ESCOLA PREPARATÓRIA DE CADETES DO EXÉRCITO

ÁLGEBRA BOOLEANA E ALGORITMOS

Versão Digital

DISCIPLINA DE INTRODUÇÃO À COMPUTAÇÃO

2026

CONTEÚDO ADICIONAL

HTTPS://APOIOIC.COM.BR

HTTPS://SECRETARIAEBAULA.EB.MIL.BR/

O **APOIO IC** leva a uma página com o conteúdo digital das aulas e ferramentas de apoio.

No site do **EBAULA** há todo o conteúdo digital, listas de exercícios, exercícios resolvidos, questionários on-line, avisos gerais e orientações para as provas.

SOBRE ESTA APOSTILA

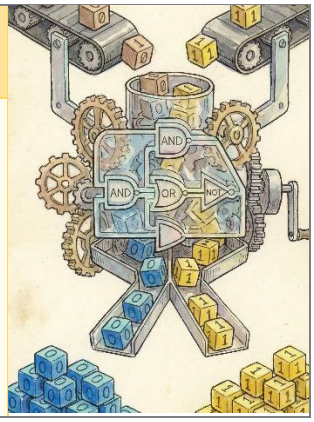
- Existem listas de exercícios ao final de cada capítulo. As soluções dos exercícios estarão disponíveis no Ambiente Virtual, após a semana de aula correspondente ao assunto.
- Há um anexo nesta apostila sobre o uso do Visualg, para teste dos algoritmos produzidos em pseudocódigo.
- Os algoritmos desta apostila são identificados no formato <cap>_<sequencial>, como o algoritmo “03_01 Area_Triangulo”, e há um arquivo correspondente em uma pasta do programa Visualg nomeado “03_01 Area_Triangulo.alg”.
- Há um anexo sobre a instalação e uso de um programa para codificação em Linguagem Python, além de um tutorial de como converter os algoritmos para esta linguagem.
- Os números entre colchetes, como [1], são referências bibliográficas, no formato padrão IEEE versão 11.29.2023.
- Versão desta apostila: 2.2 / 2026

Disciplina de Introdução à Computação. *Apostila de Álgebra Booleana e Algoritmos*. Campinas: Escola Preparatória de Cadetes do Exército / Exército Brasileiro, 2026.

SUMÁRIO

1. Álgebra Booleana

Introduz a Álgebra Booleana, base fundamental para o tratamento lógico de sistemas binários. Apresenta os principais operadores lógicos, E (AND), OU (OR) e NÃO (NOT), e suas respectivas tabelas-verdade, essenciais para compreender o funcionamento dos circuitos digitais. Além disso, explora os postulados, propriedades e teoremas da álgebra booleana, que possibilitam a simplificação de expressões lógicas, tornando mais eficiente o desenvolvimento e a implementação de sistemas digitais e programas computacionais.



A Álgebra Booleana é um formalismo para o tratamento lógico de sistemas binários, cujas variáveis podem assumir apenas dois valores: VERDADEIRO (V) ou FALSO (F). Recebeu essa denominação em homenagem ao matemático britânico George Boole, que no ano de 1847 introduziu os primeiros conceitos desta álgebra [1]. Quase um século depois veio a ser adotada como base da computação, devido à definição das operações a serem executadas por circuitos eletrônicos.

Essa álgebra é empregada na computação, pois todas as funções de um computador como abrir um arquivo, executar um vídeo ou acessar a internet, são executadas por meio de sinais elétricos percorrendo os circuitos eletrônicos do equipamento. Esses sinais elétricos normalmente assumem o valor de +5 Volts ou 0 Volts, podendo ser associados, respectivamente, aos valores lógicos V e F. Nesta disciplina, adotaremos os valores 1 para indicar um valor lógico VERDADEIRO e 0 para indicar um valor lógico FALSO.

Os circuitos eletrônicos do computador são implementados por meio de portas lógicas [2], que são basicamente componentes eletrônicos (transistores) que executam as operações lógicas. A álgebra booleana auxilia na simplificação desses circuitos, permitindo a implementação com o menor número possível de recursos (portas lógicas), reduzindo sua dimensão e custos.

A compreensão de seus operadores básicos, composição de uma expressão lógica, sua representação por tabela-verdade e simplificação, é um exercício importante para a fixação da lógica booleana, essencial para a lógica de programação.

1.1 Operadores

Os operadores lógicos (ou operadores booleanos) representam as funções que são aplicadas sobre uma ou mais variáveis (cada variável recebe um valor lógico) resultando em uma respectiva saída ou resultado (um único valor lógico).

Operador E (AND)

Realiza a operação denominada Multiplicação Lógica, que consiste em combinar os valores das variáveis de entrada para produzir uma única saída. O resultado é 1 somente quando todos os valores de entrada forem 1. Equivale a um circuito em série com duas chaves. Somente quando as duas estiverem fechadas (valor 1) que haverá um sinal na saída (valor 1).

A chave fechada no circuito, corresponde ao valor lógico 1 em um sistema lógico, enquanto a chave aberta corresponde a 0. A saída (no exemplo abaixo, a lâmpada) com energia corresponde à 1, e sem energia corresponde à 0.

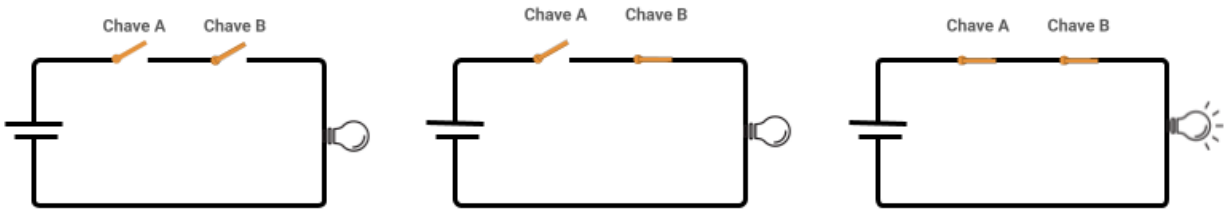


Figura 1: circuito em série.

A função booleana, do resultado produzido pelo operador E aplicado em duas variáveis, é representada a seguir:

$$S = A \cdot B$$

A tabela-verdade, que representa todas as possíveis combinações de valores das variáveis de entrada, indicando os respectivos resultados, é representada abaixo.

Tabela 1: Tabela-Verdade do Operador E

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

Esse operador pode ser aplicado a mais de duas variáveis, permanecendo a mesma regra de "somente quando todas as entradas forem 1, o resultado será 1". Ou seja, para a operação:

$$S = A \cdot B \cdot C$$

A saída S somente será 1 quando todas as entradas forem 1.

Atenção: Por critério didático desta disciplina, o operador de multiplicação lógica (E) **não deve ser omitido** nas expressões booleanas. Desta forma, **A.B** não será considerado equivalente a **AB** (como adotado em outros materiais sobre o assunto).

Operador OU (OR)

Executa a operação denominada Adição Lógica, que consiste em combinar os valores das variáveis de entrada para produzir uma única saída. O resultado é 1 quando, ao menos, um dos valores de entrada for 1. Equivale a um circuito em paralelo com duas chaves. Quando uma das duas estiver fechada (valor 1) haverá um sinal na saída (valor 1).

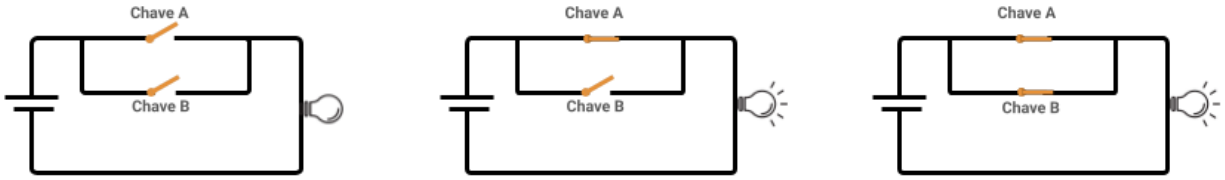


Figura 2: circuito em paralelo

A função booleana, do resultado produzido pelo operador **OU** aplicado em duas variáveis, é representada abaixo.

$$S = A + B$$

A tabela-verdade, que representa todas as possíveis combinações de valores das variáveis de entrada, indicando os respectivos resultados, é representada abaixo.

Tabela 2: Tabela-Verdade do Operador OU

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

Esse operador pode ser aplicado a mais de duas variáveis, permanecendo a mesma regra de "se ao menos uma entrada for 1, o resultado será 1". Ou seja, para a operação:

$$S = A + B + C$$

A saída S somente será 0 quando todas as entradas forem 0.

Operador NÃO (NOT)

Executa a operação denominada Complementação Lógica, que consiste em alterar o valor da variável de entrada. Se o valor de entrada for 0, o resultado é 1, e se o valor de entrada por 1, o resultado é 0. Equivale a um circuito com uma chave, conforme a figura abaixo. Quando a chave estiver aberta (valor 0) haverá um sinal na saída (valor 1).

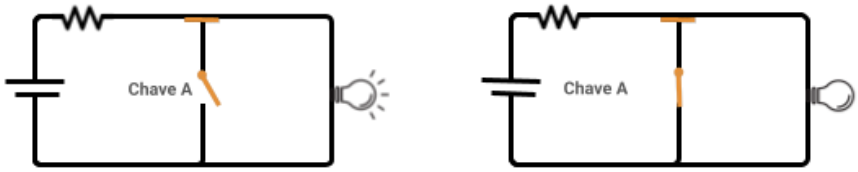


Figura 3: circuito inversor do sinal.

A função booleana, do resultado produzido pelo operador **NÃO**, é representada abaixo.

$$S = \overline{A}$$

A tabela-verdade, que representa todas as possíveis combinações de valores da variável de entrada, indicando os respectivos resultados, é representada abaixo.

Tabela 3: Tabela-Verdade do Operador NÃO

A	S
0	1
1	0

Esse operador é unário, ou seja, somente pode ser aplicado a uma variável.

Expressão Lógica

Uma expressão lógica (ou expressão booleana) é uma combinação de variáveis e operadores booleanos. Exemplos:

$$S = A \cdot B + C \cdot D$$

$$S = A + B \cdot \overline{C} + A$$

$$S = A + B + C$$

Uma expressão como $A + B \,.\, \bar{C}$ lê-se como “A ou B e não C”.

Precedência

Indica a ordem de resolução da expressão lógica, ou seja, quais operações devem ser realizadas primeiro.

Tabela 4: Ordem de Precedência

Ordem	S
1	Parênteses
2	NÃO
3	E
4	OU

Assim como em outros ramos da matemática, pode-se utilizar parênteses para alterar a precedências das operações, de forma que se deve começar a resolver as operações dentro dos parênteses mais internos. Não havendo parênteses, ou dentro de um mesmo nível de parênteses, primeiro se resolve a operação NÃO, seguido pela operação E, e por último a operação OU. Exemplo:

$S = A \,.\, B + (D + E \,.\, F \,.\, G) + \overline{C}$

$S = A \,.\, B + (D + S1) + \overline{C}$

$S = A \,.\, B + S2 + \overline{C}$

$S = A \,.\, B + S2 + S3$

$S = S4 + S2 + S3$

$S = S5$

S1 a S5 indicam as etapas de resolução. Havendo os valores lógicos (0 ou 1) definidos para cada uma das variáveis (A, B, C, D, E, F e G), basta substituir esses valores em cada variável, para se obter o valor da saída S (0 ou 1).

Por fim, pode-se haver uma expressão como:

$$S = \overline{A + B}$$

Em um caso como esse, deve-se resolver a operação OU primeiro. Havendo somente um valor resultante, aplica-se a operação de NEGAÇÃO sobre esse valor, uma vez que esse operador é unário.

1.2 Tabela-Verdade

Tabelas-verdade (TV) são recursos utilizados para visualizar de forma prática todas as combinações de valores de variáveis de uma expressão lógica e o valor do resultado para cada uma dessas combinações.

A tabela-verdade é construída para indicar as possíveis combinações de valores das variáveis, com os respectivos valores resultantes de saída. As primeiras colunas correspondem

a cada uma das variáveis de entrada. Cada coluna seguinte, corresponde a uma etapa da resolução da expressão. E a última coluna indica o resultado final.

Cada linha corresponde a uma combinação distinta de valores lógicos de entrada. Para saber a quantidade de linhas que deve haver em uma tabela-verdade, basta utilizar a expressão:

$$L = 2^n$$

Onde *L* é a quantidade de linhas, e *n* a quantidade de variáveis.

Exemplo:

$$S = A + B\,.\,(C + A).\overline{B}$$

Essa expressão possui 3 variáveis (A, B e C), portanto, terá um total de 8 (2^3) linhas. Dessa forma, as 3 primeiras colunas serão as variáveis A, B e C.

Para construir corretamente a tabela-verdade, é fundamental compreender a ordem de precedência dos operadores na expressão booleana. A resolução da expressão segue a ordem de precedência dos operadores: primeiramente, resolve-se a operação dentro dos parênteses (coluna 4). Em seguida, aplica-se a operação de negação (coluna 5). Depois, realiza-se a multiplicação dos três valores (coluna 6). Por fim, a operação de adição é executada (coluna 7), resultando no valor final da expressão. Veja como fica construída essa tabela-verdade:

Tabela 5: *Tabela*-Verdade da Expressão S

A	B	C	(C + A)	$\overline{B}$	B . (C + A). $\overline{B}$	S
0	0	0	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	1	1	0	1
1	0	1	1	1	0	1
1	1	0	1	0	0	1
1	1	1	1	0	0	1

Representa a expressão completa:

$A + B\,.\,(C + A).\overline{B}$

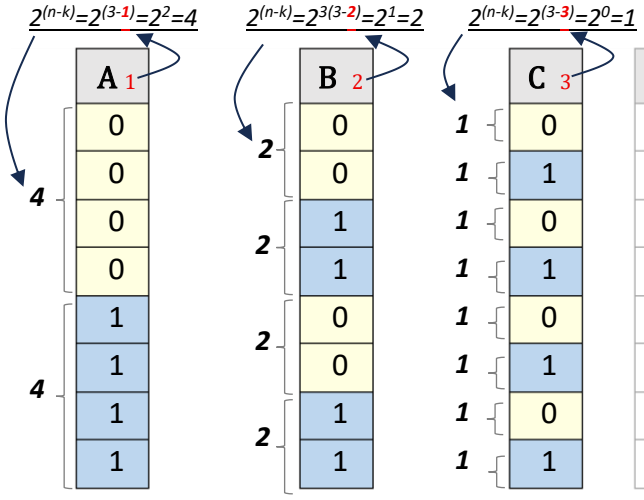
Perceba também que cada linha possui uma combinação diferente de valores de entrada. Como as variáveis booleanas só podem assumir valores 0 ou 1, as linhas da tabela-verdade devem cobrir todas as combinações possíveis. Tradicionalmente, as entradas são listadas em ordem binária crescente, começando com todos os zeros na primeira linha e terminando com todos os uns na última linha, garantindo que cada combinação seja única.

Observe o padrão de alternância de valores 0 e 1 nas colunas das variáveis de entrada da tabela-verdade (Tabela 6). Para uma expressão com 'n' variáveis, a alternância de valores para a k-ésima coluna é dada por $2^{(n-k)}$. Considerando a expressão com 3 variáveis:

- a primeira coluna (variável A), terá uma alternância de quatro ($2^{3-1} = 2^2$) valores 0 (zero) seguidos de 4 valores 1 (um).
- A segunda coluna (variável B), terá uma alternância de dois ($2^{3-2} = 2^1$) valores 0 (zero) seguidos de 2 valores 1 (um), até o preenchimento total da coluna.
- A terceira coluna (variável C), terá uma alternância de um ($2^{3-3} = 2^0$) valor 0 (zero) seguido de um valor 1 (um).

A alternância de valores nas colunas das variáveis, que deve sempre ser seguida na construção de uma TV, está expressa na Tabela 6.

Tabela 6: alternância de valores nas colunas de valores das variáveis



(C + A)	$\overline{B}$	B . (C + A) . $\overline{B}$	S
0	1	0	0
1	1	0	0
0	0	0	0
1	0	0	0
1	1	0	1
1	1	0	1
1	0	0	1
1	0	0	1

Observe que para a resolução de cada uma das etapas, basta verificar qual é o valor das variáveis, em cada linha, e realizar a devida operação substituindo com os valores associados para cada entrada.

A obtenção (desenvolvimento) da tabela-verdade, a partir de uma expressão lógica, é conhecida como “Avaliação de Expressão Booleana”.

1.3 Postulados, Propriedades e Teoremas

A álgebra booleana apresenta postulados (axiomas) a partir dos quais são definidas propriedades. Esses recursos podem ser utilizados no processo de simplificação de expressões lógicas, como será visto a seguir.

Postulados

Adição Lógica (OU)			
$0 + 0 = 0$	$0 + 1 = 1$	$1 + 0 = 1$	$1 + 1 = 1$

Multiplicação Lógica (E)			
$0 . 0 = 0$	$0 . 1 = 0$	$1 . 0 = 0$	$1 . 1 = 1$

Complemento (NÃO)	
Se $A = 1$, então $\overline{A} = 0$	Se $A = 0$, então $\overline{A} = 1$

Propriedades

Tabela 7: Propriedades definidas a partir dos postulados

Propriedade	Complemento	Adição	Multiplicação
Identidade	$\overline{\overline{A}} = A$	$A + 0 = A$ $A + 1 = 1$ $A + A = A$ $A + \overline{A} = 1$	$A \,.\, 0 = 0$ $A \,.\, 1 = A$ $A \,.\, A = A$ $A \,.\, \overline{A} = 0$
Comutativa		$A + B = B + A$	$A \,.\, B = B \,.\, A$
Associativa		$A + (B + C) =$ $(A + B) + C =$ $A + B + C$	$A.\,(B.\,C) =$ $(A.\,B).\,C =$ $A.\,B.\,C$
Distributiva		$A + (B.\,C) =$ $(A + B).\,(A + C)$	$A.\,(B + C) =$ $A.\,B + A.\,C$

Tabela 8: Outras propriedades

Absorção	$A + (A.\,B) = A$ $A.\,(A + B) = A$
Identidade	$A + \overline{A}.\,B = A + B$
	$\overline{A} + A.\,B = \overline{A} + B$

Teorema de De Morgan

O complemento de um produto lógico equivale à soma das negações de cada variável.

Para duas variáveis:

$$\overline{A.\,B} = \overline{A} + \overline{B}$$

Para n variáveis:

$$\overline{A.\,B.\,C.\,(\ldots).\,Z} = \overline{A} + \overline{B} + \overline{C} + (\ldots) + \overline{Z}$$

O complemento de uma soma lógica equivale ao produto das negações de cada variável.

Para duas variáveis:

$$\overline{A + B} = \overline{A} \,.\, \overline{B}$$

Para n variáveis:

$$\overline{A + B + C + (\ldots) + Z} = \overline{A}.\ \overline{B}.\,\overline{C}.\,(\ldots).\,\overline{Z}$$

1.4 Simplificação de Expressões

As expressões booleanas, que serão convertidas em circuitos digitais, podem se tornar extensas e complexas. Existem técnicas para reduzir essas expressões, para se obter circuitos equivalentes que apresentarão as mesmas saídas em relação à expressão inicial.

Fatoração

São aplicados **os postulados, propriedades e teoremas** de forma a simplificar a expressão.

Exemplo 1

$S = A.B.C + A.\overline{C} + A.\overline{B}$	Aplicada esta propriedade →	
$S = A.(B.C + \overline{C} + \overline{B})$	← *Resulta em*	Distributiva da multiplicação
$S = A.(\overline{B} + B.C + \overline{C})$		Comutativa da Adição
$S = A.(\overline{B} + C + \overline{C})$		Identidade
$S = A.(\overline{B} + 1)$		Identidade da adição
$S = A.(1)$		Identidade da adição
$S = A$		Identidade da multiplicação

A aplicação da propriedade é sobre a linha anterior, resultando na próxima linha, seguindo até o momento de maior simplificação.

Exemplo 2

$S = \overline{C}.(\overline{D.B} + B.D) + D.C$	Aplicada esta propriedade →	
$S = \overline{C}.(\overline{D.B} + D.B) + D.C$	← *Resulta em*	Comutativa da multiplicação
$S = \overline{C}.(1) + D.C$		Identidade da adição
$S = \overline{C} + D.C$		Identidade da multiplicação
$S = \overline{C} + D$		Identidade

Uma solução alternativa para o exemplo 2:

$S = \overline{C}.(\overline{D.B} + B.D) + D.C$	Aplicada esta propriedade →	
$S = \overline{C}.(\overline{D} + \overline{B} + D.B) + D.C$	← *Resulta em*	De Morgan
$S = \overline{C}.(\overline{D} + \overline{B} + D) + D.C$		Identidade
$S = \overline{C}.(1 + \overline{B}) + D.C$		Identidade da adição
$S = \overline{C}.(1) + D.C$		Identidade da adição
$S = \overline{C} + D.C$		Identidade da multiplicação
$S = \overline{C} + D$		Identidade

Exemplo 3

$S = B + \overline{\overline{C} + \overline{B}}$	Aplicada esta propriedade →	
$S = B + \overline{\overline{C}.\overline{B}}$	← *Resulta em*	De Morgan
$S = B + C.B$		Identidade do Complemento
$S = B$		Absorção

1.5 Mapas de Karnaugh (MK)

Como alternativa ao método de simplificação por fatoração, há um método baseado na identificação visual de agrupamentos de mintermos, por meio de tabelas denominadas mapas de Karnaugh (ou, resumidamente, mapa K). Método inicialmente desenvolvido por Edward W. Veitch, foi aperfeiçoado pelo físico e matemático Maurice Karnaugh na década de 50 [3], para simplificação de circuitos lógicos, essencial para a eletrônica.

A aplicação desse método consiste em utilizar um diagrama, com as variáveis identificadas ao seu redor, sendo uma saída correspondente à intersecção de cada conjunto de valores de variáveis possíveis. Dessa forma, deve-se preencher o diagrama com 0 ou 1, de acordo com o resultado de cada linha da tabela-verdade. O passo seguinte é identificar agrupamentos (grupos de 32, 16, oitavas, quadras ou pares de casas) de resultados com o valor 1. Esses agrupamentos correspondem às variáveis, e são adicionadas para formar a expressão final, simplificada.

Os mapas de Karnaugh podem ser utilizados para muitas variáveis, porém, o emprego deste método para até 6 variáveis é o mais usual e costuma ser o suficiente para a maioria das situações de simplificação de circuitos eletrônicos.

A seguir, serão demonstradas as construções de mapas de Karnaugh para 2, 3 e 4 variáveis. Começando pelo desenho do mapa, a transposição de valores de saída da TV, os agrupamentos dos valores 1, o relacionamento da região dos agrupamentos com a expressão correspondente e finalizando com uma **soma de produtos** que será a expressão final simplificada.

Mapa de Karnaugh para 2 Variáveis

Considere uma expressão S com duas variáveis: A e B.

Linha	A	B	(...)	S
1	0	0	---	S_1
2	0	1		S_2
3	1	0		S_3
4	1	1		S_4

	$\overline{B}$	B
$\overline{A}$	$\overline{A}\ \overline{B}$ S_1 Linha 1	$\overline{A}\ B$ S_2 Linha 2
A	$A\ \overline{B}$ S_3 Linha 3	$A\ B$ S_4 Linha 4

Figura 4: tabela-verdade e mapa de Karnaugh para 2 variáveis

Primeiro passo: observar os valores dos resultados para cada linha da tabela-verdade (coluna "S"), e preencher adequadamente no mapa de Karnaugh. Veja um exemplo a seguir:

Linha	A	B	(...)	S
1	0	0	---	1
2	0	1		0
3	1	0		1
4	1	1		1

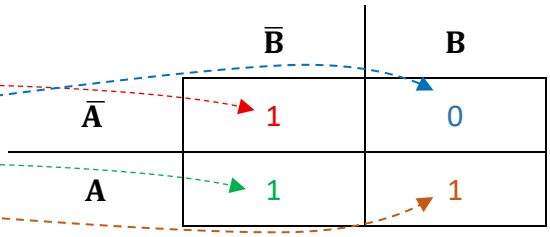


Figura 5: transpor valores de MK2V para o diagrama de Karnaugh

Segundo passo: agrupar as regiões onde S = 1 na menor quantidade de PARES possíveis. Um par é uma região onde há dois valores 1 vizinhos (na horizontal ou vertical). Nesse caso, um mesmo valor 1 pode estar em mais de um par (Figura 6). É preferível formar pares do que deixar um valor 1 isolado (Figura 7). Apenas será considerado um valor 1 isoladamente se tiver um único valor 1 como resultado na tabela-verdade, ou se houver valores 1 em diagonal no mapa

de Karnaugh, o que impede a formação de pares (Figura 11). No exemplo atual, o agrupamento seria como o demonstrado na Figura 6:

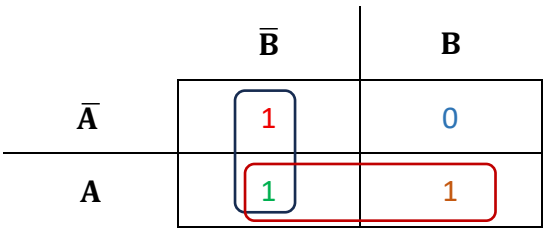


Figura 6
*Agrupamentos de pares, com sobreposição (**correto**)*

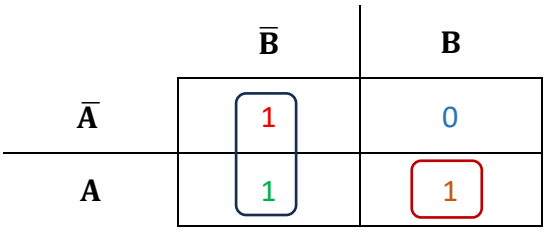
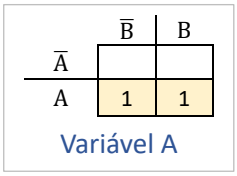
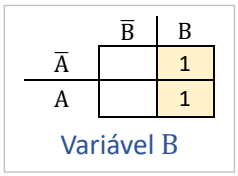
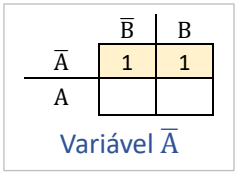


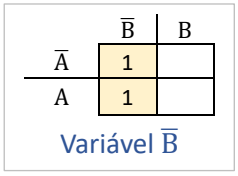
Figura 7
*Um par e um valor isolado (**incorreto**)*

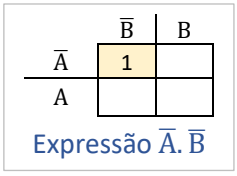
Terceiro passo: verificar a qual expressão de variáveis que corresponde a região demarcada, e realizar uma soma de todas essas expressões, resultando na expressão final simplificada.

