 mostra o código de um Bloco denominado $PoligonoRegularCircunscrito$ que possui os parâmetros: número de lados n, coordenadas x e y do centro e $raio$ da circunferência circunscrita. Esse bloco serve para construção de polígonos regulares circunscritos e utilizando uma chamada $PoligonoRegularCircunscrito(360, x_c, y_c, raio)$[2] é possível construir uma pseudo circunferência de raio r e centro (x_c, y_c). No exemplo de utilização na figura foram construídos triângulo, quadrado, pentágono, hexágono e a pseudo circunferência, todos circunscritos numa circunferência de raio 100.

[1] Ângulos interno e externo são suplementares. O ângulo interno corresponde a $180^o - \frac{360^o}{n}$ (ângulo externo)

[2] Antes do traçado do polígono, há um giro de metade do ângulo interno para que o eixo x seja bissetriz do primeiro ângulo

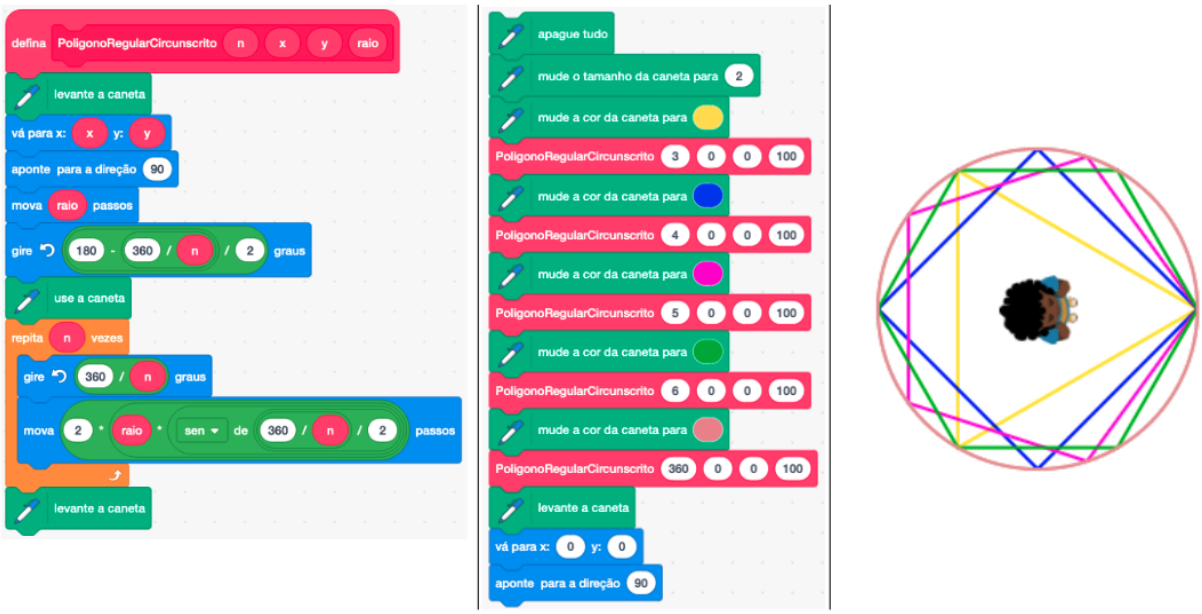

Figura 17 – Polígonos Regulares Circunscritos em Scratch (Elaborado pelo autor)

2.5 Geogebra - Polígonos Regulares Circunscritos

Nesta seção serão abordadas construções de polígonos regulares circunscritos utilizando o **Geogebra Geometria** disponível em: https://geogebra.org/geometry. As construções serão realizadas em modo álgebra com a entrada dos comandos (diferente do modo ferramenta em que as entradas são por botões e diálogos).

Serão definidos o ponto A, a circunferência C e o Polígono P de n lados (os pontos que definem o polígono estarão em uma lista L de n elementos). Todos os elementos, exceto n, estarão vinculados ao ponto A, centro da circunferência, para que não fiquem independentes (movimentados isoladamente). As definições dos elementos seguem:

- Número de lados n do polígono - será utilizado o comando *ControleDeslizante(<Mínimo>, <Máximo>, <Incremento>, <Velocidade>, <Comprimento>, <Ângulo true | false>, <Horizontal true | false>, <Animar true | false)>, <Aleatório true | false>)* para criar um controle deslizante para a variável n:
  n=ControleDeslizante(3, 20, 1, 1, 100, false, true, false, false)

- Ponto A centro da circunferência - um ponto em Geogebra é definido por um par de coordenadas entre parênteses (x,y) atribuído a uma letra maiúscula (letras minúsculas são utilizadas para vetores): **A=(0,0)**

- Círculo C - será utilizado o comando *Círculo(<Ponto>, <Raio>)* para criar o círculo C com centro no ponto A e raio 1: **C=Círculo(A, 1)**

- Lista L de pontos do polígono regular circunscrito - será utilizado o comando *Sequência(<Expressão>, <Variável>, <Valor Inicial>, <Valor Final>)* para criar a lista de pontos

que definirão o polígono circunscrito. A variável de controle i percorrerá os valores de 1 a n. Para cada valor de i será gerado um ponto na lista com as coordenadas $(cos(i * 360^o/n) + x(A), sin(i * 360^o/n) + y(A))$ que será a <*Expressão*> do comando: **L=Sequência**$((\mathbf{cos(i/n * 360^o)} + \mathbf{x(A)}, \mathbf{sin(i/n * 360^o)}) + \mathbf{y(A)}, \mathbf{i}, \mathbf{1}, \mathbf{n})$ Note que as coordenadas do ponto A obtidas por x(A) e y(A) foram acrescidas a cada ponto para que o polígono fique vinculado ao círculo. Utilize Alt + O (Mac OS: Ctrl + O) para o símbolo de grau.

- Polígono P - será utilizado o comando *Polígono(<Lista de Pontos>)* para criar um polígono com os pontos que estão na lista L: **P=Polígono(L)**

A figura 18 exibe a tela do Geogebra Geometria com o controle deslizante na posição n=6 e a imagem de um Hexâgono circunscrito.

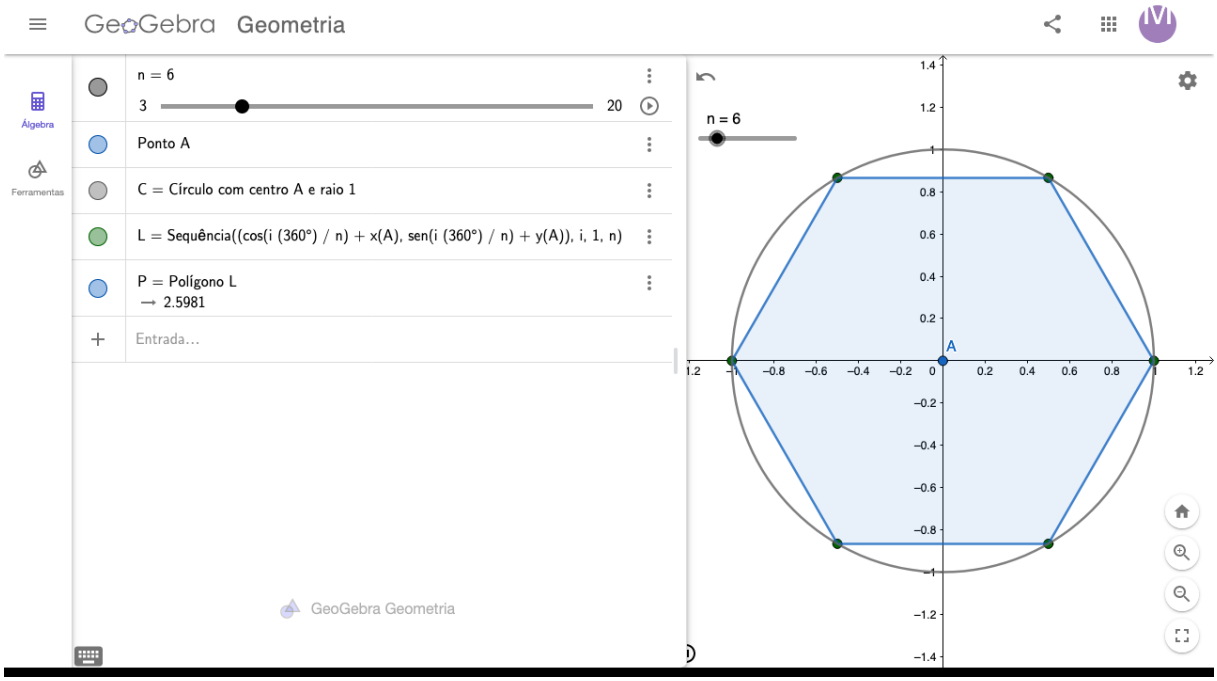

Figura 18 – Polígonos Regulares Circunscritos em Geogebra (Elaborado pelo autor)

Geogebra também possui recursos para abordagem Construcionista de Papert utilizando a Geometria da Tartaruga. Veja em: https://www.geogebra.org/m/bfhwq8ds.

3 ALGORITMOS E PROGRAMAÇÃO

Este capítulo trata conceitos de algoritmos e programação. Primeiramente, são vistos como dados podem ser representados no computador e quais as operações possíveis de serem realizadas com esses dados. Após a abordagem dos dados, são apresentados comandos de atribuição e comandos de interação. Os comandos de atribuição servem para armazenar e manipular dados na memória do computador. Já os comandos de interação servem para troca de dados e informações entre o computador e o usuário. Após a abordagem dos comandos de atribuição e interação são vistas as estruturas de controle condicional e de controle de repetições. Essas estruturas permitem alterar a forma natural de processamento que é sequêncial. O controle condicional condiciona a execução de um grupo de instruções ao teste de uma condição resultar em verdadeiro ou falso. Já as estruturas de controle de repetições permitem que um conjunto de instruções seja repetido de acordo com algum critério. Por fim, são abordados conceitos de procedimentos e funções, ou seja, formas de construções de novas abstrações (novos de comandos).

> Papert considera que o computador deve permitir a construção do conhecimento através do aprender fazendo e do pensar sobre o que se está fazendo, possibilitando por intermédio do ato de programar o computador a ação reflexiva do educando sobre um resultado e sobre seu próprio pensamento.
> (CAMPOS, 2013, p. 16)

Cormen (2002) define “Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída.”. Usualmente, um algoritmo é escrito em pseudocódigo para, posteriormente, ser codificado em uma linguagem de programação. Entretanto, na programação de computadores não há obrigatoriedade de se construir algoritmos. Quando já se tem o pensamento computacional desenvolvido, as construções de programas podem ser diretas.

Fazendo uma analogia entre uma receita de bolo e um algoritmo: os ingredientes e o próprio bolo seriam equivalentes aos dados de entrada e de saída em um algoritmo; os passos da receita para produção do bolo seriam, no algoritmo, a sequência de comandos e estruturas de controles para a partir dos dados da entrada produzir a saída.

Na elaboração de um algoritmo computacional existe uma certa “liberdade controlada". A liberdade de se desenvolver uma lógica correta, na forma de expressão do pensamento computacional do autor. Controlada porque os dados que serão manipulados devem estar alinhados aos tipos de dados que a linguagem de programação dispõe. Também, os processos, ou ações, de tratamento desses dados devem ser aqueles que a linguagem de programação implementa. Resumindo, programas devem ser escritos na linguagem e no formato que o computador seja capaz de processar.

Neste capítulo serão abordados conceitos de tipos de dados, variáveis, operações, comandos e estruturas de controles. Primeiramente, de forma genérica. Em seguida, particularizando para a programação Scratch. Nos apêndices estão as abordagens em Portugol e Python.

3.1 Tipos de Dados, Variáveis e Operações

A computação tenta imitar a matemática em diversos aspectos. Álgebra, aritmética, lógica matemática e outras áreas da matemática têm suas equivalentes na computação. Na matemática existem os conjuntos $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{I}, \mathbb{R}, \mathbb{C}$ para representar grandezas. Na computação existem tipos simples e estruturas de dados para as representações dos dados que serão processados. Tipos Simples de Dados são utilizados para representar dados elementares, tais como: $logico$ (falso ou verdadeiro), $caractere$ (letras, sinais de pontuação, dígitos e símbolos), $inteiro$, $real$, $complexo$, dentre outros. Já as Estruturas de Dados são utilizadas para representar agrupamentos de dados, por exemplo: $cadeia$ (cadeia de caracteres ou *string* - agrupamento de caracteres), $lista$, $matriz$, $dicionario$, $tabela$, $pilha$, $fila$, $arvore$, $grafo$, etc.

As implementações computacionais de tipos numéricos têm restrições. Representações de tipos $inteiro$ e $real$ em **computação numérica** possuem limites inferiores e superiores. Para valores do tipo $real$, além dos limites inferiores e superiores, existem restrições na precisão da representação. Ou seja, um tipo real em computação numérica poderá representar de forma exata somente alguns valores racionais e nenhum valor irracional. A tabela 1 mostra algumas limitações de representações de tipos $inteiro$ e $real$ do Visual C da Microsoft. Para computar racionais (frações) e/ou irracionais de forma mais precisa existe a **computação simbólica ou algébrica**, onde os valores são representados e processados simbolicamente, similar à forma que escrevemos, por exemplo $\sqrt{2},\ ou\ \frac{1}{3}$. A computação simbólica ou algébrica também é utilizada no processamento de grandes números como será abordado no capítulo 7 com a utilização do Geogebra CAS.

Tipo	Bits	Min	Max
char	8	-128	127
int	32	-2147483648	2147483647
float	48	1.175494351e-38F	3.402823466e+38F

Tabela 1 – Alguns tipos de dados do Visual C (Fonte: Microsoft)

> *A Computação Numérica envolve não só as operações aritméticas básicas de adição, subtração, multiplicação, divisão e radiciação, mas também procedimentos mais sofisticados, a exemplo da determinação das raízes de um polinômio por meio de métodos de aproximação. Essa modalidade de computação envolve apenas números: os dados iniciais e os resultados finais são números.*
> ...
> *Por outro lado, um programa de computação algébrica mostrará resultado como sendo* $\sqrt{2}\sqrt{3} = \sqrt{6}$*, algo bastante próximo do que se faz em sala de*

> *aula. Assim, um programa desse tipo efetua cálculos com radicais sem a necessidade de representá-los em forma decimal aproximada, pois ?conhece? as regras e propriedades algébricas dos objetos envolvidos. Além disso, é possível operar com objetos matemáticos que não são números. Por exemplo, um programa de computação algébrica pode desenvolver identidades algébricas como* $(a+b)^2 = a^2 + 2ab + b^2$ *ou* $(a+b)(a-b) = a^2 - b^2$*, sem atribuir quaisquer valores numéricos para a e b.* (ANDRADE, RPM83)

Na computação os dados estão representados em memória associados a um tipo de dado e a um nome ou identificador da variável. O tipo de dado define e delimita quais operações são possíveis de serem realizadas com o dado. Portanto, uma variável na computação é um identificador de uma posição de memória associada a algum tipo de dado. Por exemplo, o valor 65 [0100 0001] em um byte[1] na memória do computador pode estar representando o valor inteiro 65, a letra "A" ou parte de algum outro dado.

Variáveis do tipo $inteiro$ podem ser operadas aritmeticamente com as operações: adição (+), subtração (-), multiplicação (*), divisão inteira (/), resto da divisão inteira (mod). Dependendo da linguagem de programação a divisão pode ter tratamentos distintos para operar divisão inteira e divisão real. Por exemplo, na Linguagem de Programação Pascal: $7\ div\ 2 = 3$ e $7/2 = 3.5$.

Variáveis do tipo $real$ podem realizar operações: adição (+), subtração (-), multiplicação (*) e divisão (/). Algumas linguagens implementam a potenciação com o operador ^ ou **. Para realizar cálculos de potenciação em Linguagens que não implementem essa operação, via de regra, as funções e^x e $ln(x)$ são utilizadas na forma $\mid a \mid^b = e^{b \cdot ln|a|}$.

$$\mid a \mid^b = x \leftrightarrow ln \mid a \mid^b = ln(x) \leftrightarrow b \cdot ln \mid a \mid = ln(x) \leftrightarrow e^{b \cdot ln|a|} = e^{ln(x)} \leftrightarrow e^{b \cdot ln|a|} = x \quad (3.1)$$

Variáveis do tipo $caractere$ são utilizadas no tratamento de letras, dígitos, sinais de pontuação, caracteres especiais, etc. Usualmente caracteres são manipulados na forma estruturada $cadeia$ (agrupamentos de caracteres).

Caracteres podem ser representados em diferentes padrões de codificações, tais como: ASCII - *American Standard Code for Information Interchange* (Código Padrão Americano para o Intercâmbio de Informação); EBCDIC - *Extended Binary Coded Decimal Interchange Code* (Decimal Codificado em Binário Estendido); UTF-8 - *8-bit Unicode Transformation Format* (Formato de Transformação Unicode); dentre outros. A figura 19 exibe valores decimais para letras, dígitos, sinais de pontuação e caracteres especiais no sistema de codificação ASCII.

[1] Os termos *byte* e *bit* são acrônimos para *binary term* (termo binário) e *binary digit* (dígito binário), respectivamente. Um byte corresponde a 8 bits

DEC	CARACTERE	DEC	CARACTERE	DEC	CARACTERE	DEC	CARACTERE	DEC	CARACTERE	DEC	CARACTERE
32		48	0	64	@	80	P	97	a	114	r
33	!	49	1	65	A	81	Q	98	b	115	s
34	"	50	2	66	B	82	R	99	c	116	t
35	#	51	3	67	C	83	S	100	d	117	u
36	$	52	4	68	D	84	T	101	e	118	v
37	%	53	5	69	E	85	U	102	f	119	w
38	&	54	6	70	F	86	V	103	g	120	x
39	'	55	7	71	G	87	W	104	h	121	y
40	(	56	8	72	H	88	X	105	i	122	z
41	)	57	9	73	I	89	Y	106	j	123	{
42	*	58	:	74	J	90	Z	107	k	124	|
43	+	59	;	75	K	91	[	108	l	125	}
44	,	60	<	76	L	92	\	109	m	126	~
45	-	61	=	77	M	93	]	110	n	127	
46	.	62	>	78	N	94	^	111	o		
47	/	63	?	79	O	95	_	112	p		

Figura 19 – Tabela parcial ASCII - *American Standard Code for Information Interchange*
(Elaborado pelo autor)

Variáveis do tipo $booleano$ (ou lógico) representam um valor $falso$ ou $verdadeiro$ e são básicas no controle do fluxo do processamento. O fluxo natural de execução de instruções de um algoritmo ou programa é sequêncial, da primeira à última instrução. Entretanto, as estruturas de controles condicionais e de repetições podem modificar essa forma sequencial, seja condicionando a execução de determinadas instruções à satisfação de alguma condição lógica, seja repetindo determinadas instruções por um número de vezes ou até que alguma condição seja alcançada. Esses controles serão abordados nas próximas seções deste capítulo.

Variáveis do tipo $booleano$ podem ser operadas com as operações: disjunção inclusiva (ou), conjunção (e), negação (não). Essas operações são detalhadas no capítulo 4.

Ambos os tipos $inteiro$, $real$, $caractere$ e $booleano$ tem ordenação bem definida e, portanto, podem ser operados relacionalmente, dois a dois elementos de um mesmo tipo, pelos operadores $=, \neq, <, \leq, >, \geq$. Para os tipos $inteiro$ e $real$, o critério de ordenação é o da matemática. Para o tipo $caractere$, a ordenação seguida é a disposição do elemento no Sistema de codificação. Por exemplo: se a codificação for ASCII, '0'<'A'<'a' (vide figura 19). Para o tipo $booleano$ $falso < verdadeiro$, pois $falso$ equivale a 0 e $verdadeiro$ equivale a 1.

No escopo da abordagem de algortimos e programação deste trabalho serão tratados apenas dois tipos estruturados de dados: $cadeia$ (agrupamentos de caracteres) e $lista$ (agrupamentos de dados de um mesmo tipo).

As operações mais comuns para o tipo $cadeia$ são: a concatenação, usualmente realizada pelo operador $+$; a manipulação elemento a elemento, tratada na forma indexada $cadeia[i]$ (caractere da posição i) ou por alguma função definida; função para identificar o tamanho de uma $cadeia$; dentre outras. Em algumas linguagens de programação é possível operar relacionalmente (comparar) valores de duas variáveis do tipo $cadeia$. O Scratch não implementa essa funcionalidade (comparar cadeias de caracteres).

Variáveis indexadas podem ser $vetor$, $arranjo$ (*array*) ou $matriz$, dependendo da linguagem de programação. Listas são variáveis indexadas em um nível de abstração mais elevado pois já existem algumas operações implementadas, tais como: remover elemento de uma posição, incluir elemento em alguma posição, além de pesquisar ocorrência de valor nela.

SCRATCH - Tipos de Dados, Variáveis e Operações

O Scratch tem suporte a três tipos de dados: booleanos, números e *strings*. Um tipo booleano pode ser utilizado para testar uma ou mais condições e, de acordo com o resultado, fazer o programa selecionar um caminho diferente de execução. Uma variável numérica pode armazenar tanto valores inteiros quanto valores decimais, Scratch não faz distinção. Uma *string* corresponde a uma sequência de caracteres e pode ser usado para armazenar nomes, endereços, etc. O Scratch faz a conversão automática entre tipos de dados. (MARJI, 2014)

Blocos Scratch associados a valores numéricos ou *strings* tem forma elíptica (ou retangular com laterais arredondadas) e blocos associados a um valor booleano tem forma hexagonal.

Variáveis

A **Aba Código, categoria Variáveis**, contém elementos para criar e manipular variáveis em um projeto Scratch. Na figura 20 à esquerda estão blocos da categoria Variáveis (8) da aba Código (1) vinculado a um ator ou cenário de um projeto Scratch. O botão Criar uma Variável (2) permite criar variáveis para o projeto. A tela de criação de uma nova variável (9) solicita nome da variável, escopo de visibilidade (todos os atores ou apenas um ator) e se será uma variável armazenada na nuvem. Utilizaremos apenas variáveis globais (visibilidade a todos atores) nos projetos deste trabalho.

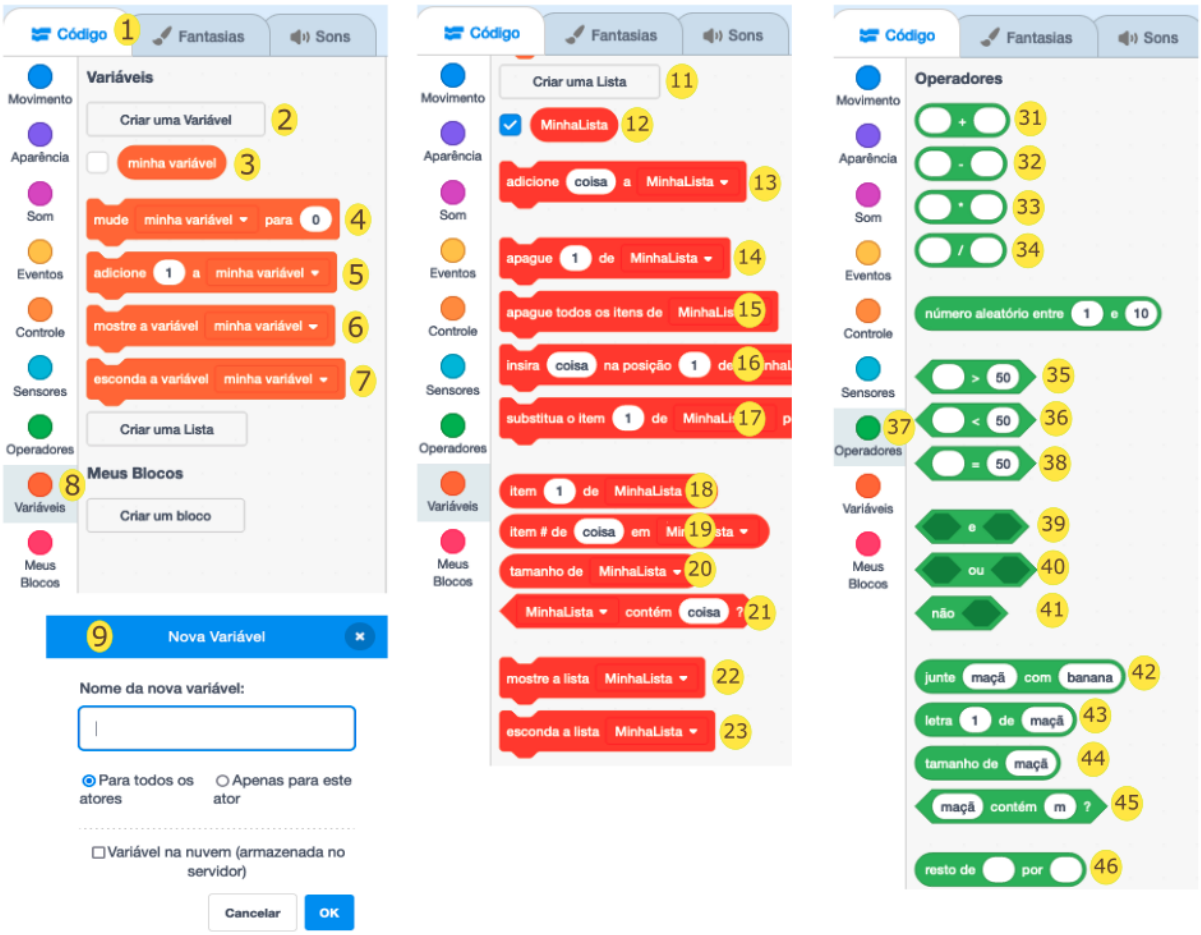

Figura 20 – Blocos das Categorias Variáveis e Operadores do Scratch (Elaborado pelo autor)

Quando uma variável é criada ela passa a ser um bloco Scratch em forma elíptica na categoria Variáveis (3). Ela pode ser arrastada para área de código e fazer parte de blocos de expressões ou de controle do processamento. Clicando com o botão direito do mouse sobre o bloco da variável é possível trocar o nome ou apagar. A caixa de checagem ao lado do nome da variável (12) serve para habilitar ou desabilitar a exibição da variável no palco.

Os blocos Scratch para manipular variáveis são: mude (4) - para mudar o valor de uma variável; adicione (5) - para adicionar um valor à variável; mostre (6) - exibir a variável no palco; esconda (7) - ocultar a exibição da variável no palco.

Listas

Ainda na **Aba Código, Categoria Variáveis**, estão os elementos para criar e manipular **listas** em um projeto Scratch. Ao centro da figura 20 estão blocos da categoria Variáveis para operar Listas. O botão Criar uma Lista (11) permite criar listas em um projeto. A criação de listas é análoga à criação de variáveis e listas assumem forma elíptica (12). Os blocos Scratch para manipular listas são: adicione (13) - para adicionar elemento ao final da lista; apague (14) - para apagar elemento de uma posição um valor à variável; apague todos (15) - para apagar todos os elementos; insira na posição (16) - inserir elemento em uma posição; substitua (17) - substituir valor de um elemento de uma posição; item (18) - valor do elemento de uma posição; item # de (19) - a posição de um valor na lista; tamanho de (20) - tamanho da lista; contém (21)- testa se um valor consta na lista; mostre (22) - exibir a lista no palco; esconda (23) - ocultar a exibição da lista no palco.

Marji (2014, p.245) define "Uma lista é um contêiner em que você pode armazenar e acessar vários valores. Podemos pensar nela como uma cômoda com várias gavetas, em que cada gaveta armazena um único item".

Operadores

Na **Aba Código, categoria Operadores**, estão blocos para realizar operações aritméticas, relacionais e booleanas, dentre outras, em um projeto Scratch. Na figura 20 à direita estão blocos da categoria Operadores (37).

Os **operadores aritméticos** para valores numéricos são os blocos elípticos para adição $+$ (31), subtração $-$ (32), multiplicação $*$ (33) e divisão $/$ (34). Ainda, para manipulações de valores numéricos existem blocos para cálculo do resto (46) e de funções matemáticas (módulo, arredondamento para baixo, arredondamento para cima, raiz quadrada, sen, cos, tg, arcsen, arcos,arctg, ln, logaritmo, e elevado à, 10 elevado à)

Os blocos dos **operadores relacionais** são: $>$ (35), $<$ (36) e $=$ (37). Eles resultam em um valor booleano e, portanto, são representados por blocos em formato hexagonal. Não existem blocos prontos para $\leq, \neq$ e $\geq$. Então, as operações $\leq, \neq$ e $\geq$ devem ser construídas por combinações de operações relacionais e operações lógicas, assunto abordado no capítulo 4.

Os blocos dos **operadores booleanos** são: e (39), ou (40) e nao (41). Eles operam e resultam valores booleanos. Portanto, tanto os operandos quanto o resultado têm formatos hexagonais.

Os blocos para operar $strings$ são: junte (42) - concatena $strings$; letra (43) - acessa uma posição da $string$; tamanho (44) - tamanho da $string$; contém (45) - pesquisa se uma $string$ ocorre dentro de outra $string$. Os blocos junte e letra resultam $string$, portanto tem formato elíptico. O bloco tamanho também tem formato elíptico pois resulta um número. Já o bloco contém tem como resultado um valor booleano, por isso tem a representação hexagonal.

3.2 Interação e Manipulação de Dados

Na seção anterior foi visto que um algoritmo é uma sequência de comandos e estruturas de controles para, a partir dos dados da entrada, produzir alguma informação na saída. Nesta seção será visto como a interação entre o computador e o usuário é representada em um algoritmo. Também, como os dados são armazenados em variáveis na memória do computador e manipulados.

Nos algoritmos deste trabalho a interação entre o computador e o usuário será tratada de forma genérica com as diretivas **entrada** e **saida**. A entrada sempre será realizada para alguma variável e a saída poderá ser de literal, variável ou combinação desses. Conteúdos entre aspas são literais e serão reproduzidos tais quais escritos. As variáveis devem ser definidas na seção **Var** por um nome de variável associado a algum tipo ou estrutura de dados. O algoritmo 1 apresenta o formato de um algortimo simples que solicita o nome do usuário e faz uma saudação.

Algoritmo 1: Entrada e saída de dados

```
Var Nome: cadeia
Inicio
    Saída: "Qual o seu nome?"
    Entrada: Nome
    Saída: "Olá, ", Nome, "!"
Fim
```

O valor de um variável pode ser utilizado numa saída ou como um operando parte de uma expressão. Para armazenar um valor em uma variável deve ser utilizada a atribuição $\leftarrow$. [1] O algoritmo 2 exemplifica o uso de variáveis em expressões e atribuições. A lógica consiste nas leituras do preço de um produto e de um percentual de desconto que deverá ser aplicado. O cálculo é processado e o preço com desconto exibido.

[1] Algumas linguagens de programação sobrecarregam o operador $=$, utilizando-o tanto como operador de atribuição quanto como operador relacional (comparação de igualdade)

Algoritmo 2: Atribuição de valor a variável

Var $preco, desconto, precocomdesconto$: real
Inicio
Saída: "Qual Preço Original do Produto?)"
Entrada: $preco$
Saída: "Qual o desconto em percentual?)"
Entrada: $desconto$
$precocomdesconto \leftarrow preco * (1 - (desconto/100))$
Saída: "Preço com desconto é: ", $precocomdesconto$
Fim

SCRATCH - Interação

Na figura 21 à esquerda estão blocos da categoria Aparência (1) da aba Código vinculado a um ator ou cenário de um projeto Scratch. Para **saída** de dados poderão ser utilizados os blocos: diga por segundos (2); diga (3); pense por segundos (4); pense (5). Já na figura 21 à direita estão blocos da categoria Sensores (11) da aba Código vinculado a um ator ou cenário de um projeto Scratch. Para **entrada** serão utilizados os blocos: pergunte e espere (12); resposta (13). Após o processamento de um bloco pergunte e espere (12) o valor digitado fica na variável interna $resposta$. É prudente realizar o armazenamento de $resposta$ em alguma variável previamente definida.

Figura 21 – Blocos das Categorias Aparência e Sensores do Scratch (Elaborado pelo autor)

Passo a passo para codificar o Algoritmo 1 em Scratch - veja figura 23

Abra a página do Scratch https://scratch.mit.edu/. Crie um novo projeto Scratch. Crie uma variável $Nome$, conforme já descrito anteriormente. Arraste os seguintes blocos para área de código vinculado ao ator *cat*: **Categoria Eventos** - (1) quando for clicado; **Categoria Sensores**

- (2) pergunte e espere e (3) resposta; **Categoria Variáveis** - (4) mude e (5) Nome[1]; **Categoria Operadores** - (6) junte com; **Categoria Aparência** - (7) diga.

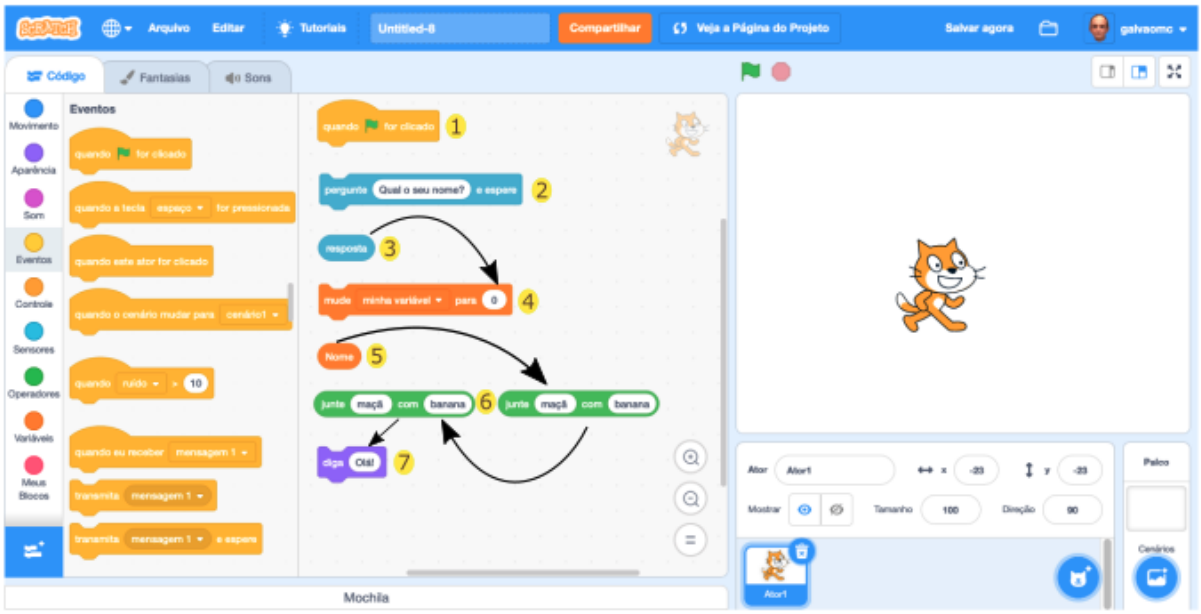

Figura 22 – Codificação do Algoritmo 1 em Scratch (Elaborado pelo autor)

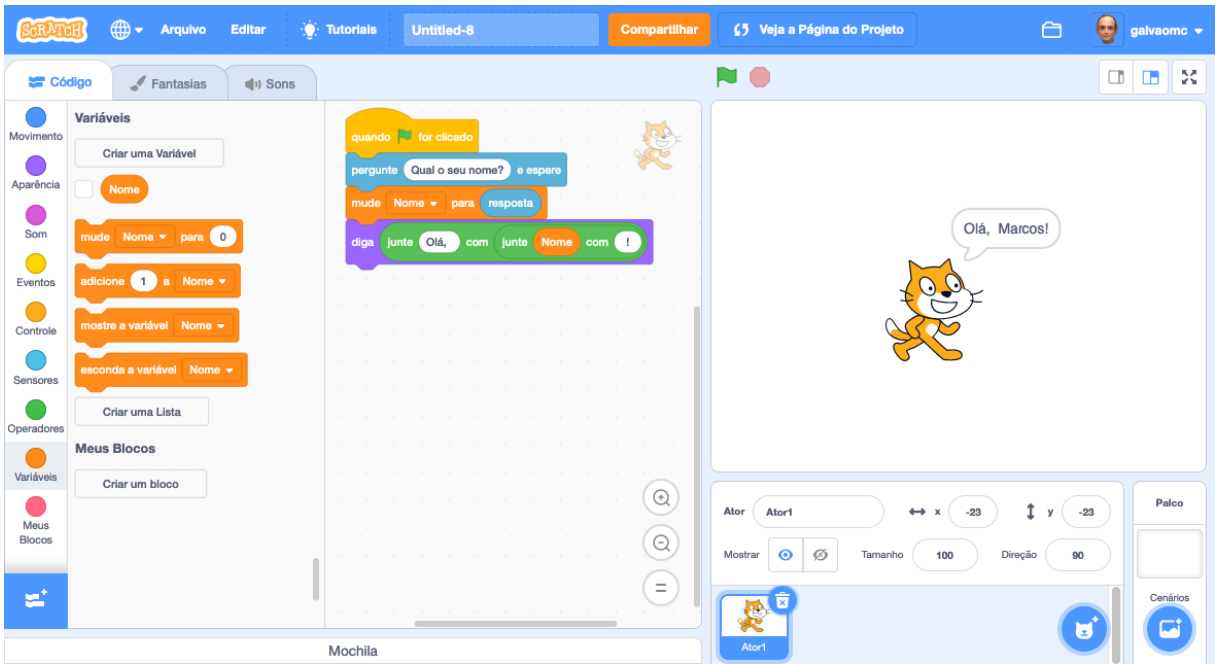

Figura 23 – Primeiro Projeto Scratch (Elaborado pelo autor)

Em seguida, antes de encaixar todos os blocos na sequência, prepare os seguintes blocos: a) encaixe o bloco resposta (3) no bloco mude (4) e, na lista desse bloco mude, selecione a variável $Nome$; b) encaixe o bloco Nome (5) na primeira lacuna do segundo bloco junte com

[1] Variável deverá estar previamente criada

(6) e preencha a segunda lacuna com um ponto de exclamação. Mova esse bloco junte com para segunda lacuna do primeiro bloco junte com e na primeira lacuna do primeiro bloco preencha "Olá, " (sem aspas); c) Arraste a composição de blocos junte com para dentro do bloco diga (7). Finalmente, encaixe todos os blocos na sequência, seguindo a lógica do algoritmo 1. O resultado deverá ser idêntico ao da figura 23. Clique na bandeira verde para execução e não esqueça de gravar o projeto, caso tenha interesse de utilizá-lo novamente.

Proceda de modo análogo para o algoritmo 2. Crie novo projeto, encaixe os blocos seguindo a lógica do algoritmo. Execute para diversos valores de mercadorias e descontos. Grave o projeto, caso tenha interesse de utilizá-lo novamente.

3.3 Estrutura de Controle Condicional

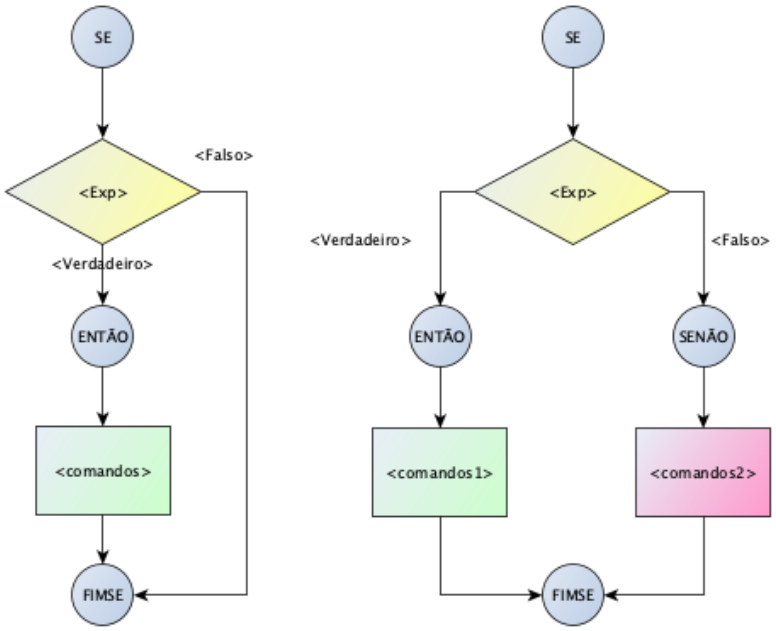

Figura 24 – Fluxos das Estruturas de Controle Condicional (Elaborado pelo autor)

A estrutura de controle condicional possibilita condicionar a execução de instruções a um valor lógico ($falso$ ou $verdadeiro$). A figura 24 apresenta em forma de fluxo as suas duas formas:

- **Forma simples (se/então)** - os comandos vinculados à condição serão executados caso o teste resulte $verdadeiro$ no momento do processamento. Para um resultado $falso$, nada será processado;
- **Forma composta (se/então/senão)** - apresenta dois conjuntos de comandos, mutuamente exclusivos. Os comandos vinculados ao se/então serão executados caso a expressão tenha valor $verdadeiro$ no momento da verificação. Caso o valor seja $falso$, os comandos vinculados ao se/senão que serão executados.

O algoritmo 3 faz a leitura de um valor (positivo ou negativo) e exibe o valor absoluto. A estrutura de controle condicional é utilizada para testar o valor da variável $Valor$ lida. Se o $Valor$ for maior ou igual a zero, nada será feito. Entretanto, se o valor for negativo, será calculado $Valor * (-1)$ e o resultado atribuído para a variável $Valor$.

Algoritmo 3: Estrutura condicional simples - Se/Entao

Var $Valor$: real
Inicio
  Saída: "Informe um valor (positivo ou negativo)"
  Entrada: $Valor$
  se $Valor < 0$ **então**
    $Valor \leftarrow Valor * (-1)$
  fim se
  Saída: "Valor absoluto é:", $Valor$
Fim

O algoritmo 4 é um exemplo para a estrutura de controle condicional composta. A construção é para testar se um aluno está aprovado ou reprovado - o critério de aprovação é nota maior ou igual a 7. Os passos do algoritmo consistem em: fazer a leitura da $Nota$ e testar a condição $Nota \geq 7$. Caso o teste da expressão resulte $Verdadeiro$, uma mensagem informando que o aluno está aprovado será exibida. Caso contrário ($falso$), a mensagem será que o aluno está reprovado.

Algoritmo 4: Estrutura condicional composta - Se/Entao/Senao

Var $Nota$: real
Inicio
  Saída: "Informe a nota do aluno"
  Entrada: $Nota$
  se $(Nota \geq 7)$ **então**
    Saída: "Aprovado!"
  senão
    Saída: "Reprovado!"
  fim se
Fim

SCRATCH - Estruturas de Controle Condicional

A figura 25 apresenta à esquerda o algoritmo 3 e à direita o algoritmo 4, ambos codificados em Scratch. Nos blocos de operações do Scratch não existe um bloco para teste simultâneo das condições maior ou igual. Logo, essa operação precisa ser construída pela negação (complemento) da expressão relacional $Nota < 7$. Outra forma seria combinar o teste de igualdade (=)

Figura 25 – Exemplos de Controle Condicional em Scratch (Elaborado pelo autor)

com o teste maior que (>) utilizando o operador de disjunção "ou" $((Nota = 7)\ ou\ (Nota > 7))$ - O capítulo 4 aborda essas construções no contexto da Álgebra de Boole.