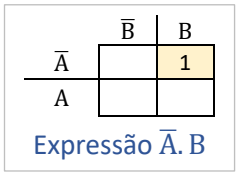


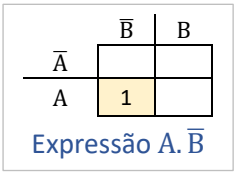












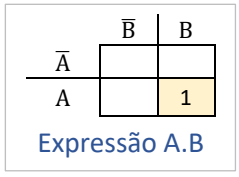


Figura 8: expressão correspondentes a cada região em um mapa de Karnaugh para 2 variáveis

Considerando o exemplo anterior:

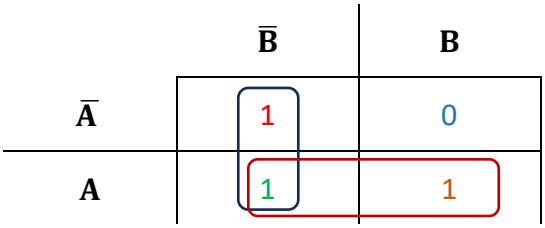


Figura 9
Os dois agrupamentos em par, ao lado, correspondem às regiões de $\overline{A}$ e de $\overline{B}$. Portanto, a soma destes termos resulta na expressão: $A + \overline{B}$

Outro exemplo:

As saídas da tabela-verdade à esquerda correspondem a um mapa de Karnaugh, à direita:

Linha	A	B	S
1	0	0	1
2	0	1	0
3	1	0	0
4	1	1	1

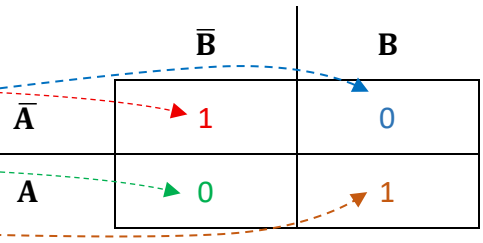


Figura 10: mapa de Karnaugh do exemplo com 2 variáveis

Construído o mapa de Karnaugh, o próximo passo é identificar as regiões e associar a variável ou expressão equivalente a cada região:

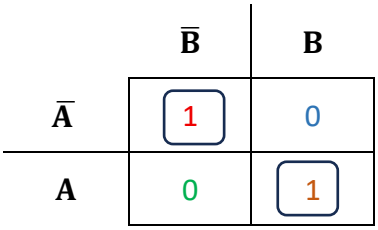


Figura 11

Dois valores 1 isolados, em diagonal, impossibilitando agrupamento em par. Um valor 1 corresponde ao termo $\overline{A}.\overline{B}$. O outro valor 1 corresponde ao termo $A.B$. A soma dos termos será: $\overline{A}.\overline{B} + A.B$

Depreende-se dos dois exemplos apresentados que a lógica fundamental na simplificação de uma função booleana, utilizando Mapas de Karnaugh, é a busca pelos maiores agrupamentos possíveis de células com valor '1'. A razão para maximizar o tamanho dos agrupamentos reside na regra de simplificação booleana que permite a eliminação de variáveis.

Mapa de Karnaugh para 3 Variáveis

Considere uma expressão S com três variáveis: A, B e C. A tabela-verdade com 3 variáveis possui 8 linhas (saídas de S1 até S8). Dessa forma, o mapa de Karnaugh é preenchido conforme a Figura 12.Cada bloco de 4 linhas de saída da TV corresponde a uma linha com 4 células no MK. Notar também que, quando um bloco de 4 saídas da TV é disposto no MK, há sempre uma inversão do penúltimo com o último valor (seta vermelha), S_4 seguido de S_3; ou S_8 seguido de S_7:

Linha	A	B	C	S
1	0	0	0	S_1
2	0	0	1	S_2
3	0	1	0	S_3
4	0	1	1	S_4
5	1	0	0	S_5
6	1	0	1	S_6
7	1	1	0	S_7
8	1	1	1	S_8

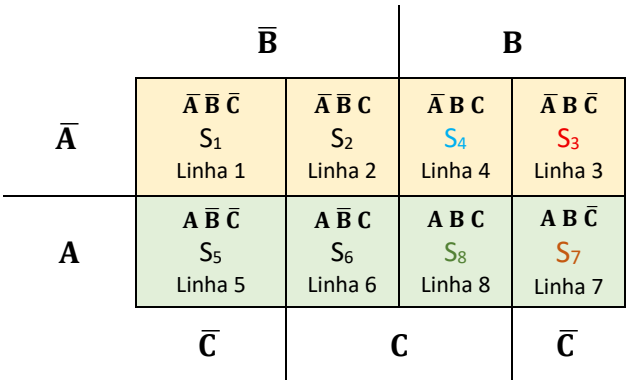
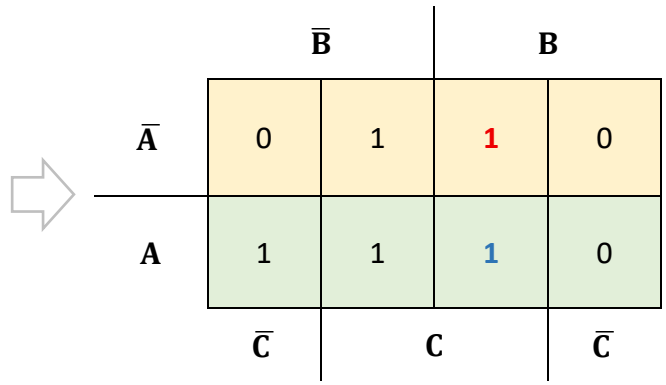


Figura 12: relação de valores de saída da tabela-verdade e mapa de Karnaugh para 3 variáveis

Primeiro passo: Observar os valores dos resultados para cada linha, e preencher adequadamente no mapa de Karnaugh. Observe um exemplo:

Linha	A	B	C	S
1	0	0	0	0
2	0	0	1	1
3	0	1	0	0
4	0	1	1	1
5	1	0	0	1
6	1	0	1	1
7	1	1	0	0
8	1	1	1	1



Segundo passo: Agrupar as regiões onde S = 1 na menor quantidade de QUADRAS possíveis. Uma quadra é uma região onde há quatro valores 1 vizinhos (na horizontal ou vertical, ou dispostos em uma região em formato de quadrado). Não sendo possível agrupar em quadras, agrupa-se um valor 1 em par. E em último caso, deixa-se um valor 1 isolado. Atenção para a quadra e pares que estão "quebrados", dispostos em áreas distintas no mapa. Observe que a região correspondente ao ($\overline{C}$) está desdobrada nas duas extremidades (Figura 13).

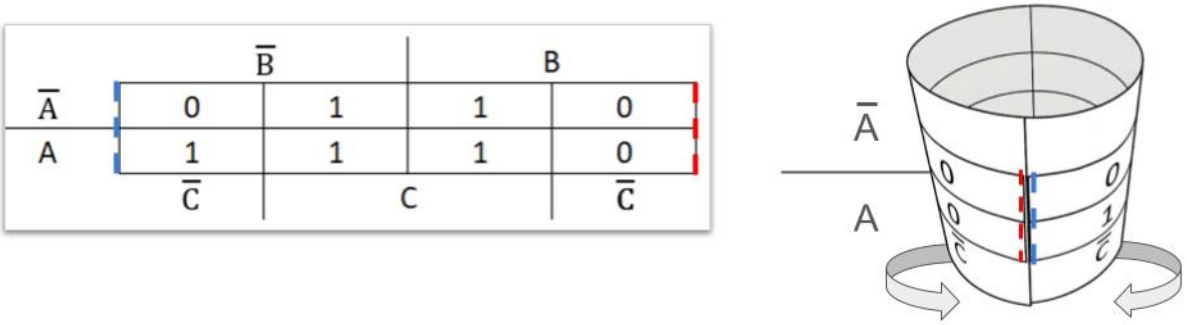
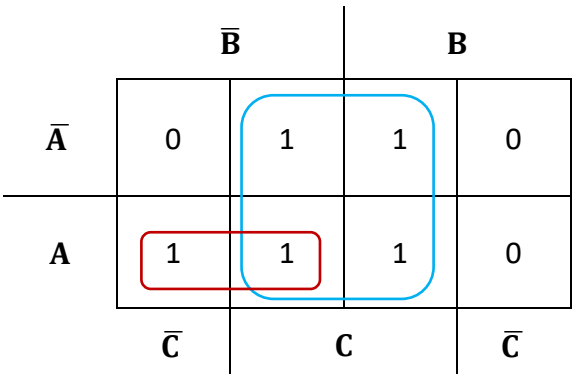


Figura 13: áreas desdobradas em um mapa de Karnaugh de 3 variáveis



Procure evidenciar com clareza cada uma das regiões. No MK ao lado foram identificados dois agrupamentos: uma quadra e um par, com um elemento de intersecção.

Figura 14: identificação de agrupamentos em um MK de 3 variáveis

Terceiro passo: verificar a qual expressão corresponde a região demarcada, e realizar uma soma de todas essas expressões, resultando na expressão final simplificada.

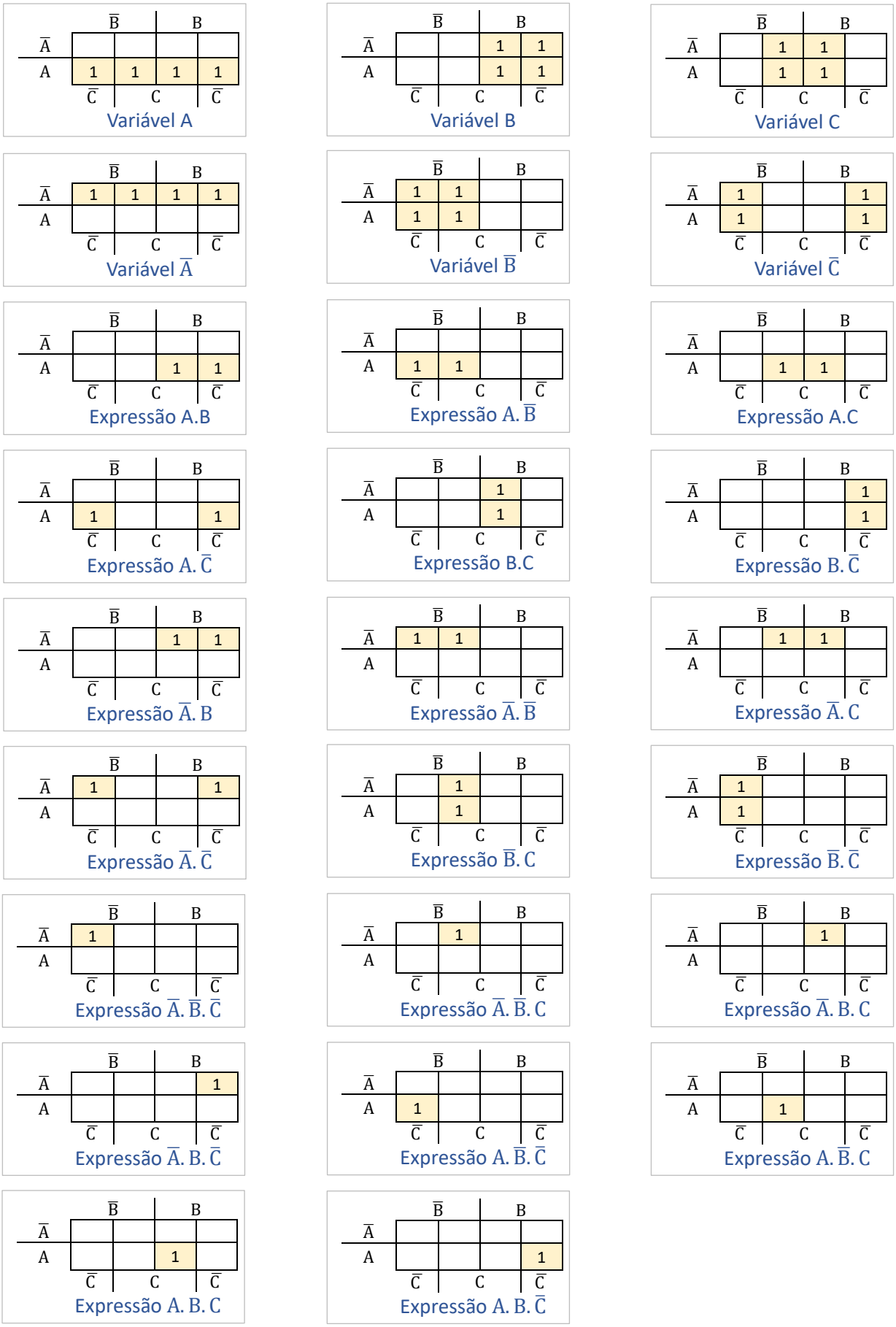


Figura 15: regiões do mapa de Karnaugh para 3 variáveis

É necessário, então, identificar as expressões correspondentes a cada agrupamento, com base no exposto na Figura 15.

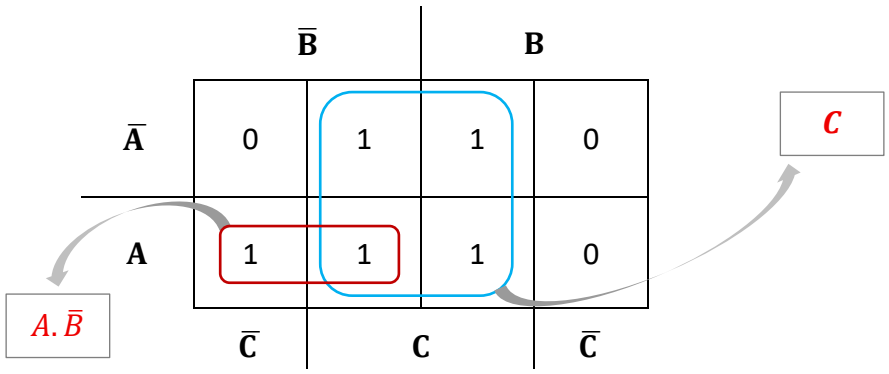


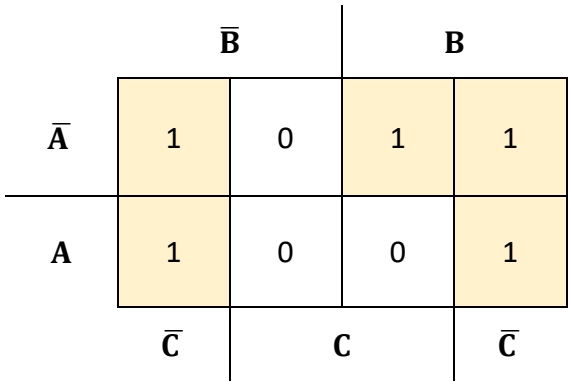
Figura 16: agrupamentos e identificação de regiões correspondentes no MK com 3 variáveis

A quadra 1 corresponde a C. E o par 2 corresponde à $A.\bar{B}$ Nesse caso, a expressão final simplificada é definida como:

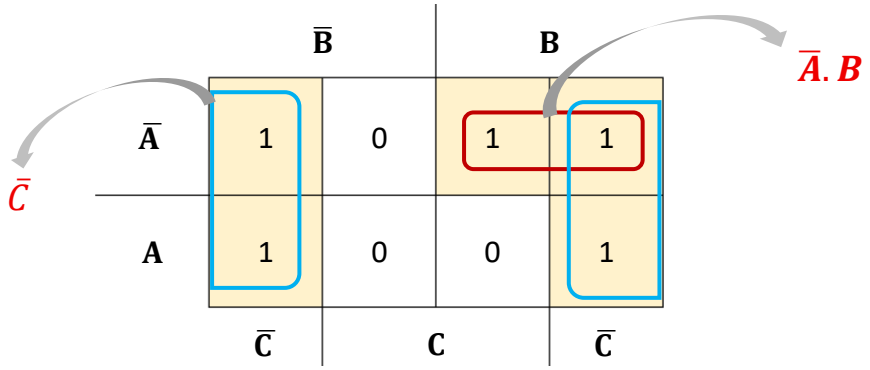
$$S = C + A.\bar{B}$$

Neste outro exemplo, o MK foi preenchido conforme os valores de saída da TV:

Linha	A	B	C	S
1	0	0	0	1
2	0	0	1	0
3	0	1	0	1
4	0	1	1	1
5	1	0	0	1
6	1	0	1	0
7	1	1	0	1
8	1	1	1	0



Depois, foram identificadas as expressões correspondentes a cada agrupamento, com base na Figura 15.



A quadra 1 corresponde à $\bar{C}$, que é desdobrada no mapa, e o par 2 corresponde à $\bar{A}.B$. Nesse caso, a expressão final simplificada é definida como:

$$S = \bar{C} + \bar{A}.B$$

Mapa de Karnaugh para 4 Variáveis

Considere uma expressão S com quatro variáveis: A, B, C e D. A tabela-verdade com 4 variáveis possui 16 linhas (saídas de S_1 até S_{16}). Dessa forma, o mapa de Karnaugh é preenchido conforme a Figura 17. Cada bloco de 4 linhas de saída da TV corresponde a uma linha com 4 células no MK, sendo que a 3ª e 4ª linhas são invertidas (seta azul). Notar também que, quando um bloco de 4 saídas da TV é disposto no MK, há sempre uma inversão do penúltimo com o último valor (seta vermelha), por exemplo S_4 seguido de S_3; ou S_{16} seguido de S_{15}:

Linha	A	B	C	D	S
1	0	0	0	0	S_1
2	0	0	0	1	S_2
3	0	0	1	0	S_3
4	0	0	1	1	S_4
5	0	1	0	0	S_5
6	0	1	0	1	S_6
7	0	1	1	0	S_7
8	0	1	1	1	S_8
9	1	0	0	0	S_9
10	1	0	0	1	S_{10}
11	1	0	1	0	S_{11}
12	1	0	1	1	S_{12}
13	1	1	0	0	S_{13}
14	1	1	0	1	S_{14}
15	1	1	1	0	S_{15}
16	1	1	1	1	S_{16}

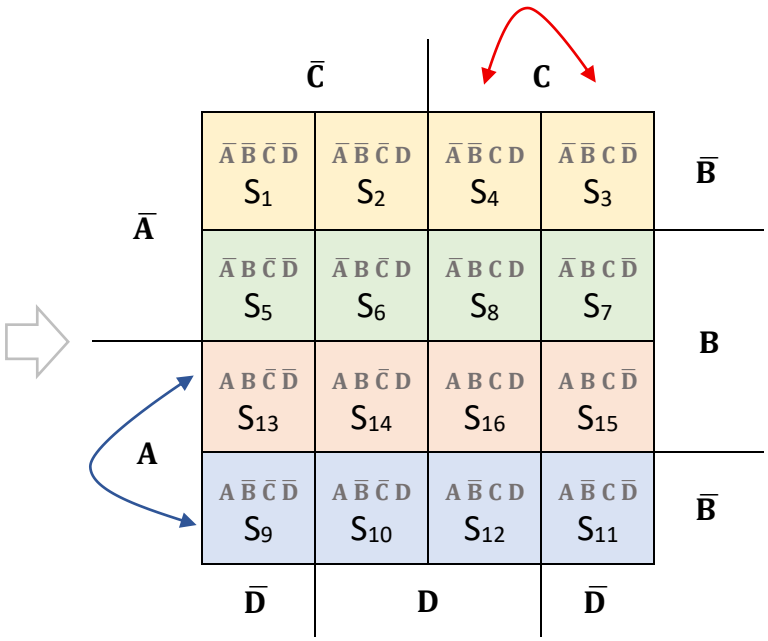
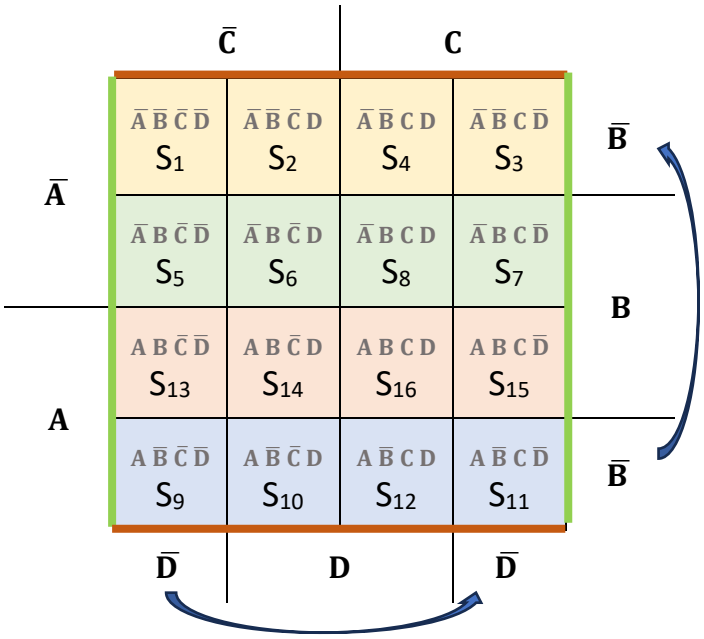
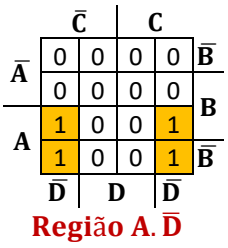


Figura 17: mapa de Karnaugh para 4 variáveis



Observe que as regiões referentes a $\bar{B}$ e a $\bar{D}$ possuem áreas não contíguas (desdobradas) no MK. Isso deve ser considerado nos agrupamentos de oitavas, quadras e duplas. Abaixo um exemplo de agrupamento (quadra) em área desdobrada:

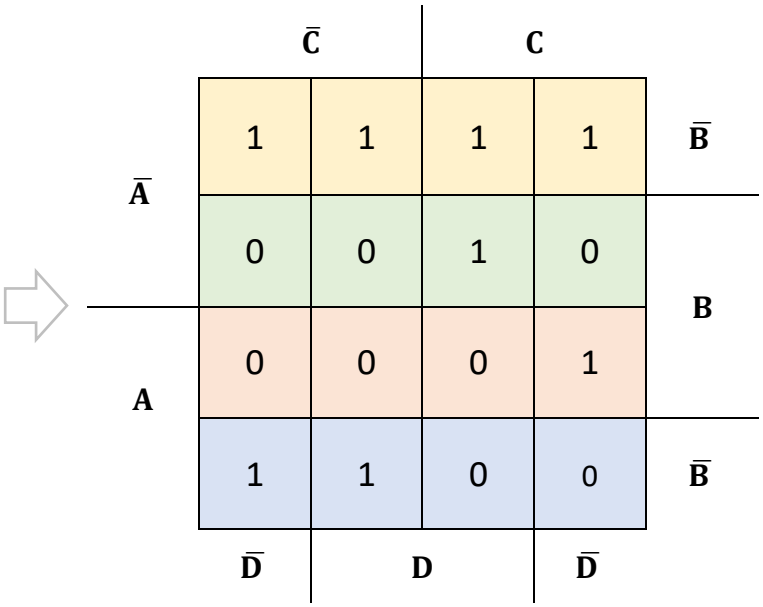


Região A. $\bar{D}$

Figura 18: áreas desdobradas em mapa de Karnaugh para 4 variáveis

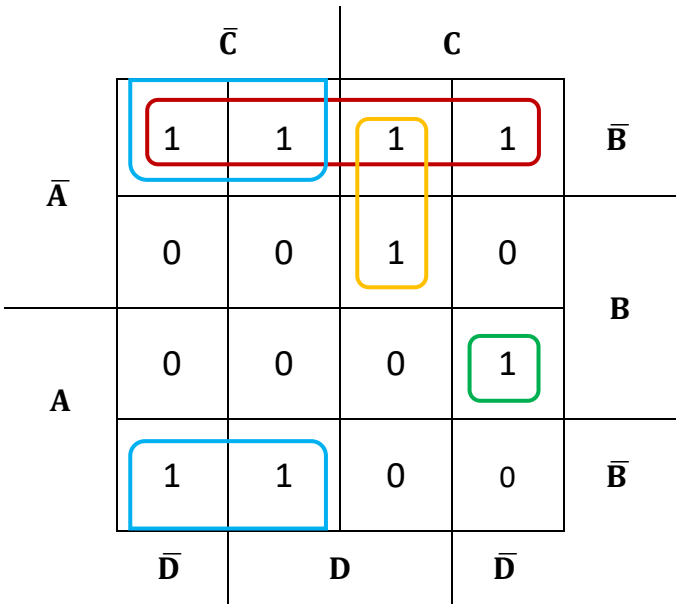
Primeiro passo: Observar os valores dos resultados para cada linha e preencher adequadamente no mapa de Karnaugh.