A forma mais simples, entretanto, seria utilizar apenas a condição $Nota < 7$ e fazer a inversão dos blocos, ficando diga "Reprovado" vinculado ao **se/então** e diga "Aprovado" vinculado ao **se/senão**. Assim como na matemática, um problema computacional pode ter múltiplas construções de soluções, ou seja, a expressão do pensamento computacional em diferentes formas.

Diversas atividades poderão ser propostas para alunos trabalharem com a Estruturas de Controle Condicional. Algumas delas: ler um valor e afirmar se ele é par ou ímpar; ler os coeficientes A, B e C de uma equação do segundo grau da forma: $a \cdot x^2 + b \cdot x + c = 0$, calcular e exibir as raízes reais, se existirem (fórmula de Bhaskara); ler três valores e afirmar qual tipo de triângulo formam, caso formem (desigualdade triangular e Pitágoras); etc.

3.4 Estruturas de Controle de Repetições

As estruturas de controle de repetição servem para que um conjunto de instruções seja repetido por um determinado número de vezes, enquanto uma condição não for atingida ou até que uma condição seja atingida. **As variações das formas de estruturas de repetições dependem da implementação de cada linguagem de programação**.

A figura 26 mostra três estruturas de controle de repetição genéricas (nenhuma delas está presente nos blocos Scratch), a primeira com teste no início (antes do conjunto de instruções ser executado), a segunda com teste ao final (após uma execução do conjunto de instruções) e a terceira com variável de controle:

- A estrutura **ENQUANTO** <*exp*> FAÇA <*comandos*> FIMENQ testa <*exp*> e encerra se o resultado for $falso$. Caso <*exp*> seja $verdadeiro$, <*comandos*> serão executados e um novo teste será realizado.

- A estrutura **REPITA** <*comandos*> ATÉ <*exp*> executa <*comandos*> e realiza o teste de <*exp*>, encerrando caso o resultado seja $verdadeiro$. Caso o resultado seja $falso$, uma nova iteração é realizada processando <*comandos*> e realizando novo teste de <*exp*>.
- A estrutura **PARA** (<*exp1*>; <*exp2*>; <*exp3*>) FAÇA <*comandos*> FIMPARA executa <*exp1*>, realiza o teste de <*exp2*> e encerra se for $falso$. Caso seja $verdadeiro$, processa comandos e <*exp3*> para realizar um novo teste de <exp2>.

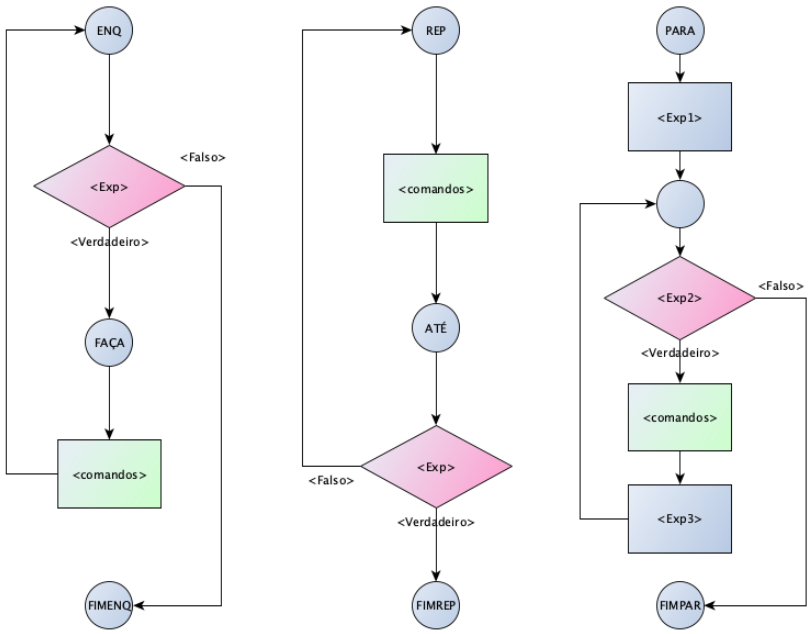

Figura 26 – Fluxos das Estruturas de Controle Controle de Repetição (Elaborado pelo autor)

Algoritmo 5: Estrutura Enquanto - exibição de 1 a 10

Var i: inteiro
Inicio
  $i \leftarrow 0$
  Enquanto $i < 10$
    $i \leftarrow i + 1$
    Saída: i
  Fim Enquanto
Fim

Ambos algoritmos 5 e 6 utilizam a estrutura de repetição **ENQUANTO** e exibem, respectivamente, os valores de 1 a 10 e de 10 a 1. Especial atenção deve ser dada à manipulação da variável que controla a estrutura de repetição (valor inicial que a variável terá, local em que ela será utilizada, variação que sofrerá e qual será a condição de parada da repetição). Estru-

turas mal elaboradas podem causar os chamados "*LOOPS*[1] INFINITOS" ou seja, condições de parada inatingíveis e, consequentemente, o programa só encerrará de modo forçado.

Algoritmo 6: Estrutura Enquanto - exibição de 10 a 1

Var i: inteiro
Inicio
  $i \leftarrow 10$
  Enquanto $i > 0$
    Saída: i
    $i \leftarrow i - 1$
  Fim Enquanto
Fim

O algoritmo 7 utiliza uma estrutura de controle de repetição **REPITA ATÉ** (com teste ao final) para ler valores e classificá-los como par ou ímpar. Um valor zero encerra a repetição.

Algoritmo 7: Estrutura Repita até - Par ou ímpar em loop

Var x: inteiro
Inicio
  repita
    Saída: "Informe um valor para saber se é par ou ímpar (0 para sair)"
    Entrada: x
    `// Testa se 2 | x (2 divide x)`
    se $(x \ mod \ 2 \ = \ 0)$ **então**
      Saída: x," é PAR"
    senão
      Saída: x," é ÍMPAR"
    fim se
  até $x = 0$
Fim

O algoritmo 8 utiliza uma estrutura de controle de repetição **PARA** (com variável de controle) para exibir os números pares de 2 a 20.

[1] Laços de repetições

Algoritmo 8: Estrutura Para - exibição dos pares de 2 a 20

Var i: inteiro
Inicio
`// Variável i terá valores 2, 4, 6, ..., 20`
para $(i \leftarrow 2;\ i \leq 20;\ i \leftarrow i + 2)$ **faça**
Saída: i,“, ”
fim para
Fim

SCRATCH - Estruturas de Controle de Repetição

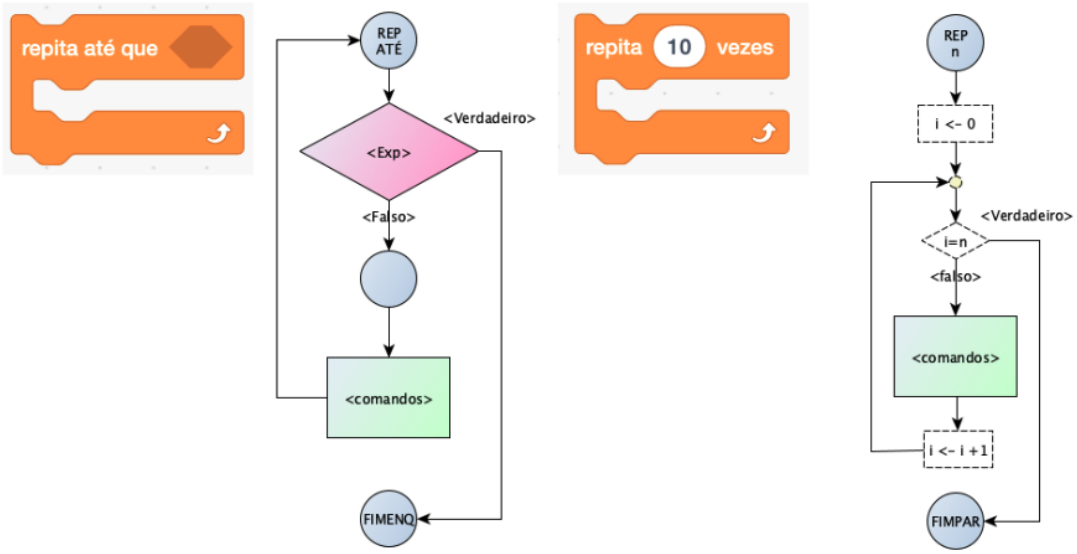

Figura 27 – Fluxos das Estruturas de Repetição do Scratch (Elaborado pelo autor)

A figura 27 mostra duas das estruturas de controle de repetição do Scratch que serão utilizadas neste trabalho: REPITA ATÉ QUE[1] <exp> e REPITA <n> VEZES. A primeira é semelhante a estrutura ENQUANTO da figura 26, mas com a inversão[2] da condição de finalização. Já a estrutura REPITA n VEZES é uma forma abreviada da estrutura PARA utilizada para repetir a execução de <$comandos$> por <n> vezes. A contagem é realizada de forma implícita. O uso desse bloco foi demonstrado e justificado no capítulo 2 na abordagem de construções geométricas.

[1] não confundi-la com o REPITA ATÉ que realiza teste após iterações
[2] um algoritmo implementado com ENQUANTO <exp> será convertido em REPITA ATÉ QUE **nao**(<exp>)

Algoritmo 9: Estrutura Repita até que - exibição de 1 a 10

Var i: inteiro
Inicio
  $i \leftarrow 0$
  repita até que $i = 10$
    $i \leftarrow i + 1$
    Saída: i
  fim repita
Fim

Algoritmo 10: Estrutura Repita n vezes - exibição de 1 a 10

Var i: inteiro
Inicio
  $i \leftarrow 0$
  repita 10 **vezes**
    $i \leftarrow i + 1$
    Saída: i
  fim repita
Fim

3.5 Construindo Abstrações

Na abordagem de construções geometricas do capítulo 2 foram vistas construções de blocos para desenhar triângulo equilátero, quadrado e qualquer polígono regular, ou seja, construções de novas abstrações para uso na programação. No paradigma de programação funcional ou procedural as linguagens de programação permitem construções de funções e procedimentos. Uma **função** é um módulo que processa algo e devolve um valor de retorno. Já o **procedimento** não há valor de retorno[1] à chamada. Os módulos (procedimento ou funções) devem se comunicam com o programa ou outros módulos por meio de parâmentros que podem ser passados por cópia (valor) ou referência (endereço) e podem ter variáveis declaradas em seu escopo (variáveis locais). Algumas vantagens da programação modular: diminuição de complexidade dos programas, facilitando desenvolvimento e manutenção; Facilidade de testes e isolamento de erros; Reuso de código; Dentre outras.

[1] pode haver retorno de dados de um procedimento por meio de parâmetros passados por referência ou endereço

Algoritmo 11: Procedimento exibe números pares até n

```
Procedimento ExibeParesAte(n : inteiro)
Var i: inteiro
Inicio
    para (i ← 2; i ≤ n; i ← i + 2) faça
        Saída: i,“, ”
    fim para
Fim
Inicio
    ExibeParesAte(10)
Fim
```

O algoritmo 11 mostra a declaração do Procedimento $ExibeParesAte$ que recebe um parâmetro inteiro n e exibe todos os pares de 2 até o par menor ou igual a n.

Algoritmo 12: Função calcula dobro

```
Var i : real
    Função Dobro(n : real) : real
    Inicio
        retorne 2 ∗ n
    Fim
Inicio
    Saída: “Entre com valor:”
    Entrada: i
    Saída: “O dobro de ”, i, “ é ”, Dobro(i)
Fim
```

Conforme já abordado, Scratch só permite construções de blocos (procedimentos). Nos apêndices de Portugol e Python podem ser vistas tanto construções de funções como de procedimentos.

4 LÓGICA E ÁLGEBRA DE BOOLE

Este capítulo trata da lógica utilizada na programação, notadamente da Álgebra de Boole que trabalha num contexto binário (falso ou verdadeiro, 0 ou 1). São abordados com exemplos e Diagramas de Venn, os operadores lógicos ou booleanos da disjunção inclusiva ou união - "ou", da conjunção ou interseção - "e", da negação ou complemento - "não". Ao final, são apresentadas as Leis de De Morgan.

A Lógica Computacional teve origens com o matemático Leonhard Euler (1707-1783) com a introdução da representação gráfica das relações entre sentenças e, com base nesse trabalho, John Venn (1834-1923) estabeleceu o que hoje é conhecido como Diagrama de Venn . (COELHO, 2014)

> "George Boole (1815-1864) mostrou que a Álgebra poderia libertar-se dos números e trabalhar também com outros tipos de entes, por exemplo os **conjuntos** e as **proposições da Lógica**. Esta descoberta de Boole, juntamente com os trabalhos de outros matemáticos como William Rowan Hamilton (1805-1865), Hermann Günther Grassmann (1809-1877) e Arthur Cayley (1821-1895), produziu uma revolução conceitual conhecida como "**a liberação da Álgebra**". (GARBI, 2010, p 380)

Para Boyer (2012), a história da lógica está dividida em três estágios: lógica grega, lógica escolástica e lógica matemática. Nos dois primeiros estágios os teoremas lógicos eram expressos em linguagem natural. No terceiro estágio o sistema é puramente formal e o início desse estágio foi marcado pelas obras de Boole: *The Mathematical Analysis of Logic* (1847) e *A Investigation of the Laws of Thought* (1854). Também, de acordo com Coelho (2014), a obra *Formal Logic* do matemático Augustus De Morgan (1806-1871) foi relevante para a lógica. Nela constam as Leis de De Morgan que serão abordadas neste capítulo.

Boole propôs substituir os números da Álgebra por conjuntos e trabalhar com duas formas de associação. Tomados dois conjuntos quaisquer A e B, são definidas as operações de soma $(+)$ ou união, tratada por $A + B$ ou $A \cup B$, e de produto $(\cdot)$ ou interseção, tratada por $A \cdot B$ ou $A \cap B$. Também são definidos o conjunto vazio, representado pelo 0 ou $\emptyset$, e o conjunto universo (representado pelo 1). Assim, o complemento de A, representado por Á ou $\bar{A}$, são os entes de I que não pertencem a A. A figura 28 mostra os diagramas de Venn para as operações propostas por Boole. (GARBI, 2010)

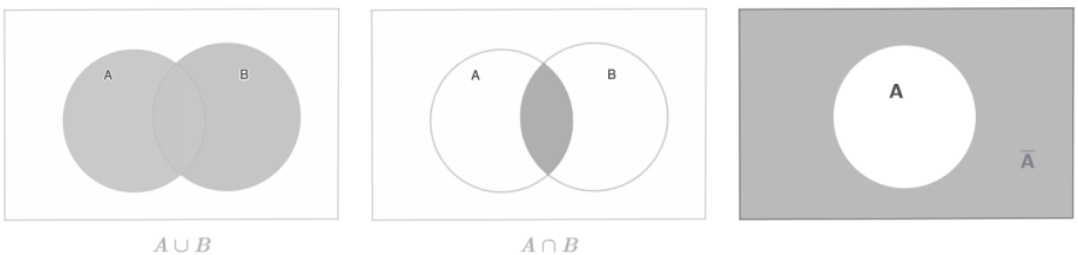

Figura 28 – Diagramas Venn: Disjunção, Conjunção e Complemento (Elaborado pelo autor)

Segundo Garbi (2010), as obras de Boole propuseram que a lógica poderia ser estudada por manipulação simbólica como álgebra. As deduções de Aristóteles (silogismos) que eram puramente verbais ou retóricas, expressas por meio de palavrras pderiam ter construções algébricas e manipulação simbólica. Essa ideia deu origem à **Lógica Simbólica ou Lógica Matemática**. A figura 29 mostra um "dicionário" de conversão.

Ou A ou B	$A + B$
Ambos, A e B	AB
NÃO A	Á
Nem A nem B	$(A + B)'$
Alguns A são B	$AB \neq 0$
Alguns A não são B	$AB' \neq 0$
Todo A é B (ou A implica B)	$A \subset B$

Figura 29 – Lógica em termos de álgebra dos conjuntos (GARBI, 2010, p.385)

> Assim, o famoso silogismo: "todos os homens (H) são mortais (M); Sócrates (S) é um homem; Portanto Sócrates é mortal" pode ser convertido em $H \subset M$, portanto $H + M = M$; $S \subset H$, portanto $S + H = H$; (substituindo $S + H$ por H na primeira relação) $S + H + M = M$; e, como $H + M = M$, $S + M = M$ o que significa que $S \subset M$, ou seja, Sócrates pertence ao conjunto M (dos mortais). (GARBI, 2010, p. 385)

A lógica de programação utiliza parte dessa lógica desenvolvida por Boole. A seguir serão abordadas construções com os chamados operadores booleanos: **e (and), ou (or) e não (not)**.

4.1 Disjunção Inclusiva - operador ou

O operador da disjunção inclusiva na álgebra de Boole é equivalente à operação de união de conjuntos. Para que a união de dois conjuntos seja um conjunto vazio ambos conjuntos operandos devem ser vazios.

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A ou B
Falso	Falso	Falso
Falso	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro

Tabela 2 – Tabela da Verdade (OU) - Disjunção inclusiva

Seja o critério para aprovação em uma escola a nota do aluno maior ou igual a 7. A expressão algébrica que representa esse critério: $(Nota > 7)\ ou\ (Nota = 7)$.

A tabela 3 mostra a Tabela da Verdade para os valores de notas 3, 7 e 9.

Nota	$(Nota > 7)$	$(Nota = 7)$	$(Nota > 7)\ ou\ (Nota = 7)$
3	Falso	Falso	Falso
7	Falso	Verdadeiro	Verdadeiro
9	Verdadeiro	Falso	Verdadeiro

Tabela 3 – Tabela da Verdade (OU) - Exemplo Aprovação

O Diagrama de Venn para as notas 3, 7, 9 e os conjuntos $A = \{x \in \mathbb{R}, x = 7\}$ e $B = \{x \in \mathbb{R}, x > 7\}$[1] está na figura 30. Como a operação **ou** equivale a união dos conjuntos, tanto a nota 7 quanto a nota 9 resultam no critério de aprovação $verdadeiro$, diferentemente da nota 3 que resulta em $falso$.

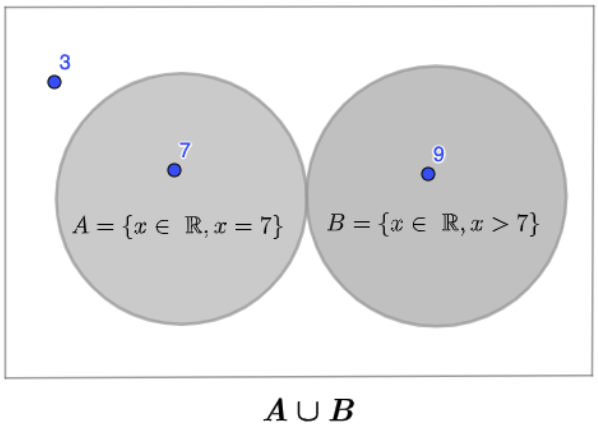

Figura 30 – Diagrama de Venn - Exemplo operador OU: Aprovação (Elaborado pelo autor)

Portanto, as análises da operação booleana **ou** referente à disjunção dos conjuntos $A = \{x \in \mathbb{R}, x = 7\}$, $B = \{x \in \mathbb{R}, x > 7\}$ e $A \cup B$ para os valores 3, 7 e 9 seguem:

- $Nota = 3$: $(3 = 7)\ ou\ (3 > 7) \implies falso\ ou\ falso =$ **falso**, equivale a $3 \notin A$, $3 \notin B$ e $3 \notin A \cup B$;
- $Nota = 7$: $(7 = 7)\ ou\ (7 > 7) \implies verdadeiro\ ou\ falso =$ **verdadeiro**, equivale a $7 \in A$, $7 \notin B$ e $7 \in A \cup B$;
- $Nota = 9$: $(9 = 7)\ ou\ (9 > 7) \implies falso\ ou\ verdadeiro =$ **verdadeiro**, equivale a $9 \notin A$, $9 \in B$ e $9 \in A \cup B$.

4.2 Conjunção - operador e

O operador da conjunção na álgebra de Boole é equivalente à operação de interseção de conjuntos. Para que a intersecção de dois conjuntos não resulte em um conjunto vazio ambos conjuntos operandos devem ser não vazios.

[1] Conjuntos A e B desse exemplo tem interseção vazia

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A e B
Falso	Falso	Falso
Falso	Verdadeiro	Falso
Verdadeiro	Falso	Falso
Verdadeiro	Verdadeiro	Verdadeiro

Tabela 4 – Tabela da Verdade (E) - Conjunção

Seja o critério para estabelecer conceito B a um aluno de uma escola que a nota seja maior que 7 e menor que 9. A expressão algébrica que representa esse critério é: $(Nota > 7)\ e\ (Nota < 9)$.

A tabela 5 mostra a Tabela da Verdade para os valores de notas 5, 8 e 9.

Nota	$(Nota > 7)$	$(Nota < 9)$	$(Nota > 7)\ e\ (Nota < 9)$
5	Falso	Verdadeiro	Falso
8	Verdadeiro	Verdadeiro	Verdadeiro
9	Verdadeiro	Falso	Falso

Tabela 5 – Tabela da Verdade (E) - Exemplo Conceito B

O Diagrama de Venn para as notas 5, 8, 9 e os conjuntos $A = \{x \in \mathbb{R}, x > 7\}$ e $B = \{x \in \mathbb{R}, x < 9\}$ está na figura 31. Tanto a nota 5 quanto a nota 9 resultam $falso$ no critério de pertinência à interseção e, consequentemente, $falso$ para o enquadramento como conceito B. Já a avaliação da expressão para a nota 8 resulta em $verdadeiro$.

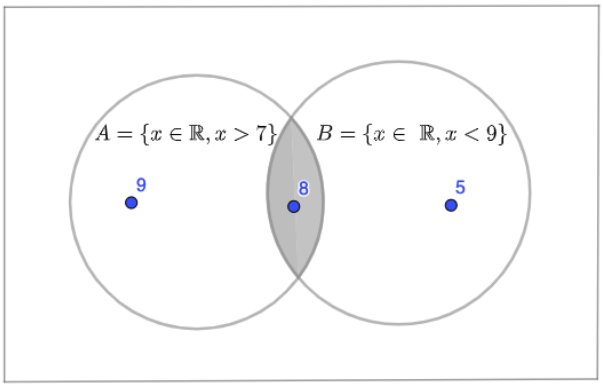

Figura 31 – Diagrama de Venn - Exemplo operador E: Conceito B (Elaborado pelo autor)

Portanto, as análises da operação booleana **e** referente à conjunção dos conjuntos $A = \{x \in \mathbb{R}, x > 7\}$, $B = \{x \in \mathbb{R}, x < 9\}$ e $A \cap B$ para os valores 5, 8 e 9 seguem:

- $Nota = 5$: $(5 > 7)\ e\ (5 < 9) \implies falso\ e\ verdadeiro =$ **falso**, equivale a $5 \notin A$, $5 \in B$ e $5 \notin A \cap B$;
- $Nota = 8$: $(8 > 7)\ e\ (8 < 9) \implies verdadeiro\ e\ verdadeiro =$ **verdadeiro**, equivale a $8 \in A$, $8 \in B$ e $8 \in A \cap B$;

- $Nota = 9$: $(9 > 7)\ e\ (9 < 9) \implies verdadeiro\ e\ falso =$ **falso**, equivale a $9 \in A$, $9 \notin B$ e $9 \notin A \cap B$;.

4.3 Negação ou Complemento - operador não

O operador da negação na álgebra de Boole é equivalente ao complemento ou conjunto complementar. Como já visto anteriormente, o conjunto vazio $\emptyset$ (representado pelo 0) e o conjunto universo (representado pelo 1) são complementares um do outro.

A	$\overline{A}$
0	1
1	0

A	NAO(A)
Falso	Verdadeiro
Verdadeiro	Falso

Tabela 6 – Tabela da Verdade (NAO) - Negação

Seja o critério anteriormente exemplificado que para aprovação em uma escola a nota do aluno maior ou igual a 7, ou seja, se a nota for inferior a 7 o aluno estará reprovado. As expressões algébricas que representam a reprovação e aprovação são, respectivamete: A=$(Nota < 7)$ e $\overline{A} = NAO(Npta < 7)$. A tabela 7 mostra a Tabela da Verdade para os valores de notas 5 e 8.

Nota	$(Nota < 7)$	NAO$(Nota < 7)$
5	Verdadeiro	Falso
8	Falso	Verdadeiro

Tabela 7 – Tabela da Verdade (NAO) - Exemplo $Nota < 7$

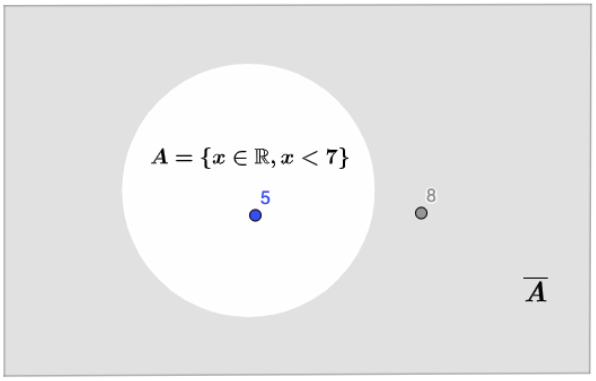

Figura 32 – Diagrama de Venn - Exemplo Negação B (Elaborado pelo autor)

O Diagrama de Venn para as notas 5, 8 e os conjuntos $A = \{x \in \mathbb{R}, x < 7\}$ e $\overline{A} = \{x \in \mathbb{R}, x \notin A\}$ está na figura está na figura 32. A nota 5 está no conjunto A e a nota 8 está em $\overline{A}$ (complementar de A). Ou seja, a nota 5 quando aplicada à expressão $NAO(Nota < 7)$ resulta em $falso$, diferentemente da nota 8 que resulta em $verdadeiro$.

Portanto, as análises da operação booleana **nao** aplicada ao conjunto $A = \{x \in \mathbb{R}, x < 7\}$ que resulta em $\overline{A}$ para os valores 5 e 8 seguem:

- $Nota = 5$: $NAO(5 < 7) \implies NAO(verdadeiro) =$ **falso**, equivale a $5 \in A$ e $5 \notin \overline{A}$;
- $Nota = 8$: $NAO(8 < 7) \implies NAO(falso) =$ **verdadeiro**, equivale a $8 \notin A$ e $8 \in \overline{A}$.

4.4 As Leis de De Morgan

As Leis de De Morgan são formuladas pelas duas equações que seguem:

$$(A \cap B)^{'} \leftrightarrow A^{'} \cup B^{'} \tag{4.1}$$

O complementar da interseção de dois conjuntos é equivalente à união dos complementares de cada conjunto: $(A \cap B)^{'} \leftrightarrow A^{'} \cup B^{'}$[1]. Em algebra Booleana seria: NAO(A E B) $\leftrightarrow$ NAO(A) OU NAO(B). A figura 33 exibe o Diagrama de Venn para a expressão: $NAO((x > 7)\ E\ (x < 9))$ que equivale a $(x \leq 7) OU (x \geq 9)$.

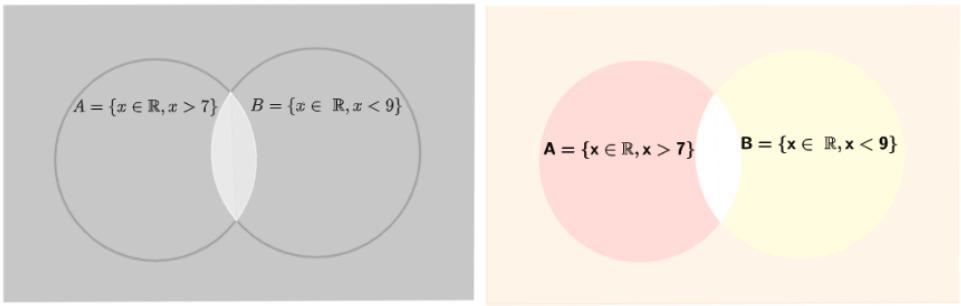

Figura 33 – Diagrama De Morgan 01 (Elaborado pelo autor)

$$(A \cup B)^{'} \leftrightarrow A^{'} \cap B^{'} \tag{4.2}$$

O complementar da união de dois conjuntos é equivalente à interseção dos complementares de cada conjunto: $(A \cup B)^{'} \leftrightarrow A^{'} \cap B^{'}$. Em algebra Booleana seria: NAO(A OU B) $\leftrightarrow$ NAO(A) E NAO(B). A figura 34 exibe o Diagrama de Venn para a expressão: $NAO((x < 3)\ OU\ (x > 7))$ que equivale a $(x \geq 3)\ E\ (x \leq 7)$.

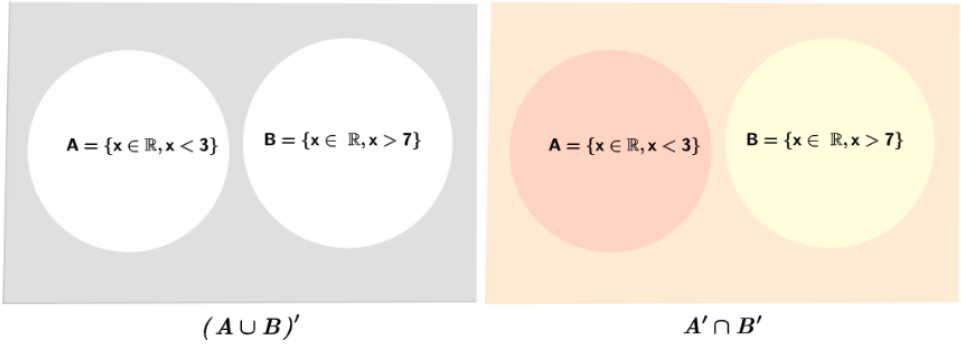

Figura 34 – Diagrama De Morgan 02 (Elaborado pelo autor)

[1] A' em amarelo e B' em vermelho. Em laranja a área complementar comum a ambos.

5 SEQUÊNCIAS E SOLUÇÕES COMPUTACIONAIS ITERATIVAS

Este capítulo tem como foco o desenvolvimento de soluções computacionais iterativas. Soluções com iterações [1] utilizam estruturas de repetições. A abordagem é focada na matemática que envolve o triângulo de Pascal. Primeiramente, o triângulo de Pascal é definido e contextualizado. Em seguida, as sequências dos números naturais, da soma dos n primeiros números naturais e de Fibonacci são caracterizadas, contextualizadas na história e abordadas em projetos.

5.1 Triângulo de Pascal

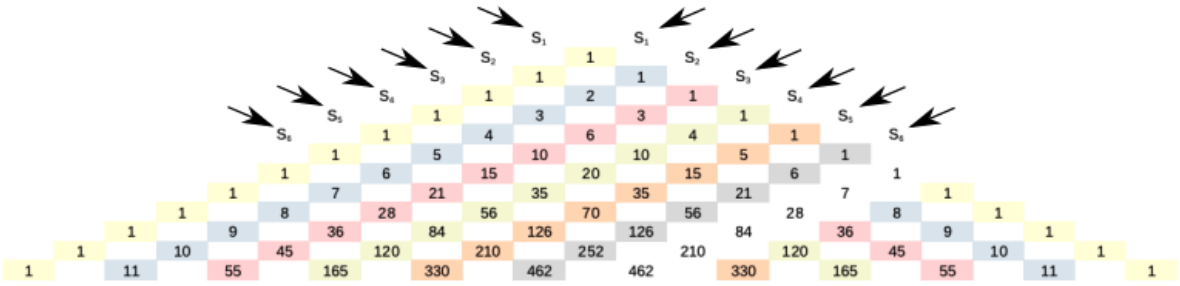

Figura 35 – Triângulo de Pascal e algumas sequências em destaque (Elaborado pelo autor)

Blaise Pascal (1623-1662) foi contemporâneo de René Descartes (1596-1650), Pierre de Fermat (1601-1665) e Marin Mersenne (1588-1648). Aos quatorze anos, Pascal passou a acompanhar seu pai nas reuniões informais da Academia de ritmo em Paris. Dentre os feitos de Pascal está o projeto de uma máquina de calcular mecânica para ajudar o pai nos cálculos. Na Figura 35 temos o Triângulo de Pascal e algumas sequências destacadas: $S_1 = \{1, 1, 1, 1, 1, 1, ...\}$, $S_2 = \{1, 2, 3, 4, 5, 6, ...\}$, $S_3 = \{1, 3, 6, 10, 15, 21, ...\}$, etc. Embora o triângulo receba o nome de Pascal, há registros desse triângulo nas obras dos chineses Yang Hui (viveu por volta de 1261-1275) e Zhu Shijie (viveu em 1280-1303) (BOYER, 2012)

Para melhor caracterizar a construção e algumas propriedades do Triângulo de Pascal, utilizaremos a representação na forma de uma matriz triangular inferior (Figura 36) com a coluna 0 e a diagonal principal preenchidas pelo valor 1. Um termo que não esteja na coluna 0, nem na diagonal principal, será obtido a partir da adição do elemento imediatamente acima com antecessor dele.

[1] Iterações e interações são coisas distintas. Iterações são repetições e interações são diálogos (verbo interagir)

$Pascal_{(i,j)}$ pode ser definida:

$$Pascal_{(i,j)} = \begin{cases} 1, se\ j = 0\ ou\ i = j \\ Pascal_{(i-1,j)} + Pascal_{(i-1,j-1)},\ se\ (j \neq 0)\ e\ (i \neq j) \end{cases} \tag{5.1}$$

i \ j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	∑Linha(i)	
0	1																		1 =	2^0
1	1	1																	2 =	2^1
2	1	2	1																4 =	2^2
3	1	3	3	1															8 =	2^3
4	1	4	6	4	1														16 =	2^4
5	1	5	10	10	5	1													32 =	2^5
6	1	6	15	20	15	6	1												64 =	2^6
7	1	7	21	35	35	21	7	1											128 =	2^7
8	1	8	28	56	70	56	28	8	1										256 =	2^8
9	1	9	36	84	126	126	84	36	9	1									512 =	2^9
10	1	10	45	120	210	252	210	120	45	10	1								1024 =	2^{10}
11	1	11	55	165	330	462	462	330	165	55	11	1							2048 =	2^{11}
12	1	12	66	220	495	792	924	792	495	220	66	12	1						4096 =	2^{12}
13	1	13	78	286	715	1287	1716	1716	1287	715	286	78	13	1					8192 =	2^{13}
14	1	14	91	364	1001	2002	3003	3432	3003	2002	1001	364	91	14	1				16384 =	2^{14}
15	1	15	105	455	1365	3003	5005	6435	6435	5005	3003	1365	455	105	15	1			32768 =	2^{15}
16	1	16	120	560	1820	4368	8008	11440	12870	11440	8008	4368	1820	560	120	16	1		65536 =	2^{16}
17	1	17	136	680	2380	6188	12376	19448	24310	24310	19448	12376	6188	2380	680	136	17	1	131072 =	2^{17}

Figura 36 – Triângulo de Pascal na Forma Matricial(Elaborado pelo autor)

Triângulo de Pascal: Representação por Conjuntos

Outra construção do triângulo de Pascal seria sob a ótica de conjuntos, conforme Figura 37. Os elementos do Triângulo de Pascal são conjuntos de conjuntos. O conjunto unitário composto pelo elemento conjunto vazio estará na coluna 0 de cada linha i e o conjunto unitário composto pelo conjunto Universo com i elementos (U_i) estará na diagonal principal. O elemento de uma posição (linha i, coluna j) que não esteja na coluna 0 (conjunto vazio), nem na diagonal (conjunto Universo), será o conjunto composto pelos subconjuntos de j elementos do conjunto Universo U_i. Sendo e_i um novo elemento acrescido a cada nova linha i, a construção do Triângulo de Pascal em Conjuntos será:

$$PascalSet_{(i,j)} \quad = \quad \begin{cases} \{\emptyset\}, se\ j = 0 \\ \{x \cup \{e_i\}, x \in PascalSet_{(i-1,j-1)}\},\ se\ j = i\ e\ j \neq 0 \\ PascalSet_{(i-1,j)} \cup \\ \quad \{x \cup \{e_i\}, \forall x \in PascalSet_{(i-1,j-1)}\},\ se\ 0 < j \leq i \end{cases} \tag{5.2}$$

i \ j	0	1	2	3	4	5
0	{Ø}					
1	{Ø}	{{a}}				
2	{Ø}	{{a},{b}}	{{a,b}}			
3	{Ø}	{{a},{b},{c}}	{{a,b},{a,c},{b,c}}	{{a,b,c}}		
4	{Ø}	{{a},{b},{c},{d}}	{{a,b},{a,c},{b,c},{a,d},{b,d},{c,d}}	{{a,b,c},{a,b,d},{a,c,d},{b,c,d}}	{{a,b,c,d}}	
5	{Ø}	{{a},{b},{c},{d},{e}}	{{a,b},{a,c},{b,c},{a,d},{b,d},{c,d}, {a,e},{b,e},{c,e},{d,e}}	{{a,b,c},{a,b,d},{a,c,d},{b,c,d}, {a,b,e},{a,c,e},{b,c,e},{a,d,e}, {b,d,e},{c,d,e}}	{{a,b,c,d},{a,b,c,e},{a,b,d,e}, {a,c,d,e},{b,c,d,e}}	{{a,b,c,d,e}}

Figura 37 – Triângulo de Pascal construído como relação de conjuntos (Elaborado pelo autor)

Dessa construção podemos chegar à fórmula:

$$Pascal_{(i,j)} = |PascalSet_{(i,j)}| = \begin{pmatrix} i \\ j \end{pmatrix} = \frac{i!}{j! \cdot (i-j)!} \tag{5.3}$$

Com a representação do Triângulo de Pascal em Conjuntos a dedução da fórmula da soma dos termos de uma linha do Triângulo de Pascal vista na última coluna da Figura 36 é direta - n elementos que podem estar ou não no conjunto - 2^n. O valor também está associado ao número de representações distintas que se pode ter com n dígitos binários - vide capítulo 8.

$$\sum_{j=0}^{\mathrm{i}} Pascal_{(i,j)} \;=\; \sum_{j=0}^{\mathrm{i}} |PascalSet_{(i,j)}| \;=\; 2^i$$

Relação de Stifel

Michael Stifel (1487-1567), quase um século antes do nascimento de Pascal, publicou o hoje chamado "Triângulo de Pascal" na página de rosto de uma de suas obras (BOYER, 2012). A relação de Stifel está representada na equação 5.4:

$$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} i-1 \\ j-1 \end{pmatrix} + \begin{pmatrix} i-1 \\ j \end{pmatrix} \tag{5.4}$$

Demonstração:

$$\begin{pmatrix} i-1 \\ j-1 \end{pmatrix} + \begin{pmatrix} i-1 \\ j \end{pmatrix} = \frac{(i-1)!}{(i-j)! \cdot (j-1)!} + \frac{(i-1)!}{(i-j-1)! \cdot j!} = \tag{5.5}$$

$$= \frac{(i-1)! \cdot j}{(i-j)! \cdot j!} + \frac{(i-1)! \cdot (i-j)}{(i-j)! \cdot j!} = \frac{i!}{(i-j)! \cdot j!} = \begin{pmatrix} i \\ j \end{pmatrix}$$

A própria definição recursiva de $PascalSet_{(i,j)}$ na equação 5.2 já seria uma demonstração para a Relação de Stifel. Observe na figura 37 que um elemento $PascalSet_{(i,j)}$ é definido a

partir dos elementos de $PascalSet_{(i-1,j)}$ escritos em verde sublinhados e dos elementos de $PascalSet_{(i-1,j-1)}$ escritos em vermelho.

Nas seções seguintes serão tratadas as sequências $S_2 = \{1, 2, 3, 4, 5, 6, ...\}$ e $S_3 = \{1, 3, 6, 10, 15, 21, ...\}$, identificadas na Figura 35. O Triângulo de Pascal ainda será retomado nas abordagens da sequência de Fibonacci e no Pequeno Teorema de Fermat.

5.2 Naturais, Axiomas de Peano e Indução Finita

Em uma evolução demorada a humanidade apoderou-se lentamente do modelo abstrato de contagem (um, dois, três, quatro, ...). As tribos mais rudimentares contam apenas um, dois e muitos. Graças a notável síntese feita pelo matemático italiano Giuseppe Peano, no limiar do século 20, hoje podemos descrever de forma concisa e precisamente o conjunto $\mathbb{N}$ dos números naturais. (LIMA, 2013)

1. Todo número natural tem um único sucessor;
2. Números naturais diferentes têm sucessores diferentes;
3. Existe um único número natural, chamado um e representado pelo símbolo 1, que não é sucessor de nenhum outro;
4. Seja X um conjunto de números naturais (isto é, $X \subset \mathbb{N}$). Se $1 \in X$ e se, além disso, o sucessor de todo elemento de X ainda pertence a X, então $X = \mathbb{N}$.

Giuseppe Peano (1858-1932) formulou os Axiomas de Peano pela primeira vez em 1889, na *Arithmetices principia nova methodo exposita*, na tentativa de reduzir a aritmética comum a puro simbolismo formal. O novo método postulacional atingiu novo nível de precisão, sem ambiguidade de sentido e sem hipóteses ocultas. Peano também contribuiu à lógica simbólica, tema bastante abordado em sua época. (BOYER, 2012)

> O último dos axiomas de Peano é conhecido como o axioma da indução e é a base de um método eficiente para demonstração de proposições referentes a números naturais (demonstrações por indução ou recorrência). Enunciado sob a forma de propriedades, se formula: Seja $P(n)$ uma propriedade relativa ao número natural n. Suponhamos que: (i) P(1) é válida; ii) Para todo $n \in \mathbb{N}$, a validez de $P(n)$ implica a validez de $P(n')$, onde n' é o sucessor de n. Então $P(n)$ é válida qualquer que seja o número natural n. (LIMA, 2013, p.25)

5.3 Soma dos Naturais e Gauss

Garbi (2010) narra a conhecida façanha de Carl Friedrich Gauss (1777-1855) quando menino. O professor de aritmética Büttner costumava passar enfadonhos exercícios envolvendo operações aritméticas. Um dia, esperando manter os alunos ocupados por bastante tempo, mandou que somassem de 1 a 100. Diferentemente dos colegas que mergulharam nos cálculos

mecanicamente, Gauss decidiu avaliar o problema e observou que as 100 parcelas poderiam ser agrupadas em cinquenta pares de valores iguais a 101 (1 e 100, 2 e 99, 3 e 98, ..., 50 e 51). Ao ver a resolução e perceber o potencial matemático de Gauss, o professor Büttner passou a lhe entregar livros mais avançados e colocou seu assistente de 17 anos, Johann Martin Chistian Bartels, para orientá-lo. Gauss recebeu de seus colegas apelidos de Titã, Colosso de Rodes e **Príncipe dos Matemáticos**.