Linha	A	B	C	D	S
1	0	0	0	0	1
2	0	0	0	1	1
3	0	0	1	0	1
4	0	0	1	1	1
5	0	1	0	0	0
6	0	1	0	1	0
7	0	1	1	0	0
8	0	1	1	1	1
9	1	0	0	0	1
10	1	0	0	1	1
11	1	0	1	0	0
12	1	0	1	1	0
13	1	1	0	0	0
14	1	1	0	1	0
15	1	1	1	0	1
16	1	1	1	1	0

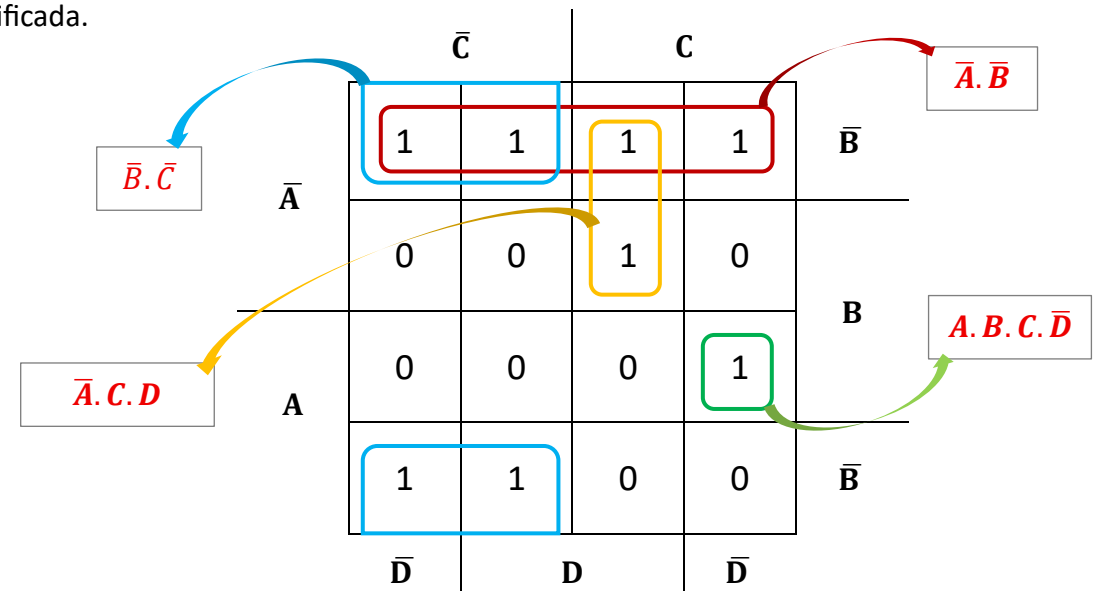


Segundo passo: Agrupar as regiões onde S = 1 na menor quantidade de OITAVAS possíveis. Uma oitava é uma região onde há oito valores 1 vizinhos (dispostos em uma região de 2x4 ou 4x2 casas). Não sendo possível agrupar em oitavas, agrupa-se em quadras (2x2, 1x4 ou 4x1). Ainda não sendo possível, agrupa-se em um par (2x1 ou 1x2)e, em último caso, deixa-se um valor 1 isolado. Atenção para as oitavas, quadras e pares que estão "quebrados", dispostos em áreas distintas no mapa.

Procure evidenciar com clareza cada uma das regiões. No MK ao lado foram identificados quatro agrupamentos: duas quadras (azul e vermelho), com duas células de intersecção. Além destes, um par (laranja) e um valor um isolado (verde).



Terceiro passo: verificar a qual expressão de variáveis que corresponde a região demarcada, e realizar uma soma de todas essas expressões, resultando na expressão final simplificada.



Assim, as quadras correspondem às regiões $\bar{C}.\bar{B}$ (desdobrada no mapa) e $\bar{A}.\bar{B}$. O par corresponde à região $\bar{A}.C.\bar{D}$. O valor isolado corresponde à região $A.B.C.\bar{D}$ A expressão final deve ser construída com a soma, na ordem, das oitavas, quadras, duplas e valores isolados. Assim, a expressão final simplificada é definida como:

$$S = \bar{A}.\bar{B} + \bar{B}.\bar{C} + \bar{A}.C.D + A.B.C.\bar{D}$$

Mapa de Karnaugh para 5 Variáveis

Considere uma expressão S com cinco variáveis: A, B, C, D e E.

A tabela-verdade com 5 variáveis possui 32 linhas (saídas de S1 até S32).

Dessa forma, o mapa de Karnaugh é preenchido conforme a figura abaixo.

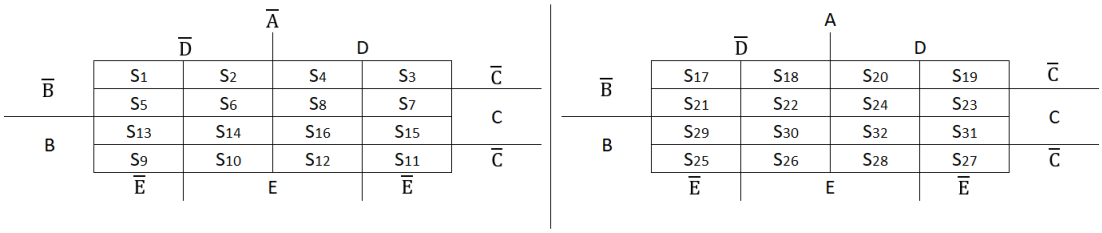


Figura 19: tabela-verdade e mapa de Karnaugh para 5 variáveis

Primeiro passo: Observar os valores dos resultados para cada linha, e preencher adequadamente no mapa de Karnaugh.

Segundo passo: Agrupar as regiões onde S = 1 na menor quantidade de GRUPOS DE 16 CASAS possíveis. Esse agrupamento consiste em uma região onde há dezesseis valores 1 vizinhos (dispostos em uma região de 4x4 ou 2x8 casas). Não sendo possível agrupar em grupos de 16, agrupa-se em oitavas. Não havendo mais oitavas, agrupa-se em quadras. Ainda não sendo possível, agrupa-se em um par. E em último caso, deixa-se um valor 1 isolado. Atenção para grupos que estão "quebrados", dispostos em áreas distintas no mapa, e, principalmente, grupos com valores divididos entre as duas áreas do mapa.

Terceiro passo: verificar a qual expressão de variáveis que corresponde a região demarcada, e realizar uma soma de todas essas expressões, resultando na expressão final simplificada.

Mapa de Karnaugh para 6 Variáveis

Considere uma expressão S com seis variáveis: A, B, C, D, E e F.

A tabela-verdade com 6 variáveis possui 64 linhas (saídas de S1 até S64).

Dessa forma, o mapa de Karnaugh é preenchido conforme a figura abaixo.

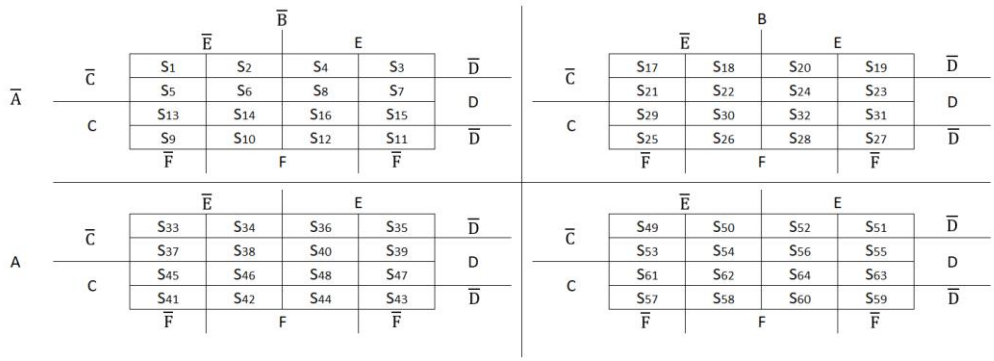


Figura 20: tabela-verdade e mapa de Karnaugh para 6 variáveis

Primeiro passo: Observar os valores dos resultados para cada linha, e preencher adequadamente no mapa de Karnaugh.

Segundo passo: Agrupar as regiões onde S = 1 na menor quantidade de GRUPOS DE 32 CASAS possíveis. Esse agrupamento consiste em uma região onde há trinta e dois valores 1 vizinhos (dispostos em uma região de 4x8 ou 2x16 casas). Não sendo possível agrupar em grupos de 32, agrupa-se em grupos de 16. Em seguida, procura-se agrupar em oitavas. Não havendo mais oitavas, agrupa-se em quadras. Ainda não sendo possível, agrupa-se em um par. E em último caso, deixa-se um valor 1 isolado. Atenção para grupos que estão "quebrados", dispostos em áreas distintas no mapa, e, principalmente, grupos com valores divididos entre as quatro (ou pelo menos duas) áreas do mapa.

Terceiro passo: verificar a qual expressão de variáveis que corresponde a região demarcada, e realizar uma soma de todas essas expressões, resultando na expressão final simplificada.

1.6 Forma Normal

Caso haja a tabela-verdade da expressão, com a indicação de cada valor de saída para todas as combinações de valores de entrada, é possível obter a expressão booleana. A obtenção da expressão lógica a partir da tabela-verdade, é conhecido como "Derivação de Expressão Booleana".

Essa expressão pode ser composta a partir de diversos operadores além dos já apresentados, como os operadores XOR (OU Exclusivo) e o NAND (Not And ou Não E). Porém, quando a expressão é composta apenas pelos operadores E, OU e NÃO, é dito que a expressão está na Forma Normal (FN), uma forma padronizada de representar expressões booleanas.

Forma Normal Conjuntiva (FNC)

Uma expressão está na FNC, se:

- Está na Forma Normal, ou seja, contém apenas os operadores E, OU e NÃO;
- Não há dupla negação, ou seja, deve-se aplicar a propriedade de identidade do complemento;

- O operador de negação não incide indiretamente sobre uma variável, ou seja, deve-se aplicar o teorema de De Morgan;
- O operador OU não incide sobre o operador E, ou seja, A OU (B E C), deve receber a propriedade distributiva de adição, ficando (A OU B) E (A OU C).

Também conhecida como Produto de Somas, ou Produto de Maxtermos, a FNC corresponde a identificar, na tabela-verdade, as saídas que apresentam valor 0, e realizar uma multiplicação (conjunção) das respectivas combinações de entrada, sendo, para cada entrada, somados (disjunção) o valor de cada variável, sendo realizado o complemento para as variáveis com valor 1. Exemplo:

*Tabela 9:*Tabela-verdade de uma expressão booleana S

Linha	A	B	C	S
1	0	0	0	0
2	0	0	1	1
3	0	1	0	0
4	0	1	1	1
5	1	0	0	1
6	1	0	1	0
7	1	1	0	0
8	1	1	1	1

Primeiro passo: identificar as saídas que são 0. No exemplo acima, corresponde às linhas 1, 3, 6 e 7.

Segundo passo: identificar cada maxtermo. No exemplo acima, cada linha identificada gerará um maxtermo baseado na soma de todas as variáveis de entrada. As variáveis que tiverem valor 1 serão negadas.

Linha 1: $A + B + C$	Linha 3: $A + \overline{B} + C$	Linha 6: $\overline{A} + B + \overline{C}$	Linha 7: $\overline{A} + \overline{B} + C$

Observe sempre a ordem alfabética das variáveis, dentro de cada maxtermo.

Terceiro passo: obter a FNC, realizando a multiplicação entre os maxtermos.

$$S = (A + B + C) \,.\, (A + \overline{B} + C) \,.\, (\overline{A} + B + \overline{C}) \,.\, (\overline{A} + \overline{B} + C)$$

Forma Normal Disjuntiva (FND)

Uma expressão está na FND, se:

- Está na Forma Normal, ou seja, contém apenas os operadores E, OU e NÃO;
- Não há dupla negação, ou seja, deve-se aplicar a propriedade de identidade do complemento;
- O operador de negação não incide indiretamente sobre uma variável, ou seja, deve-se aplicar o teorema de De Morgan;
- O operador E não incide sobre o operador OU, ou seja, A E (B OU C), deve receber a propriedade distributiva de multiplicação, ficando (A E B) OU (A E C).

Também conhecida como Soma de Produtos, ou Soma de Mintermos, a FND corresponde a identificar na tabela-verdade as saídas que apresentam valor 1, e realizar uma adição

(disjunção) das respectivas combinações de entrada, sendo, para cada entrada, multiplicados (conjunção) o valor de cada variável, sendo realizado o complemento para as variáveis com valor 0. Exemplo:

Tabela 10: Tabela-verdade de uma expressão booleana

Linha	A	B	C	S
1	0	0	0	0
2	0	0	1	1
3	0	1	0	0
4	0	1	1	1
5	1	0	0	1
6	1	0	1	0
7	1	1	0	0
8	1	1	1	1

Primeiro passo: identificar as saídas que são 1. No exemplo acima, corresponde às linhas 2, 4, 5 e 8.

Segundo passo: identificar cada mintermo. No exemplo acima, cada linha identificada gerará um mintermo baseado na multiplicação de todas as variáveis de entrada. As variáveis que tiverem valor 0 serão negadas.

Linha 2: $\overline{A} . \overline{B} . C$	Linha 4: $\overline{A} . B . C$	Linha 5: $A . \overline{B} . \overline{C}$	Linha 8: $A . B . C$

Terceiro passo: obter a FND, realizando a soma de produtos.

$$S = (A.B.\overline{C}) + (\overline{A}.B.C) + (A.\overline{B}.\overline{C}) + (A.B.C)$$

Uma vez obtida a expressão booleana, agora é possível simplificá-la pela técnica de fatoração, utilizando postulados, propriedades e teoremas. Deixando, portanto, de estar na Forma Normal.

Os assuntos tratados até esse ponto podem ser relacionados na figura a seguir:

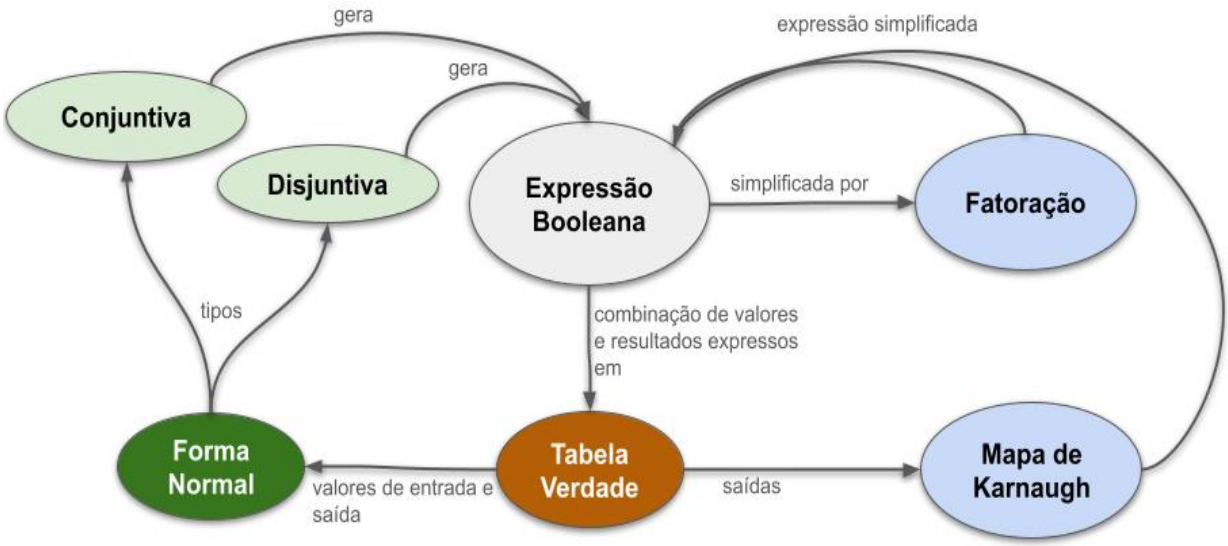


Figura 21: relacionamento entre os elementos do capítulo 1

LISTA DE EXERCÍCIOS – CAPÍTULO 1

Desenvolva a tabela verdade para as seguintes expressões booleanas:

1.1 $C.(A.C + C) + A.B$

1.2 $(\overline{B.B} + \overline{C.B}) + \overline{A}.\overline{B}$

1.3 $\overline{(A + (\overline{A} + B.C).C}$

1.4 $C.(B.D + C.B) + A.C$

1.5 $(\overline{B.A} + B.C).D + C.C$

1.6 $(\overline{\overline{A}.D + B.C}).A + \overline{A}.C$

Por meio de postulados, propriedades e teoremas, simplifique as seguintes expressões booleanas:

1.7 $\overline{C}.(\overline{D.B} + B.D) + D.C$

1.8 $\overline{A.D} + \overline{(C.B + D.\overline{B}).C}$

1.9 $(\overline{D.D} + \overline{B}.C).C + D.\overline{A}$

1.10 $\overline{(D.B + \overline{A}).C} + A.(E + A)$

1.11 $(\overline{D.A} + C.(\overline{A.E} + C.B)$

1.12 $B.\Big((E.D + C.B).\Big(A + \big(\bar{A} + (\bar{A}.\bar{B})\big)\Big) + D.D.B\Big) + A.(B + C)$

Derivação de expressão booleana

1.13 Obtenha a FNC e FND a partir da tabelas-verdade:

A	B	C	S
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

FNC:

_____

FND:

_____

1.14 Obtenha a FNC e FND a partir da tabelas-verdade:

A	B	C	D	S
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

FNC:

_____

_____

_____

FND:

_____

_____

_____

1.15 Obtenha a FNC e FND a partir da tabelas-verdade:

A	B	C	D	S
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

FNC:

_____

_____

_____

FND:

_____

_____

_____

Com base em uma tabela de 2 variáveis, obtenha: 1) a expressão simplificada por meio do Mapa de Karnaugh e 2) compare com a FNC ou FND:

1.16

A	B	S
0	0	0
0	1	0
1	0	1
1	1	1

1.17

A	B	S
0	0	1
0	1	1
1	0	1
1	1	0

1.18

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Com base em uma tabela de 3 variáveis, obtenha: 1) a expressão simplificada por meio do Mapa de Karnaugh e 2) compare com a FNC ou FND:

1.19

A	B	C	S
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

1.20

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

1.21

A	B	C	S
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Com base em uma tabela de 4 variáveis, obtenha: 1) a expressão simplificada por meio do Mapa de Karnaugh e 2) compare com a FNC ou FND:

1.22

A	B	C	D	S
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

1.23

A	B	C	D	S
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

1.24

A	B	C	D	S
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

2. Introdução a Algoritmos

- *Aborda os fundamentos dos algoritmos, definindo seu conceito e destacando sua importância na lógica de programação.*
- *Apresenta os diferentes tipos de algoritmos e discute como eles estruturam a resolução de problemas computacionais por meio de uma sequência lógica de instruções.*
- *Introduz o conceito de controle de fluxo.*

2.1. Conceito de Algoritmo

Etimologicamente, a palavra “algoritmo” é derivada do nome Mohammed ibn Musa Al-Khowarizmi, que foi um astrólogo e matemático árabe do século IX. Ele foi responsável por introduzir o sistema de numeração indiano no Ocidente. Esta notação, durante a Idade Média, tornou-se conhecida como algorismos, sendo o sistema numérico decimal que utilizamos nos dias atuais [4].

Um algoritmo é definido como sendo um processo de cálculo matemático ou da descrição sistemática da resolução de um grupo de problemas semelhantes, sendo, também, regras formais para a obtenção de um resultado ou da solução de um problema[5].

Um algoritmo pode ser definido como uma sequência finita de passos ou conjunto de instruções, organizados logicamente, com a finalidade de execução de uma tarefa ou resolução de um problema específico.

Muitas atividades do nosso cotidiano são executadas seguindo um algoritmo, ou seja, as ações que nos conduzem a um determinado resultado são organizadas seguindo o raciocínio lógico adequado à solução desejada. Exemplo: algoritmo de deslocamento para o trabalho:

- *Passo 1: abrir a porta da casa.*
- *Passo 2: ir até a garagem.*
- *Passo 3: destravar a porta do carro.*
- *Passo 4: dar partida no carro.*
- *Passo 5: retirar o carro da garagem.*
- *Passo 6: pegar o trajeto menos congestionado.*
- *Passo 7: estacionar o carro na garagem do trabalho.*
- *Passo 8: abrir a porta do escritório.*

Um algoritmo se constitui em uma solução possível para certo problema, sendo que outras soluções poderão existir. Para a resolução do problema descrito acima, passos poderiam ser acrescentados ou suprimidos, assim como poderia ser dado um maior detalhamento em algum passo, ou seja, um refinamento daquele passo específico. Como por exemplo, se o escritório estiver situado em um prédio, haveria a necessidade de pegar o elevador antes de entrar no escritório. Outra situação possível, seria o fato do estacionamento na garagem do trabalho acontecesse de maneira mais complexa, como a utilização de senhas de acesso ou manobristas, necessitando, dessa forma, de um maior detalhamento. Essa situação também pode ocorrer na

discriminação dos passos a serem dados para obtenção do trajeto menos congestionado entre a casa e o trabalho. Sendo possível a existência de vários algoritmos para resolução de um mesmo problema, alguns poderão ser mais eficientes do que outros, bem como possuírem diferentes graus de complexidade.

Em nosso dia a dia fazemos uso, mesmo de maneira inconsciente, de algoritmos para as mais diversas tarefas, como por exemplo: substituir uma lâmpada queimada, trocar o pneu do carro, na confecção de receitas de culinária ou na utilização do manual de instruções de um equipamento para sua instalação, montagem ou desmontagem.

2.2. Algoritmos e a Lógica de Programação

Na construção de algoritmos, utilizamos o raciocínio lógico para a elaboração dos passos nele descritos.

Raciocínio lógico é a sequência coerente, regular e necessária de acontecimentos, de coisas ou fatos e que é a maneira de raciocínio particular que cabe a um indivíduo ou a um grupo. E lógica é definida como a ciência dos princípios formais do raciocínio [5].

A lógica pode ser vista como a arte de pensar corretamente, pois ela visa colocar ordem no pensamento, sendo que utilizamos a lógica de forma natural nas atividades que executamos no nosso dia a dia [6].

Para que os computadores sejam capazes de interpretar os algoritmos, é necessário transformar a sequência de passos escritos em linguagem natural, para uma linguagem que possa ser entendida pelo computador. Essas linguagens são chamadas de linguagens de programação. Existem diversas linguagens de programação em uso atualmente. As linguagens de programação (Pascal, C, Cobol, Java, Python etc), chamadas linguagens de alto nível, são constituídas por símbolos que mais se aproximam da linguagem humana, e possibilitam que algoritmos sejam escritos de uma forma mais compreensível ao ser humano e, posteriormente, por meio de um processo de tradução (compilação ou interpretação[1]), esse algoritmo é convertido para a linguagem de máquina.

Os programas de computadores (software) são algoritmos escritos em uma linguagem de programação, possibilitando sua execução por parte de um computador. De maneira geral, um programa é um algoritmo escrito em uma linguagem que o computador entende. As instruções que compõem o programa são armazenadas na memória do computador para que possam ser executadas.

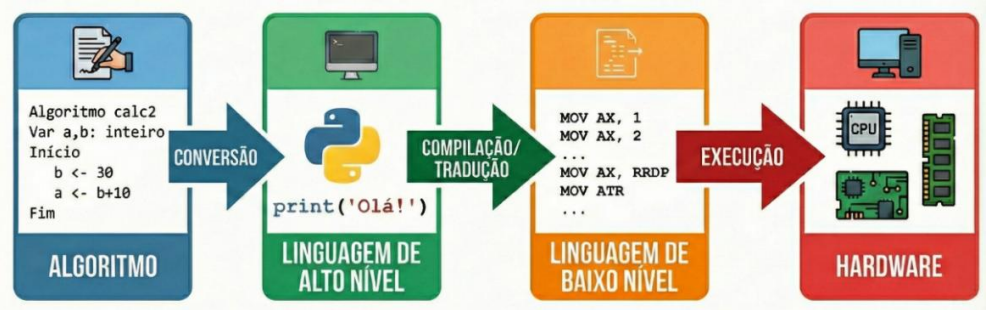


Figura 22: algoritmo e níveis de linguagem de programação

[1] **Compilação:** *converte integralmente o código-fonte de um programa para linguagem de máquina antes de sua execução, gerando um arquivo executável independente.* ***Interpretação:*** *o código-fonte é traduzido e executado linha a linha em tempo real por um programa intermediário, sem a geração prévia de um arquivo executável independente.*

A lógica de programação pode ser definida como um conjunto de técnicas para encadear pensamentos a fim de atingir determinado objetivo. A finalidade de toda programação é construir algoritmos possíveis de serem executados por um computador. Dessa forma, a lógica de programação é essencial para pessoas que desejam trabalhar com desenvolvimento de programas para computadores[6].

Sobre o papel dos algoritmos na computação, alguns autores [7] consideram o algoritmo como uma ferramenta para resolver um problema computacional bem específico, sendo uma sequência de etapas que recebe um valor ou conjunto de valores como entrada e o transforma em saída. Os autores citam como exemplo um algoritmo para resolver o problema da ordenação, onde é fornecido como entrada um conjunto de valores desordenados e, por meio de processamentos, o algoritmo apresenta como saída os valores em uma forma ordenada.

As estruturas básicas para a construção de algoritmos são:

- **Sequenciação**: conjuntos de ações que devem ser executadas sequencialmente, ou seja, todas as instruções são executadas uma após o término da anterior, de cima para baixo e da esquerda para a direita.
- **Seleção**: o fluxo de execução das instruções a ser seguido é determinado pelo resultado da avaliação de uma ou mais condições, sendo elas uma expressão condicional e/ou lógica que pode ser verdadeira ou falsa[8].No capítulo 6 estas estruturas serão apresentadas.
- **Repetição:** o fluxo é alterado por meio da repetição de trechos de código em um número pré-definido de vezes ou enquanto uma determinada condição for satisfeita [9]. No capítulo 7 desta apostila serão apresentadas as estruturas de repetição.

Além das estruturas básicas mencionadas, o fluxo de instruções também pode ser alterado por meio de sub-rotinas (tais como funções e procedimentos), que são códigos situados em pontos distantes do local onde são chamados e, quando tal invocação ocorre, o fluxo de execução sofre um desvio para onde está situada a sub-rotina, sendo ela executada, e retornando o fluxo de instruções para o ponto de onde foram chamadas no código principal. As sub-rotinas (funções e procedimentos) serão abordadas no capítulo 8 desta apostila.

Outras formas de controle de fluxo de execução que podem ser encontradas nas linguagens de programação são os comandos ou funções específicas para este fim. Como exemplo, podemos citar a função Interrompa, em Pseudocódigo. Essa função interrompe a execução do laço, desviando o fluxo de execução para fora da estrutura, seguindo a sequência normal de execução.

2.3. Tipos de Algoritmos

Os tipos de algoritmos estão relacionados às diversas formas gráficas que eles podem ser representados. A seguir serão abordadas algumas maneiras de representação de algoritmos.

Descrição narrativa: descreve-se o algoritmo de maneira natural utilizando qualquer idioma. É o que fazemos quando escrevemos uma receita de bolo, por exemplo. Esse tipo de representação tem a vantagem da simplicidade e fácil compreensão, porém apresenta os problemas de imprecisão, ambiguidades causadas pela própria língua, que proporcionam diferentes interpretações e pouca confiabilidade.

Exemplo de algoritmo que verifica se um número inteiro é par ou ímpar, usando descrição narrativa:

> ***Início***
>
> *(1) Obter o número.*
>
> *(2) Obter o resto da divisão do número por 2.*
>
> *(3a) Se o resto for zero, informe que número é par.*
>
> *(3b) Caso contrário, informe que número é ímpar.*
>
> ***Fim***

Fluxograma (ou Diagrama de Blocos): utilização de símbolos gráficos padronizados para representar as diversas ações do algoritmo. Possui a vantagem de que uma figura pode dizer o que seria necessário a utilização de várias palavras, porém, apresenta a desvantagem da necessidade do entendimento do significado de cada símbolo utilizado, além de se tornar complicado para representação de algoritmos grandes.

Exemplo de algoritmo que verifica se um número é par ou ímpar, usando fluxograma:

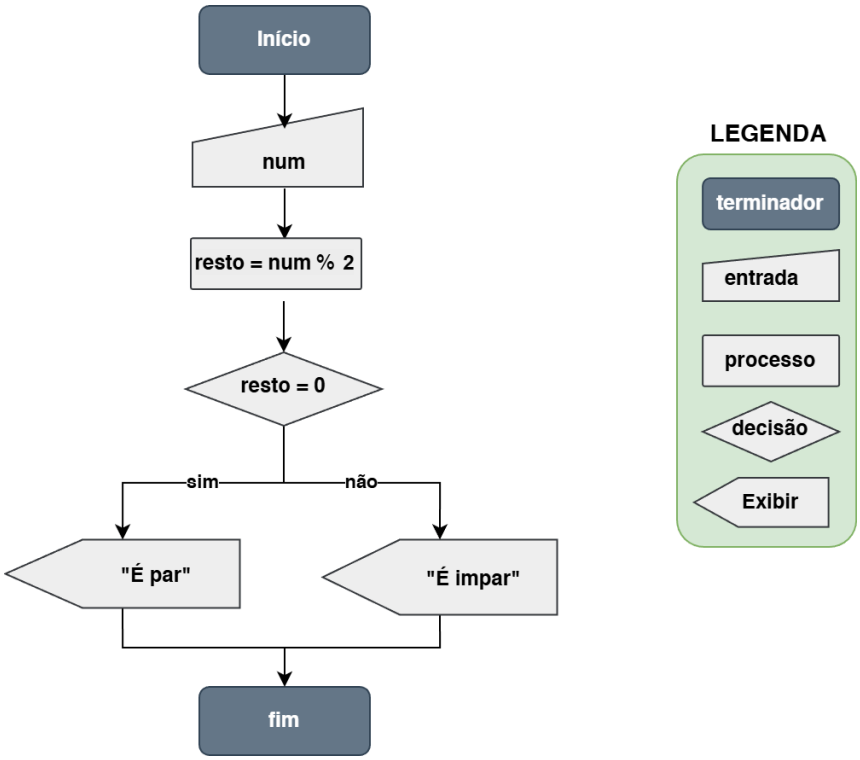


Figura 23: fluxograma [6]

Pseudocódigo: definição de uma pseudolinguagem de programação para representar um algoritmo, cujas instruções são escritas em qualquer idioma, porém com regras que restringem e estruturam seu uso. Não apresenta as ambiguidades de uma língua nem os rigores de uma linguagem de programação. Sua conversão para uma linguagem de programação qualquer se dá de maneira quase direta. Um pseudocódigo escrito em português é também chamado português estruturado ou portugol (português + algol). Algol (Algorithmic Language) é uma alusão a uma linguagem de programação de alto nível.

Os algoritmos não são concebidos para execução imediata por computadores, mas sim para servirem como uma estrutura lógica fundamental, atuando como um rascunho detalhado prévio à codificação. Eles operam em um nível de abstração focado exclusivamente na resolução passo a passo de um problema, isentos das regras rígidas de sintaxe técnica; portanto, sua função é orientar o programador, que deverá traduzir essa lógica universal para uma das diversas linguagens de programação existentes, materializando o conceito abstrato em um software funcional. **Entretanto, para fins didáticos, poderão ser utilizados softwares para a verificação sintática e execução de pseudocódigos, como o Visualg**[2]. Há um anexo desta apostila que trata somente da instalação, uso e guia de referência deste software.

Somente após esse algoritmo ser escrito em uma linguagem de programação, com a consequente conversão em um código, é que ele poderá ser executado. Portanto, o pseudocódigo pode ser considerado um intermediário entre a linguagem falada e a linguagem de programação. Assim como é necessário o conhecimento do significado dos símbolos no fluxograma, há, também, a necessidade de conhecimento das regras sintáticas do pseudocódigo.

Exemplo de algoritmo que verifica se um número é par ou impar, usando pseudocódigo:

```
Início
    Leia (num).
    resto ← num % 2
    se resto=0 então
        escreva(“Par”)
    senão
        escreva(“Ímpar”)
    fimSe
Fim
```

2 Academic Free License -AFL

LISTA DE EXERCÍCIOS – CAPÍTULO 2

2.1 Utilizando a descrição narrativa, represente um algoritmo para a troca de um pneu furado de um automóvel.

2.2 Elabore um fluxograma que leia duas notas de um aluno (N1 e N2). O algoritmo deve calcular a média aritmética e verificar se o aluno foi aprovado.

2.3 Construa um algoritmo, utilizando a descrição narrativa, para detalhar a apresentação de uma turma de aula, pelo Chefe de Turma, quando da chegada de um professor militar em sala.

2.4 Descreva um algoritmo para realizar uma ligação telefônica.

2.5 Construa um algoritmo, utilizando a descrição narrativa, para calcular o IMC de uma pessoa.

2.6 Descreva, de forma detalhada, uma sequência lógica de passos para trocar uma lâmpada queimada.

2.7 Suponha que você tenha uma caixa cheia de bolas e que nessa caixa existem bolas verdes e bolas vermelhas. Além disso, você tem também duas caixas vazias. Vamos chamar a caixa que contém as bolas de “caixa 1” e as duas caixas vazias de “caixa 2” e “caixa 3”. Neste contexto, escreva um algoritmo que defina como tirar todas as bolas da “caixa 1” colocando as bolas verdes na “caixa 2” e as bolas vermelhas na “caixa 3”.

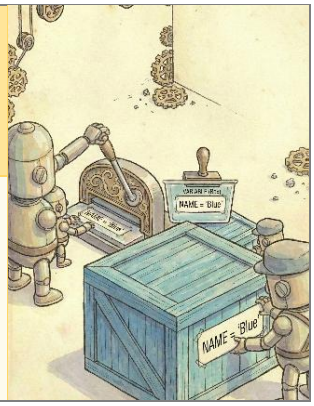
2.8 Descreva as medidas de segurança necessárias para devolução de uma pistola para a reserva de armamento.

2.9 Faça um algoritmo para somar dois números e multiplicar o resultado pelo primeiro número.

2.10 Identifique os dados de entrada, processamento e saída no algoritmo abaixo:

```
Início
Receba código do material
Receba valor do material
Receba Quantidade de material
Calcule o valor total do material (Quantidade * Valor do material)
Mostre o código do material e seu valor total
Fim
```

3. Variáveis, Constantes e Tipos de Dados



- *Variáveis são espaços na memória do computador para armazenar dados de um determinado tipo, são nomeadas e podem ser modificados durante a execução de um programa.*
- *Constantes são valores que não se alteram durante a execução do programa, sendo usualmente definidas no início do código.*

3.1. Variáveis

Para que um computador seja capaz de realizar o processamento dos dados de entrada e transformá-los em dados de saída, tanto os dados a serem manipulados como as instruções que constituem o programa (algoritmo) que está sendo executado, devem estar armazenados na sua memória.

Variável, na lógica de programação, refere-se a um espaço na memória principal do computador, cujo valor armazenado nesse espaço pode ser modificado ao longo da execução do algoritmo (programa). Embora uma variável possa assumir diferentes valores, ela armazena apenas um valor em cada instante. Portanto, quando atribuímos um novo valor a uma variável, o que estava armazenado nela é perdido (Figura 24).

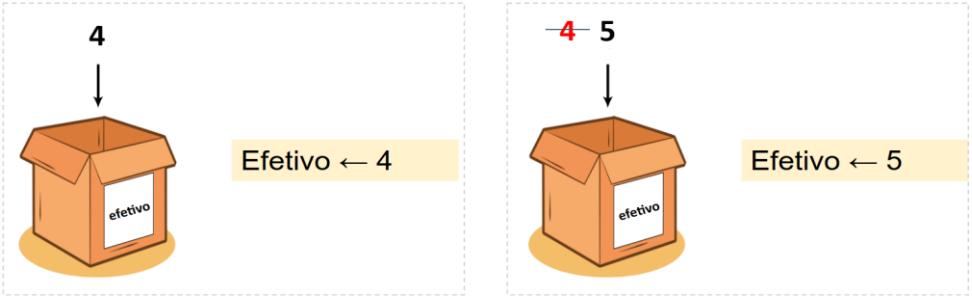


Figura 24: armazenamento de valores em variável

É importante distinguir a variável matemática da variável computacional: enquanto na Álgebra a variável funciona como uma incógnita (um valor desconhecido a ser descoberto em uma equação), em algoritmos ela atua como um espaço de memória reservado para armazenar dados.

Uma variável deve possuir um nome (identificador) e um tipo, referente ao tipo de dado armazenado.

Existem algumas regras a serem seguidas na formação dos identificadores (nomes), seja de variáveis, constantes ou qualquer outra rotina de um programa. A seguir, serão apresentadas algumas delas:

- Devem ser constituídos apenas por letras (maiúsculas e/ou minúsculas), números ou o caractere sublinhado (_).
- Não são permitidos espaços em branco nem caracteres especiais ($, %, &, @ etc).
- Não devem ser iniciados por número.

- Não podem ser utilizadas palavras reservadas da linguagem de programação que está sendo utilizada (identificadores de comandos da própria linguagem), bem como identificadores que já estejam sendo utilizados. A Tabela 11 apresenta as palavras reservadas em Pseudocódigo, que não podem ser usadas como identificadores.
- Devem ter nome sugestivo do conteúdo que armazena, como por exemplo: nomeAluno, qtdNotas, Nota1, notaIC, aprovado, etc.

Tabela 11: algumas palavras reservas em pseudocódigo

Falso	ALGORITMO	SENÃO	FIMPARA	FUNÇÃO
Verdadeiro	VAR	FIMSE	ENQUANTO	ESCOLHA
E	INÍCIO	PARA	FAÇA	CASO
OU	FIMALGORITMO	ATÉ	FIMENQUANTO	RETORNE
NÃO	SE	PASSO	PROCEDIMENTO	...

Em muitas linguagens de programação, se faz necessária a declaração das variáveis que serão utilizadas, mencionando o seu nome e o tipo de dado que será armazenado, logo no início do programa. Isso também acontece com o pseudocódigo. Ao declararmos uma variável em um programa, será alocado na memória um espaço compatível com o tipo de dado declarado.

Sintaxe do comando para declaração de variável no pseudocódigo:

Var <nome_da_variável> : <tipo_da_variável>

Var <lista_de_variáveis> : <tipo_das_variáveis>

Exemplos de declaração de variável no pseudocódigo:

Var ano: inteiro

Var nome, endereço, cidade: caractere

Para atribuir um valor a uma variável em pseudocódigo utilizamos o símbolo ← (seta para a esquerda), que pode ser representada por "<-". Exemplo de atribuição de valor a uma variável em pseudocódigo:

idade ← 50

> ***Nota:*** *no decorrer desta apostila, as sintaxes dos comandos aparecerão, por vezes, como uma palavra reservada ou comando, complementado por trechos entre < >, que representam uma forma geral. Estes símbolos vão representar nomes de variáveis, funções, procedimentos, tipos de dados, valores e demais elementos que complementam uma determinada sintaxe.*

3.2. Constantes

Diferentemente das variáveis, as constantes são constituídas por valores que não devem ser alterados durante a execução do programa. Em algumas linguagens há a possibilidade de se definir variáveis, em outras define-se uma variável com a finalidade de uso como constante.

No pseudocódigo adotado na disciplina, as constantes são definidas como variáveis, sendo uma boa prática o emprego de maiúsculas na sua nomeação, ajudando a diferenciá-las das variáveis dentro de um algoritmo. Também é uma boa prática definir o valor desta constante nas primeiras linhas após o início do algoritmo.

Sintaxe do comando para definição de constante no pseudocódigo:

```
Var  <NOME DA CONSTANTE> : <tipo>
```

No algoritmo 3.2, no final deste capítulo, há um exemplo do uso de uma “constante”.

3.3. Tipos de Dados

Quando declaramos variáveis em um algoritmo, temos a necessidade de saber qual tipo de dados que ela armazenará. Nas diversas linguagens de programação disponíveis há, de forma geral, semelhança nos tipos de dados, com algumas particularidades. A Tabela 12 apresenta os tipos de dados básicos em Pseudocódigo.

Tabela 12: Tipos de Dados em Pseudocódigo

TIPO DE DADOS		**DESCRIÇÃO**	**EXEMPLOS**
NUMÉRICOS	*inteiro*	Números inteiros	5, 75, -30, 100
	real	Números reais (ponto flutuante). Há a divisão da parte inteira da parte fracionária, podendo estar na notação científica.	4.7; 88.987 0.456; 3e2
LITERAIS	*caractere*	Conjunto de caracteres ou um único caractere escrito entre aspas duplas (“...”).	“frase” “X”
BOOLEANOS	*logico*	Dados com dois valores possíveis: verdadeiro ou falso	*VERDADEIRO* *FALSO*

Considerando as atribuições à esquerda, podemos imaginar uma variável como um espaço, nomeado e apto a armazenar um tipo específico de dado.

```
efetivo ← 50
largura ← 2.5
aprovado ← Verdadeiro
aviso ← “Olá”
```

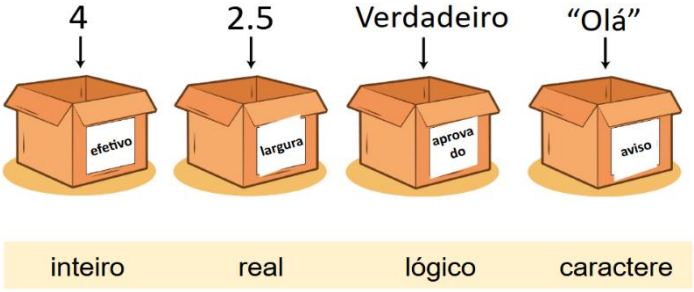


Figura 25: armazenamento de valores de variáveis

O algoritmo 03_01 demonstra a aplicação dos conceitos abordados neste tópico, quais sejam: variáveis (nomeação, tipos de dados e atribuição de valores) e constantes. Com isso, pode se produzir um pseudocódigo mais completo. Para termos um algoritmo minimamente funcional, antecipamos alguns conceitos que serão aprofundados em capítulos futuros. Há um comando *Leia* na linha 5, responsável por permitir a entrada dos valores para as variáveis “b” e “h”, que serão utilizadas como base e altura no cálculo. Na linha 6 é efetuado o cálculo da área (A), sendo que o “*” é operador de multiplicação e a “/” operador de divisão. Finalmente, na linha 7, há um comando *Escreva* que exibe o cálculo da área realizado anteriormente.

```
1  Algoritmo “03_01 Area_Triangulo”
2  Var
3      A, b, h: real
4  Inicio
5      Leia (b, h)
6      A ← (b * h)/2
7      Escreva (A)
8  FimAlgoritmo
```

Observação: quando da utilização do software Visualg para testar seus algoritmos, substitua o símbolo "←" por "<-" (menor + traço), que será o símbolo reconhecido na atribuição de valores às variáveis.

Outro exemplo de algoritmo que emprega esse mesmo conhecimento:

```
1  Algoritmo "03_02 Area_Circulo"
2  Var
3      area, r: real
4      NPI : real
5  Inicio
6      NPI <- 3.141593
7      Leia (r)
8      area ← (NPI * r * r)
9      Escreva (area)
10 FimAlgoritmo
```

Não foi utilizado a variável PI pois se trata de uma palavra reserva no Portugol/Visualg.

LISTA DE EXERCÍCIOS – CAPÍTULO 3

3.1 O que são constantes? Dê dois exemplos.

3.2 O que são variáveis? Dê dois exemplos.

3.3 Cite os tipos de dados utilizados nas variáveis dos algoritmos, exemplificando cada um deles.

3.4 Forneça o tipo dos dados listados abaixo:
a) 613
b) 613.0
c) -613
d) "613"
e) Verdadeiro
f) “Falso”
g) "Fim de Exercício"

3.5 Faça um algoritmo, em pseudocódigo, que atribua às devidas variáveis os seguintes dados:
a) 12345
b) -1122
c) 10.5
d) Verdadeiro
e) Maria

3.6 Assinale as variáveis cujos nomes estejam corretamente nomeados:
a) [ ] nome
b) [ ] telefone-celular
c) [ ] nome+sobrenome
d) [ ] 2taxa
e) [ ] telefone_celular
f) [ ] conta1

3.7 Para cada valor dado abaixo, foi definido um tipo de variável. Marque os pares “valor e tipo” definidos corretamente:

a) [ ]	valor = 2.5	tipo = real
b) [ ]	valor = “F”	tipo = inteiro
c) [ ]	valor = -2	tipo = inteiro
d) [ ]	valor = ‘M’	tipo = caractere
e) [ ]	valor = 5	tipo = caractere
f) [ ]	valor = -10.35	tipo = real
g) [ ]	valor = 38	tipo = real
h) [ ]	valor = “José”	tipo = caractere
i) [ ]	valor = 135	tipo = inteiro
j) [ ]	valor = 7.5	tipo = inteiro
k) [ ]	valor = 100	tipo = logico
l) [ ]	valor = VERDADEIRO	tipo = caractere

3.8 Assinale os comandos de atribuição realizados corretamente:

a) [ ] var sexo: texto
sexo ← “F”

b) [ ] var altura: inteiro
altura ← 1.80

c) [ ] var salario: real
salario ← 3000.00

d) [ ] var nome: caractere
nome ← “JOAQUIM”

3.9 Para os nomes de variáveis abaixo, marque (C) para os corretos e (E) para os errados. Para cada nome errado, explique o que está incorreto.

a) [ ] cidade .....

b) [ ] media idade .....

c) [ ] nome2 .....

d) [ ] endereco_nr .....

e) [ ] a3 .....

f) [ ] 4cidade .....

g) [ ] media_peso% .....

h) [ ] endereco.cep .....

i) [ ] cliente_nome .....

j) [ ] aluno-nota.....

k) [ ] B5 .....

l) [ ] 1234P .....

3.10 Considerando as variáveis abaixo, assinale **I** para inteiro, **R** para real, **C** para caractere e **L** para Lógico:

a) [ ] 1000

b) [ ] “4.56”

c) [ ] “12”

d) [ ] Verdadeiro

e) [ ] “cinco”

f) [ ] “casa8”

g) [ ] “5”

h) [ ] - 456

i) [ ] 456

j) [ ] – 4.56

k) [ ] 45.897

3.11 Considere as variáveis NotaAlu, NomeAlu, NumMatr e Sexo, utilizadas para armazenar a nota de um aluno, o nome do aluno, o número de matrícula do aluno e o sexo do aluno, respectivamente. Declare-as corretamente em pseudocódigo.

3.12 Encontre os erros existentes nas seguintes declarações de variáveis:

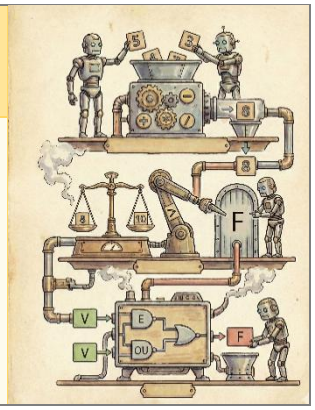
```
var
    endereço, nfilhos, valor$, xpto, c, peso: inteiro
    idade, x: caractere
    2lâmpada: logico
    valor: decimal
(...)
```

3.13 Analise o algoritmo abaixo e escreva o resultado de sua execução:

```
Algoritmo exe313
Var
    num_a,num_b,num_c: inteiro
Início
    num_a ← 5
    num_b ← 30
    num_c ← num_a
    num_b ← num_c
    num_a ← num_b
    Escreva (num_a)
    Escreva (num_b)
    Escreva (num_c)
Fim
```

4. Operadores

Operadores são símbolos utilizados para formular expressões, atuando sobre operandos (variáveis, constantes ou outras expressões). Os operadores são classificados em aritméticos, usados em expressões numéricas; relacionais, que comparam operandos; e lógicos, usados para combinar ou inverter expressões lógicas.



4.1. Operadores Aritméticos

Os operadores aritméticos são utilizados para formulação de expressões aritméticas. Os operandos que constituem uma expressão aritmética podem ser variáveis ou constantes do tipo numérico. A tabela 11 apresenta os operadores aritméticos utilizados no pseudocódigo, com as respectivas ordens de avaliação, quando da existência dos diversos tipos em uma mesma expressão. Dessa forma, os operadores com maior precedência serão avaliados primeiramente.

Tabela 13: Operadores Aritméticos

Operação	Operador	Ordem	Descrição
Exponenciação	^	1	Eleva o operando da esquerda (base) ao operando da direita (expoente). Ex.: *2^3 = 8*.
Divisão	/	2	Divide o operando da esquerda (dividendo) pelo operando da direita (divisor). Ex.: *6/3 = 2*.
Divisão inteira	\		Retorna a parte inteira da divisão. Ex.: 9\2 = *4*.
Multiplicação	*		Realiza a multiplicação dos operandos. Ex.: 12*2 = 24
Módulo	%		Retorna o resto da divisão de inteiros. Ex.: *9%2 = 1*
Adição	+	3	Realiza a soma dos operandos. (*) Ex.: 3+5 = 8
Subtração	-		Subtrai o operando da direita do operando da esquerda. Ex.: *8 – 7 = 1*.

(*) Quando forem do tipo literal, executa a concatenação dos operandos.

A precedência na execução das operações pode ser alterada por meio da utilização dos parênteses ( ). Neste caso, todos as operações que estão entre parênteses têm precedência sobre as demais, sendo que os parênteses mais internos têm precedência sobre os parênteses mais externos. Exemplo:

valor ← (((3 + 5) / 2) – ((4 * 3) ^ 2))

valor ← -140

4.2. Operadores Relacionais

Os operadores relacionais possibilitam a realização de comparações entre valores. As expressões relacionais, que constituem uma relação entre operandos do mesmo tipo, fornecem como resultado um valor do tipo lógico (Verdadeiro ou Falso).

A Tabela 14 apresenta os operadores relacionais utilizados no pseudocódigo, com as respectivas ordens de avaliação.

Quando os operandos forem numéricos, a comparação será feita com base nos seus valores; quando tratar-se de operandos do tipo literal (texto, caractere ou *string*) ela é realizada lexicograficamente; porém, para os operandos do tipo lógico (*booleano*) estará disponível apenas os operadores igualdade e desigualdade.

A precedência dos operadores relacionais é menor do que a dos operadores aritméticos. Logo, em expressões que contenham operadores relacionais e aritméticos, os operadores aritméticos são avaliados primeiramente.