Nesta seção será abordada a forma de se realizar um somatório em algoritmos iterativos, tendo por objetivo o cálculo de $\sum_{i=1}^{n} i$. O algoritmo 13 descreve uma forma de realizar esse processo. A lógica da programação desenvolvida consiste de: leitura de um valor para n; O valor da soma S_n iniciado com 0; um bloco de repetição que faz o valor de i assumir os valores de $1, 2, 3, \cdots, n-1, n$; Dentro do bloco de repetição, a cada iteração, o valor de i é acrescido ao valor da soma S_n; Finalizadas as iterações, o valor da soma S_n é exibido.

Algoritmo 13: Cálculo da soma Sn dos naturais utilizando iterações

Var i, n, S_n: Inteiro
Inicio
  Saída: "Quer somar os naturais até quanto?"
  Entrada: n
  $Sn \leftarrow 0$
  // Calcula $\sum_{i=1}^{n} i$
  para $(i \leftarrow 1;\ i \leq n;\ i \leftarrow i+1)$ **faça**
    $Sn \leftarrow Sn + i$
  fim para
  Saída: "A soma é:", S_n
Fim

Na figura 38 há um quadro que demonstra os valores de i e a soma S_n para cada iteração e, tarjada, a situação antes de processar as iterações. A variável S_n tem o valor zero antes do início das iterações e a estrutura de repetição faz a variável i assumir, a cada iteração, um dos valores de $1, 2, 3, \cdots, n$. A cada iteração o valor de S_n é acréscido de i.

i		1	2	3	4	5	6	7	8	9	10	...	n-1	n
Sn	0	1	3	6	10	15	21	28	36	45	55	...	S_{n-1}	S_{n-1}+n

Figura 38 – Iterações no cálculo da Soma S_n dos Naturais (Elaborado pelo autor)

A figura 39 mosta a implementação em Scratch. A adaptação necessária em relação ao algoritmo 13 foi a adequação da estrutura de repetição PARA para o bloco REPITA n VEZES.

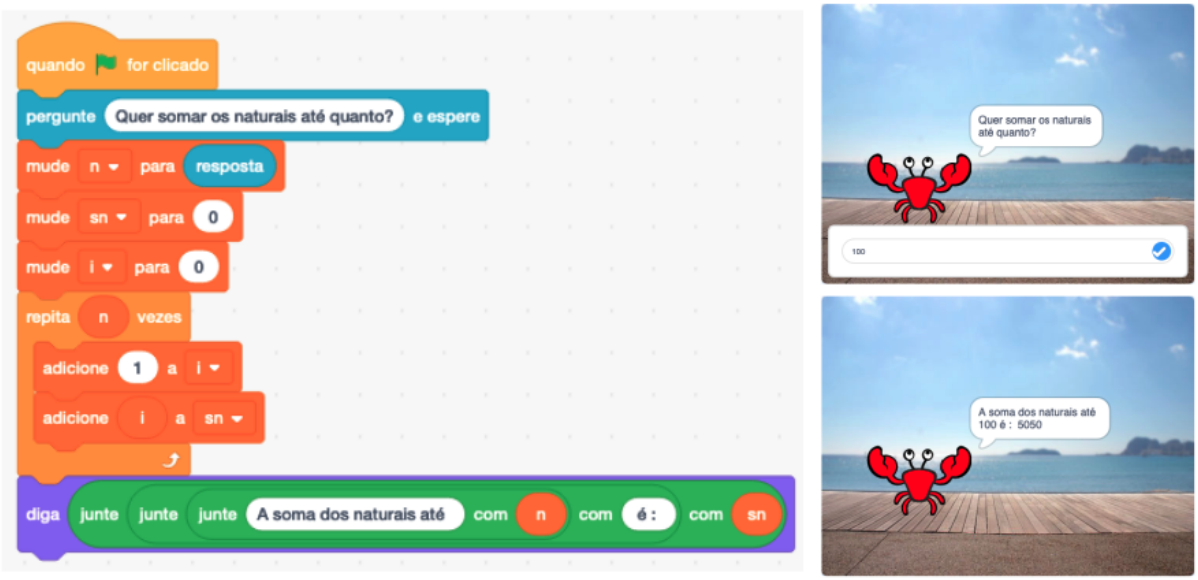

Figura 39 – Tela da Soma S_n dos Naturais em Scratch(Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/537991046/

No capítulo 6 será abordada uma solução para soma dos naturais até n na forma recursiva: $S(n) = n + S(n-1)$.

O algoritmo 13 realizou o cálculo $\sum_{i=1}^{n} i$ somando os n termos, um a um. Esse algortimo tem complexidade O(n) [1]. Ele poderia ser simplificado com a eliminação da estrutura de repetição e tendo cálculo realizado como o feito de Gauss narrado. A fórmula para o cálculo direto seria $S_n = \frac{n\cdot(n+1)}{2}$, conforme prova por Indução Finita que segue:

Prova por Indução $S_n = \frac{n\cdot(n+1)}{2}$

1. $S_1 = \frac{1\cdot(1+1)}{2} = 1$
2. $S_n = \frac{n\cdot(n+1)}{2}$ [HIPO]
3. $S_{n+1} = S_n + (n+1) \overset{HIPO_n}{=} \frac{n\cdot(n+1)}{2} + (n+1) = \frac{n\cdot(n+1)}{2} + \frac{(n+1)\cdot 2}{2} =$

$= \frac{(n+1)\cdot(n+2)}{2} = \frac{(n+1)\cdot((n+1)+1)}{2} \overset{HIPO_{n+1}}{=} S_{n+1}$

Um algoritmo implementado com a fórmula direta seria mais eficiente computacionalmente pois teria complexidade O(1) contra a complexidade O(n) da construção iterativa apresentada. Esse assunto é tratado na computação como análise de complexidade de algoritmos e de problemas.

[1] notação O grande (ou *Big O*, em inglês)

Soma dos Ímpares e Ternos Pitagóricos

Uma outra sequência que pode ser trabalhada com alunos é a sequência da soma dos números ímpares. Algumas propriedades dessa sequência podem ser discutidas e refletidas.

Ímpares (n)	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49
S	1	4	9	16	25	36	49	64	81	100	121	144	169	196	225	256	289	324	361	400	441	484	529	576	625
k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Figura 40 – Sequências dos Números Ímpares, Soma de Termos e Ternos Pitagóricos
(Elaborado pelo autor)

- A fórmula direta para cálculo da soma dos k primeiros números ímpares ou para a soma dos ímpares até o ímpar n é dada por: $S_n = \left(\frac{(n+1)}{2}\right)^2$ ou $S_k = k^2$, onde $n = 2k - 1$ e $k = \frac{n+1}{2}$

- Ternos Pitagóricos são três inteiros não nulos que satisfaçam a relação Pitagórica: $A^2 = B^2 + C^2$. Na Figura 40 podemos constatar destacados os quadrados dos ternos Pitagóricos primitivos[1]: { (3, 4, 5), (5, 12, 13), (7, 24, 25), $\cdots$ } ou seja, a cada ímpar n (ou $2k - 1$) que for um quadrado perfeito, teremos um terno Pitagórico primitivo $\left(\sqrt{n}, \frac{n-1}{2}, \frac{n+1}{2}\right)$ ou $\left(\sqrt{2k-1}, k-1, k\right)$

- O Último Teorema de Fermat é uma generalização dos ternos Pitagóricos e diz não existir inteiros não nulos que satisfaçam a relação: $A^n = B^n + C^n$ para $n > 2$. Uma obra interessante que trata do assunto sem rigor matemático é Singh (2019).

5.4 Sequência de Fibonacci

Leonardo de Pisa (cerca de 1180-1250), mais conhecido como Fibonacci ou "filho de Bonaccio", por volta de 1202 escreveu o livro *Liber abaci* (Livro do ábaco) que tratava de métodos e problemas algébricos com o uso de numerais indo-arábicos e tratando o zero como número. Do *Liber abaci*, o problema que inspirou matemáticos e que deu origem à conhecida **"sequência de Fibonacci"** foi : "Quantos pares de coelhos serão produzidos em um ano, começando com um só par, se em cada mês cada par gera um novo par que se torna produtivo a partir do segundo mês?" (BOYER, 2012, p. 182). Em notação matemática e com definição recorrente, a sequência $Fibo_n$ pode ser definida:

[1] Um terno pitagórico primitivo é um terno pitagórico em que os três números são primos entre si

Algoritmo 14: Cálculo do n−ésimo termo da sequência de Fibonacci

Var $FiboN, FiboN1, FiboN2, n$: Inteiro
Inicio
Saída: "Quer saber qual termo de Fibonacci?"
Entrada: n
$FiboN2 \leftarrow 1$
$FiboN1 \leftarrow 1$
// Realiza $n-2$ iterações para encontrar o n-ésimo termo
repita $n-2$ **vezes**
// Calcula o termo de Fibonacci da iteração
$FiboN \leftarrow FiboN1 + FiboN2$
// Atualiza valores de FiboN2 e FiboN1 para próxima iteração
$FiboN2 \leftarrow FiboN1$
$FiboN1 \leftarrow FiboN$
fim repita
Saída: "Fibo(",n,")=", $FiboN$
Fim

$$Fibo_n = \begin{cases} Fibo_1 = Fibo_2 = 1 \\ Fibo_n = Fibo_{n-1} + Fibo_{n-2}, \ \forall\, n > 2 \end{cases} \tag{5.6}$$

ou seja, $1, 1, 2, 3, 5, 8, 13, 21, \cdots, u_n, \cdots$ (onde $u_n = u_{n-1} + u_{n-2}$).

A sequência de Fibonacci terá três abordagens neste trabalho. A primeira, nesta seção, com uma solução para o cálculo de um termo da sequência utilizando processo iterativo utilizando uma técnica de programação dinâmica[1]. As outras duas abordagens estão no capítulo 6 que trata de recorrências. Uma é a abordagem da fórmula de Binet para cálculo direto de um termo da sequência junto à demonstração da obtenção. A outra é uma implementação computacional para cálculo de um termo utilizando recursividade junto à discussão da ineficiência do método.

O algoritmo 14 descreve uma forma de calcular o n-ésimo termo da sequência de Fibonacci de forma iterativa. A lógica da programação desenvolvida consiste de: leitura de qual termo n se deseja calcular o valor; As variáveis $FiboN2$ e $FiboN1$ irão guardar os dois termos antecessores a cada cálculo de $FiboN$, iniciando as duas com valor unitário; Uma estrutura de repetição para realizar $n-2$ iterações processará o cálculo do termo $FiboN$ relativo àquela iteração

[1]Programação Dinâmica é um método de resolução de problemas computacionais que busca uma solução a partir de soluções previamente calculadas e armazenadas em memória para evitar recálculo

e atualizará os valores de $FiboN2$ e $FiboN1$ para a próxima rodada. A figura 41 mostra os valores das variáveis no início e fim de cada iteração. Ao final de todas as $n-2$ iterações o valor de $FiboN$ é exibido.

Início	$Fibo_N$		2	3	5	8	13	21	34	...	$Fibo_{N-1}$
	$Fibo_{N-1}$	1	2	3	5	8	13	21	34	...	$Fibo_{N-1}$
	$Fibo_{N-2}$	1	1	2	3	5	8	13	21	...	$Fibo_{N-2}$
	termo	3	4	5	6	7	8	9	10	...	n
Fim	$Fibo_N$	2	3	5	8	13	21	34	55	...	$Fibo_{N-1}$+$Fibo_{N-2}$
	$Fibo_{N-1}$	2	3	5	8	13	21	34	55		$Fibo_N$
	$Fibo_{N-2}$	1	2	3	5	8	13	21	34	...	$Fibo_{N-1}$

Figura 41 – Iterações no cálculo do n-ésimo termo de Fibonacci (Elaborado pelo autor)

A figura 42 mosta a implementação do cálculo do n-ésimo termo da sequência de Fibonacci em Scratch. A única adaptação realizada em relação ao algoritmo 14 foi a inclusão de uma estrutura de controle condicional (se/então/senão) para tratar os casos em que o valor de n seja 1 ou 2. Esse detalhe não foi previsto no algortimo por questões de simplificação da abordagem, mas houve ressalva que o cálculo seria para $n > 2$.

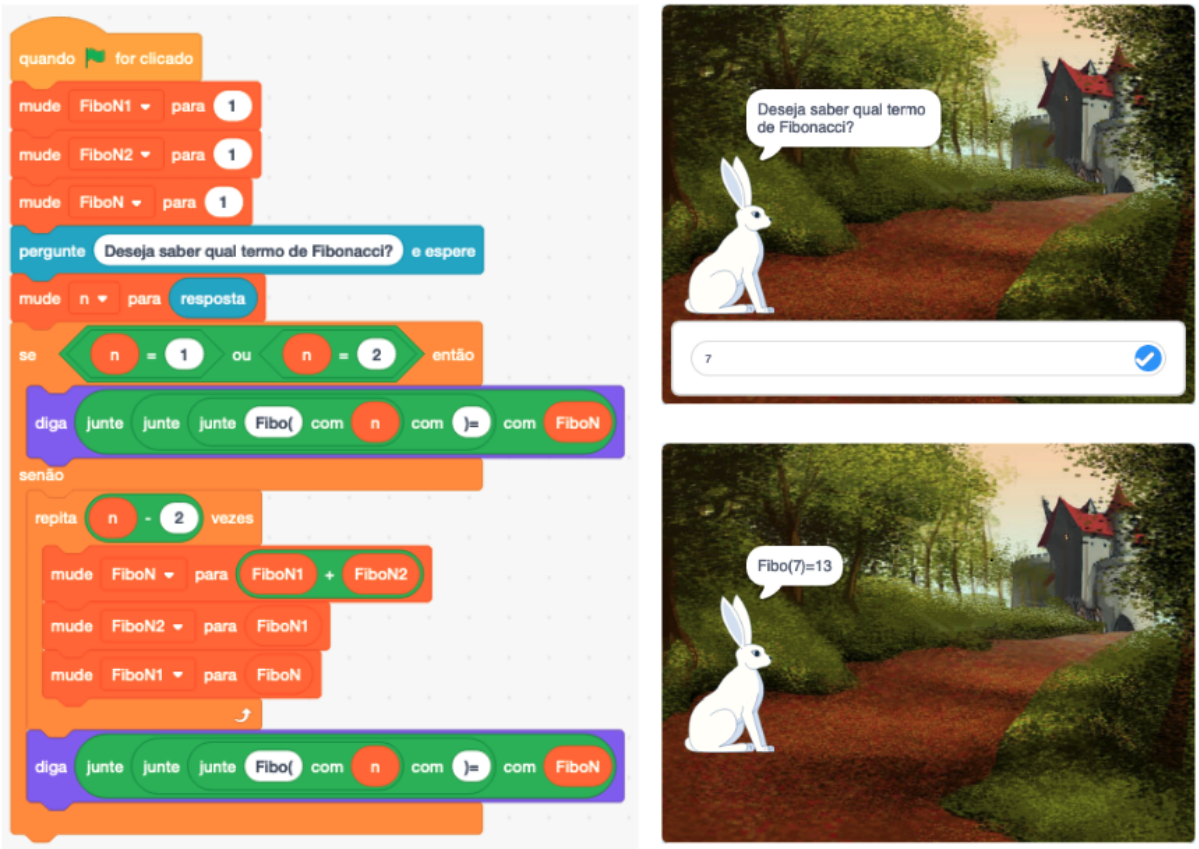

Figura 42 – Cálculo do n-ésimo termo de Fibonacci em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/548421851/

Curiosidades e conteúdos relacionados

- **Fórmula de Binet** - Para todo $n \in \mathbb{N}$, tem-se que:

$$u_n = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}} \tag{5.7}$$

  A fórmula 5.7 permite cálculo direto do n-ésimo termo da sequência de Fibonacci. A demonstração da dedução dessa fórmula está no capítulo 6.

- **Número Áureo** - O número áureo (razão de ouro ϕ [1]) pode ser aproximado pela divisão do n-ésimo termo da sequência de Fibonacci pelo termo antecessor. Quanto maior for o n, melhor será a aproximação.

$$\frac{2}{1} = 2; \frac{3}{2} = 1,5; \frac{8}{5} = 1,6; \frac{13}{8} = 1,625; \frac{6765}{4181} = 1,6180339...; \phi = \lim_{n\to\infty} \frac{u_{n+1}}{u_n} \tag{5.8}$$

A figura 43 mostra a sequência de Fibonacci no Triângulo de Pascal.

	0	1	2	3	4	5	6	7	8	9	10	11
		1	1	2	3	5	8	13	21	34	55	89
0	1											
1	1	1										
2	1	2	1									
3	1	3	3	1								
4	1	4	6	4	1							
5	1	5	10	10	5	1						
6	1	6	15	20	15	6	1					
7	1	7	21	35	35	21	7	1				
8	1	8	28	56	70	56	28	8	1			
9	1	9	36	84	126	126	84	36	9	1		
10	1	10	45	120	210	252	210	120	45	10	1	
11	1	11	55	165	330	462	462	330	165	55	11	1

Figura 43 – Sequência de Fibonacci no Triângulo de Pascal (Elaborado pelo autor)

[1] Dados $a > b > 0$, eles estarão em razão áurea se: $\frac{a+b}{a} = \frac{a}{b} = \phi$; $\phi = 2 \cdot \cos(\frac{\pi}{5})$.

6 RECORRÊNCIAS E SOLUÇÕES COMPUTACIONAIS RECURSIVAS

Este capítulo aborda recorrências tanto sob a ótica da matemática quanto da computação. No contexto da matemática são abordados os conceitos de recorrências lineares de primeira e segunda ordem. Os exemplos abordados para recorrências de primeira ordem são a soma dos n primeiros números naturais e fatorial de um número. O n-ésimo termo da sequência de Fibonacci e a fórmula de Binet são abordados na contextualização de recorrência de segunda ordem. No contexto da computação é apresentado o conceito de recursividade, uma forma alternativa de desenvolver o pensamento computacional, e são apresentados projetos em Scratch para calcular a a soma dos n primeiros números naturais, fatorial de um número e o n-ésimo termo da sequência de Fibonacci. Para o cálculo do n-ésimo termo de Fibonacci é discutida a ineficiência da solução recursiva e introduzido o conceito de complexidade computacional de um algoritmo. Por fim, o jogo Torre de Hanói e solução por recorrência é abordada.

6.1 Recorrências Lineares de Primeira

Uma **recorrência de primeira ordem** expressa x_{n+1} em função de x_n e é dita linear se, e somente se, a função for do primeiro grau. São exemplos de recorrências lineares de primeira ordem:

- **N fatorial (n!)** é uma recorrência matemática na forma $x_{n+1} = (n+1) \cdot x_n,\ x_1 = 1$. É, portanto, uma recorrência linear de primeira ordem homogênea. Desenvolvendo cada termo, aplicando o produto e simplificando, teremos:

$$\left.\begin{array}{lcl} x_2 = 2 & \cdot & x_1 \\ x_3 = 3 & \cdot & x_2 \\ x_4 = 4 & \cdot & x_3 \\ \cdots & \cdot & \cdots \\ x_n = n & \cdot & x_{n-1} \end{array}\right\} \times$$

$$x_n = 2 \cdot 3 \cdot\ 4 \cdots n \cdot x_1 \quad \therefore\ x_n = \prod_{i=1}^{n} i = n! \tag{6.1}$$

- **Soma dos naturais até n (S_n)** é uma recorrência linear de primeira ordem não homogênea, pois sua forma é $x_{n+1} = (n+1) + x_n,\ x_1 = 1$. Desenvolvendo cada termo, somando

as parcelas e simplificando, teremos:

$$\left.\begin{array}{rcl} x_2 = 2 & + & x_1 \\ x_3 = 3 & + & x_2 \\ x_4 = 4 & + & x_3 \\ \cdots & + & \cdots \\ x_n = n & + & x_{n-1} \end{array}\right\} + \qquad (6.2)$$

$$x_n = 2 + 3 + 4 + \cdots + n + x_1 \quad \therefore\ x_n = \sum_{i=1}^{n} i = S_n$$

6.2 Recorrências Lineares de Segunda Ordem

As recorrências da forma $x_{n+2} + p \cdot x_{n+1} + q \cdot x_n = 0$, com coeficientes p e q ($q \neq 0$) constantes, são ditas **recorrências lineares de segunda ordem homogêneas** e serão associadas a uma equação do segundo grau, $r^2 + p \cdot r + q = 0$ [1], chamada de equação característica (MORGADO, 2015).

TEOREMA Se as raízes de $r^2 + p \cdot r + q = 0$ são r_1 e r_2, então $a_n = C_1 \cdot r^n + C_2 \cdot r^n$ é solução da recorrência $x_{n+2} + p \cdot x_{n+1} + q \cdot x_n = 0$, quaisquer que sejam os valores das constantes C_1 e C_2.

A seguir a recorrência linear de segunda ordem que define um termo da sequência de Fibonacci será desenvolvida para se obter uma fórmula direta para cálculo do n-ésimo termo:

- **Fibonacci ($Fibo_n$):** $x_{n+2} = x_{n+1} + x_n$ com $x_1 = x_2 = 1$

  Equação característica $r^2 = r + 1$ que tem como raízes $r_1 = \frac{1+\sqrt{5}}{2}$ e $r_2 = \frac{1-\sqrt{5}}{2}$.
  Então, $Fibo_n = C_1 \cdot (\frac{1+\sqrt{5}}{2})^n + C_2 \cdot (\frac{1-\sqrt{5}}{2})^n$
  Para resolver, é mais conveniente utilizar $n = 0$ e $Fibo_0 = 0$.
  Para $n = 0$ e $Fibo_0 = 0$, teremos: $0 = C_1 + C_2$.
  Para $n = 1$ e $Fibo_1 = 1$, teremos: $1 = C_1 \cdot \frac{1+\sqrt{5}}{2} + C_2 \cdot \frac{1-\sqrt{5}}{2}$

  Resolvendo o sistema:

$$\begin{cases} C_1 + C_2 = 0 \\ C_1 \cdot \frac{1+\sqrt{5}}{2} + C_2 \cdot \frac{1-\sqrt{5}}{2} = 1 \end{cases} \qquad (6.3)$$

  Teremos:

$$C_1 = \frac{1}{\sqrt{5}}\ e\ C_2 = -\frac{1}{\sqrt{5}}$$

[1] A condição preliminar de $q \neq 0$ implica que 0 não seja raiz dessa equação. Pois, para o caso de q = 0, a recorrência seria de primeira ordem

Portanto: [Fórmula de Binet]

$$Fibo_n = \frac{1}{\sqrt{5}} \cdot (\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}} \cdot (\frac{1-\sqrt{5}}{2})^n \tag{6.4}$$

6.3 Soluções Computacionais Recursivas

Soluções computacionais recursivas são elaboradas apresentando:

- **Ponto de parada ou base da recorrência** - é a solução do problema para um caso singular em que não haja recorrência (chamada recursiva);
- **Recorrência ou Chamada Recursiva** - é a solução fazendo referência ao próprio problema em um caso mais simples até que o ponto de parada ou base da recorrência seja atingido.

A implementação de soluções computacionais recursivas irá depender da capacidade da linguagem de programação ter esse recurso implementado, ou seja, nem todas as linguagens de programação possibilitam uso de recursividade.

Para iniciar a compreensão do funcionamento da recursividade serão abordados os algoritmos 15 e 16 que divergem apenas na ordem entre a chamada recursiva e a saída do valor de n.

Algoritmo 15: Contagem recursiva 0 até n progressivamente

Procedimento $Contagem01(n : inteiro)$
Inicio
  se $n = 0$ **então**
    Saída: n
  senão
    Contagem01(n-1)
    Saída: n
  fim se
Fim

Algoritmo 16: Contagem recursiva 0 até n regressivamente

Procedimento $Contagem02(n : inteiro)$
Inicio
  se $n = 0$ **então**
    Saída: n
  senão
    Saída: n
    Contagem02(n-1)
  fim se
Fim

A figura 44 mostra os passos do processamento de $Contagem01(2)$. A execução para $n = 2$ será processada (1) e será realizada uma chamada (2) $Contagem01(1)$ sem que o procedimento anterior tenha sido concluído. A execução para n=1 novamente fará uma chamada recorrente sem ter sido concluída, dessa vez (3) $Contagem01(0)$. A execução para $n = 0$ atingirá a base da recorrência ou ponto de parada, fará a exibição de 0 (valor de n naquela instância de execução) e concluirá a execução retornando para a instrução seguinte a sua chamada (4). Essa instrução exibirá o valor 1 (valor de n) e, como não há instrução alguma após, finalizará a execução retornando à instrução seguinte à sua chamada (5). Então, o valor 2 será exibido, a execução finalizada e o retorno à instrução seguinte ao ponto que deu origem a todas as chamadas (5). Portanto, a chamada Contagem01(2) fez a exibição dos valores 0, 1, 2. Logo, o procedimento Contagem01(n) exibe os valores de 0 a n em ordem crescente.

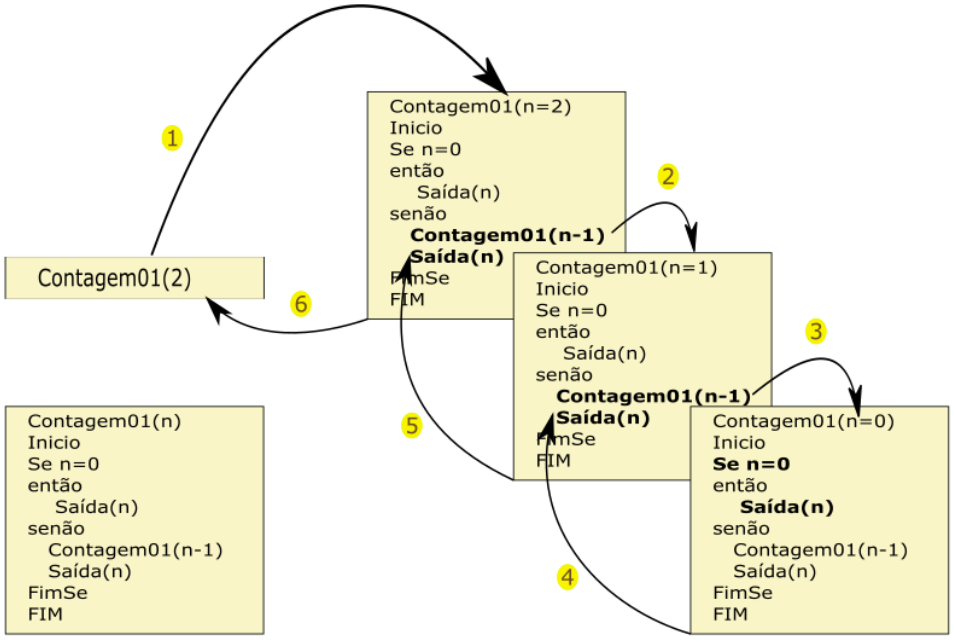

Figura 44 – Funcionamento do Algoritmo 15 contagem recursiva (Elaborado pelo autor)

O algoritmo 16 tem a instrução **Saída**(n) antes da chamada recursiva. Se o processo for desenhado da mesma forma que o ilustrado na figura 44, será percebido que uma chamada Contagem02(2) exibe os valores 2, 1, 0. Logo, o procedimento Contagem2(n) exibe os valores de 0 a n em ordem decrescente.

A figura 45 tem um código em Scratch para práticar e entender a recursividade. As mensagens estão utilizando o bloco diga n por 1 segundos para que haja tempo hábil para visualização. Esse código possui um bloco para exibição do valor de n antes da chamada recursiva e outro após, ou seja, o processamento produzirá uma contagem crescente e outra decrescente.

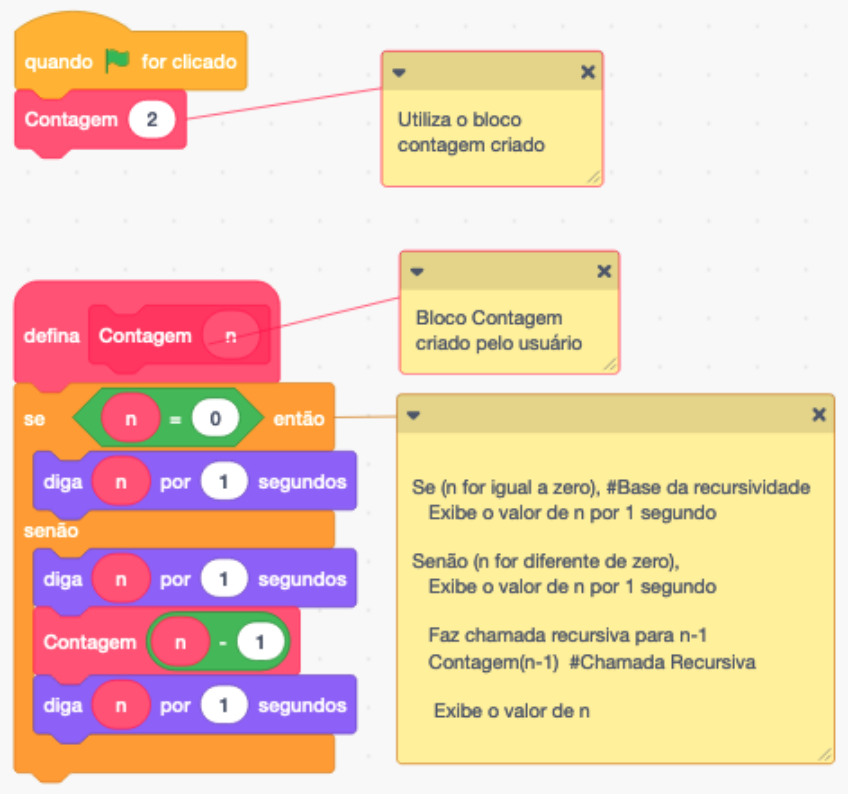

Figura 45 – Contagem Recursiva em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/548394767/

A seguir serão discutidas limitações do Scratch para implementações recursivas junto com abordagens das implementações recursivas para Soma dos Naturais até n (S_n) e Fatorial de n ($n!$). Após, será abordade a implementação computacional recursiva para o cálculo do n-ésimo termo da sequência de Fibonacci ($Fibo_n$) e discutida sua ineficiência. Por fim, será apresentado o problema da Torre de Hanoi que poderá ser utilizado como forma de estímulo para o pensamento lógico recorrente na resolução de problemas.

Fatorial de n (n!)

O algoritmo 17 mostra um código para calcular de forma recursiva o fatorial de um número. Em linguagens de programação que têm a capacidade de processamento recursivo e possibilitam a implementação de funções a codificação é direta. O Scratch não permite a implementação de funções e, portanto, a implementação necessitará de uma solução de contorno[1]: será utilizada uma variável global Fat para guardar o valor da última chamada recursiva e atualizá-lo. A implementação está na figura 46.

[1] As implementações em Portugol e Python que constam nos apêndices são mais adequadas para o estudo da recursividade

Algoritmo 17: Cálculo recursivo de Fatorial de n - $n! = n * (n-1)!$

Função $Fatorial(n : inteiro) : inteiro$
Inicio
  se $n = 0$ **então**
    retorne 1
  senão
    retorne $n * Fatorial(n-1)$
  fim se
Fim

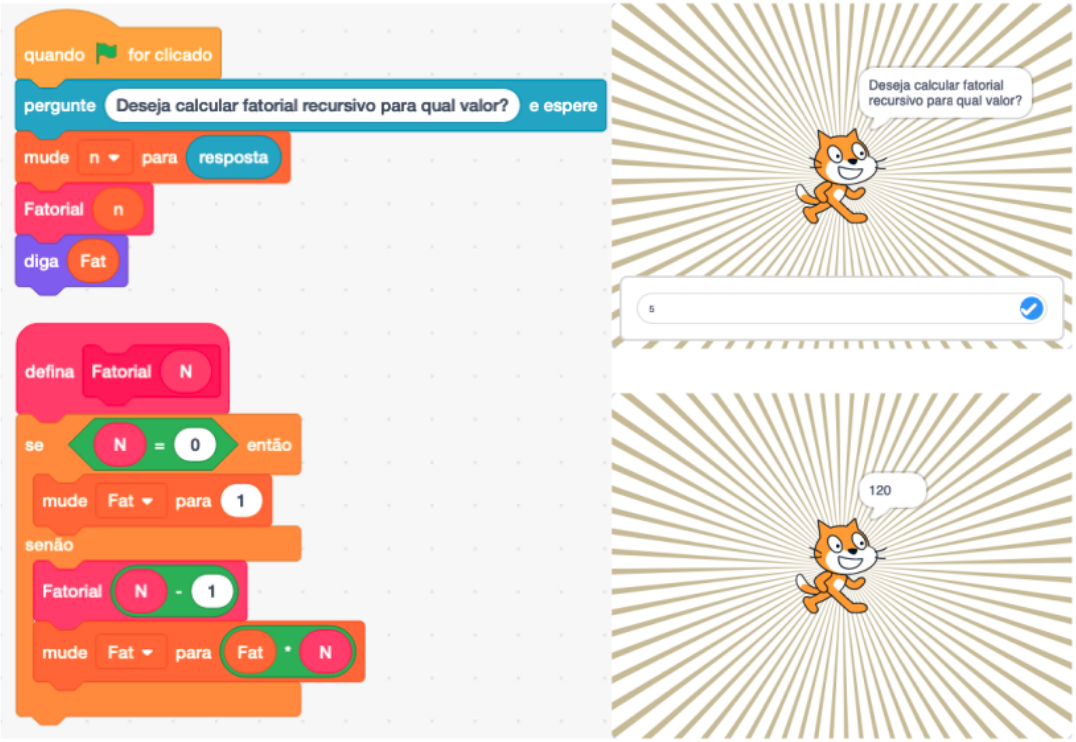

Figura 46 – Cálculo recursivo de Fatorial de n em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/546568211/

Soma dos Naturais até n (S_n)

O algoritmo 18 mostra uma função recursiva para calcular a soma dos naturais até n e a figura 47 mostra a implementação em Scratch. As mesma ressalvas feitas para a implementação do fatorial recursivo aqui se aplicam. Implementações mais adequadas em Portugol e Python estão nos respectivos apêndices. No projeto Scratch foi utilizada a variável global $SomaN$ para o processamento dos valores de retorno das chamadas recursivas.

Algoritmo 18: Cálculo recursivo da Soma dos n primeiros naturais - $S_n = n + S_{n-1}$

Função $SomaN(n : inteiro) : inteiro$
Inicio
  se $n = 0$ **então**
    retorne 0
  senão
    retorne $n + SomaN(n - 1)$
  fim se
Fim

Figura 47 – Cálculo recursivo de Fatorial de n em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/548303730/

N-ésimo Termo da Sequência de Fibonacci ($Fibo_n$)

O algoritmo 19 mostra uma função recursiva para calcular o n-ésimo termo da sequência de Fibonacci. As implementações em Portugol e Python são mais didáticas pois a implementação desse algoritmo em Scratch exigirá uma solução de contorno mais elaborada pois existirão duas chamadas recorrentes para cálculo de um termo. A figura 48 mostra a sequência de chamadas para o cálculo de $Fibo(4)$.

Algoritmo 19: Cálculo recursivo do n−ésimo termo sequência Fibonacci

Função $Fibo(n : inteiro) : inteiro$
Inicio
  se $(n = 1)$ *ou* $(n = 2)$ **então**
    retorne 1
  senão
    retorne $Fibo(n-1) + Fibo(n-2)$
  fim se
Fim

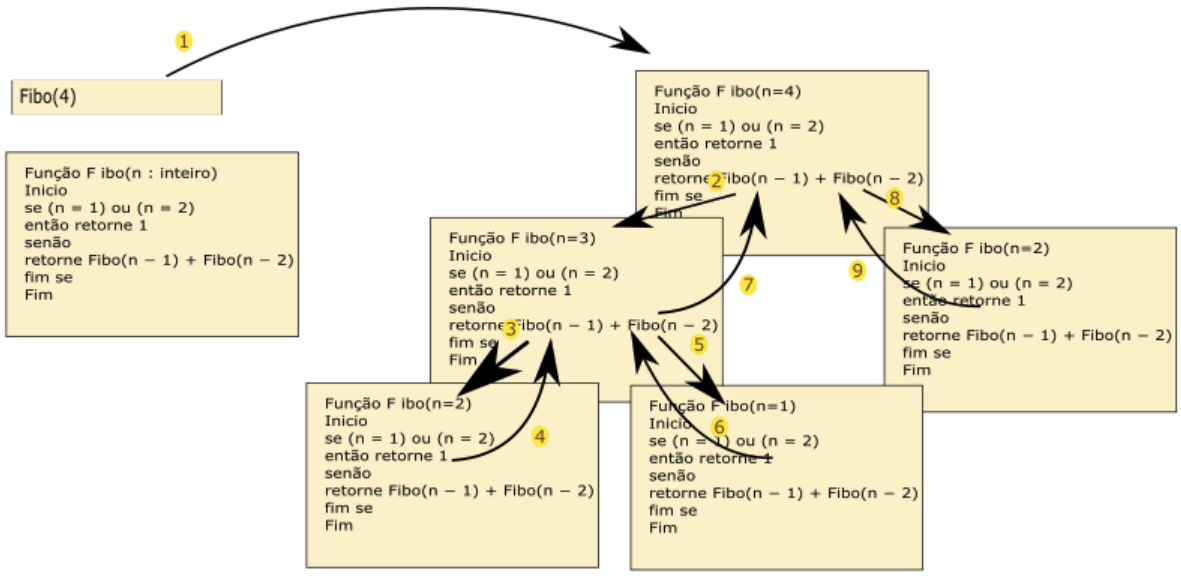

Figura 48 – Sequência de Chamadas Recursivas para $Fibo(4)$ (Elaborado pelo autor)

A solução de contorno será a utilização de uma lista nomeada $PilhaFatorial$ que será abstração de uma estrutura de dados denominada Pilha, caracterizada pela disciplina de acesso aos dados - último que entra é o primeiro a sair - UEPS (*Last In First Out*, em inglês). Assim, cada valor calculado pelo procedimento será inserido na posição 1 da lista $PilhaFatorial$ (topo da pilha) e a soma de dois termos consecutivos será calculada retirando os dois elementos do topo da pilha e re-empilhando o resultado. O projeto Scratch da figura 46 também utilizada a variável global $contagem$ para calcular o número de termos calculados.

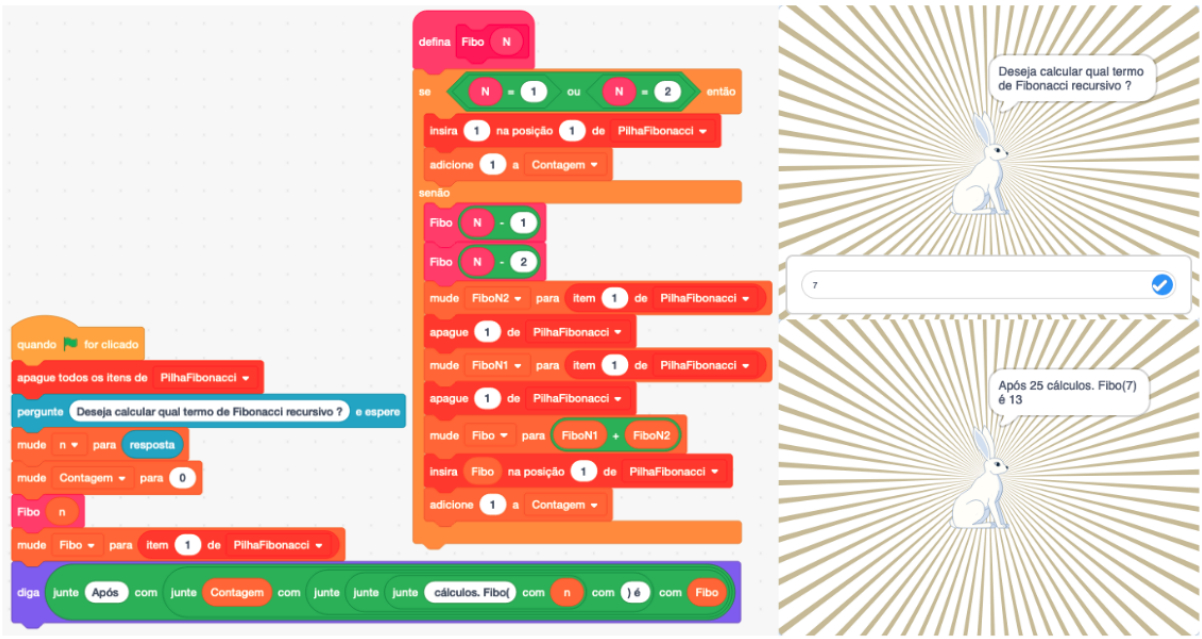

Figura 49 – Cálculo recursivo do termo de Fibonacci em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/548375360/

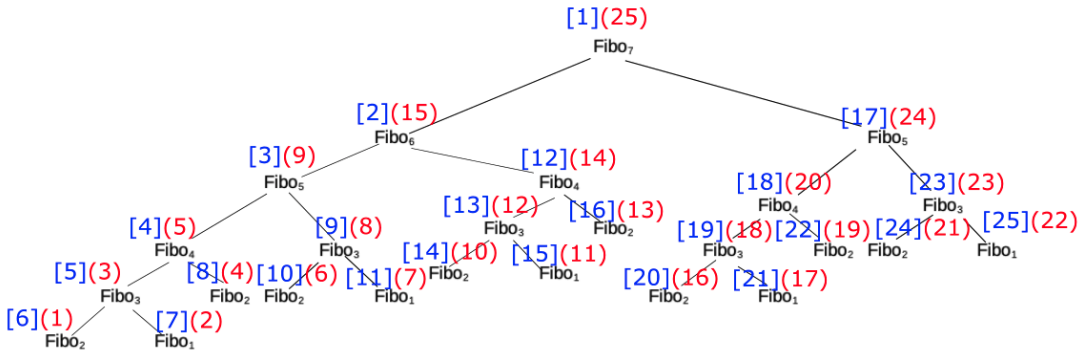

Figura 50 – Sequência de Cálculo Recursivo para Fibo(7) (Elaborado pelo autor)

A figura 50 exibe a árvore com os termos calculados e as respectivas ordens de chamada (em azul, entre colchetes) e de encerramento (em vermelho, entre parênteses). Veja que para Fibo(7) foram realizados 25 cálculos. Essa solução recursiva é extremamente ineficiente pois tem complexidade $O(2^n)$[1]. Assim, foram abordadas três formas de cálculos para um termo da sequência de Fibonacci: Fórmula de Binet com complexidade $O(1)$, pois é um cálculo direto para qualquer que seja o valor de n; Cálculo iterativo (com técnica de programação dinâmica) com complexidade $O(n)$, pois necessita de $n-2$ iterações para calcular o n-ésimo termo; e o cálculo recursivo com complexidade $O(2^n)$.

[1] quanto maior o n, mais próximo de 2^n estará o número de termos calculados

Torre de Hanoi com 3 hastes e n discos

> Em Benares, no norte da Índia, sob a grande cúpula do templo Brama, haveria um enorme jogo de Torre de Hanói, com hastes de diamantes e 64 discos de ouro. O jogo teria sido posto lá pelo deus Brama em pessoa, no momento da criação do mundo. Naquele momento os discos estariam todos empilhados em uma das hastes. Caberia aos sacerdotes mover os discos, segundo as regras. No momento em que todos os discos tiverem sidos empilhados na outra haste, terá chegado a hora do deus voltar e pôr fim ao mundo com um relâmpago. (COUTINHO, 2011, p.88)

O clássico problema da Torre de Hanoi consiste em um tabuleiro com 3 hastes: A, B, C e uma coleção de n discos de tamanhos diferentes. O problema consiste em empilhar n discos ordenadamente na haste A (o maior na base e o menor no topo), movê-los para haste C utilizando a haste B como auxiliar, seguindo os critérios:

1. Mover um disco de cada vez (um disco deve sair de uma haste e ir para outra);
2. Um disco maior não pode sobrepor um menor;
3. Deve ser realizado o menor número de movimentos (a princípio, este critério pode ser flexibilizado e ser realizada a contagem do número de movimentos realizados). O menor número de movimentos é $2^n - 1$, sendo n o número de discos.

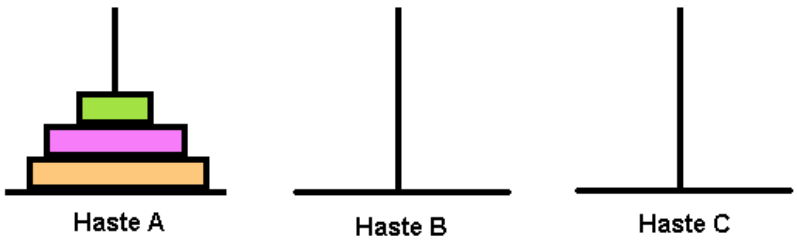

Figura 51 – Torre de Hanoi com 3 discos (Elaborado pelo autor)

O problema da Torre de Hanoi tem uma solução simples mas que requer entendimento de recorrências.

Algoritmo 20: Torre de Hanoi

Procedimento $Hanoi(n, origem, destino, auxiliar)$
Inicio
  se $n \geq 1$ **então**
    $Hanoi(n-1, origem, auxiliar, destino)$
    $MoverDisco(origem, destino)$
    $Hanoi(n-1, auxiliar, destino, origem)$
  fim se
Fim

7 DIVISIBILIDADE E NÚMEROS PRIMOS

Este capítulo têm como foco a divisibilidade de números inteiros e números primos. Inicialmente, o conceito de divisibilidade é abordado. Em seguida, são apresentados e discutidos Projetos Scratch para exibir os divisores de um número e para exibir o maior número que divide dois números - Máximo Divisor Comum (MDC), utilizando o algoritmo de Euclides. Dando sequência, são apresentados os conceitos de primalidade e coprimalidade. Dois Projetos Scratch abordam primalidade: um Projeto faz a classificação de um número como primo ou composto, baseado na lógica do projeto que exibe divisores de um número; outro projeto gera e exibe os números primos até 100 utilizando a técnica do Crivo de Eratóstenes. Após essas abordagens, o Teorema da Fatoração Única ou Teorema Fundamental da Aritmética é apresentado e seu respectivo Projeto Scratch discutido. Após, conceitos da Teoria dos Números relacionados a primalidade são apresentados, tais como: Pequeno Teorema de Fermat, Pseudoprimos, Números de Carmichael, Números de Mersenne, Números de Fermat e Primos Gêmeos. Ao final, o Geogebra CAS é utilizado em alguns cálculos de grandes números para comprovar alguns resultados apresentados.

Inicialmente, serão revisados alguns conceitos:

- Uma multiplicação[1], primitivamente, consiste em:

$$a * n = \underbrace{a + a + a + \cdots + a}_{n\ termos}$$

- Uma divisão (inteira)[2], primitivamente, consiste em[3]:

$$a/m = \underbrace{m + m + m + \cdots + m}_{n\ termos} + r\ (0 \leq r < m)$$

  onde: n é o maior número de vezes que m pode ser retirado de a e r é o que sobrou (resto ou resíduo).

  A partir das definições dos termos d, m, e r acima, serão adotadas as seguintes denominações e notações:

  - a é congruente a r módulo m: $a \equiv r\ (mod\ m)$. Ex.: $5 \equiv 1\ (mod\ 2)$
  - m divide a (caso $r = 0$): $m \mid a$. Ex.: $5 \mid 15 \ \leftrightarrow\ 15 \equiv 0\ (mod\ 5)$
  - m não divide a (caso $r \neq 0$): $m \nmid a$. Ex.: $4 \nmid 15 \ \leftrightarrow\ 15 \not\equiv 0\ (mod\ 4)$

[1] exercício que poderia ser praticado: implementar o produto de dois valores inteiros, utilizando apenas adições
[2] outro exercício que poderia ser praticado: implementar a divisão de dois valores, utilizando apenas subtrações
[3] quantos "m" cabem dentro do "a"? Houve sobra ou resíduo?

7.1 Divisores de um Número

Nesta secção será abordado um algoritmo para exibir os divisores de um número n qualquer. Esse algoritmo será a base da lógica para construção de outros algoritmos, tais como: contar os divisores de um número para classificá-lo como primo ou composto; somar os divisores de um número para classificá-lo como primo ou perfeito.

O algoritmo 21 descreve os passos para exibição dos dividores de um número n. A lógica consiste da leitura de um valor para n, uma estrutura de repetição para i assumir a cada iteração um dos valores de $1, 2, 3, \cdots, n$ e realizar teste se i divide n. Se o i dividir n (n mod $i = 0$), i é divisor de n e, consequentemente, é exibido.

Algoritmo 21: Exibe os divisores de um número

Var i, n: inteiro
Inicio
  Saída: "Quer saber os dividores de qual valor?"
  Entrada: n
  `// Testa se cada valor de i (1, 2, ···, n) divide n`
  para $(i \leftarrow 1;\ i \leq n;\ i \leftarrow i + 1)$ **faça**
    `// Se o resto da divisão de n por i for 0, i é um divisor`
    se $(n\ mod\ i)\ =\ 0$ **então**
      Saída: i
    fim se
  fim para
Fim

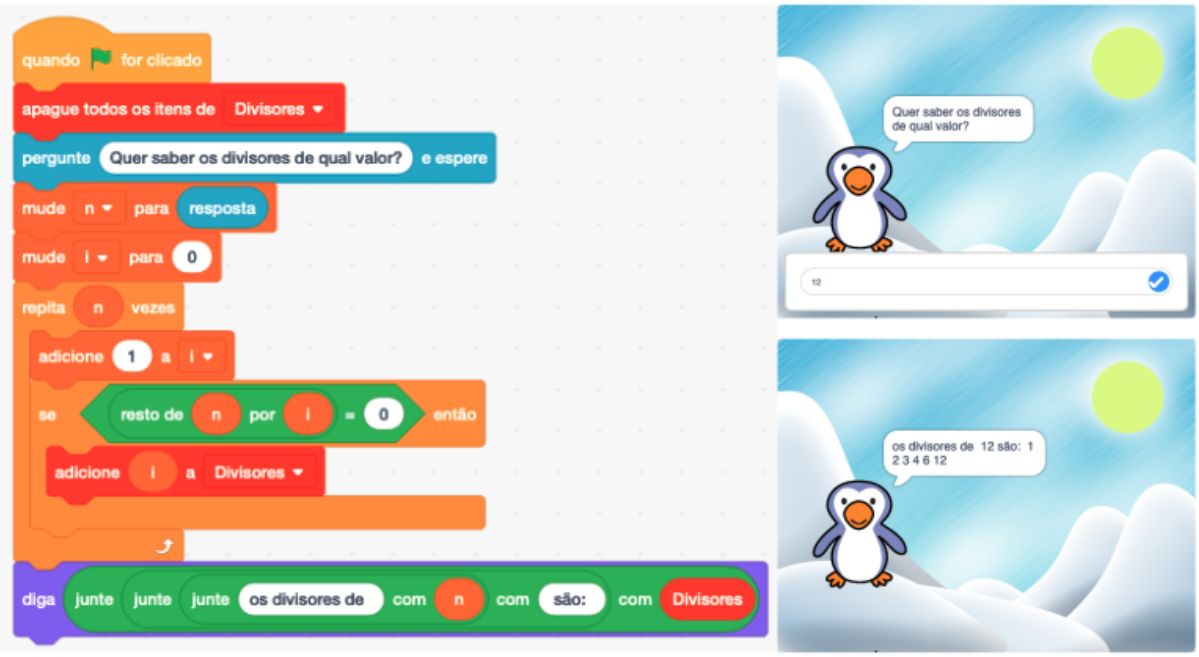

Figura 52 – Divisores de um Número em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/544765160/

A implementação em Scratch (figura 52) tem algumas adaptações mas com a mesma lógica do algortimo 21 apresentado. A estrutura de repetição **para** utilizada no algoritmo é adaptada para uma estrutura de repetição **repita** n **vezes** do Scratch, como já abordado anteriormente. Outra adequação foi a utilização da variável $Divisores$ do tipo lista para armazenar os divisores do número. No início do código todos os elementos da lista $Divisores$ são apagados. Nas iterações, cada valor de i que for divisor de n, em vez de ser exibido, é incluído na lista $Divisores$. Após todas as iterações, a lista $Divisores$ é exibida[1]

7.2 Máximo Divisor Comum (MDC) - Algoritmo de Euclides

Nesta seção será abordado o Algoritmo de Euclides para cálculo do Máximo Divisor Comum de dois números. O enunciado que consta no Livro VII dos Elementos de Euclides segue:

> **Sendo dados dois números não primos entre si, achar a maior medida comum deles**
> Sejam AB, CD os dois números dados não primos entre si. É preciso, então, achar a maior medida comum dos AB, CD.
> Se, por um lado, de fato, o CD mede o AB, mas mede também a si mesmo, portanto o CD é uma medida comum dos CD, AB. E é evidente que é também a maior; pois, nenhum maior do que o CD medirá CD.
> Se, por outro lado, o CD não mede o AB, dos AB, CD, sendo sempre subtraído de novo o menor do maior terá restado algum número; e, se não, os AB, CD serão primos entre si; o que não foi suposto. Portanto, terá restado algum número, o qual medirá o antes dele mesmo. E, por um lado, o CD, medindo o BE, reste um menor do que ele mesmo, o EA, e, por outro lado, o EA, medindo o DF, reste um menor do que ele mesmo, o FC e o CF meça o AE. Como, de fato, o CF mede o AE, e o AE mede o DF, portanto o CF medirá o DF; e mede também a si mesmo; portanto, medirá também o CD todo. E o CD mede o BE; portanto, o CF mede também o BE; e mede também o EA; portanto, medirá também o BA todo; e mede também o CD; portanto, o CF mede os AB, CD Portanto, o CF é uma medida comum dos AB, CD. Digo, então, que também é maior. Pois, se o CF não é a maior medida comum dos AB, CD, algum número medirá os números AB, CD, sendo maior do que CF. Meça. e seja o G. E como o G mede o CD, e o CD mede o BE, portanto também o G mede o BE; e mede também o BA todo; portanto, medirá também o AE restante. Mas o AE mede o DF; portanto, o G medirá também o DF; e mede também o DC todo; portanto, também medirá o CF restante, o maior, o menor; o que é impossível; portanto, nenhum número medirá os números AB, CD, sendo maior do que CF portanto, o CF é a maior medida comum dos AB, CD; [o que era preciso provar].
>
> COROLÁRIO
>
> Disso, então, é evidente que, caso um número meça dois números, também medirá a maior medida comum deles; o que era preciso provar. (EUCLIDES, 2009, p. 271)

Segundo Garbi (2010), pouco se sabe sobre Euclides sendo provável que tenha sido aluno da Academina de Platão. Euclides foi diretor da área de Matemática do Museu de Alexandria,

[1] Em Scratch, os blocos de aparência **pense** ou **diga** tratam listas como tipos simples, ou seja, exibem todos os elementos sem necessidade de fazê-lo elemento a elemento.

onde ensinou e escreveu os Elementos e outros livros por volta de 300 a.C. Nenhum autor conseguiu êxito comparável à Euclides com seus **Elementos**, composto de 13 livros. O mais antigo e influente livro de Matemática ainda em vigor nos dias de hoje.

A equação 7.1 mostra a expressão expandida para o processo descrito por Euclides:

$$a = ((((\cdots(\overbrace{(\overbrace{(r_{n-1} \cdot q_n + r_n)}^{r_{n-2}} \cdot q_{n-1} + r_{n-1})}^{r_{n-3}} \cdots) \cdot q_3 + r_3) \cdot q_2 + r_2) \cdot q_1 + r_1) \tag{7.1}$$

b (…) r_{n-2} r_{n-3} r_1 a

A figura 53 e a equação 7.2 mostram o processo para cálculo do MDC entre 36 e 15 e a equação expandida, respectivamente. Pela equação expandida é possível identificar 15 e 36 gerados a partir do valor 3, o máximo divisor comum a ambos.

quociente		2	2	2
	36	15	6	3
resto	6	3	0	

Figura 53 – Cálculo do MDC entre 36 e 15 (Elaborado pelo autor)

$$a = (\overbrace{(\overbrace{(\overbrace{(\mathbf{3} \cdot 2 + 0)}^{6} \cdot 2 + 3)}^{15} \cdot 2 + 6)}^{36}) \tag{7.2}$$

Algoritmo 22: Calcula MDC de dois valores - Algoritmo de Euclides

Var $a, b, resto$: inteiro
Inicio
  Saída: "Informe o primeiro valor:"
  Entrada: A
  Saída: "Informe o segundo valor:"
  Entrada: B
  $Resto \leftarrow A \ mod \ B$
  `// Encerra caso` $b \mid a$ `ou nova iteração entre divisor e resto`
  enquanto $Resto \neq 0$ **faça**
    $A \leftarrow B$
    $B \leftarrow Resto$
    $Resto \leftarrow A \ mod \ B$
  fim enqto
  Saída: "O MDC é: ", B
Fim

O algoritmo 22 é uma implementação iterativa do algoritmo de Euclides para cálculo do Máximo Divisor Comum de dois valores. Inicialmente, os dois valores a e b são lidos e o $resto$ da divisão é calculado [1]. Se $resto$ for zero, b divide a e, portanto, será o MDC. Entretanto, se b não divide a, uma iteração será processada: o valor de a para a ser o quociente, o valor de b passa a ser o resto e um novo $resto$ é calculado. Esse processo é repetido até $resto$ ser zero. Finalmente, b será exibido como o MDC dos valores iniciais. Cabe aqui ressaltar que os valores originais de a e b se perdem nesse processo. Caso haja interesse em mantê-los, outras variáveis deverão ser definidas e utilizadas no processo iterativo. A implementação do algortimo 22 em Scratch está na figura 54.

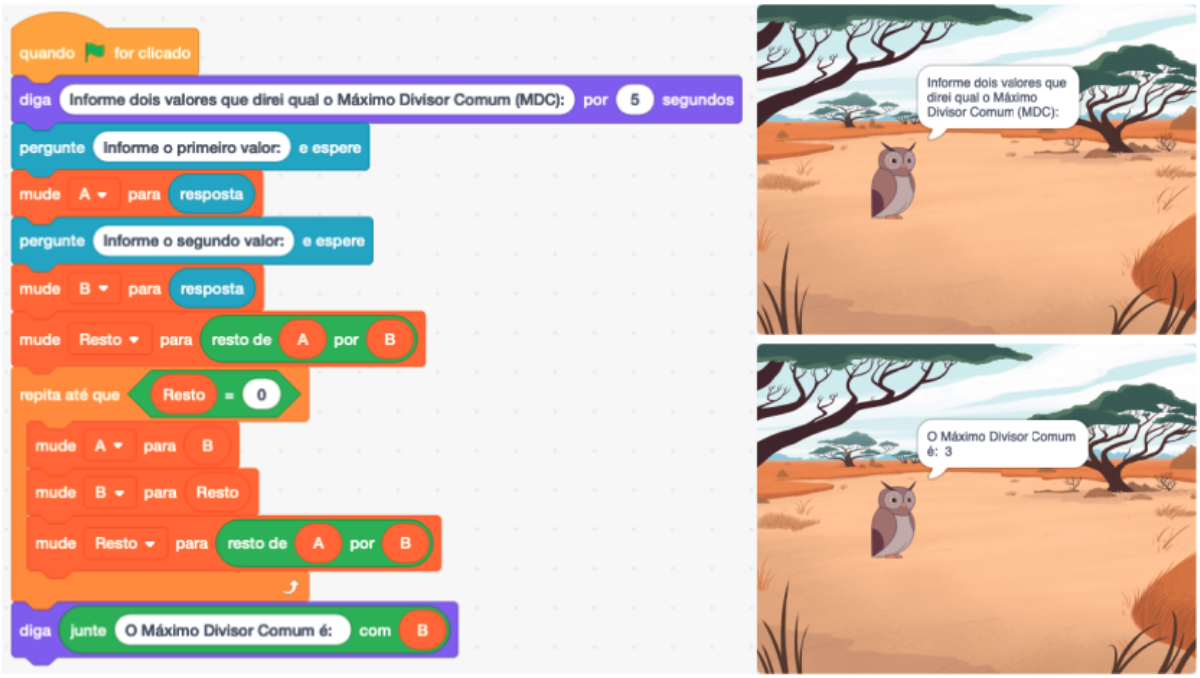

Figura 54 – Cálculo do Máximo Divisor Comum (MDC) em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/548426901/ e uma versão aprimorada com um estágio que ensina passo a passo o algoritmo de Euclides está disponível em: https://scratch.mit.edu/projects/554063400/

A equação 7.3 sintetiza o Algoritmo descrito por Euclides em forma de definição recursiva:

$$MDC(a,b) = \begin{cases} b, \ se\ b \mid a \\ MDC(b, a \bmod b), \ se\ b \nmid a \end{cases} \tag{7.3}$$

7.3 Número Primo

Os números primos podem ser considerados os tijolos fundamentais da aritmética, e a estrutura do conjunto dos números primos tem inquietado gerações

[1] se o segundo valor for maior que o primeiro não haverá problemas, apenas uma iteração a mais que fará a troca dos valores

> de matemáticos de todos os tempos. Depois de tantos anos de trabalho intenso de muitos matemáticos, a quantidade de problemas abertos e conjecturas sobre números primos é muito grande. (MOREIRA, 2010)

Um número natural diferente de 1 é denominado número **primo** quando possui unicamente dois divisores: a unidade e ele próprio. A denominação primo origina-se de primeiro, ou seja, um número primo não é gerado por nenhum outro além da unidade. Ele é um gerador de números. Um número natural diferente de 1 e que não for primo é dito **composto**, ou seja, números compostos podem ser gerados pela unidade e por outro número diferente dele próprio. Exemplos:

- **7 é primo** pois só pode ser gerado pela unidade ou por ele próprio:
  $\underbrace{1+1+1+1+1+1+1}_{7\ vezes\ 1}$ ou $\underbrace{7}_{1\ vez\ 7}$

  Note que os divisores de 7 são: 1 e 7.

- **14 é composto** pois pode ser gerado pela unidade, pelo 2, pelo 7 e por ele próprio:

  $\underbrace{1+1+1+1+1+1+1+1+1+1+1+1+1+1}_{14\ vezes\ 1}$ ou $\underbrace{14}_{1\ vez\ 14}$ ou

  $\underbrace{2+2+2+2+2+2+2}_{7\ vezes\ 2}$ ou $\underbrace{7+7}_{2\ vezes\ 7}$

  Note que os divisores de 14 são: 1, 2, 7 e 14. Como existem divisores além da unidade e do próprio valor, ele é dito composto.

O algoritmo 23 classifica um número como primo ou composto baseado na contagem do número de divisores. A lógica aplicada nele é similar a do algoritmo 21, com uma pequena adequação: em vez de exibir os divisores, uma contagem dos divisores é realizada pela variável c. Assim, teremos: a leitura do valor de n, uma estrutura de repetição para i percorrer os valores de 1 a n. A cada iteração, o valor do resto da divisão de n por i é testado. Se o resto for zero, i é divisor de n e, consequentemente, o contador c é aumentado em uma unidade. Após todas as iterações, o valor de c é testado e o número classificado como primo, se computar apenas 2 dividores, ou composto, se computar número maior de divisores.

A figura 55 mostra o projeto Scratch para classificar um número como primo ou composto. A implementação em Scratch segue a lógica do algoritmo 23 com a adequação já abordada da estrutura de controle de repetição **para** para a estrutura de controle **repita** n **vezes**

Algoritmo 23: Classifica um número como Primo ou Composto

Var i, n, c: inteiro
Inicio
  Saída: "Quer saber se qual valor é primo ou composto?"
  Entrada: n
  $c \leftarrow 0$
  `// Testa se cada valor de i (1, 2, ···, n) divide n`
  para $(i \leftarrow 1;\ i \leq n;\ i \leftarrow i+1)$ **faça**
    `// Se o resto da divisão de n por i for 0, i é um divisor`
    se $((n\ mod\ i)\ =\ 0)$ **então**
      $c \leftarrow c+1$
    fim se
  fim para
  `// Se só existirem 2 divisores, n é primo`
  se $(c\ =\ 2)$ **então**
    Saída: n," é número primo"
  senão
    Saída: n," é número composto. Possui ", c," divisores"
  fim se
Fim

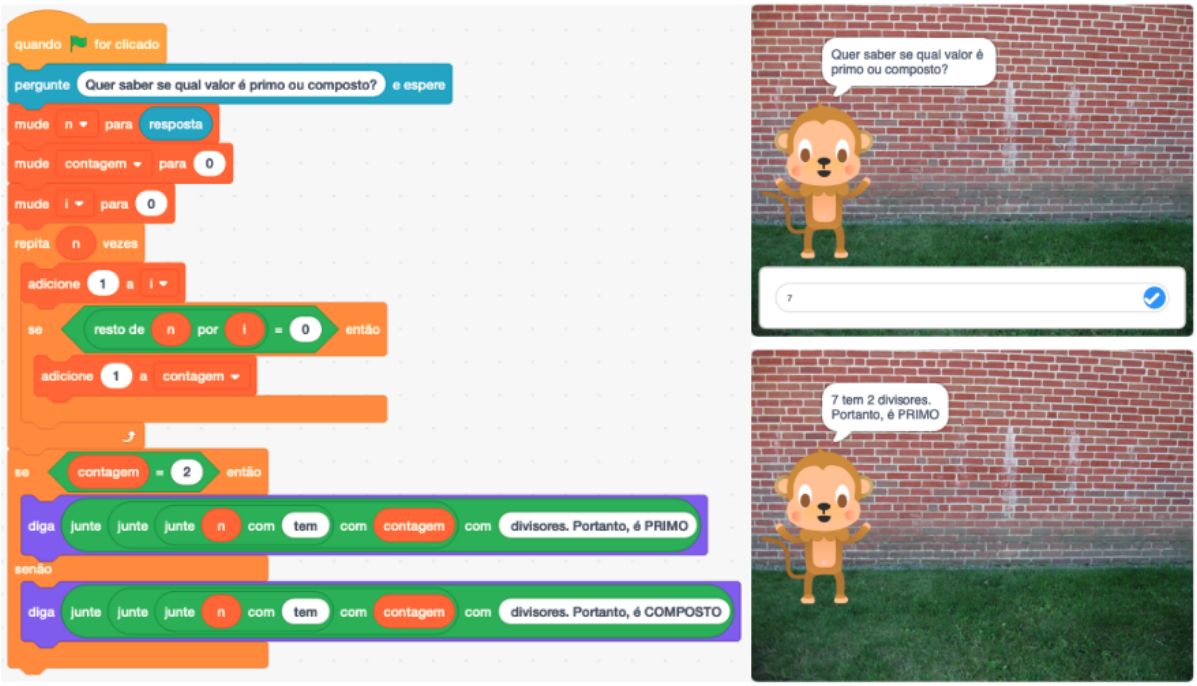

Figura 55 – Primo ou Composto em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/544764635/

> Desde os dias de Euclides, sabe-se que o número de primos é infinito, mas é evidente que a densidade dos números primos decresce quando avançamos para inteiros maiores. Tornou-se, então, um dos problemas mais famosos descrever a distribuição e primos entre números naturais. (BOYER, 2012)

Euclides, no Livro VII, define: "Um número primo é medido por uma medida só"; "Números primos entre si são os medidos por uma unidade só como medida em comum"; "Um número composto é medido por algum número";"Números compostos entre si são os medidos por algum número como medida comum"; "Um número perfeito é o igual às suas próprias partes". (EUCLIDES, 2009)

Pitágoras era fascinado pelos números perfeitos. Uma de suas descobertas foi a de que todas as potências de 2 chegavam perto mas falhavam em uma unidade[1]: $2^2 = 4$, soma dos divisores $1 + 2 = 3$; $2^3 = 8$, soma dos divisores $1 + 2 + 4 = 7$. Dois séculos depois, Euclides descobriu que os números perfeitos são sempre múltiplos de dois números, um dos quais é uma potência de 2 e o outro é a potência seguinte menos 1. $6 = 2^1 * (2^2 - 1)$, $28 = 2^2 * (2^3 - 1)$, $496 = 2^4 * (2^5 - 1)$, $8128 = 2^6 * (2^7 - 1)$. (SINGH, 2019)

Hefez (2016) denota $S(n)$ como a soma de todos os divisores naturais de n e define:

- Seja $n \in \mathbb{N}$. Tem-se que $S(n) = n + 1$ se, e somente se, n é um número primo.
- Um $n \in \mathbb{N}$ é chamado de número perfeito se $S(n) = 2 \cdot n$, ou, ainda, se n é igual a soma dos seus divisores naturais distintos dele mesmo. Exemplos: 6, 28, 496, 8128 e 3350336

A partir das definições acima e das lógicas abordadas nos algoritmos 21, 22 e 23 poderão ser propostas atividades de desenvolvimento de algoritmos: exibir os números primos entre 2 e 100; exibir os números perfeitos entre 2 e 100 (somente 6 e 28 devem ser exibidos); classificar dois números como primos entre si (coprimos) ou compostos entre si; calcular e exibir o Mínimo Múltiplo Comum - MMC de dois valores a e b.

$$MMC(a, b) = \frac{a \cdot b}{MDC(a, b)}$$

7.4 Crivo de Eratóstenes

Eratóstenes (viveu aproximadamente de 275 a 194 a.C.) deu contribuições a vários domínios do conhecimento tendo sido bem conhecido dos matemáticos pelo "crivo de Eratóstenes", um método sistemático para isolar números primos. Os naturais são dispostos em ordem e riscados os números de dois em dois a partir do dois, de três em três a partir do três, de cinco em cinco a partir do cinco e assim sucessivamente. Os números restantes a partir do 2 são números primos. (BOYER, 2012)

[1] veja sistema de numeração binário no capítulo 8

O algortimo 24 implementa o método proposto por Eratóstenes para um conjunto de valores de 1 a 100. A lógica do algoritmo consiste em gerar um vetor ou lista de 100 elementos com a correspondência entre o índice e o valor: $X[1] = 1, X[2] = 2, \cdots, X[100] = 100$. Inicialmente retira os múltiplos de $p = 2$ (a marcação será atribuindo zero ao valor retirado). Então p é incrementado para um valor não crivado e o processo de crivagem repetido enquanto o p não superar a raiz quadrada de 100 (a última crivagem necessária será a dos múltiplos de 7, último primo inferior 10 - raiz quadrada de 100).

Algoritmo 24: Números primos até 100 - Crivo de Eratóstenes

Var $i, p, crivo$: inteiro; X[1..100] de inteiro
Inicio
// Gera uma lista com os números até 100
para $(i \leftarrow 1;\ i \leq 100;\ i \leftarrow i + 1)$ **faça**
$X[i] \leftarrow i$
fim para
// Inicia crivagem de 2 até o número primo menor 10
$p \leftarrow 2$
enquanto $(p < 10)$ **faça**
// Criva os múltiplos: 2p, 3p, ...
$crivo \leftarrow 2 \cdot p$
enquanto $(crivo \leq 100)$ **faça**
$X[crivo] \leftarrow 0$
$crivo \leftarrow crivo + p$
fim enqto
// Descobre o próximo primo para crivagem
$p \leftarrow p + 1$
enquanto $X[p] = 0$ **faça**
$p \leftarrow p + 1$
fim enqto
fim enqto
// Saída de $X[i], X[i] \neq 0$
para $(i \leftarrow 2;\ i \leq 100;\ i \leftarrow i + 1)$ **faça**
se $X[i] \neq 0$ **então**
Saída: X[i]
fim se
fim para
Fim

A figura 56 mostra a implementação do algoritmo 24 em Scratch. Para melhor legibilidade foram criados dois blocos $GeraLista$ e $CrivaMultiplos$, ambos operam sobre a lista de números X. A crivagem na implementação Scratch atribui um "-" ao elemento para facilitar a visualização de saída. A figura 57 mostra as saídas dessa aplicação.

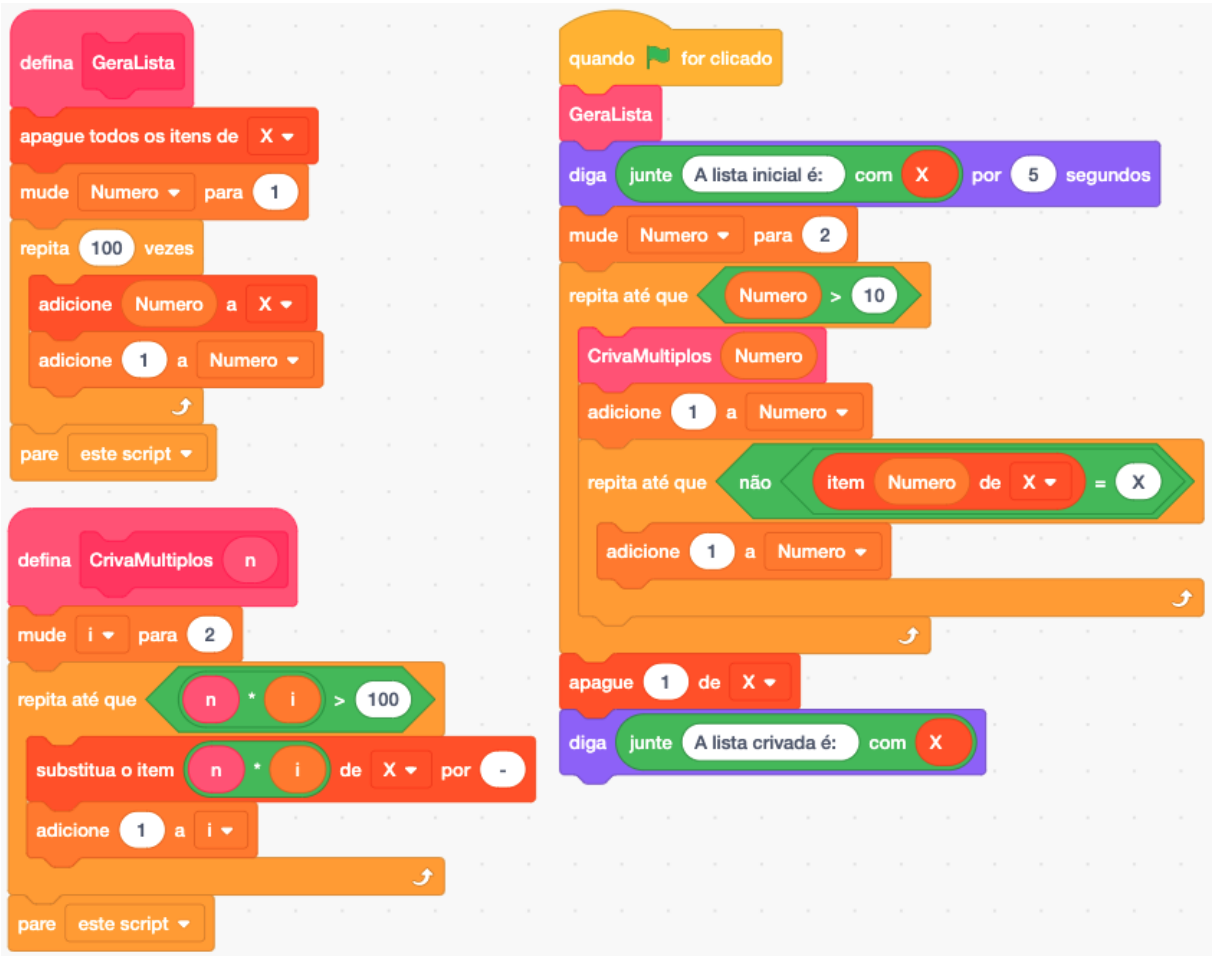

Figura 56 – Crivo de Eratóstenes em Scratch (Elaborado pelo autor)

Figura 57 – Telas do Crivo de Eratóstenes em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/543564651/

7.5 Fatoração - Teorema da Fatoração Única

Coutinho (2011) afirma que, devido à importância, o Teorema da Fatoração Única é também chamado de Teorema Fundamental da Aritmética e o primeiro a enunciá-lo na forma apresentada foi Carl Friedrich Gauss em seu famoso livro *Disquisitiones arithmeticae*. O Teorema afirma que dado um inteiro positivo $n \geq 2$ podemos sempre escrevê-lo, de modo único, na forma $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$, onde $1 < p_1 < p_2 < \cdots < p_k$ são números primos e $e_1, e_2, \cdots, e_k$ são inteiros positivos.

Algoritmo 25: Fatoração de um número n $(n > 1)$

Var $n, fatora, i, p$: inteiro
Inicio
Saída: "Quer saber os fatores de qual valor?"
Entrada: n
// $fatora$ inicia com o valor de n
$fatora \leftarrow n$
$p \leftarrow 2$
// Repete enquanto $fatora \neq 1$
enquanto $(fatora \neq 1)$ **faça**
// Se $p \mid fatora$
se $((fatora \ mod \ p) = 0)$ **então**
$i \leftarrow 0$
// Contagem em i de quantas vezes $p \mid n$
enquanto $((fatora \ mod \ p) = 0)$ **faça**
$i \leftarrow i + 1$
$fatora \leftarrow fatora \ / \ p$
fim enqto
// Exibe p e sua potência i na fatoração
Saída: p," elevado a ",i
fim se
// p é incrementado e poderá assumir valores não primos. Entretanto, valores não primos nunca serão fatores pois seus geradores primos já foram processados na fatoração
$p \leftarrow p + 1$
fim enqto
Fim

O algortimo 25 implementa a fatoração de um número. Os passos do algoritmo consistem, inicialmente, da leitura do valor n e cópia desse valor para a variável $fatora$ que sofrerá divisões sucessivas de fatores primos. Essa variável $fatora$ fará o controle da estrutura repetição mais externa que encerrará quando atingir valor 1, ou seja, quando n estiver sido fatorado. In-

ternamente à estrutura de repetição tem-se o teste se um dado p divide $fatora$. Caso divida, uma outra estrutura de repetição faz a contagem de vezes que p divide $fatora$, ou seja, identificará o expoente do número primo p. O primeiro p candidato a fatorar $fatora$ é o valor 2 e p poderá assumir valores não primos. Entretanto, valores não primos nunca serão fatores pois seus geradores primos já foram processados anteriormente.

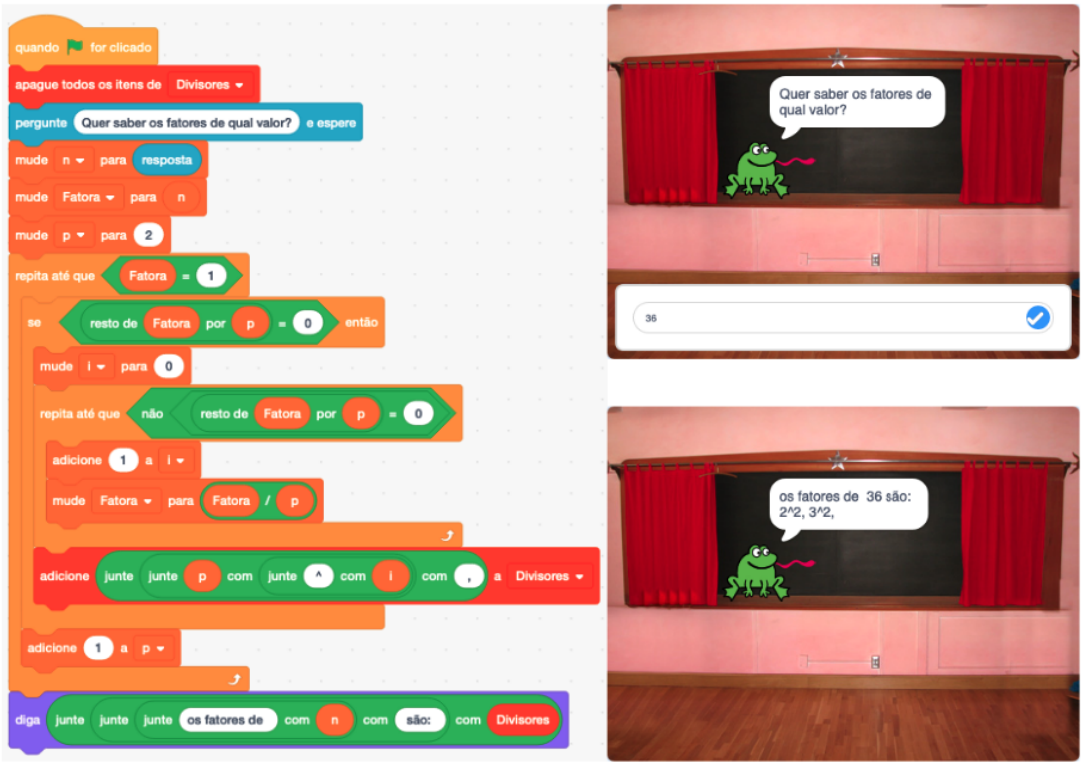

Figura 58 – Fatoração de um Número em Scratch (Elaborado pelo autor)

A figura 58 exibe o Projeto Scratch que implementa o algoritmo 25.
Este projeto está disponível em: https://scratch.mit.edu/projects/540281905/

Pequeno Teorema de Fermat

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			
0	1																		1	=	2^0
1	1	1																	2	=	2^1
2	1	2	1																4	=	2^2
3	1	3	3	1															8	=	2^3
4	1	4	6	4	1														16	=	2^4
5	1	5	10	10	5	1													32	=	2^5
6	1	6	15	20	15	6	1												64	=	2^6
7	1	7	21	35	35	21	7	1											128	=	2^7
8	1	8	28	56	70	56	28	8	1										256	=	2^8
9	1	9	36	84	126	126	84	36	9	1									512	=	2^9
10	1	10	45	120	210	252	210	120	45	10	1								1024	=	2^{10}
11	1	11	55	165	330	462	462	330	165	55	11	1							2048	=	2^{11}
12	1	12	66	220	495	792	924	792	495	220	66	12	1						4096	=	2^{12}
13	1	13	78	286	715	1287	1716	1716	1287	715	286	78	13	1					8192	=	2^{13}
14	1	14	91	364	1001	2002	3003	3432	3003	2002	1001	364	91	14	1				16384	=	2^{14}
15	1	15	105	455	1365	3003	5005	6435	6435	5005	3003	1365	455	105	15	1			32768	=	2^{15}
16	1	16	120	560	1820	4368	8008	11440	12870	11440	8008	4368	1820	560	120	16	1		65536	=	2^{16}
17	1	17	136	680	2380	6188	12376	19448	24310	24310	19448	12376	6188	2380	680	136	17	1	131072	=	2^{17}

Figura 59 – Triângulo de Pascal com destaque para abordagem do PTF (Elaborado pelo autor)

> Talvez dois milênios do tempo de Fermat, tenha havido uma hipótese chinesa que dizia que n é primo se e só se $2^n - 2$ é divisível por n, onde n é um número inteiro maior que um. Sabe-se hoje que metade dessa conjectura é falsa pois $2^{341} - 2$ é divisível por 341, e $341 = 11 \cdot 31$ é composto; mas a outra metade é verdadeira, e o "pequeno" teorema de Fermat é uma generalização disso. (BOYER, 2012, p.250)

Observe o Triângulo de Pascal da figura 59. As linhas associadas aos números primos estão tarjadas para avaliação da conjectura chinesa de que n é primo se e só se $2^n - 2$ é divisível por n. Sendo p um número primo e tendo as equações 7.4 e 7.5 que definem fatorial de p e um termo do Triângulo de Pascal (coeficiente binomial), respectivamente:

$$p! = 1 \cdot 2 \cdot 3 \cdots (p-1) \cdot p \tag{7.4}$$

$$\begin{pmatrix} p \\ q \end{pmatrix} = \frac{p!}{q! \cdot (p-q)!},\ com\ 0 \leq q \leq p \tag{7.5}$$

Se $0 < q < p$, ambos q! e (p-q)! não conterão o termo p em suas parcelas e, consequentemente, o termo $\frac{p!}{q!\cdot(p-q)!}$ será divisível por p. Daí, conclui-se que a conjectura chinesa de que se p é um número primo, então $p \mid 2^p - 2$ é correta. Entretanto, a recíproca não é verdadeira, ou seja, é falso afirmar que se $n \mid 2^n - 2$ então n é número primo. Por exemplo, $341 = 11 \cdot 31$ é um número composto e satisfaz a $341 \mid 2^{341} - 2$.

TEOREMA (PTF) - "Dado um número primo p, tem-se que p divide o número $a^p - a$, para todo $a \in \mathbb{Z}$". Ainda, se $MDC(p, a) = 1$, tem-se que: $p \mid a^{p-1} - 1$, ou seja: $a^{p-1} \equiv 1\ mod(p)$ (HEFEZ, 2016, p.135)

Pseudoprimo

Um inteiro positivo n, impar e composto, é um *pseudoprimo* para a base b (onde 1 < b < n-1) se $b^{n-1} \equiv 1 \ (mod \ n)$. Ex.: 341 é um pseudoprimo para a base 2. Entre 1 e 10^9 existem 50.847.534 primos, mas apenas 5.597 pseudoprimos para a base 2. Para as bases 2 e 3, entre 1 e 10^9 existem apenas 1.272 pseudoprimos (COUTINHO, 2011, p.106)

Números de Carmichael

Um número composto ímpar $n > 0$ é um número de Carmichael se $b^n \equiv b \ (mod \ n)$ para todo[1] $1 < b < n - 1$. Todo número primo satizfaz à equação $b^p \equiv b \ (mod \ p)$, mas não é um número de Carmichael. O menor número de Carmichael é 561. (COUTINHO, 2011)

Números de Mersenne e Números de Fermat

De acordo com Coutinho (2011), duas fórmulas foram intensamente estudadas pelos matemáticos do século XVII:

$$M(n) = 2^n - 1 \tag{7.6}$$

$$F(n) = 2^{2^n} + 1 \tag{7.7}$$

Os números na forma M(n) (equação 7.6) são conhecidos como números de Mersenne. Marin Mersenne (1588-1648) foi um frade e matemático amador do século XVII que se correspondia com muitos matemáticos, dentre eles Fermat, Descartes e Pascal. Mersenne afirmou que os números da forma M(n) seriam primos quando $n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127 \ e \ 257$ e seriam compostos para os outros 44 primos menores que 257. Como ocorria frequentemente na época, Mersenne não apresentou justificativa para esses resultados. Em 1732, erroneamente, Euler afirmou que M(41) e M(47) seriam primos. O primeiro erro da lista de Mersenne foi encontrado em 1886: Pervusin e Seelhof descobriram que 61 não consta na lista mas M(61) é primo. Posteriormente, outros erros foram encontrados: a lista omite 89 e 107, sendo M(89) e M(107) primos; a lista inclui os números 67 e 257 que são compostos. Alguns dos maiores primos conhecidos são números de Mersenne, por exemplo: M(1398296) é um primo com 420.921 algarismos.