Tabela 14: Operadores Relacionais

Comparação	Operador	Ordem	Descrição
Menor	<	1	Compara se o operando da esquerda é menor do que o operando da direita. Ex.: *a < b*.
Maior	>		Compara se o operando da esquerda é maior do que o operando da direita. Ex.: *7 > 5*.
Menor ou igual	<=		Compara se o operando da esquerda é menor ou igual ao operando da direita. Ex.: *a <= b*.
Maior ou igual	>=		Compara se o operando da esquerda é maior ou igual ao operando da direita. Ex.: *7 >= 5*.
Igualdade	=	2	Compara se dois valores são iguais. Ex.: *a=b*.
Desigualdade	< >		Compara se dois valores são diferentes. Ex.: *c<>d*.

4.3. Operadores Lógicos

Assim como os operadores relacionais, os operadores lógicos fornecem como resultado um valor também lógico (verdadeiro ou falso).

A Tabela 15 apresenta os operadores lógicos utilizados no pseudocódigo.

Tabela 15: Operadores Lógicos

Operação	Operador	Ordem	Descrição
Negação	*NÃO*	1	Inverte o valor lógico do operando examinado. Ex.: *NÃO Verdadeiro = Falso; NÃO Falso = Verdadeiro*.
Conjunção	*E*	2	Retorna falso se, pelo menos, um dos operandos for falso.
Disjunção	*OU*	3	Retorna verdadeiro se, pelo menos, um dos operandos for verdadeiro.
Disjunção exclusiva	*EOU*		Retorna verdadeiro se somente um dos operandos for verdadeiro.

Os operadores OU, E e EOU são binários (agem sobre dois operandos), já o operador NÃO é unário (age sobre um operando apenas).

Normalmente, os operadores lógicos são usados em conjunto com os operadores relacionais em suas comparações.

Os operadores lógicos possuem menor precedência do que os operadores relacionais e, consequentemente, do que os aritméticos [10]. Portanto, em expressões que contenham operadores aritméticos, relacionais e lógicos, serão avaliados primeiramente os operadores aritméticos, em seguida os operadores relacionais e, por último, os operadores lógicos.

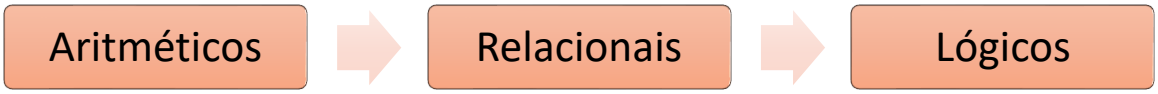


Figura 26: ordem de precedência de operadores

Exemplo:

Consideramos a seguinte expressão: **não ((7 - 4) * 3 ^ 2) + 1 >= 100 e verdadeiro**

A expressão acima é avaliada na seguinte sequência:

(1) Parênteses mais internos (7 – 4) = 3.

(2) Parênteses mais externos (3 * 3 ^ 2), obedecendo a precedência dos operadores aritméticos, primeiramente 3 ^ 2 = 9 e posteriormente, 3 * 9 = 27.

(3) Operador aritmético 27 + 1 = 28.

(4) Operador relacional 28 >= 100 = False.

(5) Operador lógico, obedecendo a precedência: não Falso = Verdadeiro.

(6) Operador lógico Verdadeiro e Verdadeiro = Verdadeiro.

Quando operadores de mesma precedência aparecerem em uma expressão, eles serão avaliados da esquerda para a direita.

Visando dar uma melhor visibilidade nas expressões, aconselha-se a utilização de parênteses delimitando os trechos da expressão que deverem ser avaliados primeiramente.

LISTA DE EXERCÍCIOS - CAPÍTULO 4

4.1 Escreva um algoritmo, em pseudocódigo, com o nome atribuição de valores, que possua as variáveis que armazenem o preço unitário, quantidade e preço total de um produto, sendo atribuídos valores iniciais às variáveis preço unitário e quantidade e o valor atribuído à variável preço total será o resultado da multiplicação do preço unitário pela quantidade.

4.2 Indique os resultados que serão obtidos nas seguintes expressões:

a) 1 / 2
b) 1 \ 2
c) 1 % 2
d) 7 \ 2
e) 9 % 2
f) (208 \ 10) % 3
g) 5 ^ 2 + 3
h) 19 + 3 * 2 ^ 2 – 23
i) 3.0 * (5.0 + 1)
j) 1 / 4 + 2
k) 29.0 / 7 + 4
l) 3 / 6.0 – 7
m) 2 > 3
n) (6 < 8) OU (3 > 7)
o) (((33 / 2) % 6) > 3) E (3 < (2 % 2))
p) NÃO (2 < 3)

4.3 Sabendo-se que A = 4, B = 8 e C = 5, informe se as expressões abaixo são verdadeiras ou falsas.

a) (A = C) OU (B > C)
b) NÃO(B = A) E ((A + 1) >= C)
c) (C = (B – A)) OU B = A

4.4 Observe o trecho de algoritmo abaixo, que contém a declaração de variáveis e suas respectivas atribuições e responda à questão a seguir:

```
Algoritmo Atribuição
Var
   num1, num2, num3, num4: inteiro
Início
   num1 ← 10
   num2 ← 5
   num3 ← 200
   num4 ← 300
Fim
```

Coloque V (Verdadeiro) ou F (Falso) nas expressões abaixo:

a) [ ] num1 > num2
b) [ ] num1 < num3
c) [ ] num1 > num4
d) [ ] num3 = num4
e) [ ] num1 + num2 > num3
f) [ ] num1 * num2 < num4
g) [ ] num3 – num4 < > num1
h) [ ] num3 / num1 < num2
i) [ ] num1 / num2 > 0 E num1 + num3 > num4
j) [ ] num1 * num2 > 40 E num3 - num1 > num4
k) [ ] num1 - num2 = 10 E num2 +num3 > num4
l) [ ] num1 + num2 < 10 E num3 - num4 = num1
m) [ ] num3 / num2 > 0 OU num1 + num3 > num4
n) [ ] num2 * num1 = 50 OU num3 - num > num4
o) [ ] num1 - num2 > 10 OU num2 + num3 > num4
p) [ ] num1 + num2 > 10 OU num3 / num1 > num4
q) [ ] num1 > num2 E num2 < num3 OU num3 < num4
r) [ ] num1 * num2 > 10 E num1 > num4 OU num3 - num1 > num4
s) [ ] num1 >= 10 OU num4 < num3 E num3 * num2 < num4
t) [ ] num1 + num2 > 10 E num4 / num2 > num3 E num3 < num4

4.5 Indique qual o resultado das expressões abaixo, sendo:

a = 5; b = 3; d = 7; p = 4; q = 5; r = 2; x = 8; y = 4; z = 6; vrLogico = Verdadeiro.

a) (z \ a + b * a) - d \ 2
b) p / r % q - q / 2
c) (z \ y + 1 = x) E vrLogico OU (y >= x)

4.6 Determine o resultado lógico (V ou F), resolvendo passo a passo as expressões lógicas abaixo.

Considere na avaliação os seguintes valores: x = 2; a = 4; b = -5; c = 3 ; d = 7.

a) não (x > 3)
b) (x > 1) E (não (b > d))
c) não (d > 0) E (c > 5)
d) não (x > 3) OU (c > 7)
e) (a > b) OU (c > b)
f) (x <= 2) E (b > 10) OU (a >= 3)
g) (x < 1) E (b >= d) E (d >= 5)
h) (d < 0) OU (c > 5)
i) não (d > 3) OU não (b > 7)
j) não ((x > C) E (a > b)) OU ((b < c) OU não (d < a) OU (b < 10))

4.7 Realize o cálculo, passo a passo, das expressões computacionais abaixo, atribuindo o valor final para a variável correspondente à expressão.

Considere os seguintes valores iniciais das variáveis: a = 1; b = 2; c = 3.

a) a ← 2 * a + 4 * b
b) b ← (3 * c + 4) –5
c) c ← (a * b) – 5 * (8 / 2 * c)
d) d ← a % b
e) e ← c \ b
f) f ← (5 * ((a ^ 2 / b * 6) + 5) – (-4 – (5 ^ 2 + a * b)))
g) g ← (a ^ 2 * (-10 + (b ^ 2 + c ^ 2) + 50 % 25 * (a ^ 2) + c))
h) h ← (5 ^ (c – a – b) * (2 ^ (a – 7) + 3 ^ 2 – 1)) \ 7
i) i ← 18 – c * ((62 – 65 + 10) * (17 – 3 * b))
j) j ← 2 ^ 6 + (c * (17 – 2 ^ 3 / a) + 6 ^ 2 / 9) / (5 – 3) ^ 2

4.8 Para cada uma das expressões aritméticas abaixo, determine o tipo de dados da variável que está no lado esquerdo do comando de atribuição, bem como o resultado da expressão que está no lado direito.

a) num_b ← 1 + 2 * 3 / 7
b) num_a ← 1 + 2 * 3
c) num_c ← 1 + 2 * 3 \ 7
d) num_d ← 3 \ 3 * 4.0
e) num_e ← num_a + num_b * num_c – num_d

4.9 Escreva as expressões abaixo no formato utilizado em pseudocódigos.

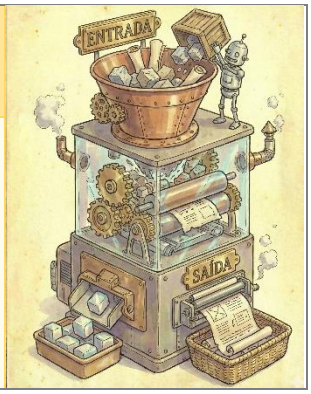
a) $\frac{2x^2-\sqrt{y}}{3+x}$

b) $\frac{1}{4y}+\left(\frac{3x}{2}\right)^3$

c) $\sqrt[3]{x^2}+2$

5. Entrada e Saída

- *"Comandos de Entrada" permitem a inserção de dados, que são geralmente armazenados em variáveis para uso no algoritmo.*
- *"Comandos de Saída" mostram mensagens ou resultados de processamento.*
- *"Comentários": são informações inseridas nos algoritmos que servirão apenas como documentação do algoritmo ou parte dele.*



O ciclo básico de um algoritmo é formado por uma sequência lógica e finita de etapas cujo objetivo é transformar dados iniciais em um resultado desejado. Esse processo começa com a **entrada de dados**, passa pelo **processamento** — que pode envolver cálculos, decisões ou manipulações — e termina com a **saída de informações** que resolvem o problema proposto (Figura 27). Essa estrutura garante que o algoritmo funcione como uma "caixa de transformação", onde a matéria-prima (dado) entra, sofre alterações controladas e sai como um produto acabado (informação).

Figura 27: ciclo básico de um algoritmo

Na prática da programação, esse conceito teórico é parcialmente materializado pelos comandos de Entrada e Saída (E/S). O comando de entrada viabiliza a etapa inicial ao capturar dados e armazená-los em variáveis, enquanto o comando de saída é o mecanismo responsável por exibir os resultados ao usuário após o processamento.

5.1. Comandos ou Funções de Entrada

O comando de entrada de dados permite a inserção, via teclado, de dados que serão utilizados pelo algoritmo. Os valores inseridos, normalmente, são armazenados em variáveis, que serão manipuladas pelo programa durante a sua execução. Em pseudocódigo, o comando **Leia** executa esta função.

Este comando pode ser utilizado para a leitura de uma variável:

```
Leia(<variável>)
```

Exemplo:
Leia(idade)

Ou, na leitura de várias variáveis em sequência:

```
Leia(<variável1>, <variável2>, ..., <variáveln>)
```

Exemplo:
Leia(nome, salario)

5.2. Comandos ou Funções de Saída

Por meio dos comandos de saída, conseguimos mostrar ao usuário os resultados do processamento dos algoritmos, bem como qualquer outro tipo de mensagem que se fizerem necessárias durante a execução do programa. Esses valores e mensagens são mostradas no monitor ou algum dispositivo de saída.

Em pseudocódigo o comando de saída é o Escreva. Os comandos de saída exibem na tela os argumentos fornecidos a essas funções, e, quando fornecidos vários argumentos estes devem ser separados por vírgula. Os argumentos poderão ser:

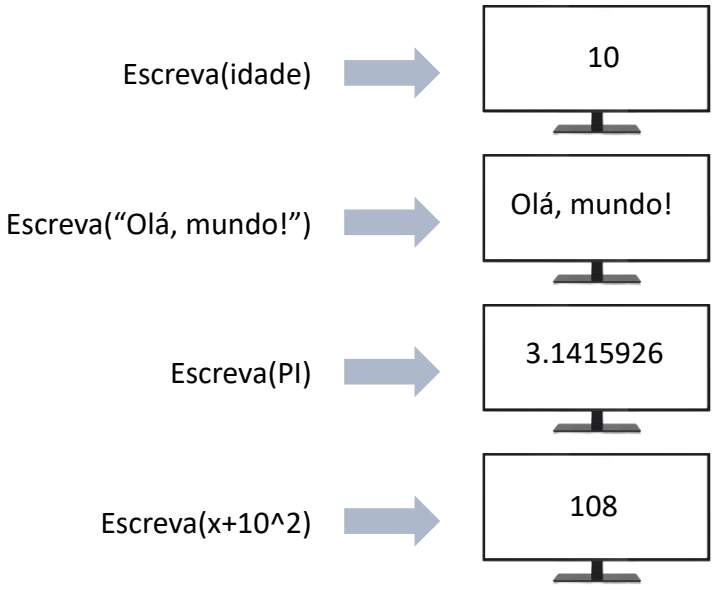
- variável: o valor contido nesta variável é exibido.
- constantes: o valor da constante é exibido.
- expressões: é resolvida e o seu resultado final é exibido.
- texto: a mensagem, entre as aspas que delimitam o texto, será exibida.

Com a utilização de um parâmetro, a sintaxe básica é:

```
Escreva(<argumento>)
```

Exemplos:

Considere a idade = 10 e x = 8):

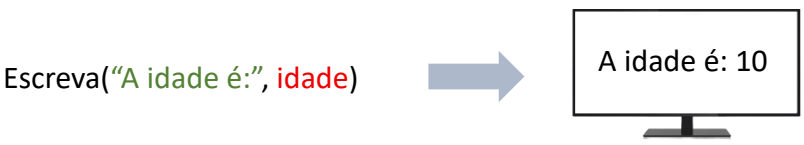


Ou, na utilização de vários argumentos:

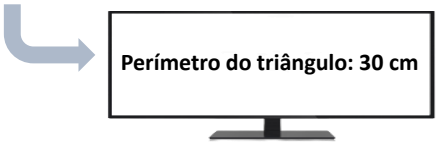
```
Escreva (<argumento1> , <argumento2> , ... , <argumenton>)
```

Exemplos:

Considere a idade = 10; nome="Pedro"; lado1 = 8; lado2 = 7 e lado3 = 5:



Escreva(“Perímetro do triângulo:”, lado1+lado2+lado3, “ cm”)

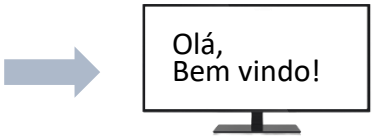


Escreva(“Aluno ”, nome)

O comando **Escreva** mantém o cursor na mesma linha. Isso significa que o próximo comando de saída continuará escrevendo exatamente onde o anterior parou. O uso comum é para criar mensagens compostas ou solicitar dados onde a resposta do usuário deve ficar na frente da pergunta.

Há uma alternativa ao escreva que é o comando **Escreval** (“Escreva linha”), que mostra o conteúdo e, ao final, salta o cursor para o início da próxima linha.

Escreval(“Olá,”)
Escreval(“Bem vindo!”)



Escreva(“Olá,”)
Escreva(“Bem vindo!”)

Atenção: é sempre uma boa prática incluir um comando escreva() antes do comando Leia(), instruindo o usuário sobre o que ele deve digitar (veja linhas 7 e 8 do algoritmo 05_01).

5.3 Comentários

Comentários são informações inseridas nos algoritmos que servirão apenas como explicações e orientações sobre o algoritmo ou parte dele. O conteúdo desses comentários não é desconsiderado no momento da execução do algoritmo, isto é, o texto que o compõem não será exibido ou terá qualquer efeito sobre a execução do código.

Comentário em Pseudocódigo:

- comentário de linha – Tudo o que for digitado desde **//** até o final da linha não é considerado. Utilizado nos comentários de apenas uma linha.
  // esta linha, a partir do símbolo de barra dupla, será considerada comentário
- comentário de bloco – Tudo o que estiver entre **/*...*/** não será considerado, para comentários que se estendam por mais de uma linha. Exemplo:

  / todo este texto será considerado comentário dentro do algoritmo*
  *E continuará sendo assim até encontrar o encerramento do comentário de bloco */*

Observação importante: no software Visualg não há possibilidade de utilização de comentários de bloco. As diversas linhas de comentário de bloco devem ser tratadas como comentários de linha, como no exemplo do algoritmo 05_01.

Veja o exemplo da aplicação de comentários no código a seguir:

```
1  Algoritmo "05_01 – exemplo de uso de comentário"
2  // Este é um primeiro exemplo de uso de comentários. No visualg, comentários
3  // em bloco exigem o símbolo de dupla barra no início em cada linha
4  Var
5     x: inteiro
6  Inicio
7     Escreva("Digite o valor de x:")
8     Leia(x)          // realiza a leitura de um valor e armazena na variável "x"
9     Escreva ("Fim do programa")
10 FimAlgoritmo
```

A palavra Algoritmo é uma palavra reservada que define o nome do algoritmo e não constitui um comentário no código. Assim como os comentários, este cabeçalho no pseudocódigo não é executado e não gera nenhum tipo de saída para o usuário.

5.4 Estrutura básica de um algoritmo.

Um algoritmo deve ter uma estrutura básica, como demonstrado no exemplo a seguir:

```
Algoritmo <nome_algoritmo>
// <comentário>
Var <variável1>,<variável2>:<tipo de variável>
      <variável3>:<tipo de variável>
Início
    <comando1>
    <comando2>
    (...)
    <comando_n>
FimAlgoritmo
```

Além da estrutura evidenciada, os valores e comandos que completam o código estão representados genericamente por <...>.

Estruturação do código

Um código-fonte, seja em pseudocódigo ou em uma linguagem de programação, exige mais do que apenas a sintaxe correta; ele necessita ser bem estruturado, com seus blocos lógicos claramente delimitados. Em muitas linguagens, essa separação é feita por marcadores especiais: usa-se o ponto e vírgula (;) para indicar o fim de uma instrução e elementos como chaves {} ou as palavras reservadas begin/end para agrupar blocos de código. No entanto, existem linguagens que utilizam a **indentação** como recurso principal para estruturar o código.

Indentação é o recuo do texto em relação à margem. Esse recurso cria uma hierarquia visual que organiza parágrafos (em textos comuns) ou blocos lógicos (em programação), melhorando drasticamente a legibilidade. O recuo pode ser feito usando espaços ou tabulação

(tecla "Tab"), que insere automaticamente um espaçamento padrão (geralmente equivalente a 4 espaços).

Nota: No pseudocódigo adotado pela disciplina de IC, a indentação é obrigatória. Ela serve tanto para definir a estrutura dos blocos quanto para garantir a clareza e organização do código (Figura 28).

```
Se x>1 então
|    Escreva(“Número maior que zero”)
Senão
|    Se x=0 então
|    |    Escreva(“Número igual a zero”)
|    Senão
|    |    Escreva(“Número menor que zero”)
|    Fimse
Fimse
```

```
Se x>1 então
Escreva(“Número maior que zero”)
Senão
Se x=0 então
Escreva(“Número igual a zero”)
Senão
Escreva(“Número menor que zero”)
Fimse
Fimse
```

Figura 28: pseudocódigo com e sem indentação

Apresentação de um algoritmo completo

Visando demonstrar a aplicação do conteúdo abordado até este tópico, será mostrado a seguir um algoritmo, em pseudocódigo, que abrange o conceito de variáveis, operadores aritméticos, comentários e comandos de entrada e saída . Este algoritmo recebe como entrada um determinado valor em segundos e oferece como saída esse valor convertido em horas, minutos e segundos.

```
1  Algoritmo “05_02 Conversor_segundos”
2  // Esse algoritmo converte um valor total de segundos em horas, minutos e segundos.
3  Var
4      total_segs, horas, minutos, segundos_restantes, segundos_restantes_final: inteiro
5  Inicio
6      // Entrada de dados
7      Escreva (“Digite a quantidade de segundos: “)
8      Leia (total_segs)
9
10     // Processamento
11     horas ← total_segs \ 3600
12     segundos_restantes ← total_segs % 3600
13     minutos ← segundos_restantes \ 60
14     segundos_restantes_final ← segundos_restantes % 60
15
16     // Saída de dados
17     Escreva (horas, "hora(s)," ,minutos, "minuto(s) e" ,segundos_restantes_final, "segundo(s).")
18 FimAlgoritmo
```

LISTA DE EXERCÍCIOS - CAPÍTULO 5

Todos os exercícios deste capítulo devem ser desenvolvidos em Pseudocódigo. O desenvolvimento e a execução do código poderão ser feitos no software Visualg.

5.1 Desenvolva um algoritmo que leia o nome de um aluno e escreva a seguinte mensagem: “Bem-vindo à disciplina de Introdução à Computação, <nome do aluno digitado>!”.

5.2 Escreva um algoritmo que receba como entrada os dados de um aluno (número, nome, idade e pelotão) e apresente esses dados na tela do computador.

5.3 Crie um algoritmo que calcule e apresente o consumo médio de combustível de um veículo. O algoritmo deverá solicitar ao usuário que informe, via teclado, a quilometragem inicial e final e a quantidade de combustível consumida.

5.4 Formule um algoritmo que leia e apresente na tela os dados de uma pessoa (nome, idade, endereço, telefone de contato).

5.5 Escreva um algoritmo que receba via teclado os valores de preço unitário e quantidade e apresente na tela o preço total. Inclua no algoritmo informações impressas na tela, informando sobre os dados a serem inseridos, assim como sobre os dados que estão sendo mostrados como saída.

5.6 Escreva um algoritmo, em pseudocódigo, que recebe como entrada, via teclado, o valor dos lados de um retângulo e calcula e mostra a área e o perímetro desse retângulo.

5.7 Escreva, em pseudocódigo, um algoritmo que leia, calcule e escreva a média ponderada de quatro números inteiros inseridos pelo usuário, via teclado, com os seguintes pesos: primeiro número digitado: peso 1; segundo número digitado peso 2, terceiro e quarto números digitados, peso 3.

5.8 Elabore um algoritmo que receba um número inteiro e escreva na tela o número fornecido, o antecessor desse número e o seu sucessor.

5.9 Faça um algoritmo que receba como entrada as medidas dos dois catetos de um triângulo retângulo e calcule e exiba a medida da hipotenusa.

5.10 Desenvolva um algoritmo, em pseudocódigo, que lê o horário (em horas e minutos) de partida e de chegada de um viajante e calcula e informa o tempo total de viagem (em horas e minutos). Assumimos que as viagens sempre são iniciadas e concluídas no mesmo dia.

5.11 Os funcionários de uma empresa recebem 12 reais por hora trabalhada e 40 reais por dependente. Sobre o salário bruto são feitos descontos de 8,5% para o INSS e 5% para IR. Faça um algoritmo que solicite ao usuário a inserção, via teclado, do nome, número de horas trabalhadas e número de dependentes de um funcionário e após a leitura, escreva qual o nome, salário bruto, os valores descontados para cada tipo de imposto e finalmente qual o salário líquido a receber.

```
Nome do funcionário: João Silva
Horas trabalhadas: 200
Dependentes: 2
========================================
Nome: João Silva
Salário Bruto: 2400,00
INSS (8,5%): 204,00
IR (5%): 120,00
Salário líquido: 2076,00
```

5.12 O preço de um veículo é calculado pela soma do preço de fábrica com o preço dos impostos (45% do preço de fábrica) e a percentagem do revendedor (28% do preço de fábrica). Faça um algoritmo que leia o nome do automóvel e o preço de fábrica e imprima o nome do automóvel, o preço de fábrica, o valor dos impostos, o valor destinado ao revendedor e o preço final. Exemplo da execução:

```
Nome do automóvel: HB20S
Preço de Fábrica: 80000,00
========================================
Automóvel: HB20S
Preço de Fábrica: 80000,00
Impostos (45%): 36000,00
Margem da revenda(28%): 22400,00
Preço Final: 138400,00
```

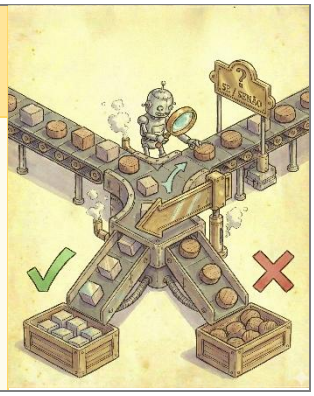
5.13 Escreva um algoritmo que receba como entradas as coordenadas de dois pares ordenados do plano cartesiano (x, y), calcule e imprima:

- a distância entre esses dois pontos;
- coeficiente angular da reta que passa por ambos os pontos;
- o coeficiente linear da reta que passa por ambos os pontos.

Dados:

- *Distância entre dois pontos* = $(((x_1-x_2)^2) + ((y_1-y_2)^2))^{1/2}$
- *Equação geral da reta:* $y = ax + b$ *a = coeficiente angular e b = coeficiente linear*
- $a = (y_2-y_1)/(x_2-x_1)$
- $b = y-ax$

6. Condicionais



- *Aborda as estruturas condicionais, que permitem que o fluxo de execução de um algoritmo seja determinado pela avaliação de uma ou mais condições.*
- *Detalha Condicionais Simples, Condicionais Compostas e Condicionais Aninhadas.*
- *Demonstra o emprego de seleção de múltipla escolha.*

As estruturas condicionais, também chamadas de estruturas de seleção ou estruturas de decisão, destinam-se a determinar a execução de uma instrução ou um conjunto de instruções, quando alguma condição específica for satisfeita.