Em 1640, Fermat escreveu uma carta para o matemático amador Frenicle enumerando os números da forma $F(n) = 2^{2^n} + 1$ (equação 7.7) para os valores de n entre 0 e 6: 3, 5, 17, 257, 65537, 4294967297 e 8446744073709551617. Em seguida conjecturou que todos os números de Fermat da forma F(n) seriam primos. Euler, quase cem anos mais tarde[2], fatorou F(5) e

[1] MDC(b,n)=1

[2] Em 1732 (BOYER, 2012)

demonstrou que não era um número primo. Até hoje, não se descobriu um número F(n) primo com n>5. Números de Mersenne são uma rica fonte de primos, diferentemente dos números de Fermat que poucos primos são conhecidos.

Primos Gêmeos

Dois números primos p e q são chamados primos gêmeos se $\mid p-q \mid = 2$. Conjectura-se, mas não se sabe demonstrar até agora, que existem infinitos pares de primos gêmeos. O matemático norueguês Viggo Brun (1885-1978) criou a teoria do crivo combinatório e provou que a soma dos inversos dos primos gêmeos $(\frac{1}{5}+\frac{1}{7})+(\frac{1}{11}+\frac{1}{13})+(\frac{1}{17}+\frac{1}{19})+(\frac{1}{29}+\frac{1}{31})+(\frac{1}{41}+\frac{1}{43})+(\frac{1}{59}+\frac{1}{61})+\cdots$ converge [1]. (MOREIRA, 2010)

[1] A Constante de Brun vale aproximadamente 1,9021605824283

7.6 Geogebra CAS

A figura 60 mostra cálculos com o Geogebra CAS para os números de Mersenne citados no parágrafo anterior. Foi utilizada a versão Geogebra Classic com habilitação apenas da janela de Cálculo Simbólico (CAS). Os cálculos de M(41), M(47), M(61), M(89), M(107) e M(67) foram exitôsos com o Geogebra CAS. Entretanto, houve falha e travamento em Fatorar(M(257)[1]). Também, consta o cálculo de F(5) e sua fatoração, demonstrando tratar-se de um número composto.

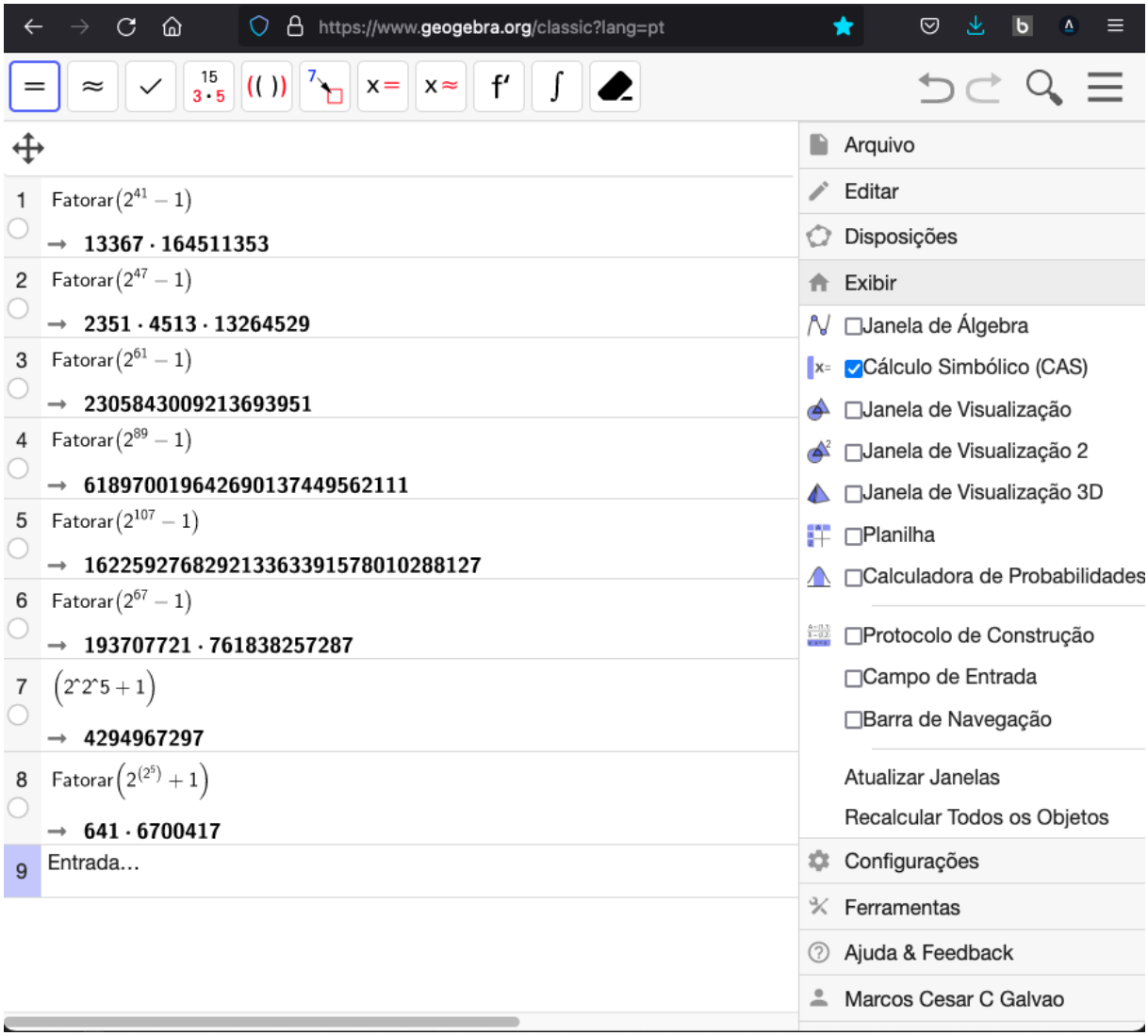

Figura 60 – Cálculos com Geogebra CAS (Elaborado pelo autor)

[1] M(257)=231584178474632390847141970017375815706539969331281128078915168015826259279871

8 SISTEMAS DE NUMERAÇÃO

Este capítulo aborda representações de números na base decimal (10), amplamente utilizada, e nas bases binária (2) e hexadecimal (16), utilizadas no mundo digital. Fazem parte dessa abordagem projetos Scratch para conversões das bases decimal em binário, binário em decimal, decimal em hexadecimal e hexadecimal em decimal. Ao final, são apresentadas formas de conversões de representações entre os sistemas binário e hexadecimal.

A figura 61 mostra os valores possíveis de serem armazenados em 1 *byte*[1] (8 bits)[2], nos sistemas decimal, binário e hexadecimal. Ambos os sistemas são posicionais, ou seja, cada algarismo além do seu valor intrínseco, possui um peso que lhe é atribuído em função da posição que ocupa no número. Os símbolos utilizados em cada sistema e o processo de contagem são:

- **Sistema Binário** - símbolos possíveis: $\{0, 1\}$, com $0 < 1$. Exemplo de contagem no sistema binário: 0, 1, 10, 11, 100, 101, 110, 111, ...
- **Sistema Decimal** - símbolos possíveis: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, com $0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9$. Exemplo de contagem no sistema decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ..., 19, 20, 21, ... 29, 30, 31, ..., 99, 100, 101, ..., 999, 1000, 1001, ...
- **Sistema Hexadecimal** - os símbolos possíveis:$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ com $0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < A < B < C < D < E < F$.
  Os valores na base decimal 10, 11, 12, 13, 14 e 15 correspondem na base hexadecimal, respectivamente, aos símbolos A, B, C, D, E e F. Exemplo de contagem no sistema hexadecimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, ..., 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21, ... 29, 2A, ..., 2F, 30, 31, ..., 99, 9A, ..., 9F, A0, A1, ...

Um número x numa base b será representado na forma $(x)_b$. Assim, $(10)_2 = (2)_{10}$, $(111)_2 = (7)_{10}$, $(1011)_2 = (B)_{16}$, $(A)_{16} = (10)_{10}$, $(D)_{16} = (13)_{10}$, $(F)_{16} = (15)_{10}$.

[1] A palavra *byte* é um acrônimo de *binary term*, ou seja, *byte* é um termo binário
[2] A palavra *bit* é um acrônimo de *binary digit*, ou seja, *bit* é um dígito binário

DEC	BIN	HEX	DEC	BIN	HEX	DEC	BIN	HEX	DEC	BIN	HEX	DEC	BIN	HEX
0	00000000	0	52	00110100	34	103	01100111	67	154	10011010	9A	205	11001101	CD
1	00000001	1	53	00110101	35	104	01101000	68	155	10011011	9B	206	11001110	CE
2	00000010	2	54	00110110	36	105	01101001	69	156	10011100	9C	207	11001111	CF
3	00000011	3	55	00110111	37	106	01101010	6A	157	10011101	9D	208	11010000	D0
4	00000100	4	56	00111000	38	107	01101011	6B	158	10011110	9E	209	11010001	D1
5	00000101	5	57	00111001	39	108	01101100	6C	159	10011111	9F	210	11010010	D2
6	00000110	6	58	00111010	3A	109	01101101	6D	160	10100000	A0	211	11010011	D3
7	00000111	7	59	00111011	3B	110	01101110	6E	161	10100001	A1	212	11010100	D4
8	00001000	8	60	00111100	3C	111	01101111	6F	162	10100010	A2	213	11010101	D5
9	00001001	9	61	00111101	3D	112	01110000	70	163	10100011	A3	214	11010110	D6
10	00001010	A	62	00111110	3E	113	01110001	71	164	10100100	A4	215	11010111	D7
11	00001011	B	63	00111111	3F	114	01110010	72	165	10100101	A5	216	11011000	D8
12	00001100	C	64	01000000	40	115	01110011	73	166	10100110	A6	217	11011001	D9
13	00001101	D	65	01000001	41	116	01110100	74	167	10100111	A7	218	11011010	DA
14	00001110	E	66	01000010	42	117	01110101	75	168	10101000	A8	219	11011011	DB
15	00001111	F	67	01000011	43	118	01110110	76	169	10101001	A9	220	11011100	DC
16	00010000	10	68	01000100	44	119	01110111	77	170	10101010	AA	221	11011101	DD
17	00010001	11	69	01000101	45	120	01111000	78	171	10101011	AB	222	11011110	DE
18	00010010	12	70	01000110	46	121	01111001	79	172	10101100	AC	223	11011111	DF
19	00010011	13	71	01000111	47	122	01111010	7A	173	10101101	AD	224	11100000	E0
20	00010100	14	72	01001000	48	123	01111011	7B	174	10101110	AE	225	11100001	E1
21	00010101	15	73	01001001	49	124	01111100	7C	175	10101111	AF	226	11100010	E2
22	00010110	16	74	01001010	4A	125	01111101	7D	176	10110000	B0	227	11100011	E3
23	00010111	17	75	01001011	4B	126	01111110	7E	177	10110001	B1	228	11100100	E4
24	00011000	18	76	01001100	4C	127	01111111	7F	178	10110010	B2	229	11100101	E5
25	00011001	19	77	01001101	4D	128	10000000	80	179	10110011	B3	230	11100110	E6
26	00011010	1A	78	01001110	4E	129	10000001	81	180	10110100	B4	231	11100111	E7
27	00011011	1B	79	01001111	4F	130	10000010	82	181	10110101	B5	232	11101000	E8
28	00011100	1C	80	01010000	50	131	10000011	83	182	10110110	B6	233	11101001	E9
29	00011101	1D	81	01010001	51	132	10000100	84	183	10110111	B7	234	11101010	EA
30	00011110	1E	82	01010010	52	133	10000101	85	184	10111000	B8	235	11101011	EB
31	00011111	1F	83	01010011	53	134	10000110	86	185	10111001	B9	236	11101100	EC
32	00100000	20	84	01010100	54	135	10000111	87	186	10111010	BA	237	11101101	ED
33	00100001	21	85	01010101	55	136	10001000	88	187	10111011	BB	238	11101110	EE
34	00100010	22	86	01010110	56	137	10001001	89	188	10111100	BC	239	11101111	EF
35	00100011	23	87	01010111	57	138	10001010	8A	189	10111101	BD	240	11110000	F0
36	00100100	24	88	01011000	58	139	10001011	8B	190	10111110	BE	241	11110001	F1
37	00100101	25	89	01011001	59	140	10001100	8C	191	10111111	BF	242	11110010	F2
38	00100110	26	90	01011010	5A	141	10001101	8D	192	11000000	C0	243	11110011	F3
39	00100111	27	91	01011011	5B	142	10001110	8E	193	11000001	C1	244	11110100	F4
40	00101000	28	92	01011100	5C	143	10001111	8F	194	11000010	C2	245	11110101	F5
41	00101001	29	93	01011101	5D	144	10010000	90	195	11000011	C3	246	11110110	F6
42	00101010	2A	94	01011110	5E	145	10010001	91	196	11000100	C4	247	11110111	F7
43	00101011	2B	95	01011111	5F	146	10010010	92	197	11000101	C5	248	11111000	F8
44	00101100	2C	96	01100000	60	147	10010011	93	198	11000110	C6	249	11111001	F9
45	00101101	2D	97	01100001	61	148	10010100	94	199	11000111	C7	250	11111010	FA
46	00101110	2E	98	01100010	62	149	10010101	95	200	11001000	C8	251	11111011	FB
47	00101111	2F	99	01100011	63	150	10010110	96	201	11001001	C9	252	11111100	FC
48	00110000	30	100	01100100	64	151	10010111	97	202	11001010	CA	253	11111101	FD
49	00110001	31	101	01100101	65	152	10011000	98	203	11001011	CB	254	11111110	FE
50	00110010	32	102	01100110	66	153	10011001	99	204	11001100	CC	255	11111111	FF
51	00110011	33												

Figura 61 – Valores possíveis para um byte nos sistemas Decimal, Binário e Hexadecimal

TEOREMA - sejam dados os números inteiros n e b, com $x > 0$ e $b > 1$. Existem números inteiros $n \geq 0$ e $0 \leq r_0, r_1, \cdots, r_n < b$, com $r_n \neq 0$, univocamente determinados, tais que: $x = r_0 + r_1 \cdot b^1 + r_2 \cdot b^2 + \cdots + r_n \cdot b^n$ (HEFEZ, 2016, p. 59)

COROLÁRIO - Do teorema acima temos que a representação de x na base b será:
$(x)_b = (r_n\ r_{n-1}\ \cdots\ r_3\ r_2\ r_1\ r_0)_b$

Exemplos:
$(1506)_{10} = 6 \cdot 10^0 + 0 \cdot 10^1 + 5 \cdot 10^2 + 1 \cdot 10^3$
$(1001101)_2 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 0 \cdot 2^5 + 1 \cdot 2^6 = (77)_{10}$
$(4D)_{16} = 13 \cdot 16^0 + 4 \cdot 16^1 = (77)_{10}$

Assim, os métodos de conversões binário para decimal e hexadecimal para decimal estão definidos.

Já as conversões da base decimal para as bases binária e hexadecimal utilizarão divisões sucessivas pela base b (2 ou 16), iniciando com x como dividendo e seguindo com os quocientes, até que um dado quociente do processo seja igual a 0, conforme figura 62. A representação de x na base b será a sequência de restos $(x)_{10} = (r_n r_{n-1} \cdots r_2 r_1 r_0)_b$.

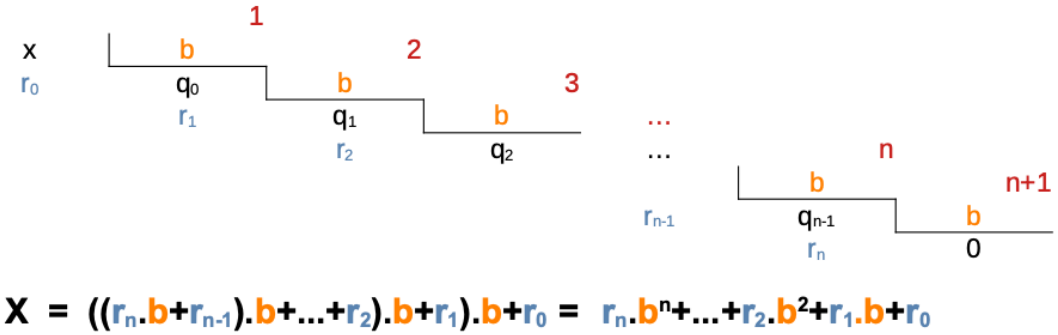

Figura 62 – Conversão de n para base b (Elaborado pelo autor)

A equação 8.1 mostra termo a termo do processo iterativo de divisões sucessivas para a representação de x numa base b:

$$\begin{aligned} X &= q_0 \cdot b + r_0 \\ q_0 &= q_1 \cdot b + r_1 \\ q_1 &= q_2 \cdot b + r_2 \\ &\cdots \\ q_{n-2} &= q_{n-1} \cdot b + r_{n-1} \\ q_{n-1} &= 0 \cdot b + r_n \end{aligned} \tag{8.1}$$

A equação 8.2 mostra a expressão expandida para a representação de x numa base b:

$$X = ((((\cdots(\overbrace{\overbrace{0 \cdot b + r_n}^{q_{n-1}} \cdot b + r_{n-1}}^{q_{n-2}}) \cdot b + \cdots) \cdot b + r_2) \cdot b + r_1) \cdot b + r_0) \tag{8.2}$$

q_0 (...) q_1 X

8.1 Conversão Inteiro Base Decimal → Binária

Algoritmo 26: Conversão de número na base decimal para base binária

Var n, i, j, Dec: inteiro; Bin[1..32] de inteiro
Inicio
  Saída: "Entre com valor Decimal para conversão em Binário?"
  Entrada: n
  $Dec \leftarrow n$
  $i \leftarrow 0$
  `// Gera Bin:  lista de restos da divisão por 2`
  repita
    $i \leftarrow i + 1$
    $Bin[i] \leftarrow n \; mod \; 2$
    $n \leftarrow n \; div \; 2$
  até $n = 0$;
  Saída: "O valor decimal ", Dec, " corresponde ao valor binário: "
  `// Exibe Bin em ordem inversa`
  para $(j \leftarrow i;\; j \geq 1;\; j \leftarrow j - 1)$ **faça**
    Saída: Bin[j]
  fim para
Fim

O algortimo 26 implementa a conversão de um número n na representação base decimal para a base binária. A conversão da base decimal para a base binária utiliza divisões sucessivas pela base 2, iniciando com n como dividendo e repetindo as divisões com os quocientes, até que um dado quociente do processo seja igual a 0. Nesse processo, os restos são guardados na lista Bin. Ressaltando que o primeiro índice da lista Bin tem valor 1. A representação de n na base 2 é a sequência de restos $(n)_{10} = (Bin_i Bin_{i-1} \cdots Bin_2 Bin_1)_2$.

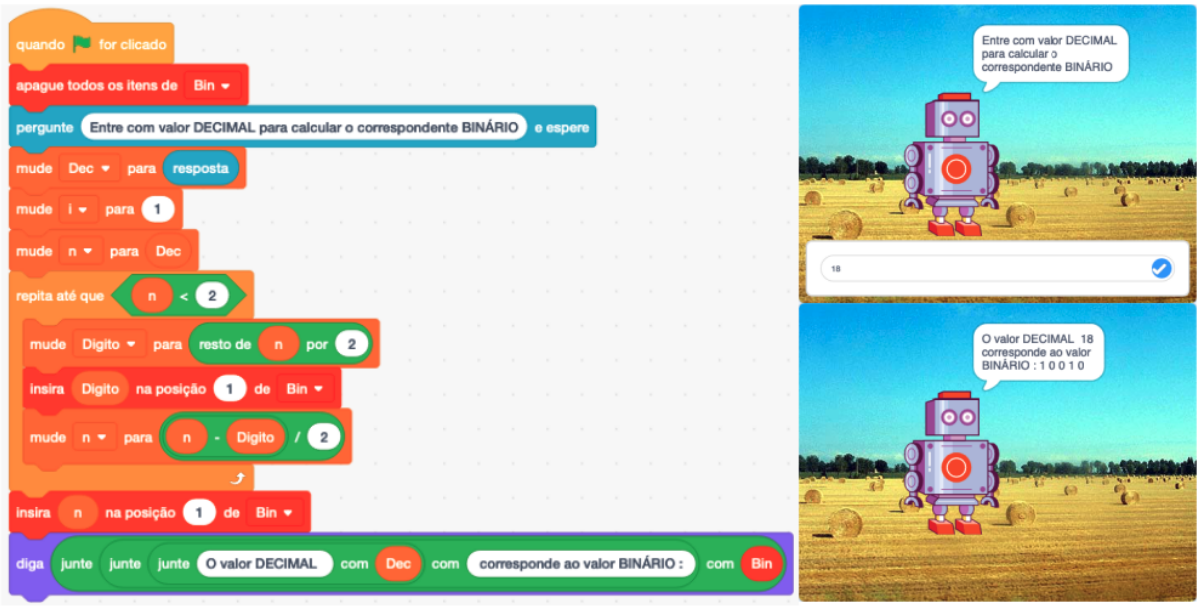

Figura 63 – Conversão de um valor Decimal para Binário em Scratch (Elaborado pelo autor)

A figura 63 mostra a implementação do algoritmo 26 em Scratch. Observe que, como em Scratch não existe a operação divisão inteira, o cálculo de n para nova iteração foi realizado com o ajuste $n \leftarrow (n - (n\%2))/2$ para uma divisão exata.

Este projeto está disponível em: https://scratch.mit.edu/projects/544758196/

8.2 Conversão Inteiro Base Binária $\rightarrow$ Decimal

Algoritmo 27: Conversão de número na base binária para base decimal

Var i, Dec: inteiro; Bin: cadeia
Inicio
Saída: "Entre com valor Binário para conversão em Decimal?"
Entrada: Bin
$Dec \leftarrow 0$
// $Dec = \sum_{i=1}^{n} Bin[i] * 2^{n-i}, \quad n = Tamanho(Bin)$
para $(i \leftarrow 1;\ i \leq tamanho(Bin);\ i \leftarrow i + 1)$ **faça**
$Dec \leftarrow Dec * 2 + caractereparainteiro(Bin[i])$
fim para
// Exibe Valor Decimal Dec
Saída: "O valor binário ", Bin, " corresponde ao valor decimal: ", Dec
Fim

O algortimo 27 implementa a conversão de um número na base binária para a base decimal. Inicialmente, a representação do número binário é lida na cadeia Bin. Em seguida, a estrutura de repetição percorre cada dígito binário fazendo somas parciais na variável Dec, conforme equação 8.2. Ao final, as representações que estão nas variáveis Bin e Dec são exibidas.

A figura 64 mostra a implementação do algoritmo 27 em Scratch.

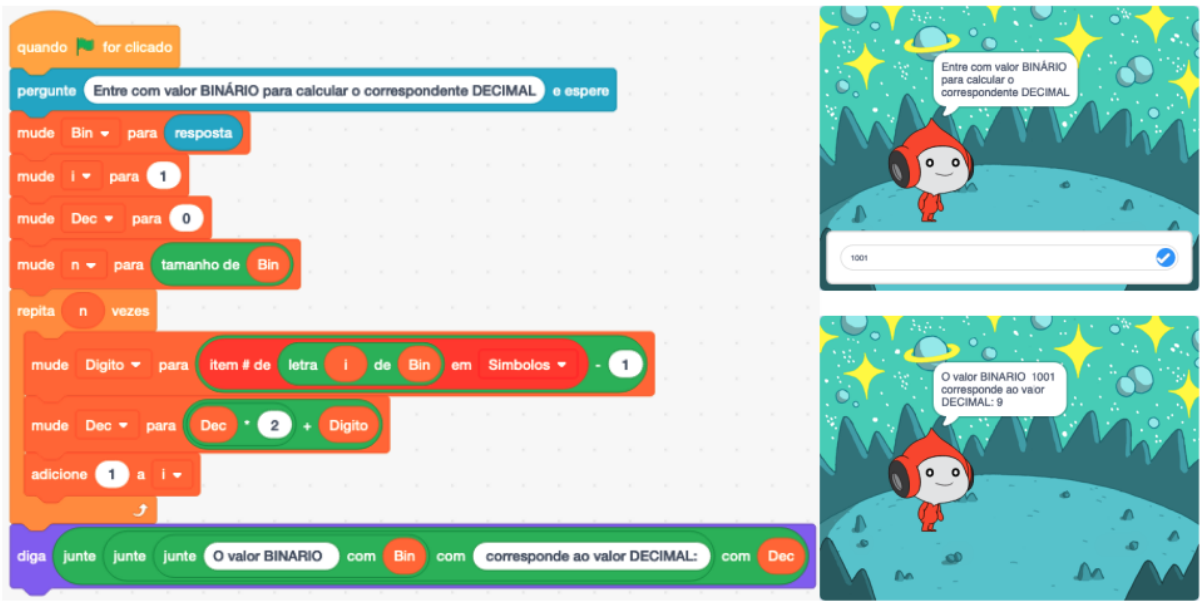

Figura 64 – Conversão de um valor Binário para Decimal em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/538630790/

8.3 Conversão Inteiro Base Decimal → Hexadecimal

Algoritmo 28: Conversão de número na base decimal para base hexadecimal

Var Dec, n: inteiro;
$Hexa$: $cadeia$
$Simbolo$={“0”,“1”,“2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “A”, “B”, “C”, “D”, “E”, “F”}
Inicio
Saída: “Entre com valor Decimal para conversão em Hexadecimal?”
Entrada: n
$Dec \leftarrow n$
// Inicia $Hexa$ com cadeia vazia
$Hexa \leftarrow$ “”
// Gera $Hexa$: lista de restos da divisão por 16
repita
// Concatena o dígito no início da cadeia $Hexa$
$Hexa \leftarrow Simbolo[(n \ mod \ 16) + 1] + Hexa$
$n \leftarrow n \ div \ 16$
até $n = 0$;
Saída: “O valor decimal ”, Dec, “ corresponde ao hexadecimal:”, Hexa
Fim

O algortimo 28 implementa a conversão de um número n na representação base decimal para a base hexadecimal.A conversão da base decimal para a base hexadecimal utiliza divisões sucessivas pela base 16, iniciando com n como dividendo e repetindo as divisões com os quocientes, até que um dado quociente do processo seja igual a 0. Nesse processo, os restos foram convertidos em caracteres e concatenados no início da cadeia Hexa que armazena a representação, ou seja, os restos das divisões já são agrupados em ordem inversa - do último para o primeiro.

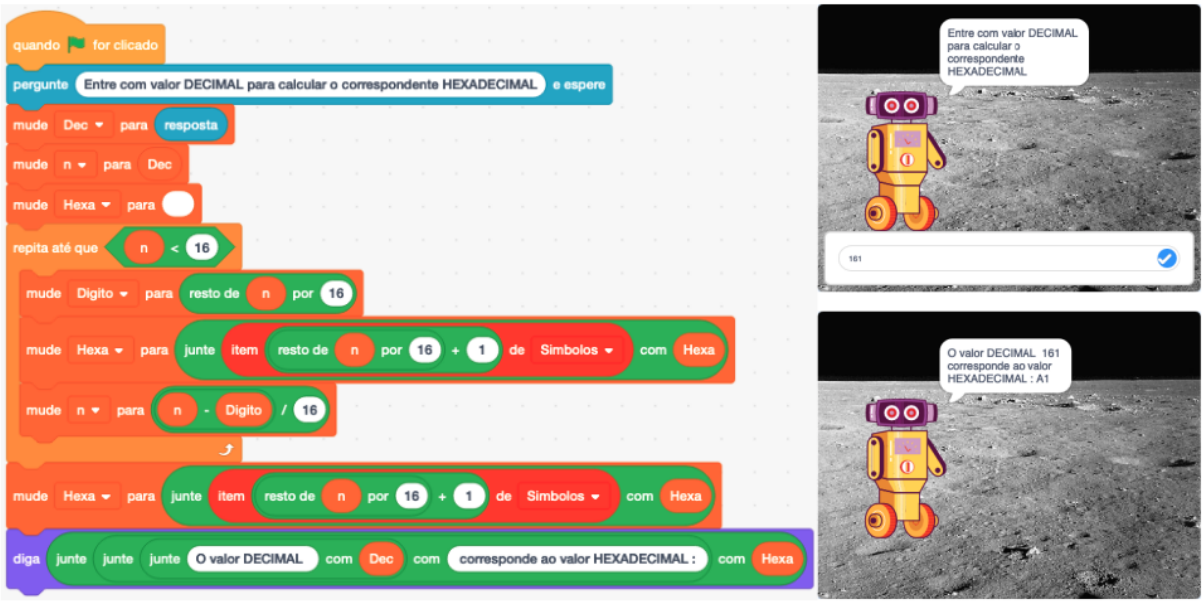

Figura 65 – Conversão valor Decimal para Hexadecimal em Scratch (Elaborado pelo autor)

A figura 65 mostra a implementação do algoritmo 28 em Scratch. Observe que, como em Scratch não existe a operação divisão inteira, o cálculo de n para nova iteração foi realizado com o ajuste $n \leftarrow (n - (n\%16))/16$ para uma divisão exata. A lista Simbolos, que não aparece na figura, tem valores $Simbolo$={“0”,“1”,“2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “A”, “B”, “C”, “D”, “E”, “F”} como definido no algoritmo.

Este projeto está disponível em: https://scratch.mit.edu/projects/551222610/

8.4 Conversão Inteiro Base Hexadecimal → Decimal

O algortimo 29 implementa a conversão de um número na base hexadecimal para a base decimal. Inicialmente, a representação do número hexadecimal é lida na cadeia Hexa. Em seguida, a estrutura de repetição percorre cada dígito hexadecimal fazendo somas parciais na variável Dec, conforme equação 8.2. Ao final, as representações que estão nas variáveis Hexa e Dec são exibidas.

Algoritmo 29: Conversão de número na base hexadecimal para base decimal

Var $Dec, n : inteiro; Hexa : cadeia$

$Simbolo$={"0","1","2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F"}

Inicio

Saída: "Entre com valor Hexadecimal para conversão em Decimal?"

Entrada: $Hexa$

$Dec \leftarrow 0$

// $Dec = \sum_{i=1}^{n} Hexa[i] * 16^{n-i}, \quad n = Tamanho(Hexa)$

para $(i \leftarrow 1;\ i \leq tamanho(Hexa);\ i \leftarrow i + 1)$ **faça**

$Dec \leftarrow Dec * 16 + Posicao(Hexa[i], Simbolo) - 1$

fim para

// Exibe Valor Decimal Dec

Saída: "O valor Hexadecimal ", $Hexa$, " corresponde ao valor decimal: ", Dec

Fim

A figura 66 mostra a implementação do algoritmo 29 em Scratch.

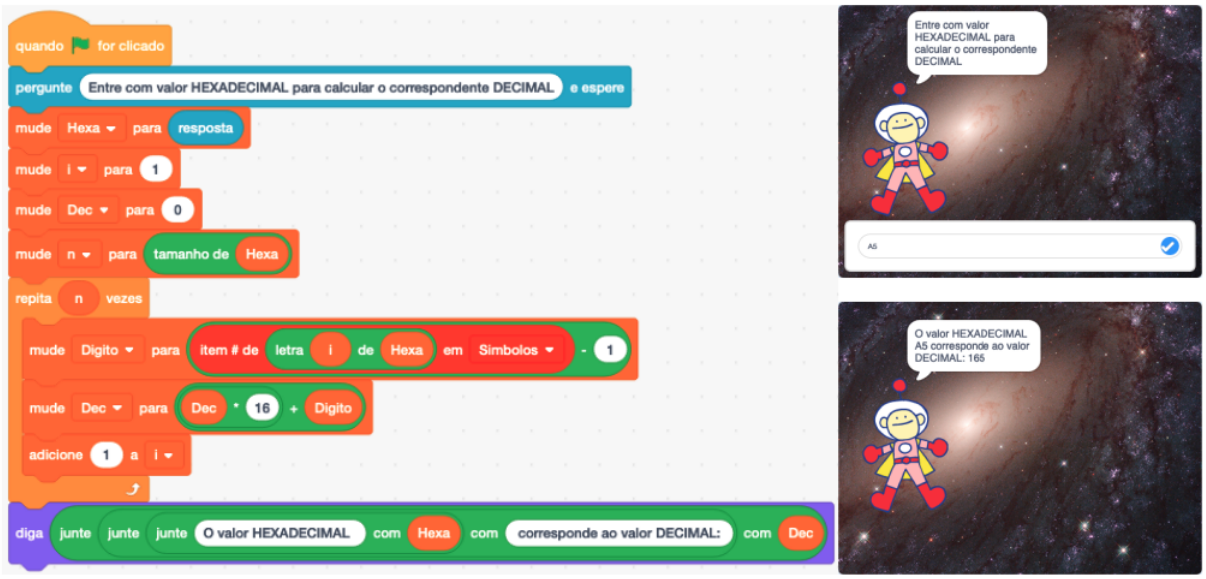

Figura 66 – Conversão valor Hexadecimal para Decimal em Scratch (Elaborado pelo autor)

Este projeto está disponível em: https://scratch.mit.edu/projects/545102468/

Conversões Binário $\leftrightarrow$ Hexadecimal

O processo de conversão binário para hexadecimal pode ser realizado adotando o critério: cada grupo de 4 bits, iniciando os agrupamentos da direita para esquerda, irá gerar um dígito hexadecimal.

Exemplos:

$$(01110001)_2 = \overbrace{0|1|1|1}^{7} \quad \overbrace{0|0|0|1}^{1} = (71)_{16} \qquad (CB)_2 = \overbrace{1|1|0|0}^{C} \quad \overbrace{1|0|1|1}^{B} = (CB)_{16}$$

De forma inversa, no processo de conversão hexadecimal para binário, cada dígito hexadecimal deve ser convertido em 4 dígitos binários (bits).
Exemplos:

$$(A2)_{16} = \overbrace{1|0|1|0}^{A} \quad \overbrace{0^{1}|0|1|0}^{2} = (10100010)_2 \qquad (60)_{16} = \overbrace{0|1|1|0}^{6} \quad \overbrace{0^{1}|0|0|0}^{0} = (01100000)_2$$

[1] Ressalta-se que os zeros a esquerda devem ser representados pois no encadeamento dos bits farão diferença na representação.

CONSIDERAÇÕES FINAIS

Este trabalho apresentou propostas para utilizar programas de computadores para ensino e aprendizagem da matemática desenvolvendo o pensamento computacional. As modalidades propostas foram “Laboratório de Matemática”, “Jogos e Gamificação” e “Construção de Algoritmos e Programação”. A utilização de tecnologias no ensino está fundamentada no documento Base Nacional Comum Curricular - BNCC que nortea as propostas pedagógicas de todas as escolas públicas e privadas de Educação Infantil, Ensino Fundamental e Ensino Médio, em todo Brasil.

O *software* Geogebra foi utilizado nas modalidades Geogebra Calculadora Gráfica para estudo do comportamento de uma função quadrática, Geogebra Geometria para estudo de polígonos regulares circuncêntricos e Geogebra CAS (*Computer Algebra System*) - para fatorar alguns Números de Fermat e Mersenne. Scratch foi utilizado em construções Geométricas seguindo a proposta Construcionista de Papert, de forma similar à Geometria da Tartaruga da linguagem LOGO. A Lógica Matemática desenvolvida por Boole e De Morgan foi abordada com Diagramas de Venn e teve importância destacada para construções de expressões das estruturas de controle condicionais e de repetições que controlam fluxos em algoritmos e programas. O Triângulo de Pascal foi utilizado como elemento matemático motivador para exploração de sequências, dentre elas: soma dos naturais e de Fibonacci que foram desenvolvidas computacionalmente nas formas iterativas e recursivas. Divisibilidade, números primos e compostos, Crivo de Eratóstenes, Algoritmo de Euclides para cálculo do Máximo Divisor Comum - MDC, Teorema Fundamental da Aritmética e Fatoração, Sistemas de Numeração nas Bases Binária, Decimal e Hexadecimal foram alguns dos diversos algoritmos discutidos e implementados em Scratch, Portugol e Python.

Também foi demonstrado ser possível mitigar barreiras de uso de tecnologias utilizando os *softwares* Geogebra, Scratch, Portugol e Python de uso gratuito, em idioma português, e com versões *online* para navegadores internet que dispensam instalações e configurações. À exceção do Python que trata-se de uma linguagem de programação universal, e portanto que tem instruções em idioma inglês, mas que possui manual em português.

Pelas razões apresentadas, pode ser concluído que no ensino da matemática na educação básica é possível utilizar Tecnologias para apresentar conceitos e estimular o aprendizado, assim como desenvolver no aluno o pensamento computacional.

Trabalhos futuros a serem realizados: Laboratório de Matemática - concepção de livro e cursos em diversos níveis abrangendo os recursos do *software* Geogebra. Na área de Programação - lançamento de cursos de programação Scratch e Python. Jogos e Gameficação - desenvolvimento de jogos com foco no Ensino e Aprendizagem da Matemática.

ÍNDICE REMISSIVO

Álgebra, 46

Algoritmo, 8, 29
Aprendizagem, 7
ASCII, 31

Binário, Sistema, 88
Binet
    Fórmula, 61
BNCC, 4
Boole, George, 46
Booleano, Tipo, 32

Cadeia, Tipo, 32
Caractere, Tipo, 31
CAS, Geogebra, 87
Complexidade Computacional, 70
Composto, Número, 77
Computação
    Algébrica, 30
    Numérica, 30
    Simbólica, 30
Conjuntos, 46, 53
Construcionismo, 22
Crivo, Eratóstenes, 79

Dados, Estruturas de, 30
Dados, Tipos Simples de, 30
De Morgan
    Augustus, 46
    Leis, 51
Decimal, Sistema, 88
Descartes, René, 52
Divisor, 73

EBCDIC, 31
Estrururas de Controle Condicional, 38
Estruturas de Controle de Repetição, 40
Euclides
    Algoritmo, 74
Euler, Leonhard, 46

Fatorial, 62
Fermat
    Último Teorema de, 58
    Números, 85
    Pequeno Teorema de, 84
    Pierre de, 52
Fibonacci
    Leonardo de Pisa, 58
    Sequência de, 58
Função, 44

Gameficação, 6
Gauss
    Carl Friedrich, 55
Geogebra, 11
    Calculadora Gráfica - Equação 2^o grau, 13
Geometria da Tartaruga, 21
GPIMEM, 5

Hanoi, Torre, 71
Hexadecimal, Sistema, 88

Inteiro, Tipo, 31

Jogos, 6
Jogos, aprendizagem baseada, 6

Lógica Matemática, 46
Lógica Simbólica, 47
LDB, Lei de Diretrizes e Base, 4
Lista, Tipo, 32
LOGO, 10

Matemática, Laboratório, 5
MDC, 74

Mersenne
    Marin, 52, 85
    Números, 85
MIT, 10
MMC, Mínimo Múltiplo Comum, 79

Número Áureo, 61

OBI, Olimpíada Brasileira de Informática, 8
Operador
    Booleano, 47
    Conjunção E, 48
    Disjunção OU, 47
    Negação NAO, 50

Papert, Seymour, 10
Pascal
    Triângulo com conjuntos, 53
    Triângulo de, 52
Pascal, Blaise, 52
Peano, Giuseppe, 55
Perfeito, Número, 79
Pitágoras
    Ternos Pitagóricos, 58
Polígono Regular, 26
Portugol, 10
Primo, Número, 77
Primos Gêmeos, 86
Principe dos Matemáticos, 56
Procedimento, 44
Programação, 8
Python, 10

Real, Tipo, 31
Recorrência
    de primeira ordem, 62
    de Segunda Ordem, 63
Recursividade, 64

SBC, 8
Scratch, 16
Scratch, Projeto, 17
Soma dos Naturais, 62
Stifel, 54
    Michael, 54
    Relação de, 54

Teorema da Fatoração Única, 82
TFA, Teorema Fundamental da Aritmética, 82
TICs,Tecnologias Digitais, 5

UNESP, 5
UNICAMP, 8
UTF-8, 31

Venn
    Diagrama, 46
    John, 46

REFERÊNCIAS

ABREU, M. D. P. d. *Laboratório de Matemática: Um Espaço para a Formação Continuada do Professor*. Santa Maria: UFSM, 1997. 06 p.

ALMEIDA, F. J. d. e. M. E. B. d. A. *Aprender construindo - A informática se transformando com os professores*. Brasil: BRASIL, 2000. Acessado em 27 de junho de 2021. Disponível em: <http://www.dominiopublico.gov.br/pesquisa/DetalheObraForm.do?select_action=&co_obra=40248>.

ALVES, F. *Gamification: como criar experiências de aprendizagem engajadoras: guia completo: do conceito à prática*. 2. ed. São Paulo: DVS Editora, 2015. ISBN 978-85-8289-102-5.

ANDRADE, L. N. d. *Breve considerações sobre a utilidade dos sistemas de computação algébrica*. Rio de Janeiro: SBM, RPM83. Acessado em 27 de junho de 2021. Disponível em: <https://rpm.org.br/cdrpm/83/1.html>.

BORBA, M. d. C. M. G. P. *Informática e educação matemática*. 6. ed. Belo Horizonte: Autêntica Editora, 2019. (Coleção Tendências em Educação Matemática). ISBN 978-85-513-0661-1.

BOYER, C. B. U. C. M. *História da matemática*. 3. ed. São Paulo: Blucher, 2012. ISBN 978-85-212-0641-5.

BRASIL. *Lei de Diretrizes e Bases da Educação Nacional*. Brasil: BRASIL, 1996. Acessado em 10 de junho de 2021. Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/l9394.htm>.

BRASIL. *Base Nacional Comum Curricular: Educação Infantil e Ensino Fundamental*. Brasil: BRASIL, 2018. Acessado em 10 de junho de 2021. Disponível em: <http://basenacionalcomum.mec.gov.br>.

CAMPOS, F. R. *Paulo Freire e Seymour Papert: Educação, tecnologias e análise do discurso*. 1. ed. Curitiba: Editora CRV, 2013. 146 p. ISBN 978-85-8042-627-4.

COELHO, R. M. *Introdução à Lógica Matemática*. Vitória/ES: [s.n.], 2014. ISBN 978-85-920580-0-5.

CORMEN, T. H. e. a. *Algoritmos: teoria e prática*. Rio de Janeiro: Elsevier, 2002. ISBN 85-352-0926-3.

COUTINHO, S. C. *Números Inteiros e Criptografia RSA*. 2. ed. Rio de Janeiro: IMPA, 2011. 226 p. (Coleção Matemática e Aplicações). ISBN 978-85-244-0124-9.

ESTEVES, A. M. d. S. e. L. F. N. *Portugol Studio: Em direção a uma comunidade aberta para pesquisa sobre o aprendizado de programação*. Itajaí-SC: SBC, 2019. Acessado em 27 de junho de 2021. Disponível em: <https://sol.sbc.org.br/index.php/wei/article/download/6656/6552/>.

EUCLIDES. *Os elementos/Euclides; tradução e introdução de Irineu Bicudo*. São Paulo: Editora UNESP, 2009. ISBN 978-85-7139-935-8.

GARBI, G. G. *A Rainha das Ciências: um passeio pelo maravilhoso mundo da matemática*. 5. ed. São Paulo: Editora Livraria da Física, 2010. ISBN 978-85-88325-61-6.

GEOGEBRA. *Geogebra Aplicativo*. Geogebra, 2021. Acessado em 27 de junho de 2021. Disponível em: <https://www.geogebra.org/about>.

HEFEZ, A. *Aritmética*. 1. ed. Rio de Janeiro: SBM, 2016. 298 p. (Coleção PROFMAT). ISBN 978-85-8337-105-2.

HUIZINGA, J. *Homo ludens: o jogo como elemento da cultura*. 8. ed. São Paulo: Perspectiva, 2014. ISBN 978-85-273-0075-9.

LIMA, E. L. *Números e Funções Reais*. 1. ed. Rio de Janeiro: SBM, 2013. 297 p. (Coleção PROFMAT). ISBN 978-85-85818-81-4.

LUCENA, R. d. S. *Laboratório de Ensino de Matemática*. Fortaleza: UAB/IFCE, 2017. 94 p. Acessado em 27 de junho de 2021. ISBN 978-85-475-0058-0. Disponível em: <https://educapes.capes.gov.br/handle/capes/429642>.

MARJI, M. *Aprenda a programar com SCRATCH: uma introdução visual à programação com jogos, arte, ciência e matemática*. 1. ed. São Paulo: Novatec Editora, 2014. ISBN 978-85-7522-312-3.

MCGONIGAL, J. *A realidade em jogo*. 1. ed. Rio de Janeiro: BestSeller, 2012. ISBN 978-85-7684-522-5.

MOREIRA, C. G. e. F. E. B. M. *Primos Gêmeos, Primos de Sophie Germain e o Teorema de Brun - Revista Matemática Universitária ns.48/49, junho/dezembro de 2010*. Rio de Janeiro: SBM-RMU, 2010. Acessado em 27 de junho de 2021. Disponível em: <https://rmu.sbm.org.br/wp-content/uploads/sites/27/2018/03/n48_n49_Artigo06.pdf>.

MORGADO, A. C. C. P. C. P. *Matemática Discreta*. 6. ed. Rio de Janeiro: SBM, 2015. 294 p. (Coleção PROFMAT). ISBN 978-85-8337-034-5.

NETO, A. C. M. *Geometria*. Rio de Janeiro: SBM, 2013. 442 p. (Coleção PROFMAT). ISBN 978-85-85818-93-7.

PAPERT, S. *Logo: computadores e educação*. 3. ed. São Paulo: Editora Brasiliense, 1988. ISBN 85-11-27001-9.

PRENSKY, M. *Aprendizagem baseada em jogos digitais*. 1. ed. São Paulo: Editora Senac, 2012. ISBN 978-85-396-0271-1.

SBC, S. B. d. C. a. *XXIII Olimpíada Brasileira de Informática - OBI2021*. São Paulo: SBC, 2021. Acessado em 27 de junho de 2021. Disponível em: <https://olimpiada.ic.unicamp.br/info/>.

SINGH, S. *O Último Teorema de Fermat: a história do enigma que confundiu as mais brilhantes mentes do mundo durante 358 anos*. 4. ed. Rio de Janeiro: BestBolso, 2019. (Edições BestBolso). ISBN 978-85-7799-428-1.

UNESP, G. *Grupo de Pesquisa em Informática, outras Mídias e Educação Matemática*. São Paulo: UNESP, 2021. Acessado em 27 de junho de 2021. Disponível em: <https://igce.rc.unesp.br/#!/pesquisa/gpimem---pesq-em-informatica-outras-midias-e-educacao-matematica/gpimem/>.

UNIVALI. *Portugol Studio*. Itajaí SC: UNIVALI, 2021. Acessado em 27 de junho de 2021. Disponível em: <http://lite.acad.univali.br/pt/portugol-studio/>.

A APÊNDICE PORTUGOL

O Portugol é uma linguagem de programação para fins didáticos que se assemelha à linguagem C. A ideia é facilitar a construção de programas sem a barreira do idioma, ou seja, elaborar programas em português. Este apêndice **não abrangerá todos os recursos do Portugol**, tendo apenas definições de elementos utilizados no corpo da dissertação. O manual do Portugol está disponível em: https://portugol-webstudio.cubos.io/ide/ajuda#

A.1 Ambiente

O Portugol Studio é um ambiente para aprender a programar, voltado para os iniciantes em programação que falam o idioma português. Possui uma sintaxe fácil, diversos exemplos e materiais de apoio à aprendizagem. Também possibilita criação de jogos e outras aplicações. (UNIVALI, 2021)

O Portugol Studio pode ser utilizado diretamente da internet, via navegador, sem a necessidade de instalação no computador, pelo endereço: https://portugol-webstudio.cubos.io/. Entretanto, caso seja do interesse, existem versões *Desktop*[1] para os sistemas operacionais Windows, macOS e Linux, ou seja, versões para serem instaladas e executadas na máquina do usuário sem conexão com a internet. Por questões de simplicidade e facilidade neste trabalho foi utilizada a versão da internet (*on-line*).

Figura 67 – Portugol Studio Principal

[1] http://lite.acad.univali.br/portugol/

A figura 67 exibe a tela de abertura do Portugol Studio. As opções **Novo Arquivo** (1) para criar um novo programa, **Abrir Arquivo** (2) para abrir um arquivo armazenado no computador do usuário, **Ajuda** (3) para acessar o manual do Portugol e **Exemplos** (4) para explorar exemplos de programas estão disponíveis na tela principal.

Figura 68 – Portugol Studio Edição

A figura 68 exibe a tela de edição de um programa no Portugol Studio. A aba (1) acessa a tela principal do Portugol Studio e a aba (2) acessa o programa em edição, no caso "DivisoresDeNPortugol.por". O Menu lateral possui botões com as funções: (3) Executar o programa, (4) Parar a execução do programa, (5) Gravar o programa no computador do usuário, (6) Abrir um programa que está armazenado no computador do usuário, (7) Criar um *link* de compartilhamento do programa editado, (8) Acessar o manual do Portugol, (9) Área de edição do programa, (10) Tela de execução do programa, onde acontecem as entradas e saídas de dados.

A.2 Tipos de Dados, Variáveis e Operações

Esta seção conecta conceitos abordados na seção 3.1 com definições do Portugol. Os tipos simples de dados dsponíveis no Portugol são: Inteiro, Real, Caracter, Lógico e Vazio. As estruturas de dados disponíveis são: Cadeia e Vetor[1].

Uma variável do tipo inteiro é declaração na forma: **inteiro** *nome_da_variavel*. Os **operadores aritméticos** dos inteiros são: adição (+), subtração (-), multiplicação (*), divisão inteira

[1]Também Matrizes que são Vetores multidimensionais

(/) e resto da divisão inteira (%). **Operadores relacionais** são: maior que (>), maior que ou igual a (>=), menor que (<), menor que ou igual a (<=), igualdade (==), desigualdade (! =).

Uma variável do tipo real é declaração na forma: **real** *nome_da_variavel*. Os **operadores aritméticos** dos reais são: adição (+), subtração (-), multiplicação (*) e divisão / (/). **Operadores relacionais** são: maior que (>), maior que ou igual a (>=), menor que (<), menor que ou igual a (<=), igualdade (==), desigualdade (! =).

Uma variável do tipo lógico é declaração na forma: **logico** *nome_da_variavel*. **Operadores booleanos** são: conjunção (e), disjunção (ou) e negação (nao).

A.3 Interação e Manipulação de Dados

Em Portugol,

- Entrada de dados é realizada com o comando: leia(<listadevariaveis>)
- Saída de dados é realizada com o comando: escreva(<listadevariaveisouliterais>)
- Atribuições são realizadas com o comando: <variavel> = <expressao>. O sinal de igual pode ser substituído por +=, -=, *=, /= que correspondem a uma auto operação. Por exemplo: $i+ = 1$ corresponde a $i = i + 1$.

Veja exemplos em A.6 e A.7.

A.4 Estrutura de Controle Condicional

A estrutura de controle condicional possibilita condicionar a execução de comandos a um valor lógico ($falso$ ou $verdadeiro$).

Estrutura de Controle Condicional Simples - Se/Então

O conjunto de comandos <$comandos$> será executado caso, no momento do processamento, <$condicao$> resulte $verdadeiro$. Para um resultado $falso$, nada será processado. Veja exemplo em A.8.

Sintaxe:

se (<*condicao*>) {
    <*comandos*>
}

Estrutura de Controle Condicional Composta - Se/Então/Senão

Apresenta dois conjuntos de comandos, mutuamente exclusivos. Os comandos <$comandos1$> serão executados caso a expressão <$condicao$> tenha valor $verdadeiro$ no momento da execução. Caso seja $falso$, os comandos <$comandos2$> que serão executados. Veja exemplo em A.9.

Sintaxe:

se (<*condicao*>) {
    <*comandos1*>
}
senao {
    <*comandos2*>
}

A.5 Estruturas de Controle de Repetições

As estruturas de controle de repetição servem para que um conjunto de instruções seja repetido de acordo com alguma lógica de controle. Em Portugol existem três estruturas de controle de repetições: com pré-teste (teste antes do bloco a ser repetido); com pós-teste (teste posterior ao bloco a ser repetido); com variável de controle.

Estrutura de Repetição - Enquanto

Estrutura de controle de repetição com pré-teste. Inicialmente <$condicao$> é testada. Sendo $verdadeiro$, <$comandos$> são processados e um novo teste realizado. Caso resulte em $falso$, as repetições são encerradas. Veja exemplo em A.10.

Sintaxe:

enquanto (<*condicao*>) {
    <*comandos*>
}

Estrutura de Repetição - Faça ... Enquanto

Estrutura de controle de repetição com pos-teste. Inicialmente <$comandos$> são processados e um teste de <$condicao$> é realizado. Sendo $verdadeiro$, nova iteração é processada. Caso resulte em $falso$, as repetições são encerradas. Veja exemplo em A.12

Sintaxe:

faca {
    <*comandos*>
} enquanto (<*condicao*>)

Estrutura de Repetição - Para

Estrutura de controle de repetição com variável de controle. Inicialmente, executa <*exp1*>, realiza o teste de <*exp2*> e encerra se for $falso$. Caso seja $verdadeiro$, processa <comandos> e <*exp3*> para realizar um novo teste de <exp2>. Veja exemplo em A.13.

Sintaxe:

para (<*exp1*>;<*exp2*>;<*exp3*>) {
    <*comandos*>
}

A.6 Algoritmo 1 em Portugol - Entrada e Saída

A figura 69 mostra o algoritmo 1 codificado em Portugol.

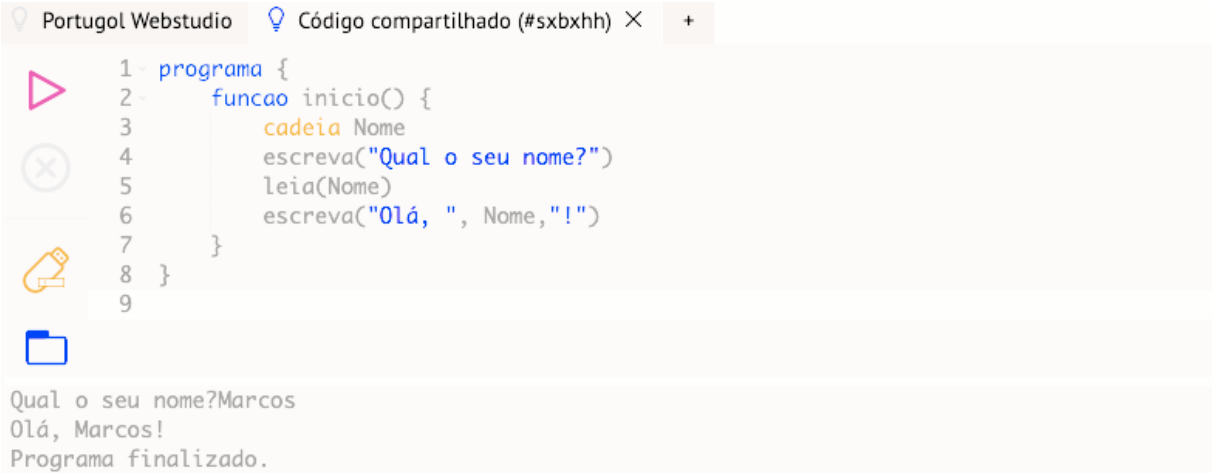

Figura 69 – Entrada e Saida em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=sxbxhh

A.7 Algoritmo 2 em Portugol - Atribuição a variável

A figura 70 mostra o algoritmo 2 codificado em Portugol.

Figura 70 – Preco Com Desconto em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=ommmhv

A.8 Algoritmo 3 em Portugol - Se/Entao

A figura 71 mostra o algoritmo 3 codificado em Portugol.

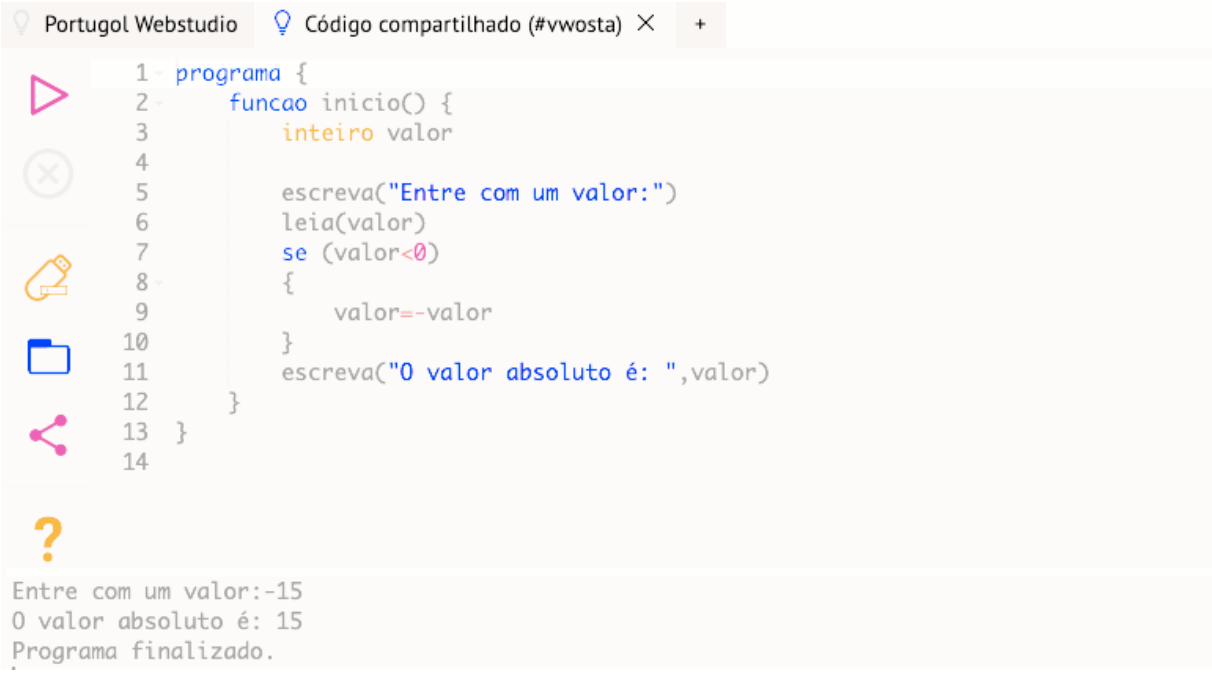

Figura 71 – Controle Condicional Simples em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=vwosta

A.9 Algoritmo 4 em Portugol - Se/Entao/Senao

A figura 72 mostra o algoritmo 4 codificado em Portugol.

Figura 72 – Controle Condicional Composta em Portugol (Elaborado pelo autor)

A expressão $(nao(nota < 7))$ pode ser substituída por $(nota >= 7)$.
Está disponível em: https://portugol-webstudio.cubos.io/ide#share=phljdg

A.10 Algoritmo 5 em Portugol - Enquanto: 1 a 10

A figura 73 mostra o algoritmo 5 codificado em Portugol. A expressão $i+=1$ é equivalente a $i = i + 1$.

Figura 73 – Exibe 1 a 10 com Enquanto em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=frdmsl

A.11 Algoritmo 6 em Portugol - Enquanto: 10 a 1

A figura 74 mostra o algoritmo 6 codificado em Portugol. A expressão $i-=1$ é equivalente a $i = i - 1$.

Figura 74 – Exibe 10 a 1 com Enquanto em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=gnynjl

A.12 Algoritmo 7 em Portugol - Repita até 0 (Par ou ímpar)

A figura 75 mostra o algoritmo 7 codificado em Portugol.

Figura 75 – Par ou Impar com repetição até 0 em Portugol (Elaborado pelo autor)

Foi utilizada $faca$ <$comandos$> $enquanto$ (<$condicao$>) que é a estrutura de repetição com pós teste em Portugol. A adequação da condição de finalização utilizada no algortimo foi realizada pela inversão da expressão. Ainda, o operador % calcula o resto da divisão inteira e os operadores relacionais == e != testam a igualdade e desigualdade, respectivamente.

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=gvaelg

A.13 Algoritmo 8 em Portugol - Para: Pares de 2 a 20

A figura 76 mostra o algoritmo 8 codificado em Portugol.

Figura 76 – Pares de 2 a 20 com Para em Portugol (Elaborado pelo autor)

O valor inicial da variável i é 2, o incremento é $+2$ e a condição para repetição é o valor de i ser menor ou igual a 20.
Está disponível em: https://portugol-webstudio.cubos.io/ide#share=jwakzn

A.14 Algoritmo 9 em Portugol - Repita até que: 1 a 10

A figura 77 mostra o algoritmo 9 codificado em Portugol.

Figura 77 – Exibição de 1 a 10 em Portugol (Elaborado pelo autor)

A estrutura de repetição *Enquanto* do Portugol foi utilizada com a inversão da expressão utilizada no algoritmo. A expressão $i+=1$ é equivalente a $i = i + 1$ e o operador relacional !=

testa a desigualdade. Portanto, as repetições acontecem enquanto o valor de i for diferente de 0.
Está disponível em: https://portugol-webstudio.cubos.io/ide#share=igvmov

A.15 Algoritmo 10 em Portugol - Repita n vezes: 1 a 10

A figura 78 mostra o algoritmo 10 codificado em Portugol.

Figura 78 – Repita 10 Vezes em Portugol (Elaborado pelo autor)

O valor inicial da variável i é 1, o incremento é $+1$ e a condição para repetição é o valor de i ser menor ou igual a 10. Portanto, o comando **escreva(i,", ")** é repetido 10 vezes.
Está disponível em: https://portugol-webstudio.cubos.io/ide#share=ythbsi

A.16 Algoritmo 11 em Portugol: Procedimento exibe pares até n

A figura 79 mostra o algoritmo 11 codificado em Portugol.

Figura 79 – Procedimento Exibe Pares em Portugol (Elaborado pelo autor)

Um Procedimento em Portugol é definido como uma função sem valor de retorno. O procedimento ExibeParesAte recebe n como parâmentro e tem a variável local i definida em seu escopo. O valor inicial de i é 2, o incremento é $+2$ e a condição para repetição é o valor de i ser menor ou igual a n. Na execução demonstrada na figura 79 o valor 10 é passado e atribuído ao parâmetro n para execução do procedimento.

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=gqxeel

A.17 Algoritmo 12 em Portugol: Função calcula dobro

A figura 80 mostra o algoritmo 12 codificado em Portugol.

Figura 80 – Função Dobro em Portugol (Elaborado pelo autor)

A função Dobro recebe um parâmetro real n e retorna um valor real. O retorno é definido pelo comando *retorne*. Na execução demonstrada na figura 80 o valor um valor para variável i é lido no programa e passado como parâmetro para a função Dobro que retorna 2 vezes esse valor.

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=nkizyz

A.18 Algoritmo 13 em Portugol - Soma iterativa dos naturais

A figura 81 mostra o algoritmo 13 codificado em Portugol.

Figura 81 – Soma iterativa dos Naturais em Portugol (Elaborado pelo autor)

A variável Sn inicia com valor 0. A estrutura de controle **para** faz a variável i variar de 1 até n com incremento 1. A cada iteração o valor de i é acrescentado à variável Sn.
Está disponível em: https://portugol-webstudio.cubos.io/ide#share=kylxwf

A.19 Algoritmo 14 em Portugol - Termo de Fibonacci iterativo

A figura 82 mostra o algoritmo 14 codificado em Portugol.

Figura 82 – Fibonacci Iterativo em Portugol (Elaborado pelo autor)

Note que se $n = 1$ ou $n = 2$ a estrutura de repetição não realizará nenhuma iteração e o valor de $FiboN$ permanecerá em 1, portanto um valor válido para ambos casos.

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=qxmgvf

A.20 Algoritmo 15 em Portugol - Contagem recursiva progressiva

A figura 83 mostra o algoritmo 15 codificado em Portugol.

Figura 83 – Procedimento Contagem01 em Portugol (Elaborado pelo autor)

Na execução demonstrada na figura 83 o valor 5 é passado e atribuído ao parâmetro n para execução do procedimento que exibe os valores de 0 a 5 progressivamente porque a escrita de n só ocorre após a finalização de cada chamada recursiva.

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=njqhuy

A.21 Algoritmo 16 - em Portugol - Contagem recursiva regressiva

A figura 84 mostra o algoritmo 16 codificado em Portugol.

Figura 84 – Procedimento Contagem02 em Portugol (Elaborado pelo autor)

Na execução demonstrada na figura 84 o valor 5 é passado e atribuído ao parâmetro n para execução do procedimento que exibe os valores de 0 a 5 regressivamente porque a escrita de n ocorre antes de cada nova chamada recursiva.

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=qdixop

A.22 Algoritmo 17 em Portugol - Fatorial recursivo

A figura 85 mostra o algoritmo 17 codificado em Portugol.

Figura 85 – Fatorial Recursivo em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=nsgnzd

A.23 Algoritmo 18 em Portugol - Soma recursiva dos naturais

A figura 86 mostra o algoritmo 18 codificado em Portugol.

Figura 86 – Soma dos Naturais Recursiva em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=zfzwpr

A.24 Algoritmo 19 em Portugol - Termo de Fibonacci recursivo

A figura 87 mostra o algoritmo 19 codificado em Portugol.

Figura 87 – Fibonacci Recursivo em Portugol(Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=fyfwjn

A.25 Algoritmo 20 em Portugol: Torre de Hanoi

A figura 88 mostra o algoritmo 20 codificado em Portugol.

Figura 88 – Hanoi em Portugol(Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=uohlyp

A.26 Algoritmo 21 em Portugol - Divisores de um número

A figura 89 mostra o algoritmo 21 codificado em Portugol.

Figura 89 – Divisores de n em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=ghinte

A.27 Algoritmo 22 em Portugol - MDC: Algoritmo de Euclides

A figura 90 mostra o algoritmo 22 codificado em Portugol.

Figura 90 – MDC em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=hkqvpx

A.28 Algoritmo 23 em Portugol - Primo ou Composto

A figura 91 mostra o algoritmo 23 codificado em Portugol.

Figura 91 – Primo ou Composto em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=pyffjj

A.29 Algoritmo 24 em Portugol - Crivo de Eratóstenes: primos até 100

A figura 92 mostra o algoritmo 24 codificado em Portugol.

Figura 92 – Crivo de Eratostenes em Portugol (Elaborado pelo autor)

Note que, para evitar um lógica adicional para ajuste de índices, o vetor x[101] foi dimensionado com 101 elementos porque em Portugol os índices de vetores iniciam em zero.
Está disponível em: https://portugol-webstudio.cubos.io/ide#share=ncqkzk

A.30 Algoritmo 25 em Portugol - Fatoração

A figura 93 mostra o algoritmo 25 codificado em Portugol.

Portugol Webstudio Código compartilhado (#xfkfzj)

```
programa {
    inteiro n, fatora, i, p
    funcao inicio() {
        escreva("Quer saber os fatores de qual valor?")
        leia(n)
        fatora=n
        p=2
        enquanto (fatora!=1) {
            se ((fatora % p) == 0) {
                i=0
                enquanto ((fatora % p) == 0) {
                    i=i+1
                    fatora=fatora / p
                }
                escreva(p,"^",i," ")
            }
            p=p+1
        }
    }
}
```

```
Quer saber os fatores de qual valor?60
2^2 3^1 5^1
Programa finalizado.
```

Figura 93 – Fatoração de n em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=xfkfzj

A.31 Algoritmo 26 em Portugol - Base decimal para binária

A figura 94 mostra o algoritmo 26 codificado em Portugol.

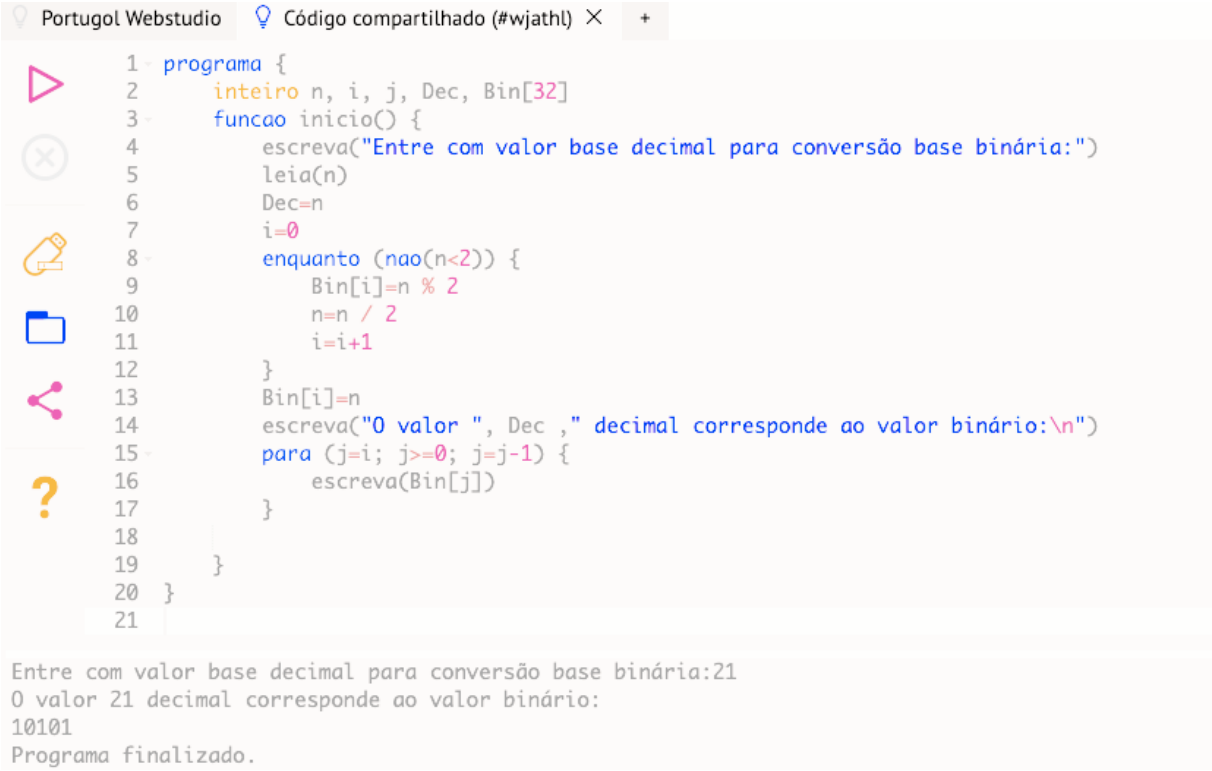

Figura 94 – Conversão base decimal para base binário em Portugol (Elaborado pelo autor)

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=wjathl

A.32 Algoritmo 27 em Portugol: Conversão binário para decimal

A figura 95 mostra o algoritmo 27 codificado em Portugol.

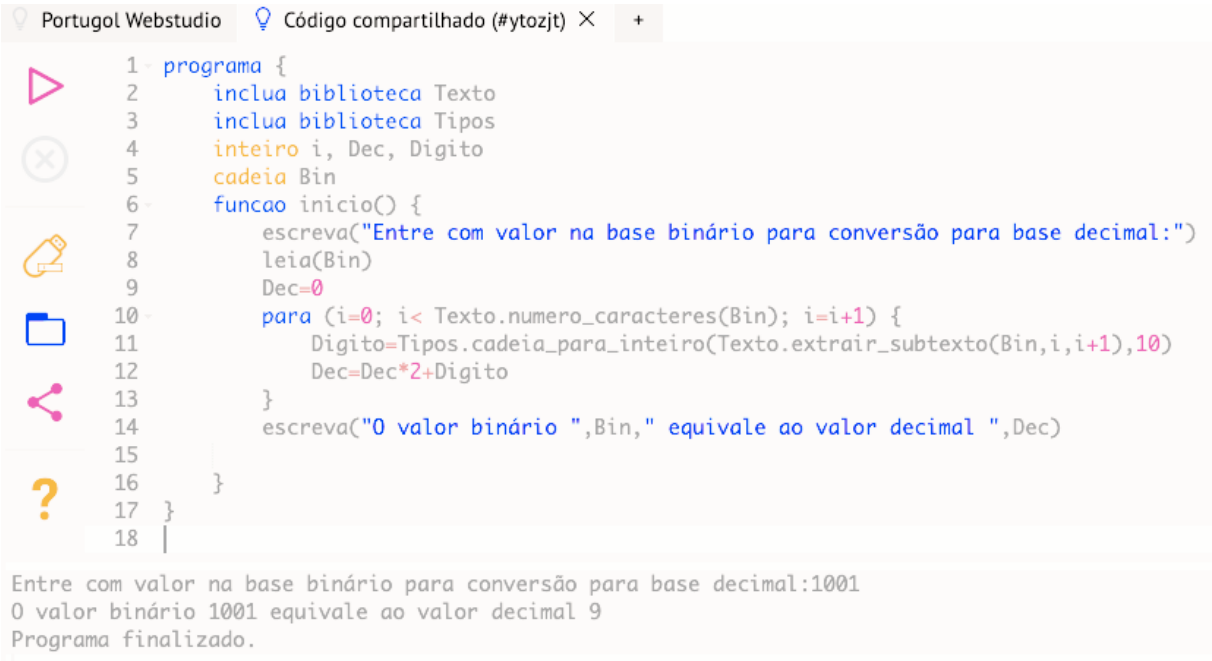

Figura 95 – Conversão base binário para base decimal em Portugol (Elaborado pelo autor)

Note que foram utilizadas duas bibliotecas: Texto e Tipo. A biblioteca Texto foi para utilização das funções *numero_caracteres(cadeia)* e *extrair_subtexto(cadeia,inicio,fim)* que retornam o tamanho da cadeia e a subcadeia de cadeia iniciando em inicio e terminando em fim, respectivamente. Já a biblioteca Texto foi para utilização da função *cadeia_para_inteiro(cadeia,base)* que retorna um valor numérico correspondente a cadeia na base.

Está disponível em: https://portugol-webstudio.cubos.io/ide#share=ytozjt

A.33 Algoritmo 28 em Portugol: Base decimal para hexadecimal

A figura 96 mostra o algoritmo 28 codificado em Portugol.

Figura 96 – Conversão base decimal para base hexadecimal em Portugol (Elaborado pelo autor)

Note que a operação *simbolo[n % 16] + Hexa* faz a concatenação do caractere equivalente ao dígito resto na base hexadecimal sempre no início da cadeia Hexa (consequentemente, evitando a necessidade de inversão dos dígitos da representação)
Está disponível em: https://portugol-webstudio.cubos.io/ide#share=vkpazt

A.34 Algoritmo 29 em Portugol: Conversão hexadecimal para decimal

A figura 97 mostra o algoritmo 29 codificado em Portugol.

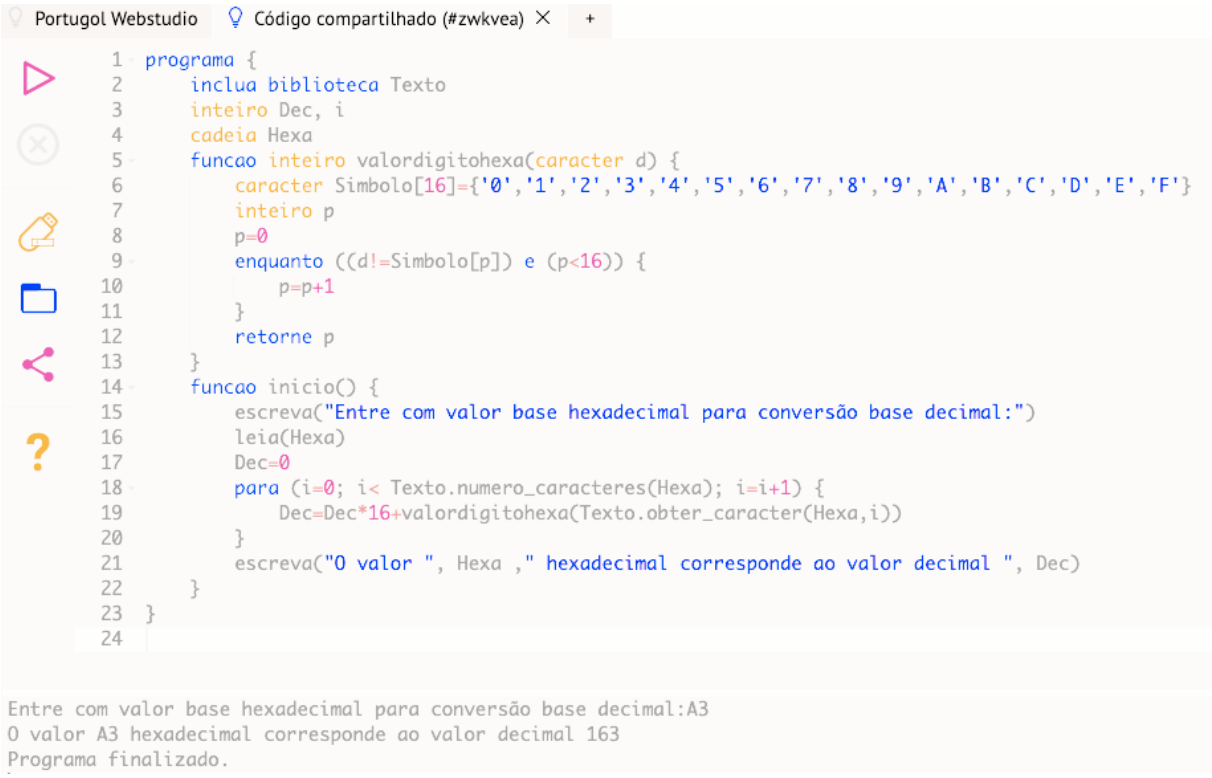

Figura 97 – Conversão base hexadecimal para base decimal em Portugol (Elaborado pelo autor)

Note que foi utilizada a biblioteca Texto para utilização das funções *numero_caracteres(cadeia)* e *obter_caractere(cadeia, posicao)* que retorna o caractere da posicao de cadeia.
Está disponível em: https://portugol-webstudio.cubos.io/ide#share=zwkvea

B APÊNDICE PYTHON

O Python foi criado no início dos anos 1990 por Guido van Rossum na Stichting Mathematisch Centrum na Holanda como um sucessor de uma linguagem chamada ABC. Python e R atualmente são as linguagens de programação mais utilizadas na área de ciência de dados.

Este apêndice **não abrangerá todos os recursos do Python**, tendo apenas definições de elementos utilizados no corpo da dissertação. A documentação da linguagem Python[1] em português está disponível em: https://docs.python.org/pt-br/3/.

B.1 Ambiente

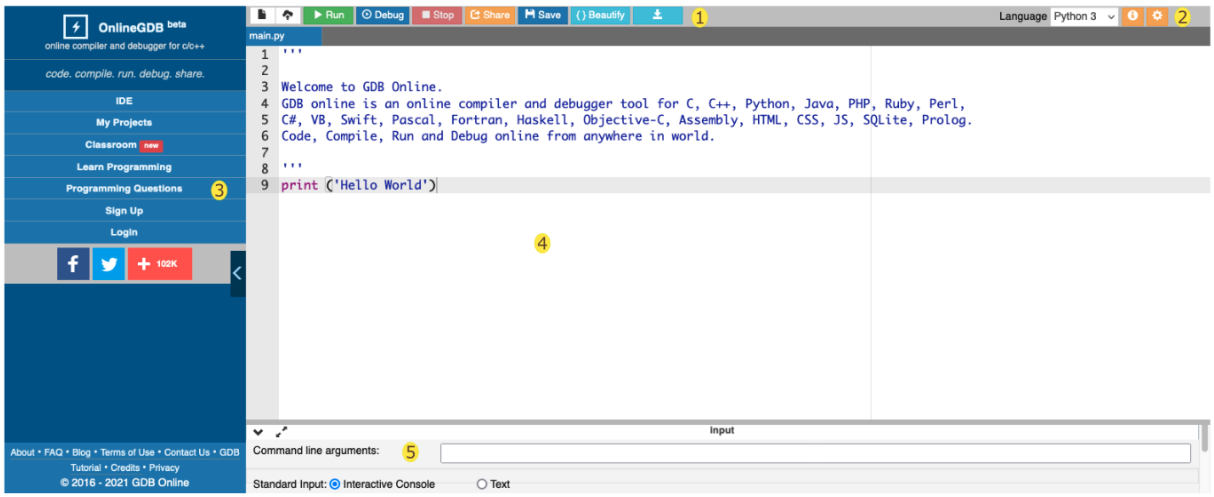

Figura 98 – Ambiente OnlineGDB

Nas implementações deste trabalho foi utilizado o ambiente OnlineGDB. GDB online é uma ferramenta web[2] para compilação e depuração para C, C++, Python, PHP, Ruby, C#, VB, Perl, Swift, Prolog, Java, Javascript, Pascal, Fortran, Haskell, Objective-C, Assembly, HTML, CSS, JS Code, SQLite, Prolog.

A figura 98 mostra o ambiente OnlineGDB. (1) Menu Principal - será utilizado o botão **Run** para executar um programa e o botão **Stop** caso seja necessário interromper a execução. (2) Menu Conficuração - será utilizada apenas a linguagem **Python 3**. (3) Menu Lateral - não será utilizado. Ele pode ser ocultado com um clique na seta ao centro. (4) Área de edição dos programas. (5) Tela de execução ou entrada de parâmetros. Será utilizado em modo tela de execução.

[1] Para referência completa e *download*, consulte https://www.python.org/
[2] Funciona no navegador (*browser*), sem necessidade de instalação e configuração

Um ambiente alternativo ao OnlineGDB para execução de programas Python é o Colaboratory ou Colab do Google[1]. A figura 99 mostra a tela do ambiente Google Colab - é necessário ter uma conta Google para utilizá-lo.

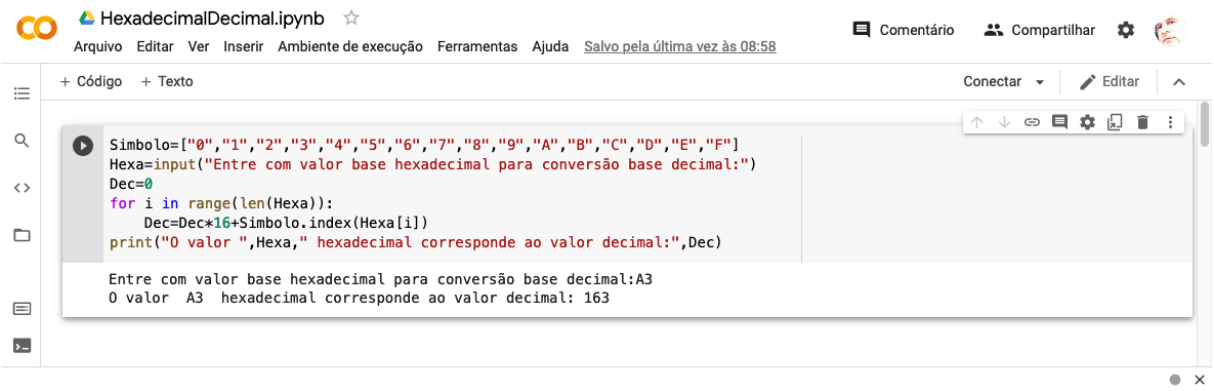

Figura 99 – Ambiente Google Colab

B.2 Tipos de Dados, Variáveis e Operações

Em Python não há declaração de variáveis, sendo os tipos inferidos em tempo de execução. Neste trabalho serão utilizados os tipos int, float, string (cadeia) e list (lista).

Os **operadores aritméticos** dos inteiros são: adição (+), subtração (-), multiplicação (*), divisão inteira (//) e resto da divisão inteira (%). **Operadores relacionais** são: maior que (>), maior que ou igual a (>=), menor que (<), menor que ou igual a (<=), igualdade (==), desigualdade (! =).

Os **operadores aritméticos** dos reais são: adição (+), subtração (-), multiplicação (*), divisão real (/) e resto da divisão inteira (%). **Operadores relacionais** são: maior que (>), maior que ou igual a (>=), menor que (<), menor que ou igual a (<=), igualdade (==), desigualdade (! =).

Operadores booleanos são: conjunção (and), disjunção (or) e negação (not).

[1]Consulte: https://colab.research.google.com/notebooks/welcome.ipynb?hl=pt-BR

B.3 Interação e Manipulação de Dados

Em Python,

- Entrada de dados é realizada com o comando: <varstring>=input(<mensagem>), ou seja, sempre o resultado de uma leitura será uma string (cadeia). Para conversões para os tipos inteiro e real devem ser utilizados int(<cadeia) ou float(<cadeia>), respectivamente.
- Saída de dados é realizada com o comando: print(<listadevariaveisouliterais>)
- Atribuições são realizadas com o comando: <variavel> = <expressao>. O sinal de igual pode ser substituído por +=, -=, *=, /= que correspondem a uma auto operação. Por exemplo: $i+=1$ corresponde a $i = i + 1$.

Veja exemplos em B.6 e B.7.