As condicionais, consideradas uma das estruturas de controle de fluxo de instruções, são de grande importância no desenvolvimento de programas de computadores, pois elas permitem o desvio do fluxo de execução do algoritmo, fazendo com que o caminho a ser seguido durante a execução do programa seja determinado por meio de testes realizados. Esses testes são constituídos por expressões relacionais e/ou lógicas, que retornarão como resultado um valor lógico (verdadeiro ou falso).

As estruturas condicionais podem ser simples ou compostas, como veremos a seguir.

6.1. Condicionais Simples

Em uma estrutura condicional simples, uma instrução ou um bloco de instruções será executado apenas se uma determinada **condição for verdadeira** (Figura 29).

Para construir estruturas condicionais simples, faremos uso de algumas palavras reservadas. A sintaxe da condicional simples em pseudocódigo é:

```
Se <condição> então
    <instrução/bloco de instruções>
Fimse
```

As palavras reservadas utilizadas no pseudocódigo para a implementação das condicionais simples são: se, então e fimse.

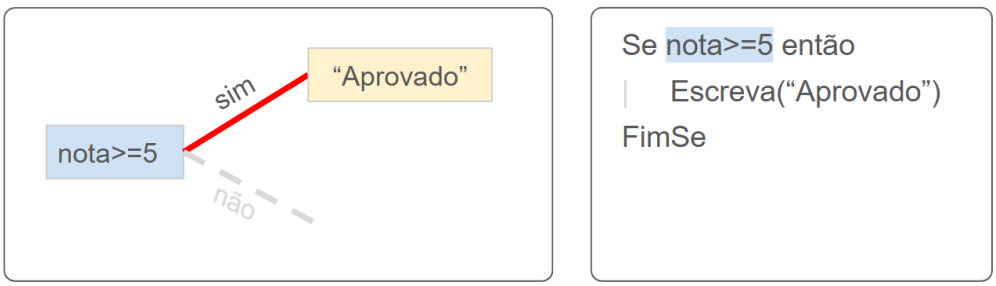


Figura 29: árvore de decisão associada à condicional simples

A seguir, será mostrado um exemplo do uso da condicional simples na elaboração de um algoritmo que calcula a média de um aluno.

Exemplo de condicional simples em pseudocódigo:

```
1  Algoritmo "06_01 media_aluno2"
2  \\ Algoritmo que calcula a média do aluno a partir das notas dos dois semestres e
3  \\ apresenta a mensagem de aprovação, caso sua nota seja igual ou superior a 5.
4  Var
5     nota1, nota2, media: real
6  Inicio
7     Escreva ("Digite a nota do primeiro semestre: ")
8     Leia (nota1)
9     Escreva ("Digite a nota do segundo semestre: ")
10    Leia (nota2)
11    media ← (nota1 + nota2)/2
12    Escreval ("A média é: ",media)
13    Se media >= 5 então
14          Escreva ("Parabéns!!!! Você foi aprovado.")
15    Fimse
16 FimAlgoritmo
```

6.2. Condicionais Compostas

Na condicional simples, um determinado trecho de código seria executado caso o resultado da expressão lógica fosse verdadeiro. Se o resultado da expressão fosse falso, ele seria ignorado.

As condicionais compostas são destinadas aos casos em que necessitamos que sejam executadas algumas instruções quando o resultado lógico da expressão for verdadeiro e outras quando o resultado for falso (Figura 30).

A sintaxe da condicional composta em pseudocódigo é:

```
Se <condição> então
    <instrução/bloco de instruções (verdade)>
Senão
    <instrução/bloco de instruções (falso)>
FimSe
```

Palavras reservadas utilizadas neste tipo de condicional no pseudocódigo: se, então, senão e fimse.

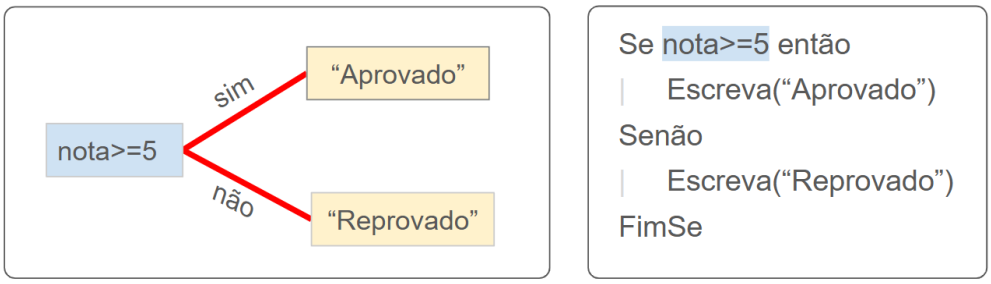


Figura 30: árvore de decisão associada à condicional composta

No algoritmo exemplificando a condicional simples (06_01), era apresentada uma mensagem somente para o caso de aprovação do aluno. No exemplo a seguir, será feita uma melhoria em tal algoritmo de modo que seja apresentada uma mensagem também para o caso de reprovação.

Exemplo de condicional composta em pseudocódigo:

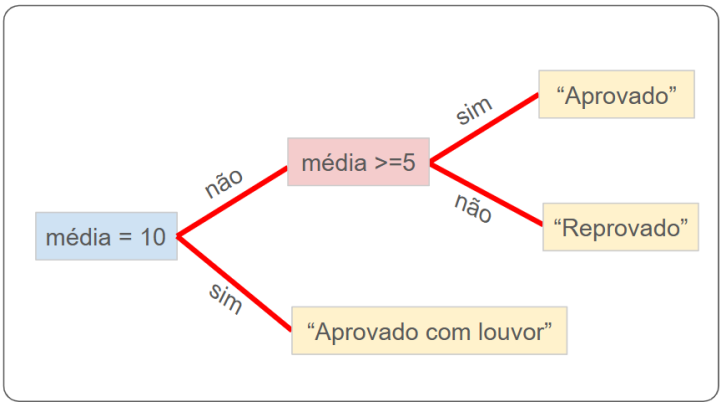
```
1  Algoritmo “06_02 media_aluno3”
2  // Algoritmo que calcula a média do aluno a partir das notas dos dois semestres e apresenta
   // a mensagem de aprovação ou reprovação, caso sua nota seja igual ou superior a 5 ou
3  // inferior a 5, respectivamente.
4  Var
5      nota1, nota2, media: real
6  Inicio
7      Escreva (“Digite a nota do primeiro semestre: “)
8      Leia (nota1)
9      Escreva (“Digite a nota do segundo semestre: “)
10     Leia (nota2)
11     media ← (nota1 + nota2)/2
12     Escreval (“A média é: “,media)
13     Se media >= 5 então
14         Escreva ("Parabéns!!!! Você foi aprovado.")
15     Senão
16         Escreva ("Sinto muito, mas você foi reprovado.")
17     FimSe
18 FimAlgoritmo
```

6.3. Condicionais Aninhadas

Uma variação das condicionais compostas são as condicionais aninhadas ou encadeadas. Elas são caracterizadas pela existência de condicionais dentro de outras condicionais, havendo, portando, mais de uma condição a ser analisada (Figura 31).

A sintaxe da condicional aninhada em pseudocódigo é:

```
Se <condição1> então
|    <instruções (condição 1 verdade)>
Senão
|    Se <condição2> então
|    |    <instruções (condição1 falso, condição2 verdade)>
|    Senão
|    |    <instruções (condição1 falso, condição2 falso)>
|    Fimse
Fimse
```




```
Se media=10 então
    Escreva("Aprovado com louvor")
Senão
    Se media>=5 então
        Escreva("Aprovado")
    Senão
        Escreva("Reprovado")
    FimSe
FimSe
```

Figura 31: árvore de decisão associada à condicionais aninhadas

A seguir, será mostrado como o algoritmo codificado anteriormente pode ser ajustado, de modo a exemplificar uma condicional composta aninhada. Na nova versão do algoritmo, serão analisadas três condições: o aluno foi aprovado com média 10 (aprovação com louvor), o aluno foi aprovado (mas não com média 10) e o aluno foi reprovado (média inferior a 5).

Exemplo de condicional composta aninhada em pseudocódigo:

```
1  Algoritmo "06_03 media_aluno4"
2  // Algoritmo que calcula a média do aluno a partir das notas dos dois semestres
3  // e apresenta a mensagem de aprovação, reprovação ou aprovação com louvor, caso sua
4  // nota seja igual ou superior a 5, inferior a 5 ou igual a 10, respectivamente.
5  Var
6      nota1, nota2, media: real
7  Inicio
8      // Entrada de dados
9      Escreva ("Digite a nota do primeiro semestre: ")
10     Leia (nota1)
11     Escreva ("Digite a nota do segundo semestre: ")
12     Leia (nota2)
13     // Processamento
14     media ← (nota1 + nota2)/2
15
16     Escreva ("A média é: ",media)                    // Saída de dados
17     Se media = 10 então
18         Escreva ("Sensacional!!!! Você foi aprovado com louvor.")      // Saída de dados
19     Senão
20         Se media >= 5 E media < 10 então
21             Escreva ("Parabéns!!!! Você foi aprovado.")                // Saída de dados
22         Senão
23             Escreva ("Sinto muito, mas você foi reprovado.")           // Saída de dados
24         Fimse
25     Fimse
26 FimAlgoritmo
```

6.4. Seleção de Múltipla Escolha

Seleção de Múltipla Escolha: este tipo de condicional é utilizado em muitas linguagens de programação, quando existe um conjunto de condições a serem testadas, sendo que para cada condição escolhida são executadas instruções diferentes. Seria como se fosse uma espécie de "Menu", em que, dependendo da opção escolhida, teríamos os resultados associados àquela escolha. A sintaxe da seleção de múltipla escolha em pseudocódigo está ao lado:

```
Escolha <Expressão/variável de escolha>
caso <valor 1>
    <instruções referentes ao caso valor 1>
caso <valor 2>
    <instruções referentes ao caso valor 2>
caso <valor 3>
    <instruções referentes ao caso valor 3>
outrocaso
    <instruções referentes ao outrocaso>
Fimescolha
```

Exemplo de algoritmo com seleção de múltipla escolha em pseudocódigo:

```
1  Algoritmo "06_04 Calculadora_Básica"
2  // Algoritmo que executa uma das quatro operações básicas, dependendo da
3  // escolha feita pelo usuário, conforme menu de opções apresentado.
4  Var
5     numero1, numero2, resultado: real
6     operacao: inteiro
7  Inicio
8     Escreva ("Digite o primeiro número: ")
9     Leia (numero1)
10    Escreva ("Digite o segundo número: ")
11    Leia (numero2)
12    Escreva ("Menu: 1 – soma; 2 – subtração; 3 – multiplicação; 4 – divisão")
13    Escreva ("Escolha a operação, conforme Menu acima: ")
14    Leia (operacao)
15    Escolha operacao
16       caso 1
17          resultado ← (numero1+numero2)
18          Escreva ("O resultado da soma é: ",resultado)
19       caso 2
20          resultado ← (numero1-numero2)
21          Escreva ("O resultado da subtração é: ",resultado)
22       caso 3
23          resultado ← (numero1*numero2)
24          Escreva ("O resultado da multiplicação é: ",resultado)
25       caso 4
26          resultado ← (numero1/numero2)
27          Escreva ("O resultado da divisão é: ",resultado)
28       outro caso
29          Escreva ("Opção inválida")
30    FimEscolha
31 FimAlgoritmo
```

LISTA DE EXERCÍCIOS - CAPÍTULO 6

Todos os exercícios deste capítulo devem ser desenvolvidos em Pseudocódigo. O desenvolvimento e a execução do código poderão ser feitos no software Visualg.

6.1 Desenvolva um algoritmo, nomeado Estrutura_Decisao_Simples, que solicita ao usuário que digite um número no teclado e, caso o número digitado seja par, mostre na tela o seguinte texto: “O número digitado é par”.

6.2 Modifique o algoritmo do exercício anterior, alterando seu nome para Estrutura_Decisao_Composta e acrescente a possibilidade de mostrar o texto “O número digitado é ímpar”, quando for essa a situação.:

6.3 Faça um algoritmo que receba como entrada, via teclado, o valor do salário e a graduação de um militar e calcula e mostra o valor do salário reajustado com um índice que varia, de acordo com a graduação, da seguinte forma: soldado: 50%, cabo: 40%, sargento: 30% e outra graduação: 20%.

6.4 Escreva e implemente um algoritmo que receba como entrada, via teclado, a idade de uma pessoa e exiba a informação se ela é menor de idade, maior de idade ou idoso, conforme ela possua menos que 18 anos, entre 18 e 65 anos ou acima de 65 anos, respectivamente. Exemplo de saída:

```
Informe sua idade: 70
Você é considerada uma pessoa idosa.
```

6.5 Escreva e implemente um algoritmo que calcule o IMC (Índice de Massa Corporal) de uma pessoa, de acordo com seu peso e altura. O programa deve informar se a pessoa está abaixo do peso (IMC menor que 20), normal (IMC entre 20 e 25), excesso de peso (entre 26 e 30), obesa (IMC entre 31 e 35) ou com obesidade mórbida (acima de 35).

Dado: fórmula para cálculo do IMC: peso (altura)2 , sendo o peso (em kg) e a altura (em metros).

6.6 Elabore um algoritmo que receba como entrada, via teclado, os valores das coordenadas (x,y) de um ponto no plano cartesiano e forneça como saída, no monitor, a localização do ponto (na origem, sobre o eixo x, sobre o eixo y, no 1o quadrante, no 2o quadrante, no 3o quadrante ou no 4º quadrante). Exemplo de saída

```
O ponto de coordenada (x,y) encontra-se no 1º quadrante.
```

6.7 Crie um algoritmo que receba como entrada o percurso a ser realizado em quilômetros e o tipo (capacidade) de viatura a ser empregada e informe o consumo estimado de combustível, sabendo-se que o consumo das viaturas do tipo 1⁄4 ton, 2,5 ton e 5 ton são, respectivamente, 8, 6 e 4 quilômetros por litro de combustível.

6.8 Formule um algoritmo que receba como entrada quatro números e conte e apresente quantos deles são negativos.

6.9 Faça um algoritmo que receba, via teclado, as medidas dos três lados de um triângulo, e informe seu tipo (equilátero, isósceles ou escaleno).

Observação: antes que seja verificado o tipo do triângulo, o algoritmo deve verificar se as medidas dos lados fornecidas pelo usuário, formam triângulo, ou seja, a medida de um lado deve ser menor do que a soma dos outros dois.

6.10 Elabore um algoritmo que receba, via teclado, um número inteiro entre 1 e 12 (representando cada mês do ano) e imprima se este mês tem 28, 30 ou 31 dias. Caso o usuário digite um número que não esteja entre 1 e 12, o algoritmo deve mostrar uma mensagem de entrada inválida e encerrar.

Observação: assuma que fevereiro sempre possui 28 dias.

6.11 Implemente um algoritmo que calcula as raízes de uma equação do segundo grau do tipo ax^2+bx+c, utilizando a fórmula de Bhaskara. O algoritmo deverá receber via teclado os valores dos índices a, b e c da equação e mostrar mensagem informando a quantidade de raízes (caso possua), com os seus respectivos valores ou que a equação não possui raízes reais.

Dado: fórmula de Bhaskara: $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$. *Exemplos de saída:*

```
Informe o valor de a: 1
Informe o valor de b: -5
Informe o valor de c: 6
-------------------------------------------------
A equação possui duas raízes: 2 e 3
```

6.12 Desenvolva um programa que receba a renda anual de um militar e calcula o imposto de renda, cujos índices variam de acordo com as seguintes faixas:

- 1ª faixa: renda menor ou igual a R$ 21.450,00: 15%.
- 2ª faixa: renda maior que R$ 21.450,00 e menor ou igual a 51.900,00: 15% sobre o teto da primeira faixa (3217.50), mais 28% do que exceder à primeira faixa.
- 3ª faixa: renda maior que 51.900,00 – alíquotas referentes aos tetos das duas faixas anteriores (11743.50), mais 31% do que exceder à segunda faixa.

```
Informe a renda: 57000.00
-------------------------------------------------
2ª faixa de imposto
Imposto devido: R$ 13171,50
Alíquiota real (%): 23,10
```

6.13 Construa um algoritmo que leia quatro números digitados pelo usuário, some o primeiro com o segundo número, some o terceiro com o quarto número. Informe qual soma é a maior ou se são iguais. Caso o usuário digitar um valor igual a zero ou negativo, em qualquer um dos quatro números, o algoritmo deve mostrar uma mensagem de valor inválido e finalizar.

6.14 Construa um programa em que o usuário informe, via teclado, as coordenadas (x,y) do centro de um círculo (em um plano cartesiano), bem como o seu raio; e forneça, também, as coordenadas (x,y) de um ponto de teste. De posse dos dados inseridos pelo usuário, o algoritmo deverá informar se o ponto de teste está situado no interior do círculo (mas não no centro), no centro do círculo, na circunferência (fronteira) ou fora do círculo.

Dados: Equação da circunferência: $r = \sqrt{(x_p - x_c)^2 + (y_p - y_c)^2}$ *, onde:*

x_p *= coordenada x do ponto de teste*
x_c *= coordenada x do centro do círculo*
y_p *= coordenada y do ponto de teste*
y_c *= coordenada y do centro do círculo*

Exemplo de execução do algoritmo:

Coordenada xc: **0**
Coordenada yc: **0**
Raio: 5
Coordenada xp: **6**
Coordenada yp: **0**

O ponto está fora do círculo

6.15 Faça um programa que receba o código da carga, o peso da carga em toneladas, calcule e apresente: o peso da carga em quilos, o valor do frete, o valor do seguro (percentual sobre o valor da carga) e o valor total a ser pago (frete e seguro).

Dados:

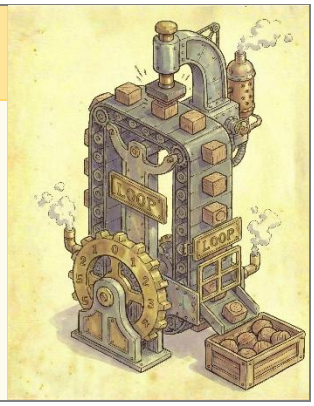
Código da carga	*Frete por quilo*	*Seguro da carga*
10 a 20	*R$ 5.00*	*3%*
21 a 30	*R$ 8.00*	*5%*
31 a 40	*R$ 10.00*	*8%*

Exemplo de execução do algoritmo:

Código da carga: **20**
Peso da carga (t): **10.5**
Valor da carga: **R$ 600000.00**

Peso da carga: 10500 Kg
Frete: R$ 52500.00
Seguro: R$ 18000.00
Valor total: R$ 70500.00

7. Estruturas de repetição



- *Estruturas de repetição são mecanismos que permitem a execução repetida de um trecho de código, otimizando o tamanho e a legibilidade dos algoritmos.*
- *As estruturas de repetição são divididas em laços contados, para um número predefinido de repetições; e laços condicionais, que repetem instruções enquanto uma condição específica for verdadeira ou falsa.*

7.1 Introdução e Tipos de Estruturas de repetição

Estruturas de repetição, laços de repetição ou loops, possibilitam a repetição de alguma parte do algoritmo, tantas vezes quantas forem necessárias, dispensando a necessidade de reescrever alguns comandos ou instruções. A existência das estruturas de repetição faz com que o tamanho do algoritmo seja reduzido, facilitando sua leitura e entendimento.

A cada execução do bloco de instruções contidas dentro dos laços de repetição é dado o nome de iteração.

Imaginemos que em um algoritmo para cálculo da média do aluno, tivéssemos a necessidade de calcular a média de uma classe inteira, composta por vinte alunos. Sem a utilização de laços de repetição teríamos a necessidade de reescrever o mesmo código várias vezes (o número de alunos da classe), ou seja, o tamanho do algoritmo mencionado seria, praticamente, multiplicado por trinta. Visando a resolução desse tipo de problema, são utilizadas as estruturas de repetição.

As estruturas de repetição podem ser de dois tipos: laços contados (repete uma instrução ou um conjunto de instruções um número preestabelecido de vezes) e laços condicionais (a instrução ou conjunto de instruções se repetirão enquanto uma determinada condição for verdadeira) [4].

Figura 32: tipos de laço de repetição

7.2 Estrutura de Repetição do tipo laço contado

As estruturas do tipo laço contado são utilizadas quando já sabemos de antemão o número de vezes que o trecho do código deverá ser repetido.

Sintaxe da estrutura de repetição do tipo laço contado em pseudocódigo:

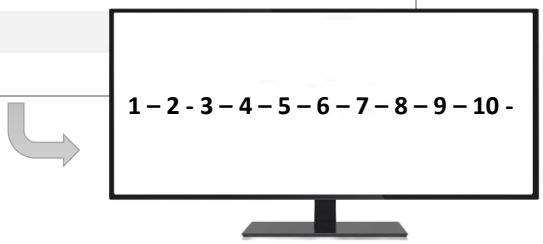
```
Para  <variável de controle> de <início*1> até <fim*1> passo <valor*1> faça
|     <instrução/bloco de instruções)>
FimPara
```

*1 *valores numéricos*

A variável de controle, partindo do valor mencionado em <início>, tem o seu valor incrementado de acordo com o que foi especificado em <passo> a cada iteração, até chegar ao valor informado em <fim>. As iterações são executadas enquanto a variável de controle não ultrapasse o valor <fim>. Desse modo, as iterações ocorrerão do valor de início ao valor fim. A variável de controle também deve ser declarada na seção "var", como do tipo inteiro.

Um primeiro exemplo de laço contado é apresentado a seguir, evidenciando o uso da variável de controle, como "contadora" das repetições realizadas.

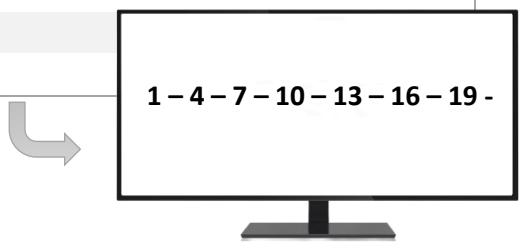
```
1  Algoritmo "07_01 Uso de Laço Contado"
2  // Algoritmo que demonstra a utilização de um laço de repetição contado ,
3  // evidenciando o uso de uma variável de controle.
4  Var
5      contador: inteiro
6  Inicio
7      Para contador de 1 até 10 passo 1 faça
8          Escreva (contador , " - ")
9      Fimpara
10 FimAlgoritmo
```



No algoritmo anterior, se fosse omitido o trecho "passo 1", seria assumido o valor 1 para passo, como padrão, com execução idêntica.

Também é possível personalizar o laço contado, para que em cada repetição o contador seja incrementado em valor diferente de 1. No algoritmo 07_02, cada repetição incrementa o contador em 3 (passo 3).

```
1  Algoritmo "07_02 Uso Laço Contado Passo 3"
2  // Algoritmo que demonstra a utilização de um laço de repetição contado ,
3  // evidenciando o passo diferente de 1, que é o valor padrão
4  Var
5      contador: inteiro
6  Inicio
7      Para contador de 1 até 20 passo 3 faça
8          Escreva (contador , " - ")
9      Fimpara
10 FimAlgoritmo
```



No algoritmo 07_02, a contagem parou em 19 porque o próximo número da sequência ultrapassaria o limite final definido no comando, que é 20.

A seguir será demonstrado, em pseudocódigo, um exemplo mais amplo de utilização da estrutura de repetição do tipo laço contado, aproveitando o algoritmo que calcula a média das notas de aluno, elaborado no capítulo sobre condicionais. Esse algoritmo será reescrito de modo que seja possível calcular a média de uma turma contendo dez alunos.

Exemplo de algoritmo contendo a implementação de uma estrutura de repetição do tipo laço contado em pseudocódigo:

```
1  Algoritmo "07_03 Média5"
2  // Algoritmo que calcula a média dos alunos a partir das notas dos dois semestres
3  // e apresenta a mensagem de aprovação (nota>=5) ou reprovação(nota<5)
4  Var
5     nota1, nota2, media: real
6     num_alunos: inteiro
7  Inicio
8     Para num_alunos de 1 até 10 passo 1 faça
9           Escreva ("Digite a nota do primeiro semestre: ")
10          Leia (nota1)
11          Escreva ("Digite a nota do segundo semestre: ")
12          Leia (nota2)
13          media ← (nota1 + nota2)/2
14          Escreva ("A média é: ",media)
15          Se media >= 5 então
16                Escreva ("Aluno aprovado.")
17          Senão
18                Escreva ("Aluno reprovado.")
19          Fimse
20     Fimpara
21 FimAlgoritmo
```

Todo este bloco de instruções será repetido 30 vezes

O algoritmo acima pode ser melhorado, visando atender classes que contenham números variados de alunos. Para tanto, será criada uma variável que receba a quantidade de alunos em uma classe e esta constituirá o campo fim no laço de repetição.

```
1  Algoritmo "07_04 Média6"
2  // Melhora algoritmo anterior, possibilitando indicar quantidade de alunos da classe
3  Var
4     nota1, nota2, media: real
5     Qtd, num_alunos: inteiro
6  Inicio
7     Escreva ("Digite a quantidade de alunos da classe: ")
8     Leia(qtd)
9     Para num_alunos de 1 até qtd passo 1 faça
10          Escreva ("Digite a nota do primeiro semestre: ")
11          Leia (nota1)
12          Escreva ("Digite a nota do segundo semestre: ")
13          Leia (nota2)
14          media ← (nota1 + nota2)/2
15          Escreva ("A média é: ",media)
16          Se media >= 5 então
17                Escreva ("Aluno aprovado.")
18          Senão
19                Escreva ("Aluno reprovado.")
20          Fimse
21     Fimpara
22 FimAlgoritmo
```

7.3 Estrutura de repetição do tipo laço condicional

Como mencionado anteriormente, a estrutura de repetição do tipo laço condicional é indicada, principalmente, para os casos em que o número de repetições a ser executado não é conhecido. Apesar disso, elas podem ser usadas também nas situações em que o número de vezes que um trecho de código deverá ser repetido já é conhecido.

ESTRUTURA ENQUANTO

Essa estrutura caracteriza-se pela realização do teste lógico no início do laço de repetição. Se o resultado do teste lógico (condição) for verdadeiro, as instruções que se encontram no interior do laço são executadas; caso contrário, o laço não será executado, ou seja, o programa passa para a próxima instrução após a estrutura.

Sintaxe da estrutura enquanto em pseudocódigo:

```
Enquanto  <condição> faça
|     <instrução/bloco de instruções)>
FimEnquanto
```

Para garantir que a repetição tenha fim, é fundamental que a variável testada na **<condição>** sofra alguma alteração dentro do **<bloco de instruções>**. Se essa atualização não ocorrer, a condição permanecerá sempre verdadeira, resultando em um **loop infinito** do algoritmo.

Imaginamos em nossos exemplos anteriores em que foi criado algoritmos que calculam a média de alunos, desejássemos que o algoritmo fosse executado indefinidamente até que o usuário resolvesse parar. Com a utilização do laço enquanto é possível resolver esse problema, conforme mostrado abaixo:

Algoritmo codificado em pseudocódigo:

```
1  Algoritmo "07_05 Media7"
2  // Algoritmo que calcula e exibe a média dos alunos a partir das notas dos
3  // dois semestres usando laço condicional enquanto.
4  Var
5     nota1, nota2, media: real
6     resp: inteiro
7  Inicio
8     resp ← 1
9     Enquanto resp = 1 faça
10       Escreva ("Digite a nota do primeiro semestre: ")
11       Leia (nota1)
12       Escreva ("Digite a nota do segundo semestre: ")
13       Leia (nota2)
14       media ← (nota1 + nota2)/2
15       Escreval ("A média é: ",media)
16       Escreval ("Digite 1 para continuar ou qualquer outro número para sair: ")
17       Leia (resp)
18    Fimenquanto
19 FimAlgoritmo
```

Enquanto a variável **resp** contiver o valor 1, o bloco de instruções dentro do laço de repetição será executado.

Como pode ser observado nos algoritmos anteriores, foi utilizada uma variável de controle do laço, chamada resp, que foi inicializada com o valor 1. Essa inicialização foi necessária para permitir que o algoritmo entrasse no laço de repetição. Pode ser constatado, também, que no interior do laço é feita a consulta ao usuário do seu desejo (parar ou continuar) e, dependendo da decisão do usuário, tal manifestação era atribuída ao valor da variável de controle, que repetiria o laço, caso seu valor fosse 1, ou sairia do laço (e do programa neste caso) caso seu valor fosse qualquer outro número diferente de 1.

ESTRUTURA REPITA

A estrutura **Repita** se diferencia por realizar o teste lógico apenas no final do laço. Isso garante uma característica fundamental: os comandos internos são executados pelo menos uma vez, pois a verificação só ocorre após a primeira passagem.

Além disso, sua lógica de parada é inversa à da estrutura *enquanto*. No *Repita*, o laço continua sendo executado enquanto a condição for falsa e só é interrompido quando o objetivo for atingido, ou seja, quando a condição se tornar verdadeira.

Sintaxe da estrutura repita em pseudocódigo:

```
Repita
|     <instrução/bloco de instruções)>
Até <condição>
```

Exemplo de algoritmo que calcula a média de alunos em pseudocódigo, utilizando a estrutura de repetição repita:

```
1  Algoritmo "07_06 Média8"
2  // Algoritmo que calcula e exibe a média dos alunos a partir das notas dos
3  // dois semestres usando laço condicional repita.
4  Var
5     nota1, nota2, media: real
6     resp: inteiro
7  Inicio
8     Repita
9        Escreva ("Digite a nota do primeiro semestre: ")
10       Leia (nota1)
11       Escreva ("Digite a nota do segundo semestre: ")
12       Leia (nota2)
13       media ← (nota1 + nota2)/2
14       Escreval ("A média é: ",media)
15       Escreval ("Digite 1 para continuar ou outro número para sair: ")
16       Leia (resp)
17    Até resp <> 1
18 FimAlgoritmo
```

Todo este bloco de instruções será repetido até que a condição contida na clausula "até" seja verdadeira.

Avalia se a condição foi satisfeita para encerrar as repetições. Se não for, executa mais uma vez o bloco de instruções dentro do laço de repetição.

COMANDO INTERROMPA

O comando interrompa atua como uma saída controlada dentro das estruturas de repetição (Para, Enquanto ou Repita). O comando interrompa aborta imediatamente a execução do laço de repetição atual, sendo que o fluxo de execução do programa salta instantaneamente para a primeira linha de código após o fim do laço (fimpara, fimenquanto ou ate). Será ignorado qualquer código dentro do bloco de repetição que estiver abaixo do comando Interrompa. Será ignorada, também, as verificações de condição que ainda faltavam para terminar o laço de forma normal.

A principal utilidade é a otimização de desempenho e a simplificação da lógica em situações de busca. Imagine um algoritmo que busca por números primos. Considere que o número avaliado seja 1.000.005. O laço de repetição irá percorrer todos os números inferiores a esse buscando um divisor exato (resto zero). Se não for utilizado o comando Interrompa, 1.000.003 repetições serão necessárias. Se utilizado o Interrompa, associado a uma condicional que verifica um divisor exato, a repetição será encerrada com o contador em 5, reduzindo enormemente o tempo de processamento.

```
1  Algoritmo "07_07b Demonstra comando Interrompa"
2  // Algoritmo que demonstra o uso do comando Interrompa
3  // Empregado dentro de um algoritmo de verificação de número primo
4  Var
5      numero, contador, resto: inteiro
6      EhPrimo: logico
7  Inicio
8      EhPrimo ← Verdadeiro          // todo número é primo até prova em contrário
9      Escreva("Digite um número para checar se é primo:")
10     Leia(numero)
11     Para contador de 2 até numero-1 faça
12         resto ← numero % contador          // verifica o resto da divisão
13         Se resto = 0 então                 // verifica se resto zero, divisão exata
14             EhPrimo ← Falso
15             Interrompa
16         FimSe
17     FimPara
18     Se EhPrimo = Verdadeiro então
19         Escreva(numero, " é primo")
20     Senão
21         Escreva(número, " não é primo")
22     FimSe
23 FimAlgoritmo
```

No arquivo de código, no Visualg, há duas versões deste algoritmo ("07_07a" e "07_07b"), com e sem interrompa para comparação do tempo de processamento.

VALIDAÇÃO DE VALORES DE ENTRADA

A validação de entrada de dados é uma técnica essencial para garantir a consistência de um algoritmo, prevenindo que informações incorretas ou fora do escopo (como uma idade negativa ou uma nota acima de 10) causem erros no processamento subsequente. O princípio é simples: não se deve confiar cegamente no que o usuário digita, mas sim verificar a consistência da informação no momento da entrada do dado.

Para implementar essa checagem, podem ser utilizados laços de repetição (tipicamente o enquanto ou repita). A lógica funciona como uma "barreira de contenção": se o valor digitado for inválido, o laço "prende" o fluxo de execução e solicita o dado novamente, repetindo o processo quantas vezes forem necessárias até que o usuário forneça uma entrada que satisfaça as regras estabelecidas.

O algoritmo 07_08 implementa um algoritmo que valida a entrada de uma nota em um pseudocódigo:

```
1  Algoritmo “07_08 Validação de Entrada”
2  // Algoritmo que demonstra o uso de laços de repetição para validar, de forma
3  // persistente, a entrada de valores em um programa.
4  Var
5    nota : real
6  Inicio
7    escreva("Digite a nota do aluno (0 a 10): ")
8    leia(nota)
9    // A condição do laço verifica valores inconsistentes:
10   // Enquanto a nota for menor que 0 ou maior que 10, repete a pergunta.
11   enquanto (nota < 0) ou (nota > 10) faca
12       escreval("Erro: Valor inválido! A nota deve ser entre 0 e 10.")
13       escreva("Tente novamente: ")
14       leia(nota)
15   fimenquanto
16   escreval("Nota válida registrada: ", nota)
17 FimAlgoritmo
```

LISTA DE EXERCÍCIOS - CAPÍTULO 7

Todos os exercícios deste capítulo devem ser desenvolvidos em Pseudocódigo. O desenvolvimento e a execução do código poderão ser feitos no software Visualg.

7.1 Desenvolva um algoritmo que receba um valor inteiro positivo digitado pelo usuário, e apresente os números inteiros maiores que zero, enquanto o somatório deles for menor que o número digitado.

7.2 Elabore um algoritmo que permita que um usuário insira, via teclado, um número inteiro, quantas vezes desejar, sendo que para cada número digitado o algoritmo deve calcular e apresentar o número digitado e o seu quadrado, caso esse número digitado seja ímpar; e o número digitado e seu cubo, caso o número digitado seja par. O algoritmo será encerrado quando o usuário digitar o valor 0, para tanto, essa informação deverá ser fornecida ao usuário, como parte integrante do algoritmo.

7.3 Construa um algoritmo que calcule e apresente a tabuada de um número escolhido pelo usuário, sendo que o usuário deverá inserir, também, o limite superior da tabuada. O algoritmo não deverá permitir que o usuário insira um valor menor que dois, tanto para o número como para o limite, mostrando a mensagem de entrada inválida e solicitando uma nova entrada, até que o valor inserido seja válido.

```
Qual tabuada você quer ver? (Digite um valor >= 2): 5
Você quer a tabuada do  5 até qual número? (>= 2): 9
========================================
 RESULTADO: Tabuada do  5 até o  9
========================================
 5 x  1 =  5
 5 x  2 =  10
 5 x  3 =  15
 5 x  4 =  20
 5 x  5 =  25
 5 x  6 =  30
 5 x  7 =  35
 5 x  8 =  40
 5 x  9 =  45
========================================
>>> Fim da execução do programa !
```

7.4 Escreva um algoritmo que apresente todos os números naturais divisíveis por 5 e que sejam inferiores a um valor digitado pelo usuário.

7.5 Utilizando o laço “enquanto” em pseudocódigo, elabore um algoritmo que imprima todas as tabuadas do 1 ao 10. A apresentação da tabuada deverá está no formato, conforme mostrado abaixo:

```
 1    2    3    4    5    6    7    8    9   10
 2    4    6    8   10   12   14   16   18   20
 3    6    9   12   15   18   21   24   27   30
 4    8   12   16   20   24   28   32   36   40
 5   10   15   20   25   30   35   40   45   50
 6   12   18   24   30   36   42   48   54   60
 7   14   21   28   35   42   49   56   63   70
 8   16   24   32   40   48   56   64   72   80
 9   18   27   36   45   54   63   72   81   90
10   20   30   40   50   60   70   80   90  100
```

7.6 Escreva um algoritmo que receba a idade e o estado civil de vários militares e, ao final, imprima a quantidade de militares casados, solteiros, separados e viúvos. O algoritmo finaliza quando for informado o valor zero para idade.

7.7 Construa um algoritmo que leia números inteiros até que seja inserido um número negativo. Ao final, o algoritmo deverá informar a média dos números digitados, o maior e o menor valor inserido.

7.8 Elabore um algoritmo que solicite ao usuário um número inteiro positivo (maior que 0). O programa deve descobrir se esse número é um Quadrado Perfeito, ou seja, se existe um número inteiro N tal que NxN seja igual ao número digitado.
*Observação: a) utilize o comando interrompa para que o laço de repetição não verifique as multiplicações além do valor que extrapole o número informado; b) não use a potenciação fracionária (ex: n ** 0.5) para achar a raiz, já que o objetivo é treinar laço.*

7.9 Desenvolva um algoritmo que lê um número inteiro positivo do teclado e informa se ele é par ou ímpar. Caso o número lido seja não positivo (menor do que 1), é apresentada uma mensagem de erro e solicitado o número novamente até que ele seja válido. Ao final da execução, o programa pergunta ao usuário se ele deseja executar o programa novamente. O usuário poderá escolher entre parar ou continuar. Neste último caso, é solicitado novamente a entrada e executa até o usuário optar por parar.

7.10 Desenvolva um algoritmo que realiza o somatório da sequência: 1, 5, 9, 13, 17, 21, ... ,53. Progressão Aritmética (PA), cujo primeiro termo é 1, último termo é 53 e razão 4.

7.11 Desenvolva um algoritmo que receba como entrada, via teclado, um número inteiro não negativo (inclui o zero), digitado pelo usuário, e calcula e apresenta o fatorial desse número. Caso o usuário digite um valor negativo, o algoritmo deverá apresentar uma mensagem de entrada inválida, solicitando que o usuário digite um valor válido (maior ou igual zero).Dado: o fatorial de um número é obtido pela multiplicação desse número por todos os seus antecessores até o 1 (um). Ex: 5! = 5 x 4 x 3 x 2 x 1 = 120. 0! = 1! = 1

7.12 Elabore um algoritmo que calcule e imprima o valor de S na série abaixo. O valor de N será digitado pelo usuário. O algoritmo deverá ficar, repetidamente, solicitando que o usuário digite o valor de n, até que o usuário digite um valor inteiro positivo.

7.13 Elabore um algoritmo que permita que o usuário insira, via teclado, os limites (inferior e superior) de uma faixa de valores e imprima os números primos contidos nesta faixa. Caso o usuário digite, como limite inferior da faixa, um número menor ou igual a 1, o algoritmo deverá imprimir a mensagem de entrada inválida e solicitar que o usuário realize uma nova entrada, até que o valor digitado seja válido. O mesmo deverá ocorrer para o caso do usuário digitar um limite superior menor do que o digitado como limite inferior da faixa.

7.14 Implemente um algoritmo que simula um cardápio e permite realizar pedidos dos produtos. Somente no início, o algoritmo mostra os produtos do cardápio na tela, contendo o código, o nome do item e o seu valor.

O usuário poderá fazer pedidos pelo código do item até que seja inserido o valor 0 (zero). Caso o usuário digite um código inexistente no cardápio (com exceção do valor 0 (zero) para sair), o programa deve ficar solicitando que o usuário digite o código de acordo com o cardápio, ou zero para sair. Para cada produto, a quantidade também deve ser informada.

No final, o algoritmo deverá apresentar na tela a quantidade, o nome e o valor de cada item que o usuário escolheu e o valor total a pagar.

Dados:

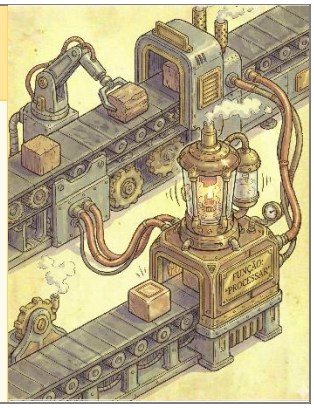
Código	*Produto*	*Valor Unitário*
1	*Refrigerante*	*R$ 5,00*
2	*Cerveja*	*R$ 9,00*
3	*Suco*	*R$ 7,00*
4	*Água*	*R$ 4,00*

Exemplo de execução do algoritmo:

```
----- CARDÁPIO -----
1 - Refrigerante  (R$ 5,00)
2 - Cerveja       (R$ 9,00)
3 - Suco          (R$ 7,00)
4 - Água          (R$ 4,00)
0 - SAIR / FINALIZAR
--------------------
Digite o código do item: 1
Qual a quantidade desejada? 3
-> Adicionado: Refrigerante
---
Digite o código do item: 8
>> Código inválido! Digite entre 1 e 4 (ou 0 para sair).
Digite o código do item: 3
Qual a quantidade desejada? 2
-> Adicionado: Suco
---
Digite o código do item: 0
================================
RESUMO DO PEDIDO
================================
Refrigerante:  3 un x R$ 5,00 = R$ 15.00
Suco:          2 un x R$ 7,00 = R$ 14.00
--------------------------------
TOTAL A PAGAR:      R$ 29.00
================================
```

8. Sub-rotinas

- *Blocos de código isolados que executam tarefas específicas dentro de um programa. Uma **função** sempre retorna um valor ao programa que a invocou, e um **procedimento** apenas executa instruções sem prover um retorno .*
- *Benefícios: modularização, reutilização do código, a redução da complexidade, o aumento da legibilidade e a facilidade de manutenção.*



8.1. Introdução à Funções e Procedimentos

Funções e **procedimentos** são sub-rotinas que executam tarefas específicas e podem ser utilizadas dentro de um programa. Elas são constituídas por porções do código que ficam, de certa maneira, isolados do programa principal.

As sub-rotinas podem ser invocadas pelo programa principal sempre que for necessária a execução das tarefas a que a função ou procedimento se destina. Ao ser invocada (chamada) uma sub-rotina pelo programa principal, pode ser necessário o fornecimento de determinados argumentos para a sua execução. Os parâmetros necessários à função ou ao procedimento são passados pelos programas que os invocam na forma de argumentos.

Vantagens da utilização de sub-rotinas:

- Permite a reutilização do código, o que possibilita o aumento da produtividade, redução da quantidade de erros e do tamanho dos programas e maior facilidade de manutenção.
- Redução da complexidade dos programas, onde tarefas complexas são decompostas em tarefas menores e com menor grau de complexidade.
- Aumento da legibilidade do programa, principalmente se o nome da sub-rotina oferece uma boa noção da tarefa que ela executa.

As linguagens de programação normalmente possuem funções e procedimentos pré-definidos, que foram implementadas pelos desenvolvedores da própria linguagem.

Sub-rotinas demasiadamente complexas podem ainda ser divididas em diversas outras sub-rotinas mais simples. A esse processo de dividir uma sub-rotina em outras dá-se o nome de refinamento sucessivo[9].

As funções e procedimentos devem ser definidos, no interior do algoritmo, antes de serem invocados. É recomendado que tal definição deva ocorrer antes do início do algoritmo, como indicado na Figura 33. Nas linguagens de programação é comum estas sub-rotinas definidas pelo usuário estejam em arquivos separados, sendo referenciadas pelo programa principal, o que faz mais sentido quando pensamos em reaproveitamento do código.

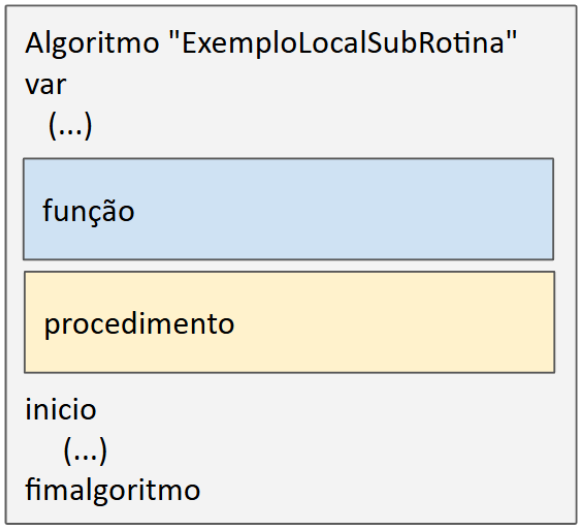


Figura 33: local de definição de sub-rotinas

A diferença entre procedimento e função reside no fato de que uma função sempre retorna um valor, como resultado de sua execução, ao programa que a invocou, enquanto um procedimento não provê qualquer retorno.

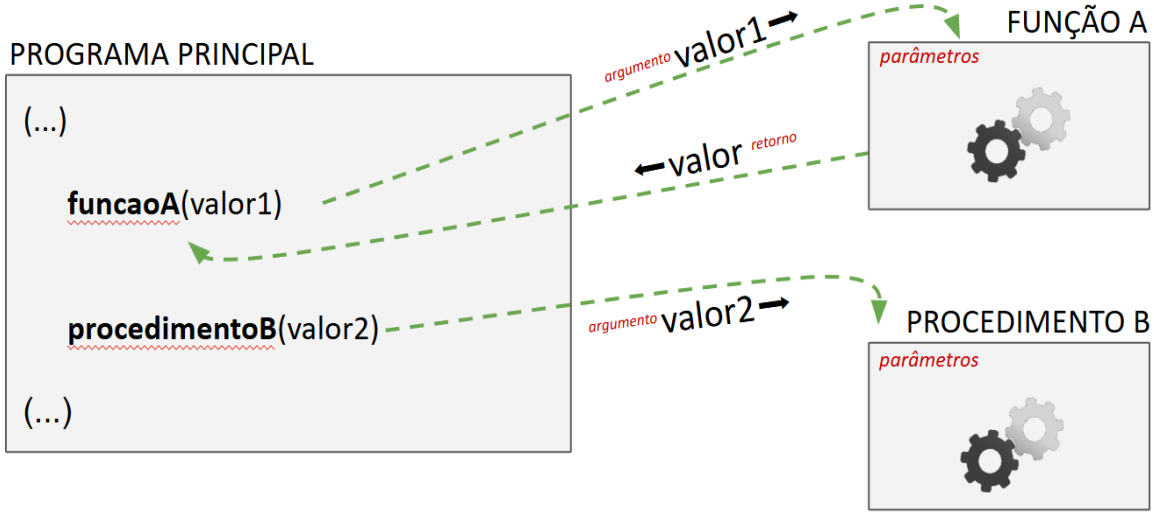


Figura 34: fluxo de sub-rotinas

Assim como as variáveis, funções e procedimentos devem ser nomeadas. Para tanto, defina nomes representativos do que executam estas sub-rotinas.

8.2. Definição de Funções

Uma função é uma unidade de código de programa autônoma desenvolvida para cumprir uma determinada tarefa em particular[6].

Funções são sub-rotinas que podem ser referenciadas (chamadas ou invocadas) em qualquer parte do programa principal, por meio de seu nome, e que retornam um valor ao programa que a invocou. Ao término da execução da função, o controle de processamento retorna automaticamente para a primeira linha de instrução após a linha que efetuou a chamada da sub-rotina.

Sintaxe de definição de função em pseudocódigo:

```
Função <nome>(<parâmetros>:<tipo de dados dos parâmetros>): <tipo de dados de retorno>
Var
   <variáveis locais da função>: <tipos das variáveis locais>
Início
   <corpo da função>
FimFuncao
```

Exemplo de algoritmo que implementa uma função em pseudocódigo:

```
1  Algoritmo "08_01 função_calculadora_potencia"
2  // Algoritmo que recebe como entrada o valor da base e do expoente e calcula e
3  // apresenta a potência, fazendo o uso de uma função para o cálculo.
4  Var
5     valor_base, valor_expoente, valor_resultado: real
6
7  // Definição da função que calcula a potência
8  Funcao calcula_potencia(base,expoente: inteiro): real
9  Var
10    resultado: real
11 Inicio
12    resultado ← (base ^ expoente)
13    retorne resultado
14 FimFuncao
15
16 // Início do programa principal
17 Inicio
18    // Entrada de dados
19    Escreva ("Digite o valor da base: ")
20    Leia (valor_base)
21    Escreva ("Digite o valor do expoente: ")
22    Leia (valor_expoente)
23   // Chamada da função
24    valor_resultado ← calcula_potencia(valor_base,valor_expoente)
25    // Saída dos dados
26    Escreva ("O valor do resultado da operação é: ", valor_resultado)
27 FimAlgoritmo
```

Local onde a função está definida no algoritmo

No exemplo acima, definimos uma função de nome calcula_potencia, que possui dois parâmetros (base e expoente) (linha 8). Na linha 12, o cálculo da potenciação é executado e o resultado é armazenado na variável chamada resultado, sendo seu valor retornado na última linha da função (linha 13). Percebam, por meio da indentação, que as instruções contidas nas linhas 12 e 13 fazem parte do que chamamos corpo da função. Entre as linhas18 e 21, que fazem

parte do programa principal, é solicitado ao usuário que digite os valores da base e do expoente. Na invocação da função (linha 23), são passados os valores da base e do expoente como argumentos, sendo o valor retornado pela função armazenado na variável valor_resultado. O resultado da operação é mostrado na linha 26. Os valores passados como argumentos na chamada da função entram na função na forma de parâmetros, que foram declarados por ocasião da sua definição.

É importante lembrar que no programa mostrado como exemplo, o trecho de código pertencente à função somente é executado quando a função for chamada em qualquer parte do programa principal. Lembrando, ainda, que tal função pode ser chamada quantas vezes forem necessárias ao longo do programa. Quando a função é chamada, acontece um desvio de fluxo de execução, onde o interpretador executa o trecho de código da função e, somente após isso, continua seu fluxo normal (sequencial).

As variáveis base, expoente e resultado são denominadas **variáveis locais**, pois elas são válidas somente no escopo da função, ou seja, elas somente terão validade enquanto a função está sendo executada. Essas variáveis não podem ser referenciadas fora do contexto da função, pois elas serão destruídas após a execução da função.

Já as variáveis valor_base, valor_expoente e valor_resultado, são chamadas **variáveis globais**, pois elas terão validade no escopo do programa principal, inclusive, dentro da função. Uma função pode fazer uso de uma variável global.

8.3. Definição de Procedimentos

Procedimentos são sub-rotinas que executam instruções sem retorno de qualquer valor ao programa que o invocou.