B.4 Estrutura de Controle Condicional

A estrutura de controle condicional possibilita condicionar a execução de comandos a um valor lógico ($falso$ ou $verdadeiro$).

Estrutura de Controle Condicional Simples - if

O conjunto de comandos <$comandos$> será executado caso, no momento do processamento, <$condicao$> resulte $verdadeiro$. Para um resultado $falso$, nada será processado. Observe que os comandos vinculados à condição, ou seja <comandos1>, devem estar **indentados** (avançados). Veja exemplo em B.8.

Sintaxe:

if (<*condicao*>):
    <*comandos1*>

Estrutura de Controle Condicional Composta - if/else

Apresenta dois conjuntos de comandos, mutuamente exclusivos. Os comandos <$comandos1$> serão executados caso a expressão <$condicao$> tenha valor $verdadeiro$ no momento da execução. Caso seja $falso$, os comandos <$comandos2$> que serão executados. Observe que <comandos1> e <comandos2> devem estar **indentados** (margem avançada). Veja exemplo em B.9.

Sintaxe:

if (<*condicao*>):
    <*comandos1*>
else:
    <*comandos2*>

B.5 Estruturas de Controle de Repetições

Python possui dois tipos de estruturas de repetições: com pré-teste (teste antes do bloco a ser repetido) - **while** e com variável de controle - **for**.

Estrutura de Repetição - while

Estrutura de controle de repetição com pré-teste: <$condicao$> é testada. Sendo $verdadeiro$, <$comandos$> são processados e um novo teste realizado. Encerra caso <$condicao$> resulte em $falso$. Observe que <comandos> devem ter margem maior que a do **while**. Veja exemplo em B.10.

Sintaxe:

while (<*condicao*>):
    <*comandos*>

Estrutura de Repetição - for

Estrutura de controle de repetição com variável de controle. É utilizada para iterações com valores de uma sequência que pode ser gerada com a função range(<valor>). Exemplo: range(5) equivale a [0, 1, 2, 3, 4]. Outro forma para range seria: range(<inicio>, <fim>, <variacao>). Veja exemplo em B.13.

Sintaxe:

for <var> in <sequencia>:
    <*comandos*>

B.6 Algoritmo 1 em Python - Entrada e Saída

A figura 100 mostra o algoritmo 1 codificado em Python.

Figura 100 – Entrada e Saída em Python(Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/lGJVQgIwhS

B.7 Algoritmo 2 em Python - Atribuição a variável

A figura 101 mostra o algoritmo 2 codificado em Python.

Figura 101 – Preco com Desconto em Python (Elaborado pelo autor)

A entrada de dados é realizada com a função **input** tendo como resultado uma cadeia de caracteres. A conversão para um tipo real com é realizada com a função **float**. A atribuição em Python é realizada pelo =. Observe que não houve declaração de variáveis, sendo os tipos assumidos de acordo com os valores atribuídos.
Está disponível em: https://onlinegdb.com/8rv-F_Mkg

B.8 Algoritmo 3 em Python - Se/Entao

A figura 102 mostra o algoritmo 3 codificado em Python.

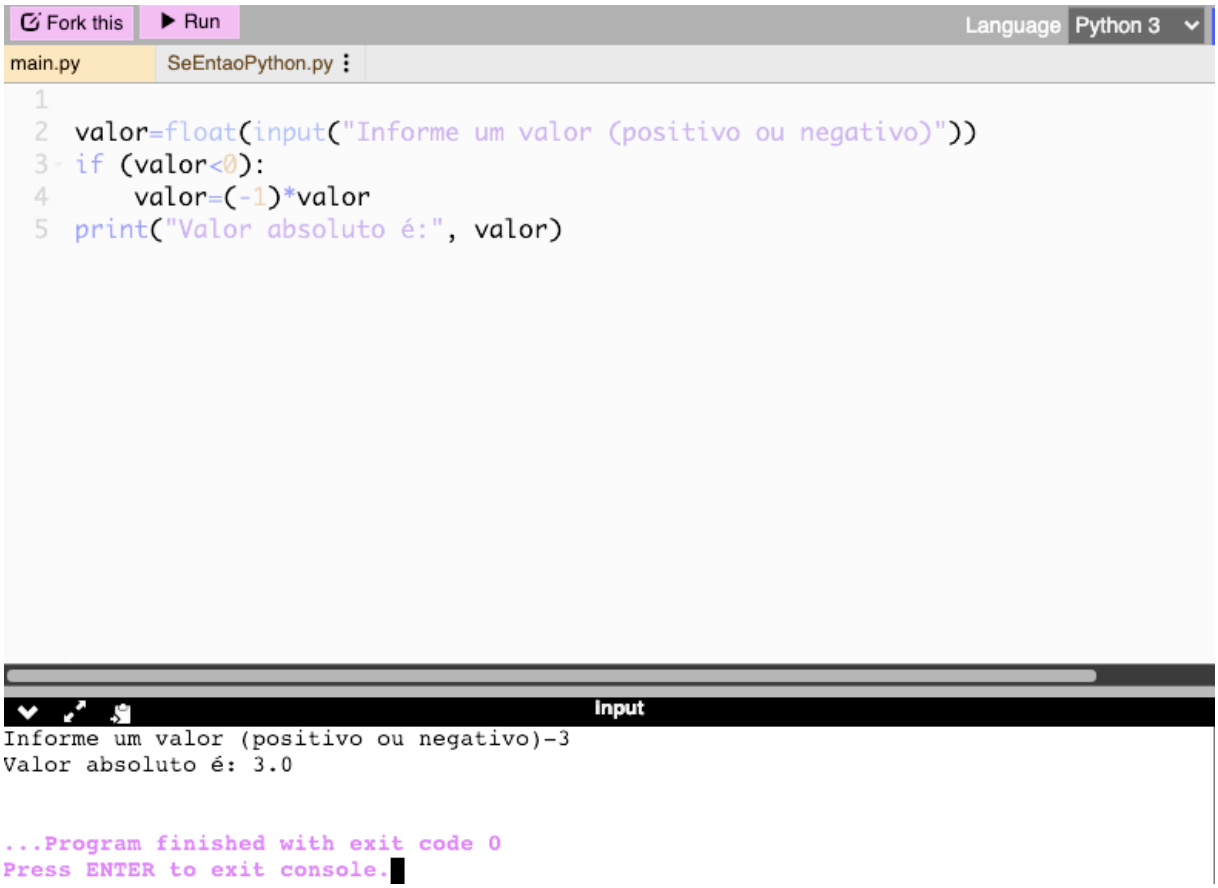

Figura 102 – Se/Entao em Python (Elaborado pelo autor)

Importante ressaltar que no Python a indentação (recuo ou afastamento da margem) define se um comando está dentro ou fora de uma estrutura. No exemplo da figura 102 apenas a expressão $valor = (-1) * valor$ está com avanço de margem e, consequentemente, vinculada à estrutura de controle condicional *if* $(valor < 0)$.
Está disponível em: https://onlinegdb.com/Xt15mUKcy

B.9 Algoritmo 4 em Python - Se/Entao/Senao

A figura 103 mostra o algoritmo 4 codificado em Python.

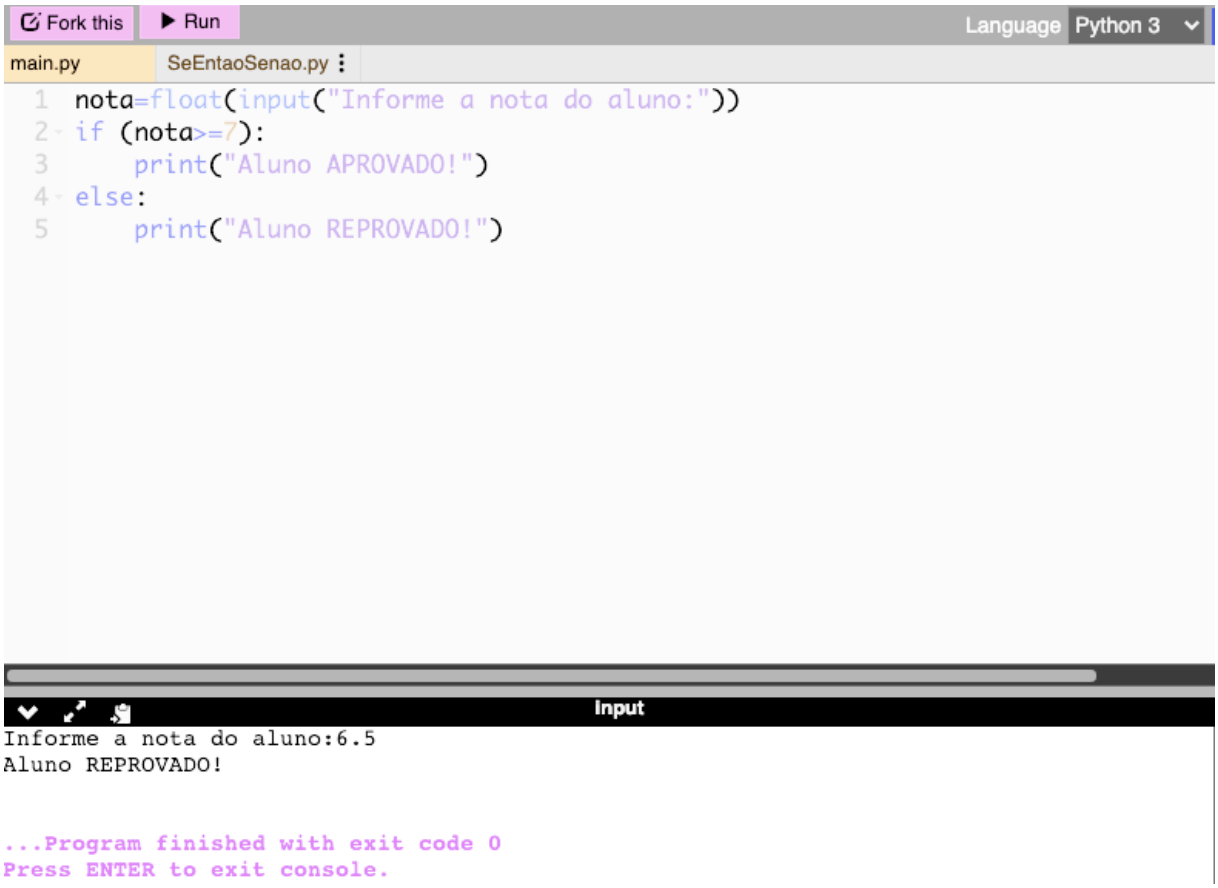

Figura 103 – Se/Entao/Senao em Python (Elaborado pelo autor)

Importante ressaltar que no Python a indentação (recuo ou afastamento da margem) define se um comando está dentro ou fora de uma estrutura de controle.
Está disponível em: https://onlinegdb.com/CuvndqvWZ

B.10 Algoritmo 5 em Python - Enquanto: 1 a 10

A figura 104 mostra o algoritmo 5 codificado em Python.

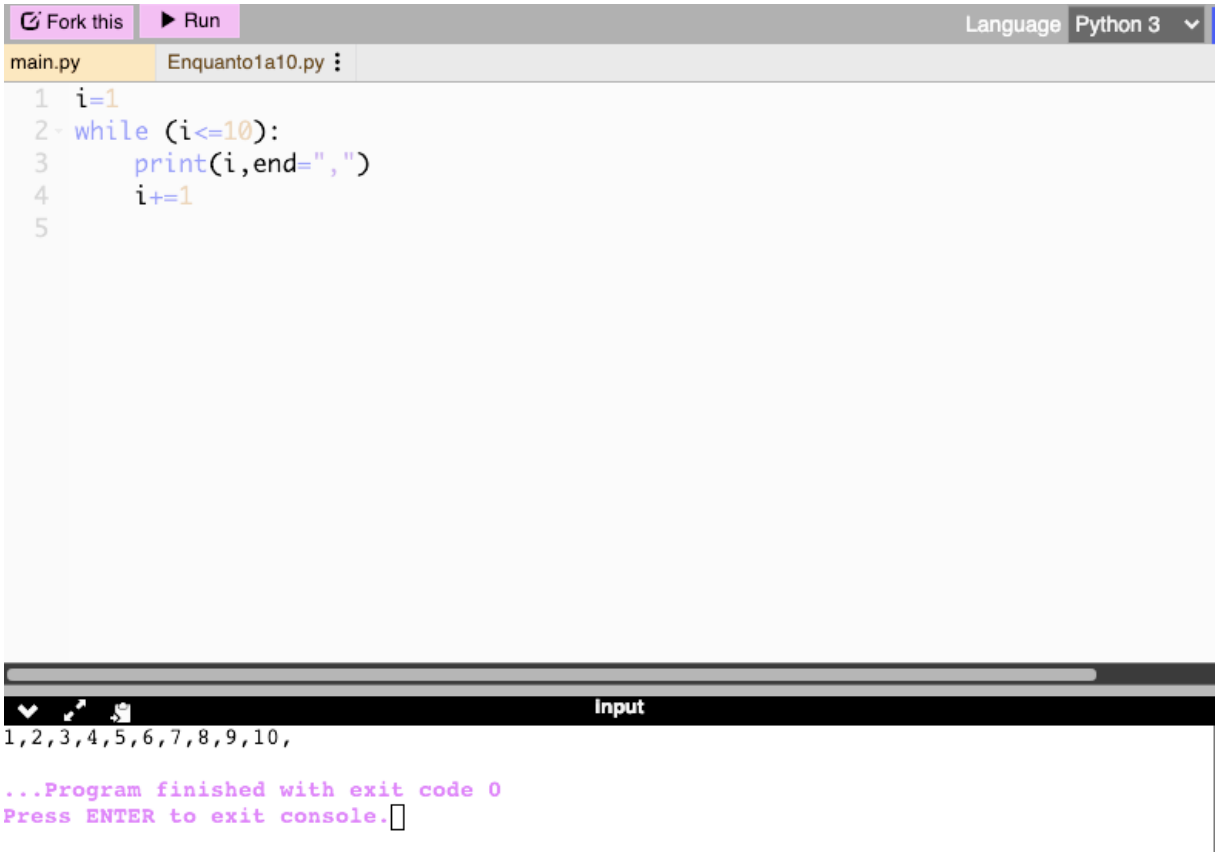

Figura 104 – Exibição de 1 a 10 com *while* em Python (Elaborado pelo autor)

Note que nessa implementação ocorreram algumas alterações em relação ao algoritmo, mas o resultado de execução foi mantido. A variável i inicia com valor 1, a condição de parada incluiu a igualdade ($<=$) e houve inversão na ordem dos comandos de exibição do valor de i e seu incremento em uma unidade. Alternativamente, a implementação poderia ter sido da mesma forma que o algoritmo 5 apresenta. Observe a indentação (recuo ou afastamento da margem) dos comandos, notadamente a exibição e incremento que estão vinculados ao comando **while**. No Python, uma indentação errada causa problema na lógica do processamento e, consequentemente, no resultado final. o **end**=“,” no comando **print** define um valor final, no caso uma vírgula.
Está disponível em: https://onlinegdb.com/92XAlf4H3

B.11 Algoritmo 6 em Python - Enquanto: 10 a 1

A figura 105 mostra o algoritmo 6 codificado em Python.

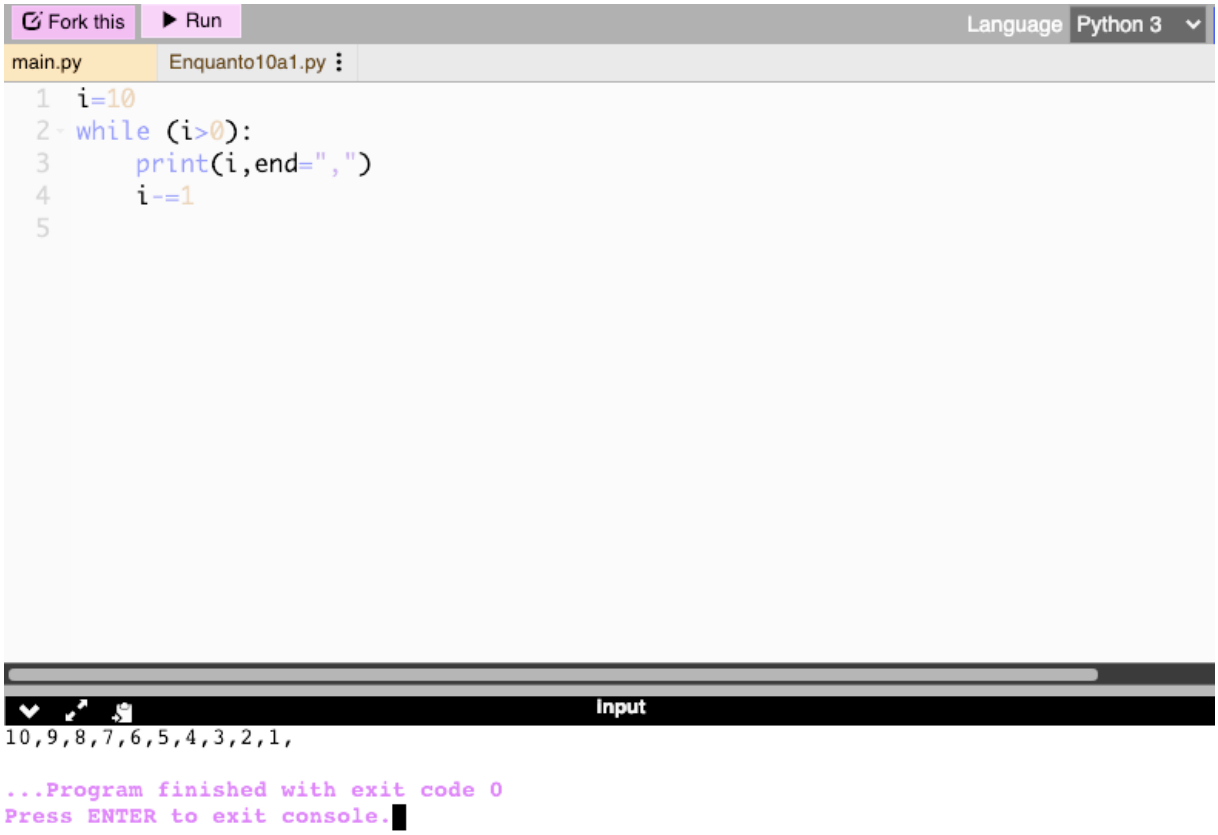

Figura 105 – Exibição de 10 a 1 com *while* em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/_1c5N3QbP

B.12 Algoritmo 7 em Python - Repita até 0 (Par ou ímpar)

A figura 106 mostra o algoritmo 7 codificado em Python.

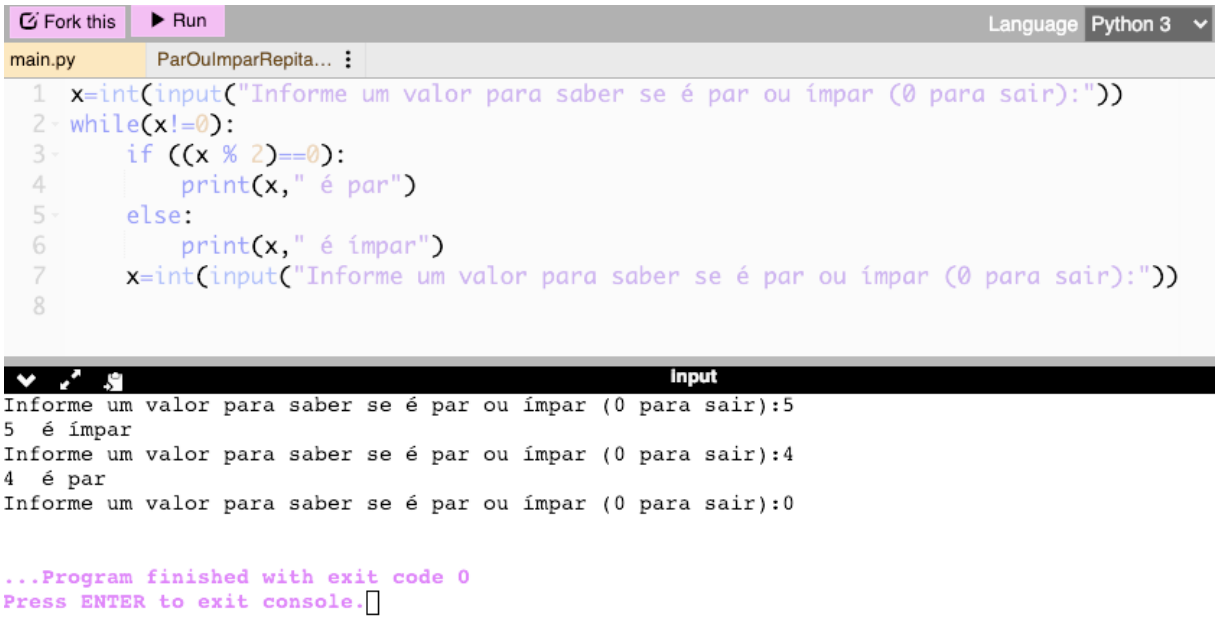

Figura 106 – Par ou Impar com repetição até 0 em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/QGeaVU0mm

B.13 Algoritmo 8 em Python - Para: pares de 2 a 20

A figura 107 mostra o algoritmo 8 codificado em Python.

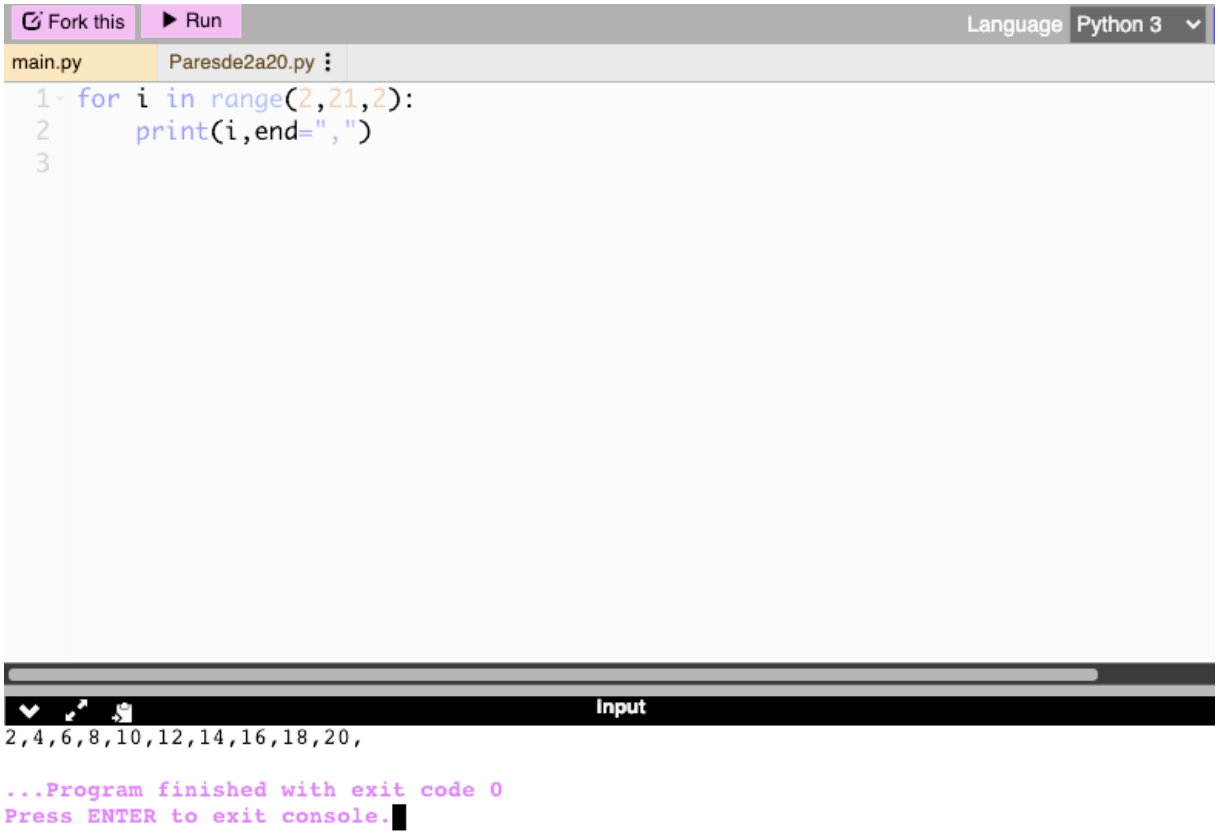

Figura 107 – Pares de 2 a 20 com *for* e *range* em Python (Elaborado pelo autor)

A função **range**$(inicio, fim, razao)$ gera uma lista a partir de $inicio$ com variação de $razao$ até o maior valor possível que seja inferior à fim, caso variação seja positiva (ou menor valor possível que seja superior a fim, caso a variação seja negativa). Por isso, se for colocado o valor 20 como final da $range$, ele não pertencerá à lista e, consequentemente, não será exibido.
Está disponível em: https://onlinegdb.com/25k1R5nBH

B.14 Algoritmo 9 em Python - Repita até que: 1 a 10

A figura 108 mostra o algoritmo 9 codificado em Python.

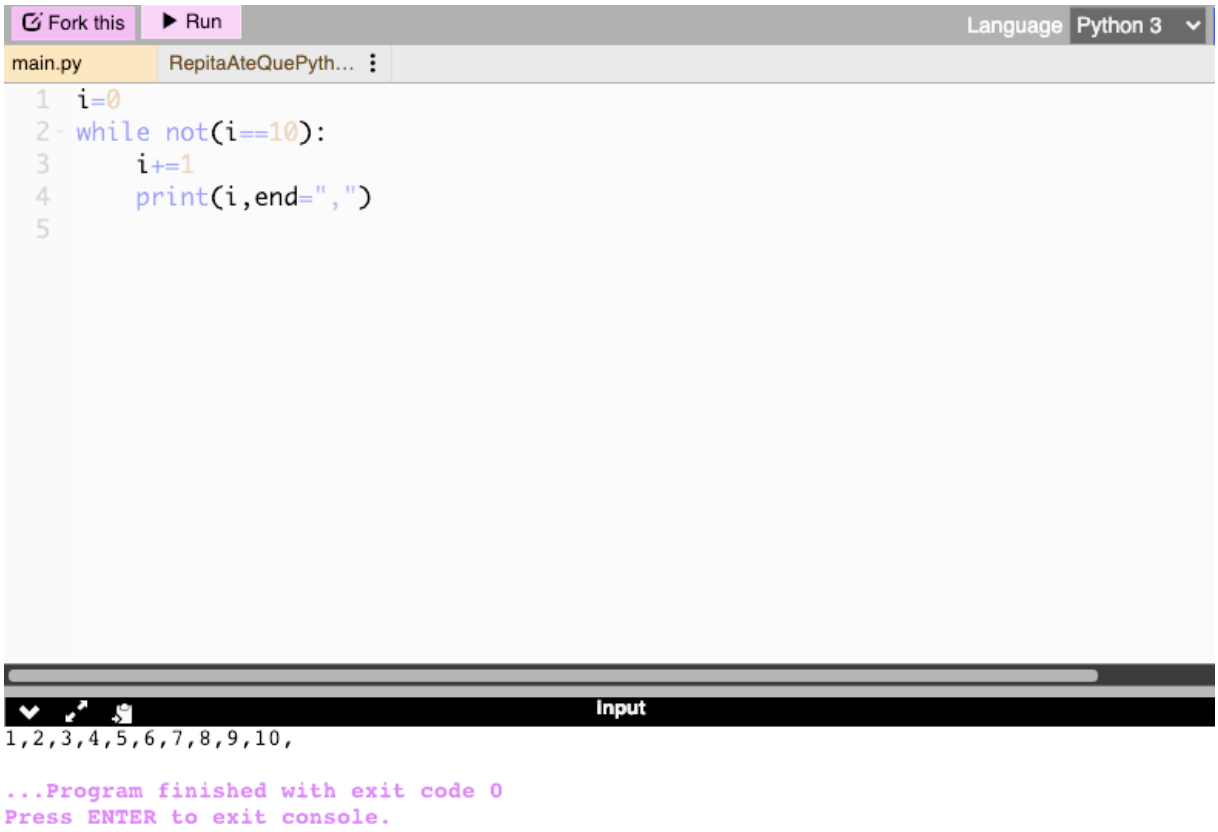

Figura 108 – Repita até que em Python (Elaborado pelo autor)

A expressão $not(i == 10)$ pode ser substituída por $(i\ != \ 10)$.
Está disponível em: https://onlinegdb.com/2lSQBh35B

B.15 Algoritmo 10 em Python - Repita n vezes: 1 a 10

A figura 109 mostra o algoritmo 10 codificado em Python.

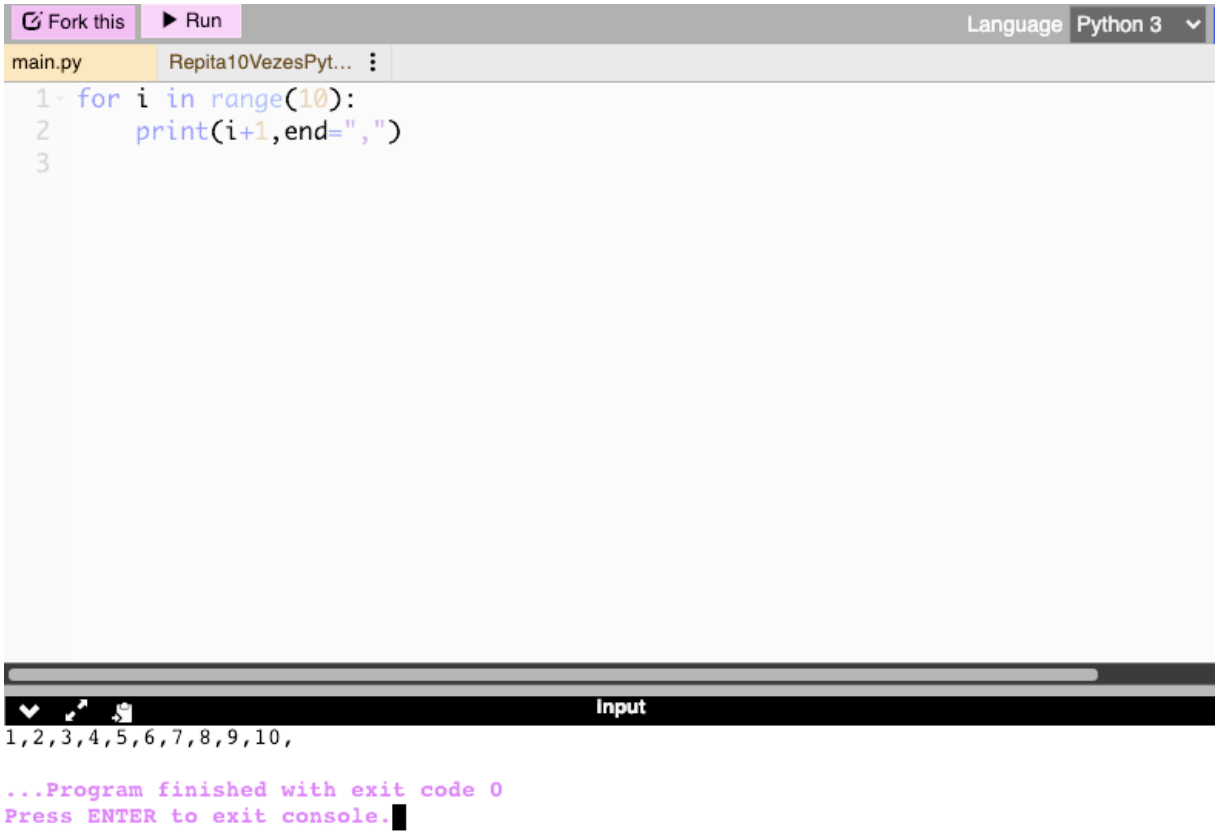

Figura 109 – Repita 10 Vezes com *for* e *range* em Python (Elaborado pelo autor)

A função **range(10)** gera uma lista com 10 elementos (valores de 0 a 9). Por isso, a exibição dos valores $(i + 1)$ sucessores de i.
Está disponível em: https://onlinegdb.com/iTaexSQbo

B.16 Algoritmo 11 em Python: Procedimento exibe pares até n

A figura 110 mostra o algoritmo 11 codificado em Python.

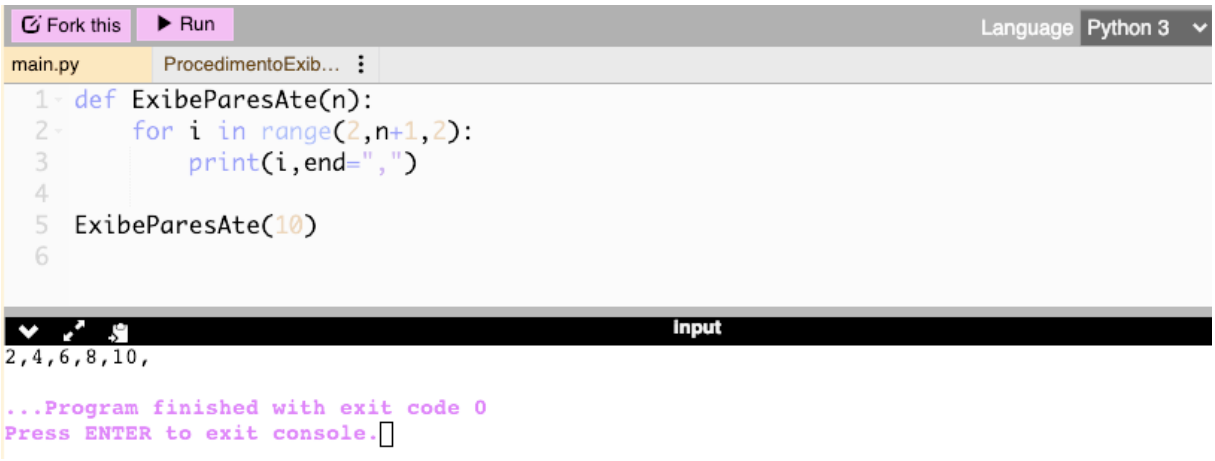

Figura 110 – Procedimento Exibe Pares Ate N em Python (Elaborado pelo autor)

O comando **def** identifica a definição de uma função, no caso um procedimento porque não há valor de retorno. Observe a indentação de todas as instruções que fazem parte da função. Está disponível em: https://onlinegdb.com/qbc3G0ldB

B.17 Algoritmo 12 em Python: Função calcula dobro

A figura 111 mostra o algoritmo 12 codificado em Python.

Figura 111 – Funcao Dobro em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/JbGrbYN5s

B.18 Algoritmo 13 em Python - Soma iterativa dos naturais

A figura 112 mostra o algoritmo 13 codificado em Python.

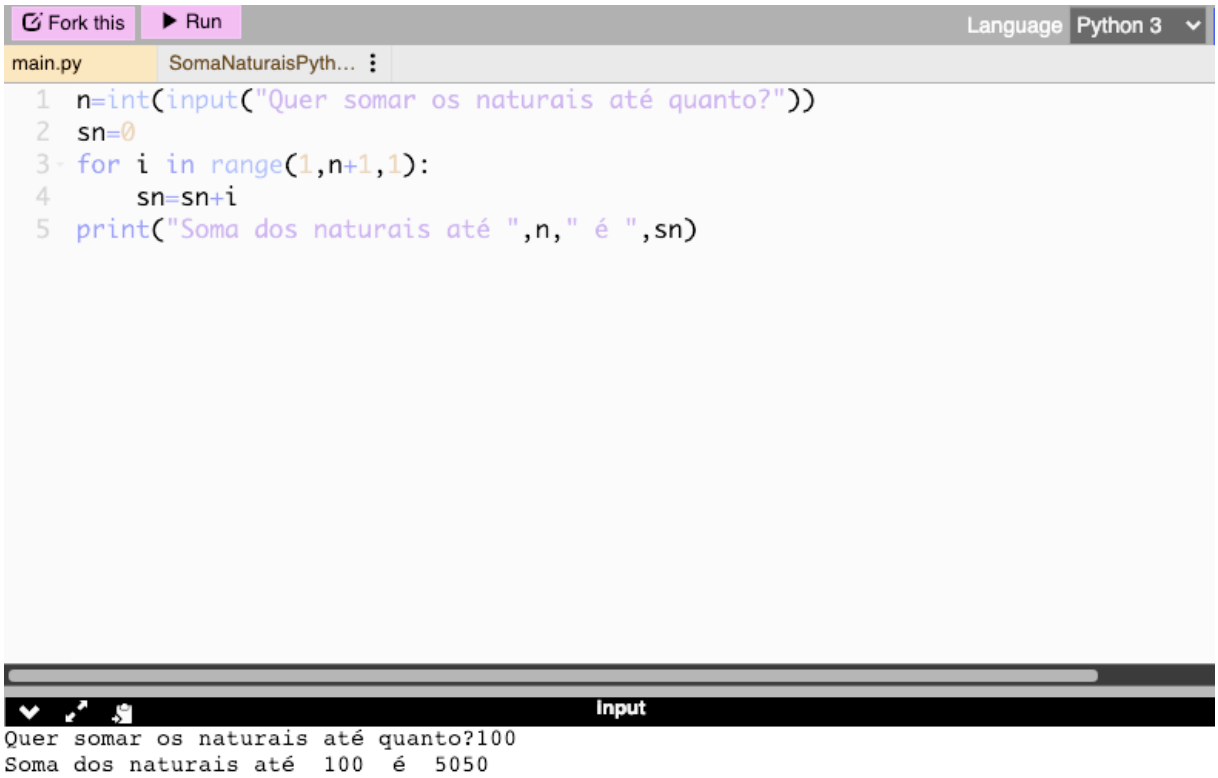

Figura 112 – Soma Iterativa dos Naturais em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/HmjqJLlhm

B.19 Algoritmo 14 em Python - Termo de Fibonacci iterativo

A figura 113 mostra o algoritmo 14 codificado em Python.

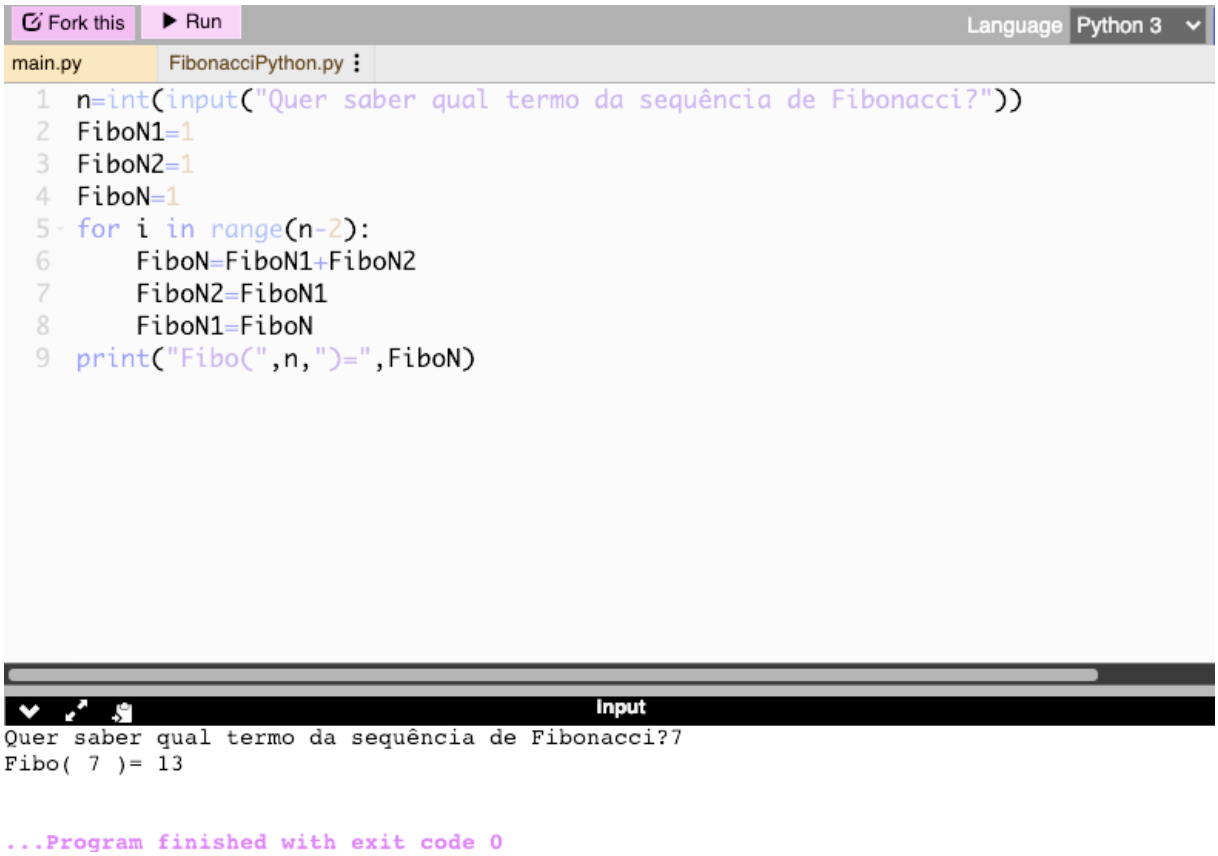

Figura 113 – Fibonacci iterativo em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/KpKD4A2zg

B.20 Algoritmo 15 em Python - Contagem recursiva progressiva

A figura 114 mostra o algoritmo 15 codificado em Python.

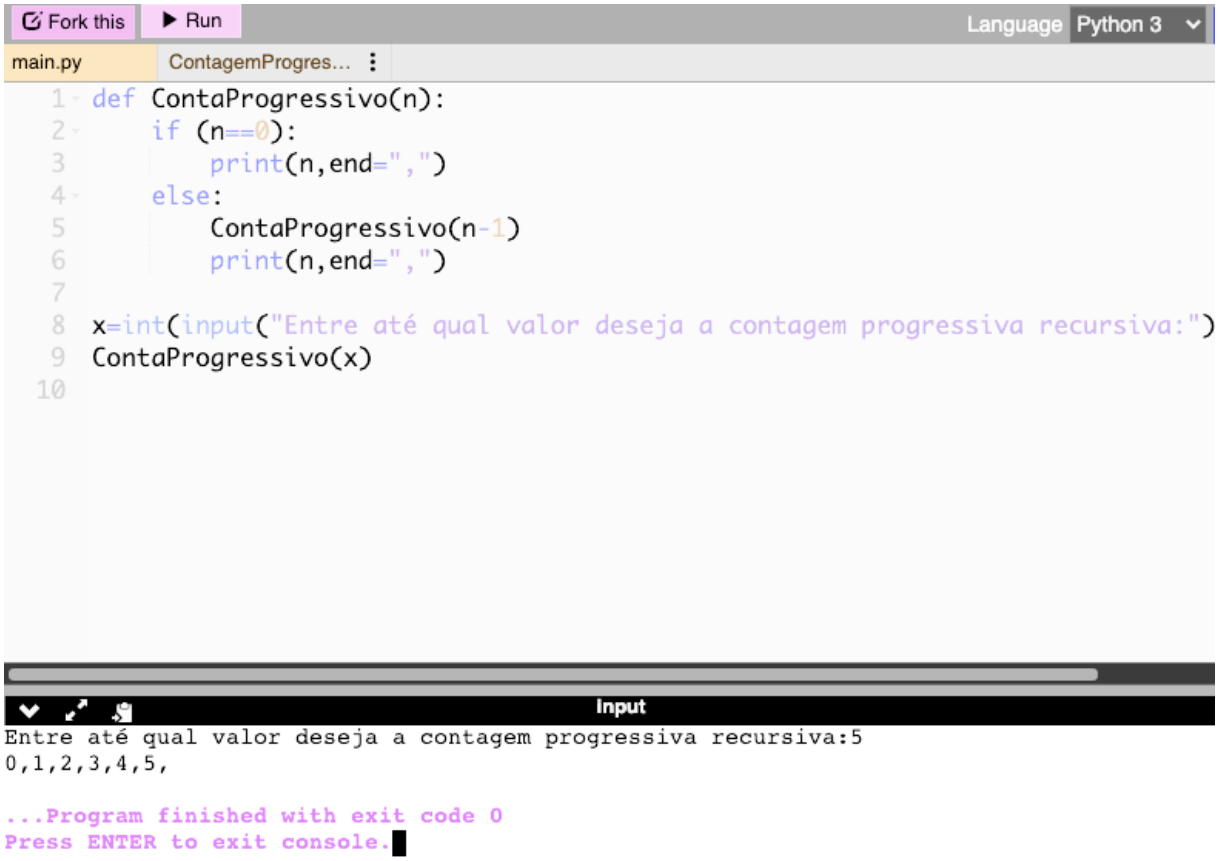

Figura 114 – Contagem Progressiva Recursiva em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/QTwrh7a1P

B.21 Algoritmo 16 em Python - Contagem recursiva regressiva

A figura 115 mostra o algoritmo 16 codificado em Python.

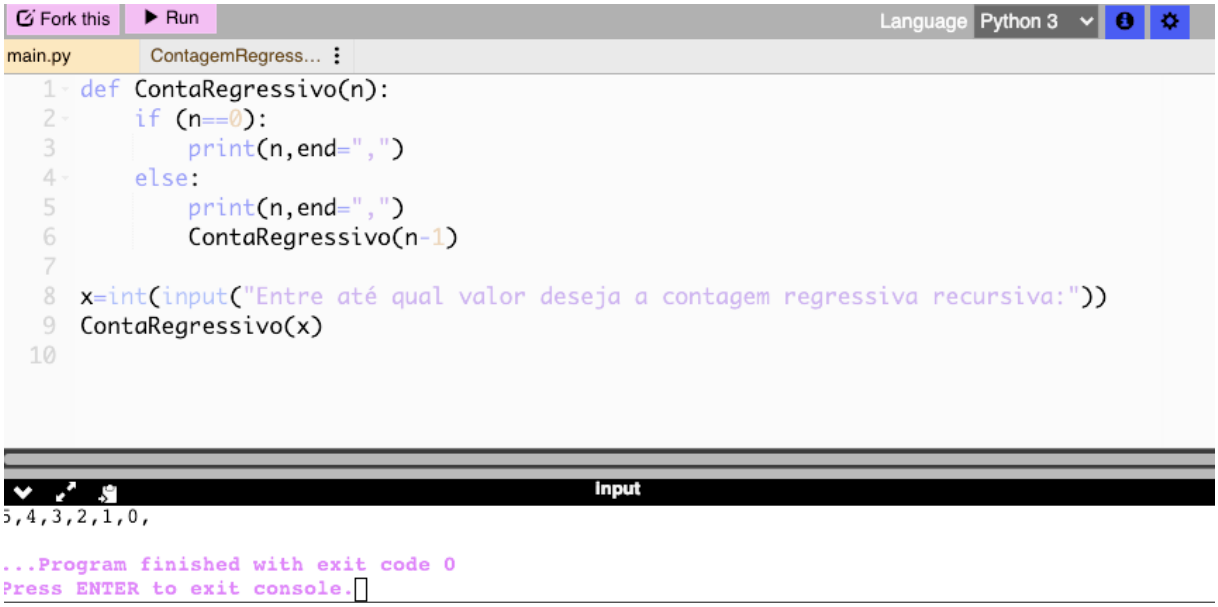

Figura 115 – Contagem Regressiva Recursiva em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/g4m5t-iUW

B.22 Algoritmo 17 em Python - Fatorial recursivo

A figura 116 mostra o algoritmo 17 codificado em Python.

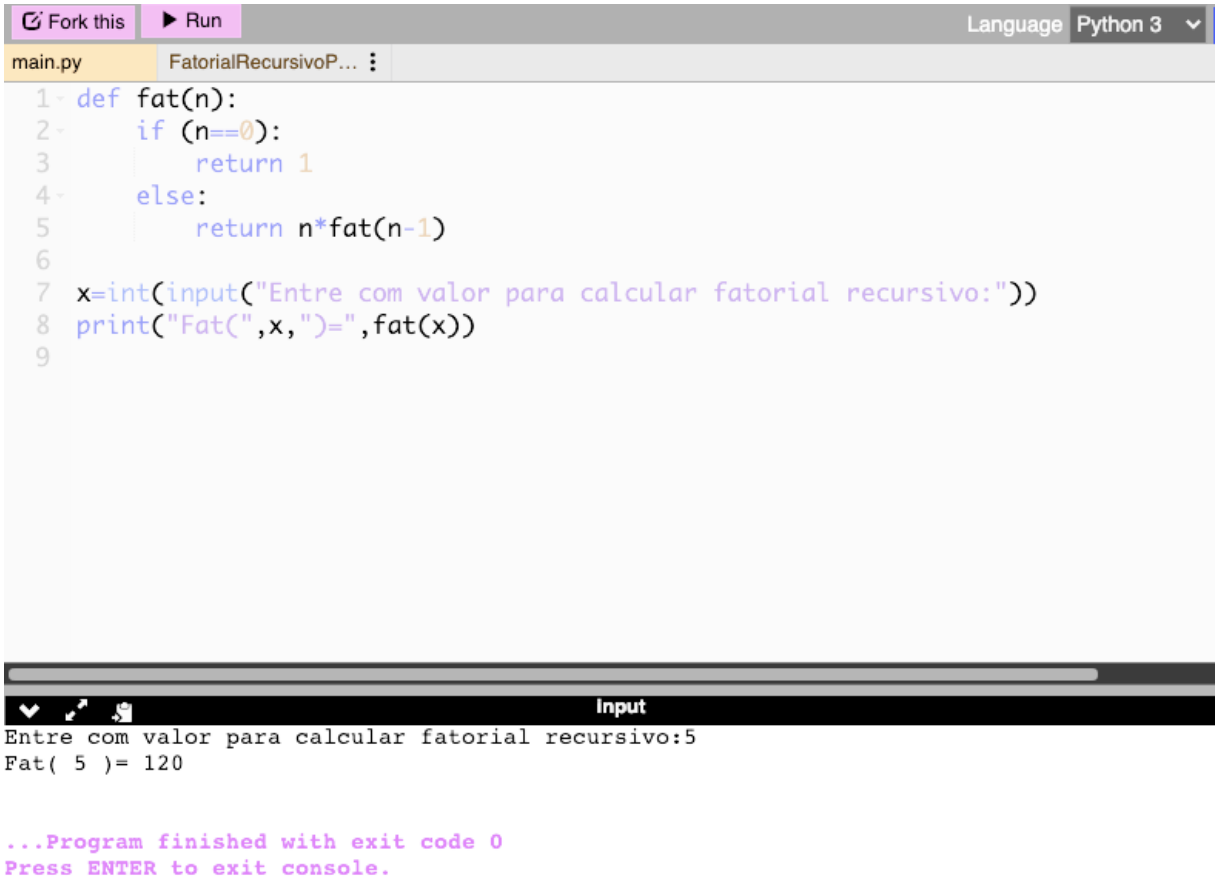

Figura 116 – Fatorial Recursivo em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/RfZLo_awX

B.23 Algoritmo 18 em Python - Soma recursiva dos naturais

A figura 117 mostra o algoritmo 18 codificado em Python.

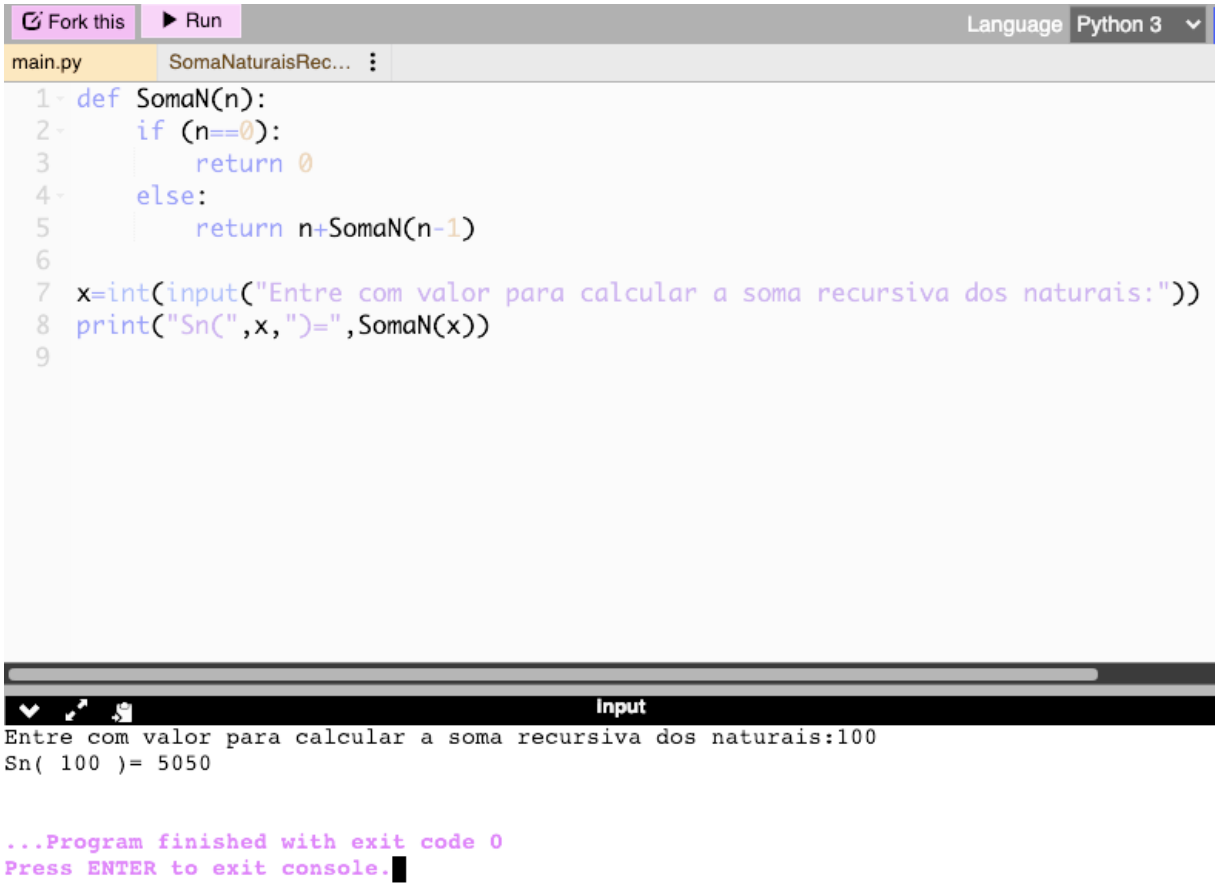

Figura 117 – Soma Recursiva dos Naturais em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/ImYFOuaDt

B.24 Algoritmo 19 em Python - Termo de Fibonacci recursivo

A figura 118 mostra o algoritmo 19 codificado em Python.

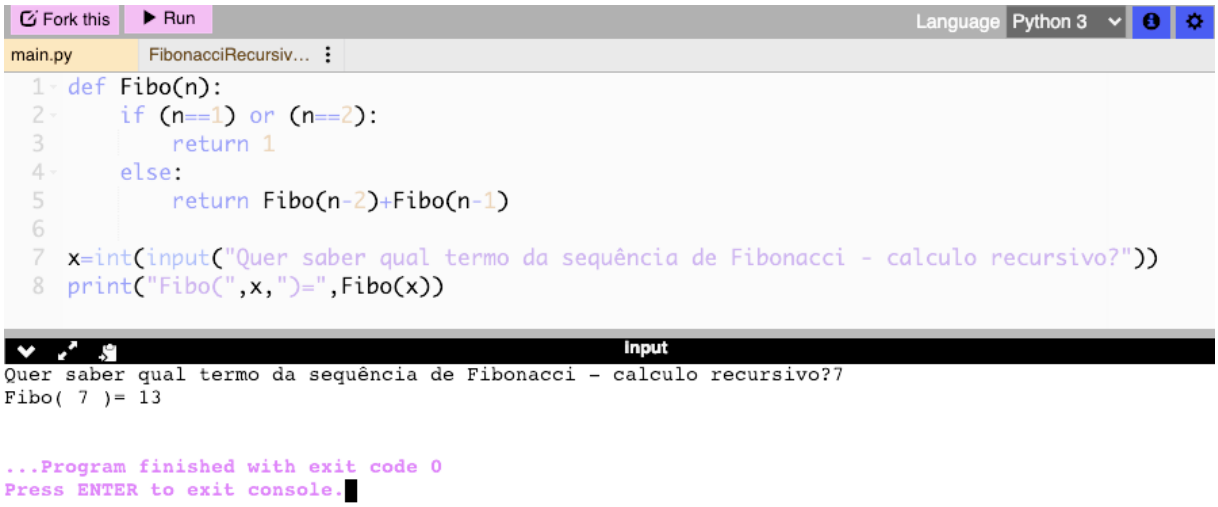

Figura 118 – Fibonacci Recursivo em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/tGNxTyTtM

B.25 Algoritmo 20 em Python: Torre de Hanoi

A figura 119 mostra o algoritmo 20 codificado em Python.

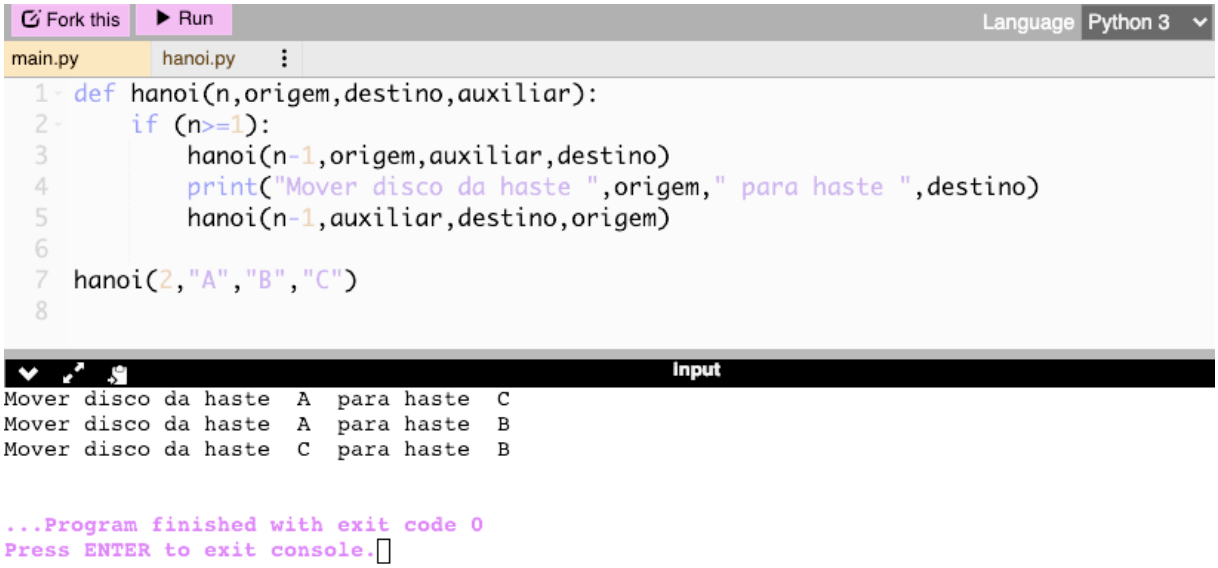

Figura 119 – Torre de Hanoi em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/OJ2NyCWiP

B.26 Algoritmo 21 em Python - Divisores de um número

A figura 120 mostra o algoritmo 21 codificado em Python.

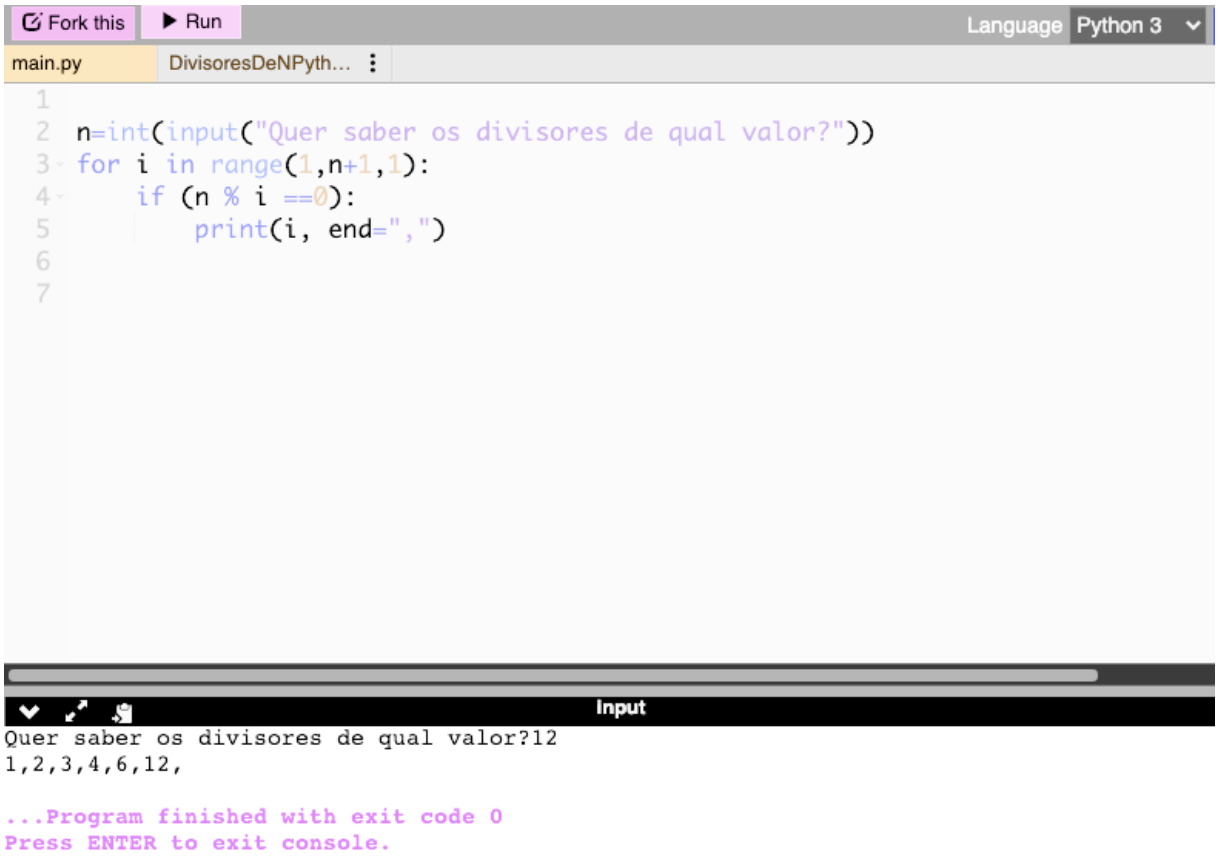

Figura 120 – Divisores de N em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/OFq_oMH7U

B.27 Algoritmo 22 em Python - MDC: Algoritmo de Euclides

A figura 121 mostra o algoritmo 22

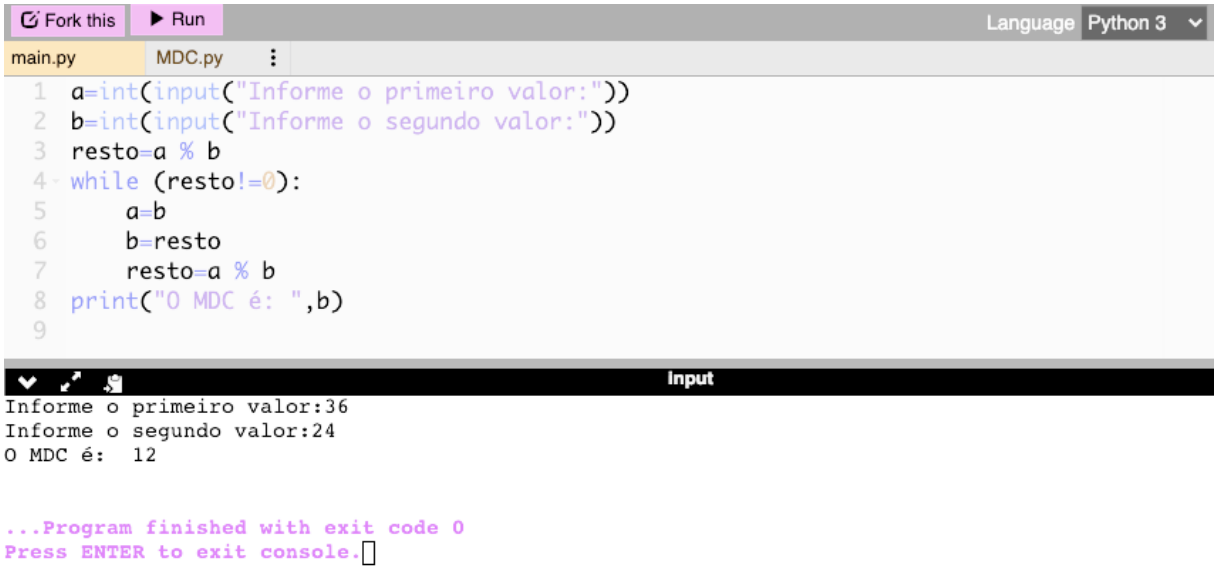

Figura 121 – MDC em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/G_ecnuYC0

B.28 Algoritmo 23 em Python - Primo ou Composto

A figura 122 mostra o algoritmo 23 codificado em Python.

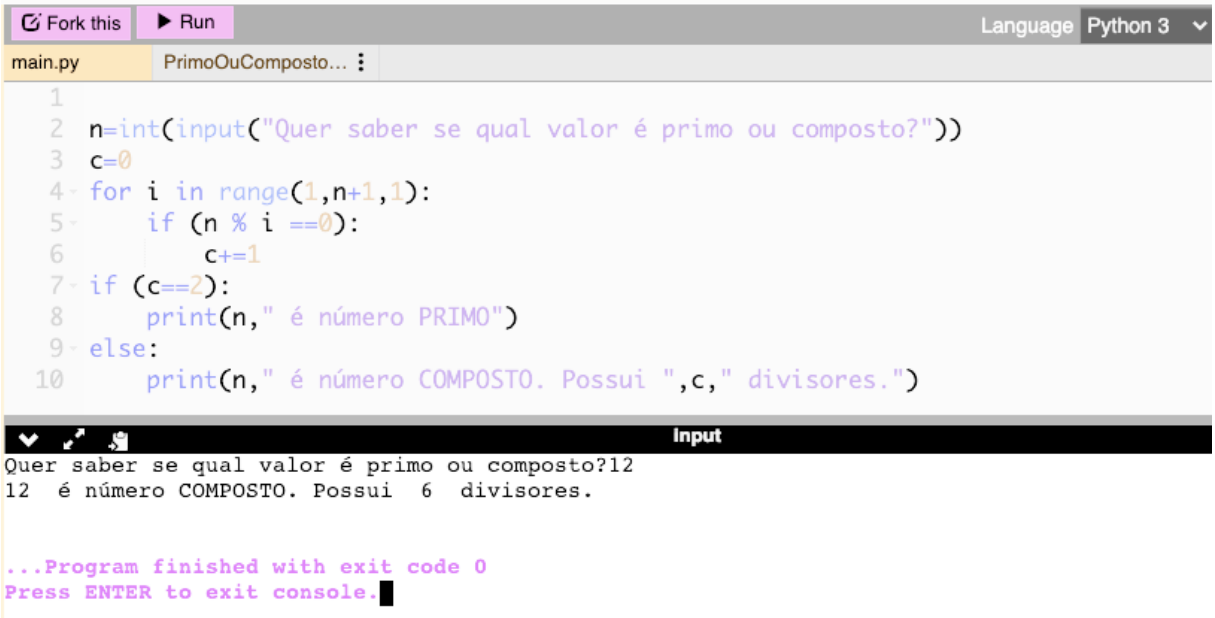

Figura 122 – Primo ou Composto em Python(Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/1P4XTmJg6

B.29 Algoritmo 24 e Python - Crivo de Eratóstenes: primos até 100

A figura 123 mostra o algoritmo 24 codificado em Python.

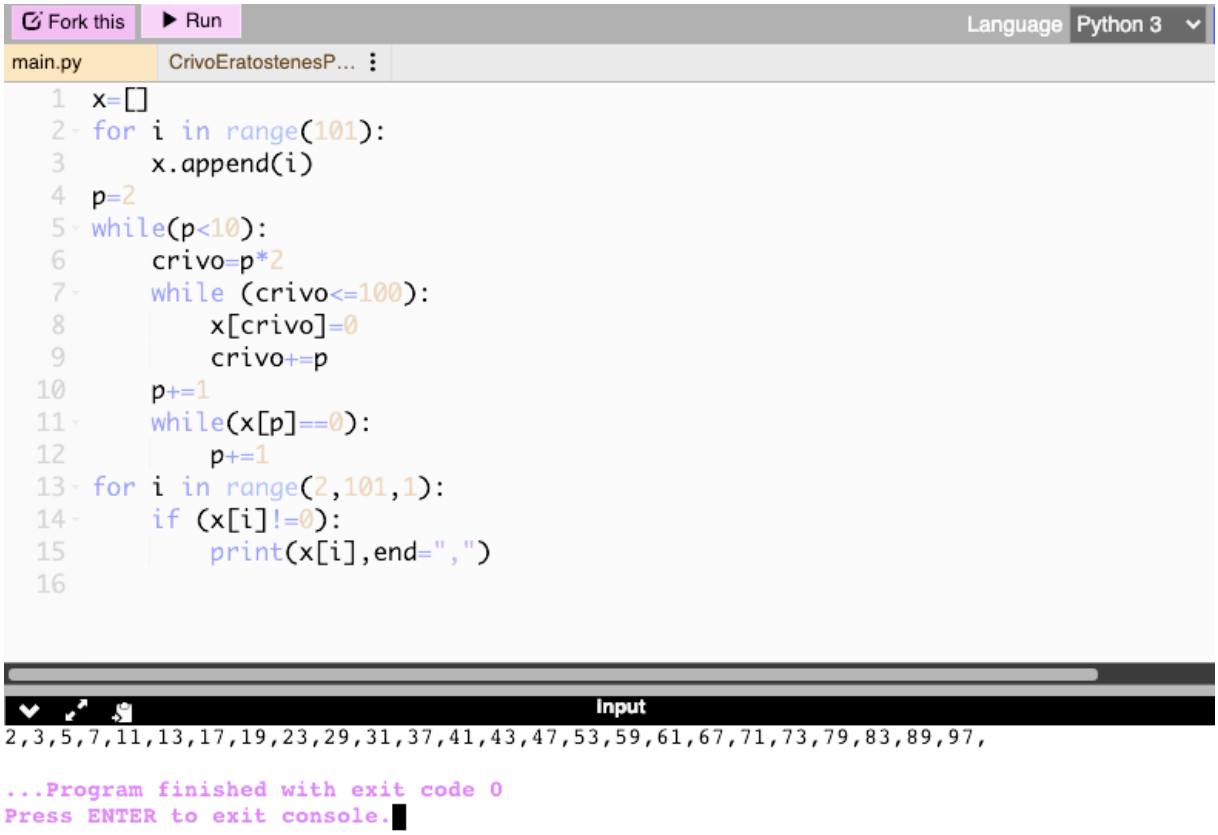

Figura 123 – Crivo de Eratostenes em Python (Elaborado pelo autor)

Está disponível em: https://onlinegdb.com/QHaedtyZC

B.30 Algoritmo 25 em Python - Fatoração

A figura 124 mostra o algoritmo 25 codificado em Python.

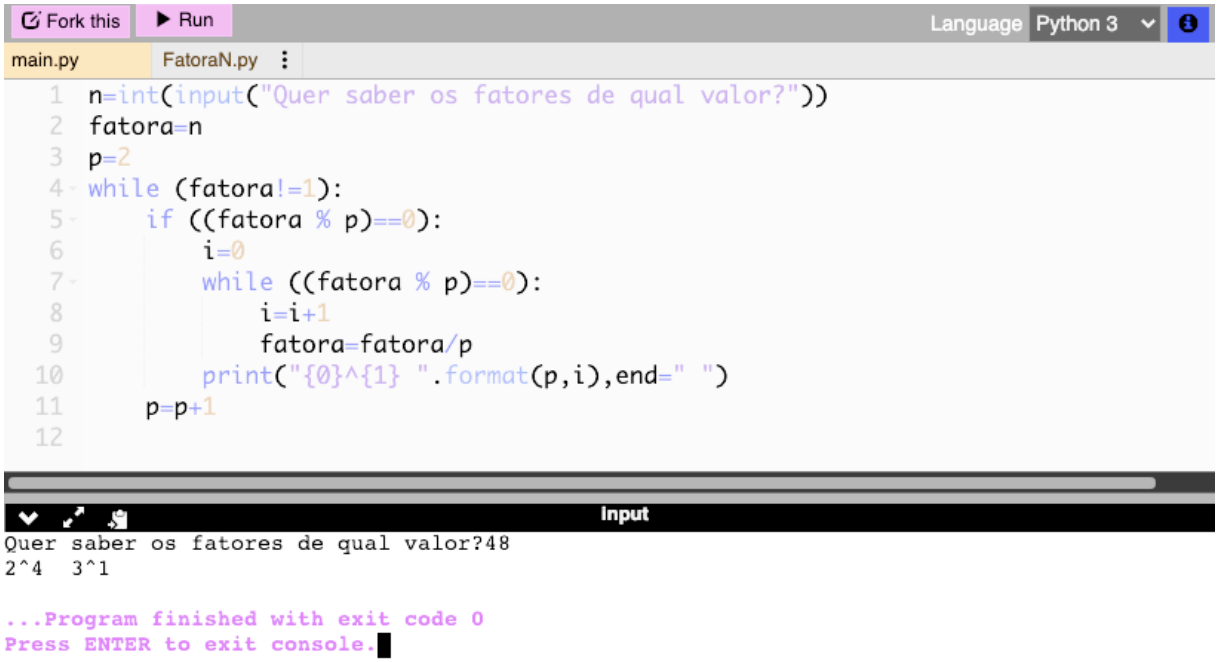

Figura 124 – Fatora N em Python (Elaborado pelo autor)

Nesse código foi utilizada uma saída formatada. Os números entre chaves definem a ordem da variável na lista de parâmetros passados no **format**.
Está disponível em: https://onlinegdb.com/bdNCa16op

B.31 Algoritmo 26 em Python - Base decimal para binária

A figura 125 mostra o algoritmo 26 codificado em Python.

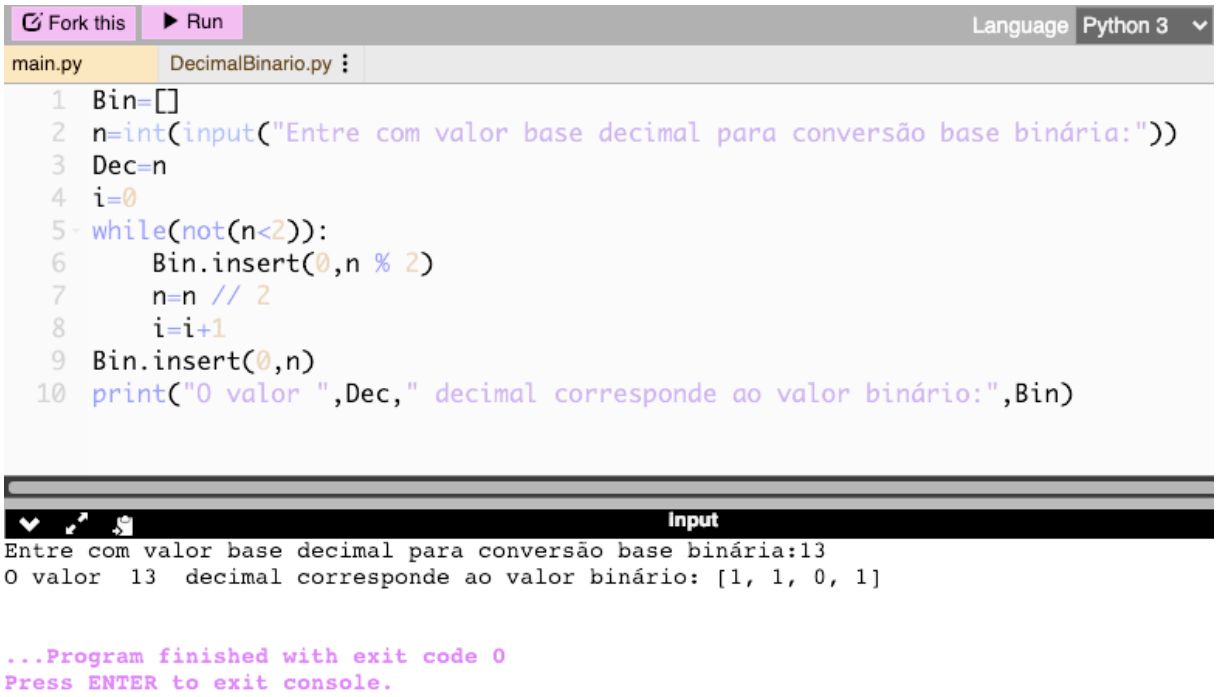

Figura 125 – Conversão decimal para binário em Python (Elaborado pelo autor)

O operador da divisão inteira em Python são duas barras (//). No código foi utilizado o método[1] **insert** para inserir na posição 0 o valor do resto da divisão por 2. Isso já ordena os restos em ordem inversa.
Está disponível em: https://onlinegdb.com/O9_aH1X2O

[1]conceito de Programação Orientada a Objetos, não abordado neste trabalho. Entenda como uma função aplicada à variável (objeto) Bin.

B.32 Algoritmo 27 em Python: Conversão binário para decimal

A figura 126 mostra o algoritmo 27 codificado em Python.

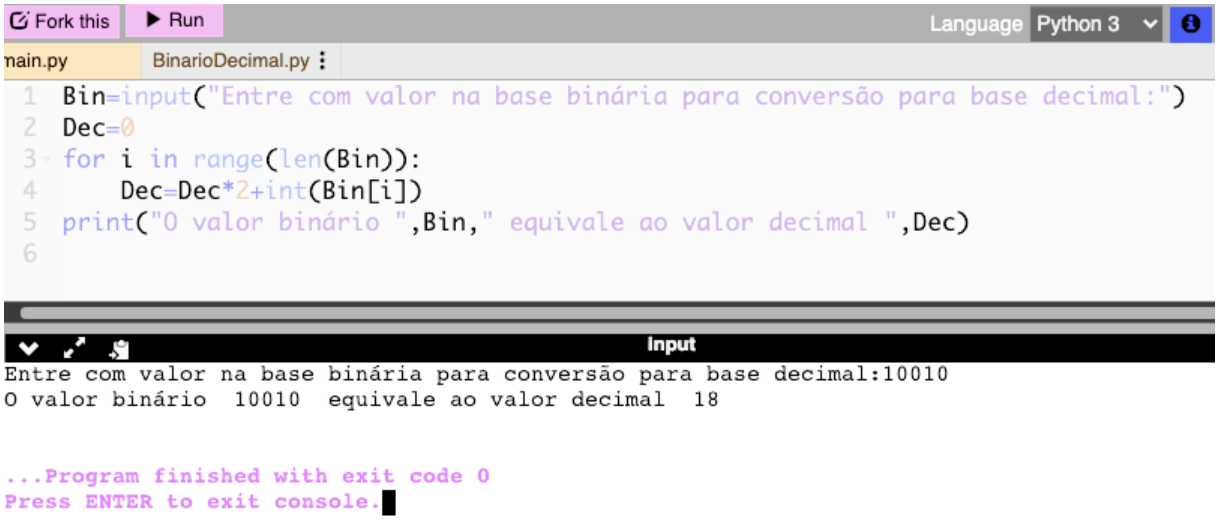

Figura 126 – Conversão binário para decimal em Python (Elaborado pelo autor)

A função **len**(Bin) retorna do tamanho de Bin e a função **int**(Bin[i]) retorna o valor inteiro correspondente ao caractere Bin[i].
Está disponível em: https://onlinegdb.com/fcr7IMjdQ

B.33 Algoritmo 28 em Python: Base decimal para hexadecimal

A figura 127 mostra o algoritmo 28 codificado em Python.

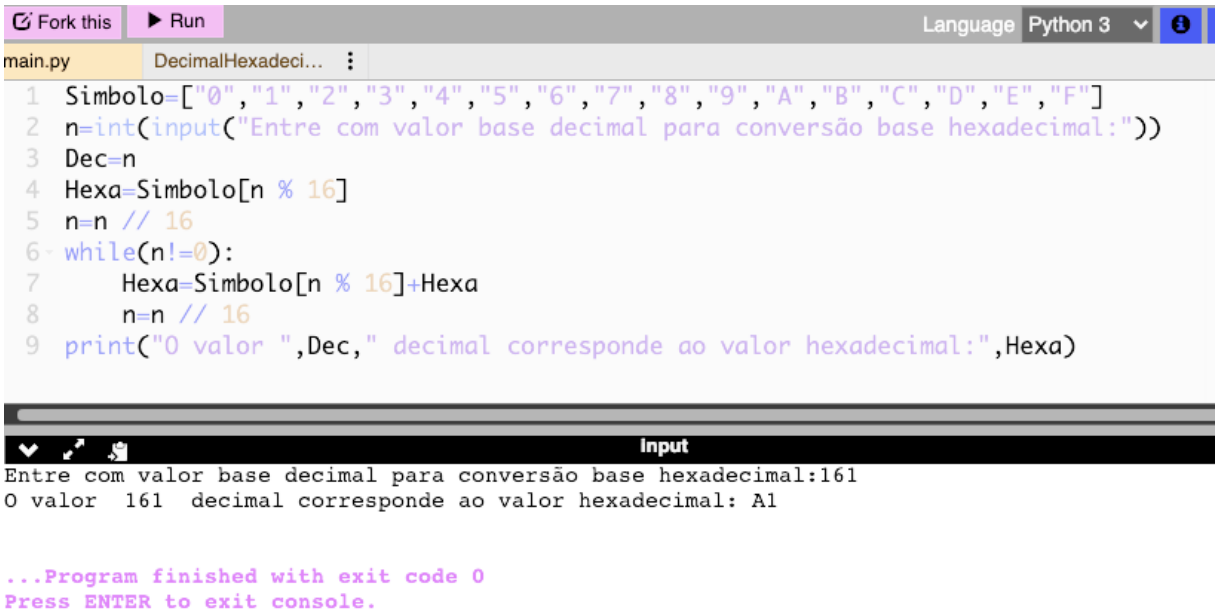

Figura 127 – Conversão decimal para hexadecimal em Python (Elaborado pelo autor)

A operação "$Hexa = Simbolo[n\%16] + Hexa$" concatena no início da cadeia de caracteres Hexa o caractere da lista Simbolo que está na posição (índice) $n\%16$.
Está disponível em: https://onlinegdb.com/KuHfsDqLd

B.34 Algoritmo 29 em Python: Conversão hexadecimal para decimal

A figura 128 mostra o algoritmo 29 codificado em Python.

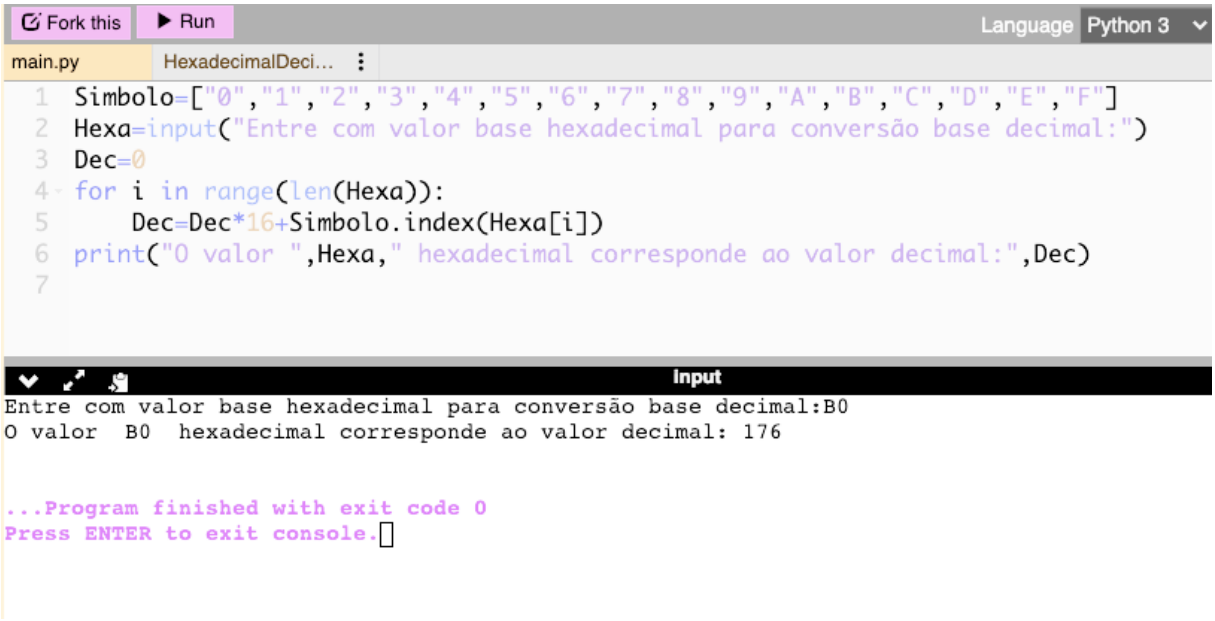

Figura 128 – Conversão hexadecimal para decimal em Python (Elaborado pelo autor)

O método[1] **index**(Hexa[i]) retorna a posição (índice) na lista Simbolo em que se encontra o valor Hexa[i], ou seja, faz a conversão de um caractere que representa um dígito Hexadecimal em um valor de 0 a 15 correspondente.
Está disponível em: https://onlinegdb.com/RyqlKDtpG

[1] conceito de Programação Orientada a Objetos, não abordado neste trabalho. Entenda como uma função aplicada à variável (objeto) Simbolo.