Um procedimento é um bloco de programa contendo início e fim e será identificado por um nome, por meio do qual será referenciado em qualquer parte do programa principal ou do programa chamador da sub-rotina [5]. Quando uma sub-rotina é chamada por um programa, ela é executada e ao seu término o controle de processamento retorna automaticamente para a primeira linha de instrução após a linha que efetuou a chamada da sub-rotina.

Sintaxe de definição de procedimento em pseudocódigo:

```
Procedimento <nome>(<parâmetros>: <tipo de dado dos parâmetros>)
Var
   <variáveis locais do procedimento>: <tipos das variáveis locais>
Início
  <corpo do procedimento>
FimProcedimento
```

Exemplo de algoritmo que implementa um procedimento em pseudocódigo:

```
1  Algoritmo "08_02 Exemplo de Procedimento"
2  // Algoritmo que demonstra o uso de procedimento
3  // Exibe um cabeçalho padrão antes do início do programa
4  Var
5        nomeUsuario: caractere
6
7  // Definição do procedimento mostra cabeçalho padrão
8  Procedimento mostraMensagem(mensagem: caractere)
9  Inicio
10    escreval("==================================")
11    escreval("= ", mensagem)
12    escreval("==================================")
13 FimProcedimento
14
15 // Início do programa principal
16 Inicio
17    mostraMensagem("Programa teste v1.0")          // Chamada do procedimento
18
19    Escreva("Digite seu nome:")
20    Leia(nomeUsuario)
21    mostraMensagem ("Olá, " , nomeUsuario, "! Seja bem vindo ao nosso sistema!")
22    mostraMensagem("Fim do programa")          // Chamada do procedimento
23 FimAlgoritmo
```

Local onde o procedimento está definido no algoritmo

Entre as linhas 7 e 15 foi definido o procedimento que formata uma mensagem entre linhas duplas. O parâmetro do procedimento é uma mensagem, do tipo caractere.

Na linha 17, iniciam-se as instruções do programa principal, solicitando o usuário que informe seu nome (linhas 21 e 22).

Nas linhas 23 e 24 são realizadas duas chamadas ao procedimento, a primeira passando como argumento a mensagem inicial para o usuário, e a segunda uma mensagem de "Fim de Programa".

A Figura 35 apresenta o fluxo da chamada, a partir do código principal, de uma função e de um procedimento

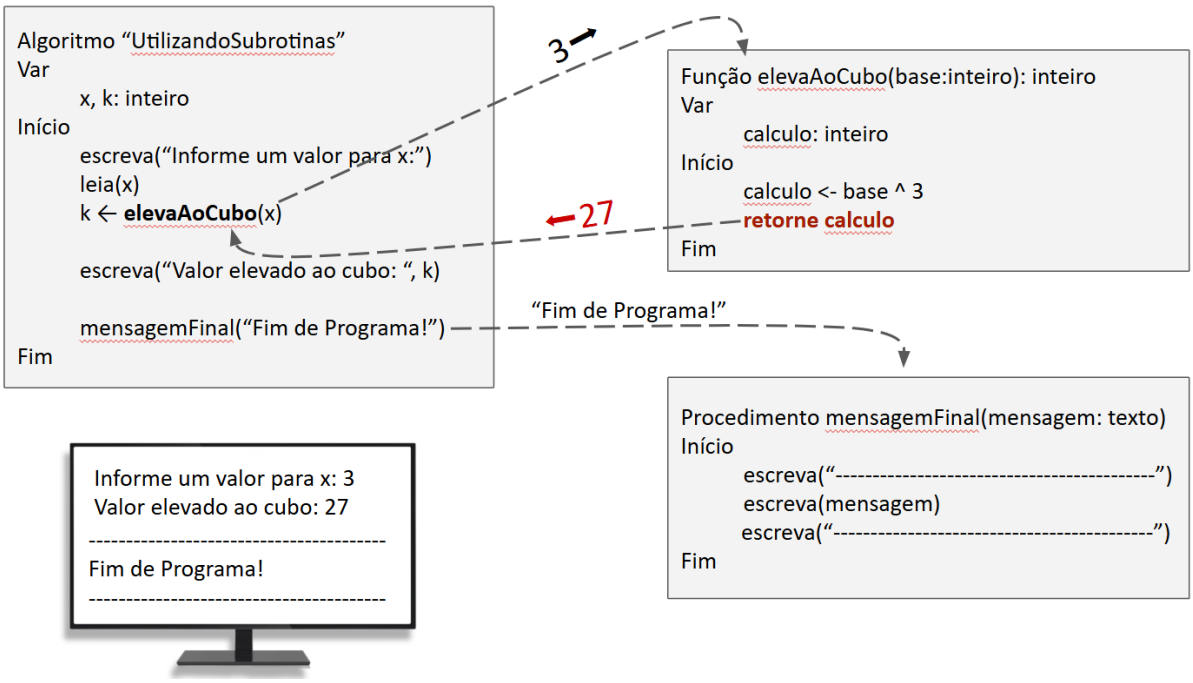


Figura 35: detalhamento do fluxo de função e de procedimento

8.4. Aplicação de Funções e Procedimentos

A utilização de sub-rotinas se constitui numa maneira simples de realizar a modularização do programa, que consiste em dividir um problema grande e complexo em problemas menores e mais simples (sub-rotinas), facilitando seu desenvolvimento, entendimento e manutenção. Os trechos de código menores e mais simples, quando integrados, possibilitam a resolução do problema como um todo. Portanto, a modularização se baseia no princípio de "dividir para conquistar"[6]. Um exemplo de utilização de sub-rotinas é um programa que confecciona a folha de pagamento dos funcionários de uma empresa, onde tal programa principal (grande e complexo) é dividido nas seguintes sub-rotinas:

Obter os dados dos funcionários.
Calcular INSS.
Calcular FGTS.
Calcular imposto de renda.
Calcular férias.
Calcular salário líquido.
Imprimir os contracheques.

Benefícios da Modularização de um Programa

Um módulo é um grupo de comandos ou instruções, que constitui um trecho de algoritmo, com uma função bem definida e o mais independente possível em relação ao resto do algoritmo e que pode ser chamado de sub-rotina.

A modularização de um programa propicia os seguintes benefícios [6]:

- Redução da complexidade (maior simplicidade), já que os vários módulos tendem a ser mais simples.
- Melhor entendimento (maior legibilidade), pois a modularização torna o programa mais fácil de entender.

- Maior manutenibilidade (facilidade de manutenção).

Aumento da possibilidade de reutilização, permitindo que uma determinada sub-rotina de um programa seja reaproveitada em outro programa.

Maior facilidade de desenvolvimento, pois é mais fácil pensar na solução de problemas menores e mais simples.

Bibliotecas de Funções e Modularização

No desenvolvimento de software, bibliotecas são conjuntos de recursos pré-codificados — como funções, classes e rotinas — organizados em módulos que podem ser integrados a novos projetos. O principal objetivo é a produtividade: ao invés de reescrever códigos do zero, o programador utiliza soluções já testadas e otimizadas.

Essas bibliotecas são geralmente categorizadas por domínio de aplicação, abrangendo áreas como cálculos matemáticos, processamento estatístico, renderização gráfica, automação, Machine Learning e Inteligência Artificial.

Quanto à origem, elas se dividem em duas categorias principais:

- Bibliotecas Nativas: Conjunto de funções internas que já vêm instaladas com a linguagem de programação, bastando apenas importá-las.
- Bibliotecas de Terceiros: Pacotes desenvolvidos por empresas ou pela comunidade *open source*.

LISTA DE EXERCÍCIOS - CAPÍTULO 8

Todos os exercícios deste capítulo devem ser desenvolvidos em Pseudocódigo. O desenvolvimento e a execução do código poderão ser feitos no software Visualg.

8.1 Escreva um programa que receba como entrada um número inteiro e calcula, por meio de uma função, o módulo (valor absoluto) desse número.

8.2 Desenvolva um programa que realiza a operação de divisão. O programa deve receber como entrada o valor do dividendo e do divisor e, por meio de uma sub-rotina, calcula e imprime o resultado da operação.

8.3 Escreva um programa que receba como entrada o valor de um ângulo em graus e apresenta seu valor em radianos, fazendo uso de uma função para o cálculo.

*A fórmula para conversão de graus para radianos é: valor em radianos = valor em graus * pi / 180. Considere o valor de pi como sendo 3.1416.*

8.4 Elabore um algoritmo que calcule e apresente o valor de um determinado termo da sequência de Fibonacci. O programa lê um número i, inserido pelo usuário, e imprime o i-ésimo termo da sequência. O programa deve contemplar uma função específica que receba o número i e retorne o valor do termo correspondente para o programa principal.

Dado: A sequência de Fibonacci é a seguinte: 1 1 2 3 5 8 13... inicia-se em 1, repete o 1 no segundo termo e, a partir do terceiro termo, o valor de qualquer termo é obtido pela soma dos dois termos anteriores. Ou seja, n = (n-1) + (n-2).

8.5 Faça um programa que leia o valor do salário atual de um militar e imprima o valor do salário reajustado. O programa principal deve solicitar ao usuário para digitar o salário do militar e chamar a função de nome Calcula_Aumento, receber o retorno desta função, imprimir o novo valor do salário e perguntar se o usuário deseja digitar outro salário ou não. Se o usuário responder que sim, o ciclo deve se repetir até que o usuário responda que não. Se o usuário digitar um valor diferente de “sim” ou “não”, o programa deverá apresentar uma mensagem de entrada inválida e solicitar uma nova entrada, até que o usuário digite uma entrada válida (sim ou não).

A função Calcula_Aumento deve seguir a seguinte regra para calcular o novo salário: se o salário atual for menor ou igual a R$ 1.000,00, deverá aplicar um aumento de 10%; se for maior que R$ 1.000,00 e menor ou igual a R$ 2.000,00, o aumento será de 8%; já se for maior que R$ 2.000,00, o aumento deverá ser de 5%. Essa função deverá calcular o novo salário e retornar o valor para o programa principal.

8.6 Modifique o algoritmo do exercício 6.11 da lista de exercícios referente ao capítulo 6 (Condicionais) da apostila de Algoritmos, de modo que o cálculo do valor de delta, o cálculo do valor das raízes (quando houverem) e a impressão dos resultados, sejam realizadas por meio de sub-rotinas, que serão chamadas pelo programa principal.

8.7 Escrever um programa que receba como entrada dois números inteiros e apresente os seguintes resultados: maior valor digitado (obtido por uma função chamada Calcula_Maior); resultado da multiplicação dos dois números digitados (obtido por uma função chamada Multiplica_Numeros); se cada número digitado é positivo ou negativo (obtido pela função Verifica_Sinal).

8.8 Desenvolva um algoritmo que receba como entrada um caractere e, por meio de uma sub-rotina, verifica e imprime a informação que o caractere digitado trata-se de uma vogal ou consoante. Deve-se partir do princípio que o usuário digitará uma letra e não um número ou qualquer outro caractere especial.

8.9 Escreva um programa que recebe como entrada dois números inteiros positivos e, por meio de uma função, calcula a soma dos números inteiros existentes entre eles (excluindo eles próprios), retornando tal valor para o programa principal. O programa deve validar a entrada, ou seja, ficar apresentando a mensagem de entrada inválida e solicitando um valor correto, até que a entrada seja válida. O programa deve tratar os casos em que não existem números entre os valores digitados, apresentando tal mensagem na tela.

8.10 Desenvolva um algoritmo que, por meio de sub-rotinas, realiza e apresenta o somatório dos N primeiros números primos, sendo que o valor de N é digitado pelo usuário. O programa deverá possuir uma função e um procedimento: a função verifica se determinado número é primo; já o procedimento, calcula o somatório e imprime o resultado. O programa principal deverá receber o valor de "N" digitado pelo usuário e invocar o procedimento que realiza e mostra o somatório dos números primos, que por sua vez chama a função que verifica se determinado número é primo. O programa não deverá permitir que o usuário digite um valor de "N" menor do que 1, mostrando repetidamente a mensagem de valor inválido, até que o usuário digite um número inteiro positivo.

Obs.: número primo: números inteiros positivos, diferentes de 1, que possuem apenas dois divisores: o 1 (um) e ele mesmo.

8.11 Utilizando os conceitos de procedimentos e funções, desenvolva um algoritmo que calcule, a partir de 2 números inteiros positivos P e M, a combinação de M elementos tomados P a P, conforme a expressão: C = M!/(P!(M – P)!)). Os valores dos diversos fatoriais existentes na expressão deverão ser obtidos por meio de uma função para isso destinada.

8.12 Elabore um algoritmo que possua uma função que lê um número positivo do teclado. Caso o usuário não forneça um número inteiro positivo, a função repete a leitura até que um número positivo seja lido. O programa principal deve chamar a função que lê um número do teclado 3 vezes, para a leitura de 3 números distintos. No final, o programa deverá imprimir os números digitados na ordem decrescente.

9. Vetores e Matrizes

- *Estruturas de dados é uma das alternativas para armazenar e organizar informações.*
- *Um **vetor** é uma variável composta que armazena uma sequência de dados do mesmo tipo, sendo, portanto, homogênea.*
- *Uma **matriz** é uma variável composta, homogênea e mutável, possuindo duas dimensões (representando linhas e colunas)*

Os dados que são utilizados por um programa de computador necessitam estar armazenados de maneira organizada, de modo a possibilitar e facilitar, o acesso a eles durante o processamento, bem como as ações de inclusão, modificação e exclusão. Para esse fim, são criadas estruturas, com características próprias, onde são armazenados os dados destinados a algum tipo de processamento.

Estrutura de dados é um modo de armazenar e organizar dados com o objetivo de facilitar acesso e modificações [7]. Nenhuma estrutura de dados única funciona bem para todas as finalidades e, por isso, é importante conhecer os pontos fortes e as limitações de várias delas.

9.1 Vetores

Vetor ou *array* é uma estrutura de dados que compreende uma sequência ordenada e indexada de elementos mutáveis. Em pseudocódigo, assim como ocorre em algumas linguagens de programação, um vetor se constitui em uma variável composta, onde é armazenado uma sequência de dados do mesmo tipo e, por isso, essas variáveis são classificadas como homogêneas.

A sintaxe de declaração de uma variável do tipo vetor em pseudocódigo é a seguinte:

Var <*nome*>: **vetor** [<*início..fim*>] **de** <*tipo de dado*>

Exemplo:

Var *notas*: **vetor** [<*1..4*>] **de** *real*

Esta declaração cria um vetor, nomeado notas, de quatro posições, cada uma indicada por um índice numérico, cada posição tendo a capacidade de armazenar uma variável do tipo real.

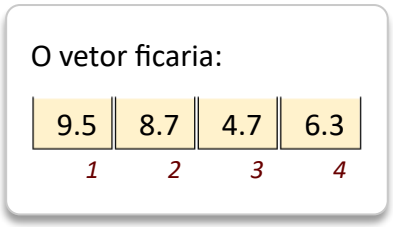
real	real	real	real	←Valores
1	*2*	*3*	*4*	*← Índice*

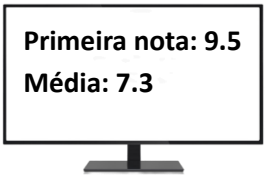
Para atribuir valores a um elemento específico de um vetor, utilizando-se do índice, deve se seguir a seguinte sintaxe:

<*nome*>[*índice*] ← <*valor*>

O algoritmo 09_01 detalha o uso de vetores.

```
1  Algoritmo "09_01 Uso de Vetor"
2  //Exemplo de preenchimento de vetor e acesso por índices
3  Var
4    // Declaração de um vetor de 4 posições de números reais
5    notas : vetor [1..4] de Real
6    media : Real
7  Inicio
8    Escreval("--- Atribuindo valores ao vetor ---")
9    // A ordem de atribuição não importa, pois usamos o índice
10   notas[1] ← 9.5
11   notas[4] ← 6.3
12   notas[3] ← 4.7
13   notas[2] ← 8.7
14
15   // Acessando um valor específico
16   Escreval("Primeira nota: ", notas[1])
17
18   // Soma-se o conteúdo de cada "gaveta" e divide pelo total
19   media <- (notas[1] + notas[2] + notas[3] + notas[4]) / 4
20   Escreval("Média calculada: ", media)
21
22 FimAlgoritmo
```





9.2 Matrizes

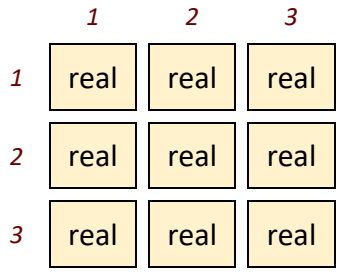
Em pseudocódigo, uma matriz se constitui uma variável composta, homogênea e mutável de duas dimensões: uma representando as linhas e a outra representando as colunas.

A sintaxe de declaração de uma variável do tipo matriz em pseudocódigo é a seguinte:

Var <*nome*>: **matriz** [<*início..fim*>] **de** <*tipo de dado*>

Exemplo:

Var *notas*: **matriz** [*<1..3> : <1..3>*] **de** *real*



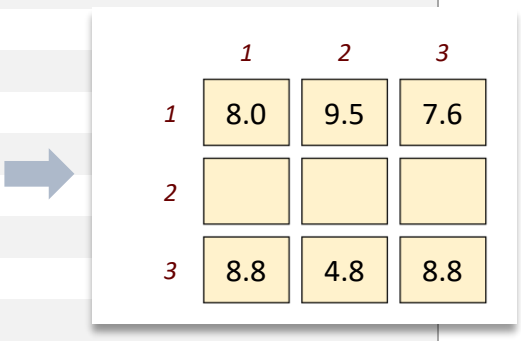
No exemplo acima, foi declarada uma variável do tipo matriz cuja dimensão é três linhas e três colunas, sendo que seus elementos são dados do tipo real. Cada um dos 9 (nove) elementos da matriz podem ser acessado por um par de índices, o primeiro indicando a linha e o segundo a coluna da posição (figura à direita).

Através do índice é possível a atribuição de valores a um elemento no vetor, conforme a sintaxe:

<*nome*>[*linha, coluna*] ← <*valor*>

O algoritmo 09_02 detalha o uso de matriz:

```
1  Algoritmo "09_02 Uso de Matriz"
2  // Descrição: Exemplo de preenchimento e leitura de Matriz (Vetor Bidimensional)
3  Var
4    // Declaração de uma Matriz 3x3 (3 Linhas, 3 Colunas)
5    notas : vetor [1..3, 1..3] de Real
6    mediaAluno3 : Real
7  Inicio
8    // Preenchendo linhas da matriz
9    Escreval("# Atribuindo valores à Matriz")
10   notas[1,1] <- 8.0
11   notas[1,2] <- 9.5
12   notas[1,3] <- 7.6
13   notas[3,1] <- 8.8
14   notas[3,2] <- 4.8
15   notas[3,3] <- 8.8
16
17   // Acessando valores randomicamente
18   Escreval("--- Leitura dos Dados ---")
19   Escreval("Aluno 1 / nota 2: ", notas[1,2])
20   Escreval("Aluno 3 / nota 3: ", notas[3,3])
21
22   // Cálculo da Média do Aluno 3 (Linha 3 completa)
23   mediaAluno3 <- (notas[3,1] + notas[3,2] + notas[3,3]) / 3
24   Escreval("Média aluno 3: ", mediaAluno3:4:2)
25
26 FimAlgoritmo
```




```
# Atribuindo valores à Matriz
Aluno 1 / nota 2:  9.5
Aluno 3 / nota 3:  8.8
Média aluno 3:  7.47
```

Observação importante! Diferente de linguagens modernas, o Visualg não possui funções internas para manipular vetores e matrizes (como ordenar, redimensionar ou buscar automaticamente).

LISTA DE EXERCÍCIOS - CAPÍTULO 9

Todos os exercícios deste capítulo devem ser desenvolvidos em Pseudocódigo. O desenvolvimento e a execução do código poderão ser feitos no software Visualg.

9.1 Implemente um algoritmo que receba números inteiros digitados pelo usuário e os armazene em vetores paralelos, na ordem em que foram digitados. Inicialmente, o usuário deverá informar a quantidade de elementos da sequência, em valor limitado a 30. Ao final, o programa deverá imprimir a sequência, em ordem decrescente, sendo um valor em cada linha.

9.2 Implemente um programa que receba, via teclado, os dados de até alunos (número, nome, pelotão e telefone) e armazene-os em um vetor. Ao final do cadastro dos dados de cada aluno, o programa deve perguntar se deseja cadastrar outro aluno ou não. Ao responder sim, o programa deve permitir o acréscimo de mais um aluno. Se o usuário responder não, o programa deverá exibir os dados dos alunos cadastrados, conforme discriminado no exemplo abaixo:

Ao lado, exemplo de saída do programa, após a realização do cadastro dos dados de alunos.

```
--- Cadastro do Aluno  1
Digite o Número do aluno: 4550
Digite o Nome do aluno: João Pedro
Digite o Pelotão: 1º
Digite o Telefone: 21 988556677
Deseja cadastrar outro aluno? (S/N): S
------------------------------------------------
--- Cadastro do Aluno  2
Digite o Número do aluno: 3339
Digite o Nome do aluno: Ana Lúcia
Digite o Pelotão: 3 ]
Digite o Telefone: 11 234567666
------------------------------------------------
Deseja cadastrar outro aluno? (S/N): N
========================================
              RELATÓRIO DE ALUNOS
========================================
NUM. | NOME         | PEL   | TELEFONE
----------------------------------------------------------------
 4550 | João Pedro   | 1º    | 21 988556677
 3339 | Ana Lúcia    | 3 º   | 11 234567666
 9999 | Carlos       | 8º    | 11 23456789
```

9.3 Escreva um algoritmo que crie e imprime uma matriz de inteiros, de 4 x 4 elementos. Permitir a entrada dos valores numéricos desta matriz e exibir a impressão da matriz de duas maneiras: 1ª) de forma contínua, na ordem em que foi informada; 2ª) no formato organizado (modelo convencional de matrizes).

9.4 Faça um pseudocódigo que recebe a idade de uma população (n=12), e calcula e apresenta o valor da média e do desvio padrão dessa população de elementos.

A fórmula para o cálculo do desvio padrão de uma população é:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}}$$

onde,

x_i = elemento i da população

μ = media

N = quantidade de elementos

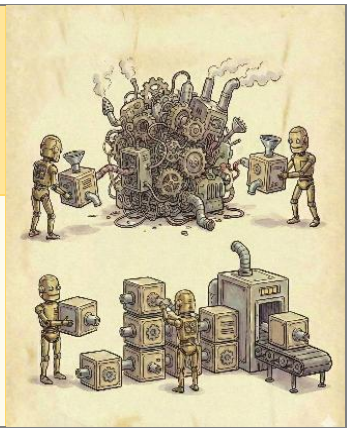
9.5 Utilizando os conceitos de procedimentos, funções e vetores, construa um algoritmo que determine, para uma série de 10 valores inteiros, digitados pelo usuário, quantos

são maiores que a média do conjunto. O programa deverá fornecer como saída: a série digitada, a média da série e a quantidade de valores acima da média.

9.6 Na álgebra linear, a multiplicação de um escalar k por uma matriz A resulta em uma nova matriz B, onde cada elemento $b_{ij} = k \times a_{ij}$. Desenvolva um pseudocódigo que:

- Leia uma matriz A[3][3].
- Leia um valor multiplicador k.
- Multiplique a matriz A pelo valor de k e armazene o resultado na matriz B.
- Apresente a matriz original e a matriz multiplicada.

10. Desenvolvimento de Algoritmos



A programação estruturada e, em particular, o método Top-Down (de cima para baixo), são metodologias essenciais no desenvolvimento de software. Este método envolve a identificação das tarefas principais, a visualização do programa principal que controla as sub-rotinas (módulos), e o detalhamento progressivo de cada sub-rotina através de refinamentos sucessivos.

10.1. Construção de Algoritmos

Com os conhecimentos adquiridos nas seções anteriores, é possível a criação de algoritmos para resolução de um grande número de problemas. Esses algoritmos poderão ser codificados em uma linguagem de programação, tornando-se, assim, um programa de computador.

Porém, no desenvolvimento de programas complexos é fundamental que as tarefas a serem executas sigam uma metodologia de programação.

O processo de programar um computador torna-se bastante simples quando aplicado o método de utilização de sub-rotinas (módulos de programas). Porém, a utilização dessas sub-rotinas deverá ser feita de forma metódica[5].

Para os autores, um método bastante adequado para a programação de um computador é a utilização do conceito de programação estruturada, pois a maior parte das linguagens de programação utilizadas atualmente também são.

Os autores mencionam, ainda, que o método mais adequado para a programação estruturada é o Top-Down (de cima para baixo), que é caracterizado basicamente pelos seguintes passos:

- Antes de iniciar a construção do programa, determinar as tarefas principais que o programa deverá executar.
- Conhecidas as tarefas principais, é possível visualizar como será o programa principal, que irá controlar todas as demais tarefas, que estarão distribuídas em sub-rotinas (módulos).
- Definido o programa principal, realizar o detalhamento de cada sub-rotina.

Com a realização dos passos do método, é definido um algoritmo para cada sub-rotina em separado, sendo que as sub-rotinas complexas devem ser decompostas em outras sub-rotinas, por meio de refinamentos sucessivos. Ao término das etapas, é possível obter uma visão do que é executado em cada módulo.

A utilização do método *Top-Down* (“de cima para baixo”) permite que seja efetuado cada módulo do programa em separado. Desta forma, cada um pode ser testado separadamente, garantindo que o programa completo esteja sem erro ao seu término.

O método Top-Down faz com que o programa tenha uma estrutura semelhante a um organograma, conforme mostrado na Figura 36.

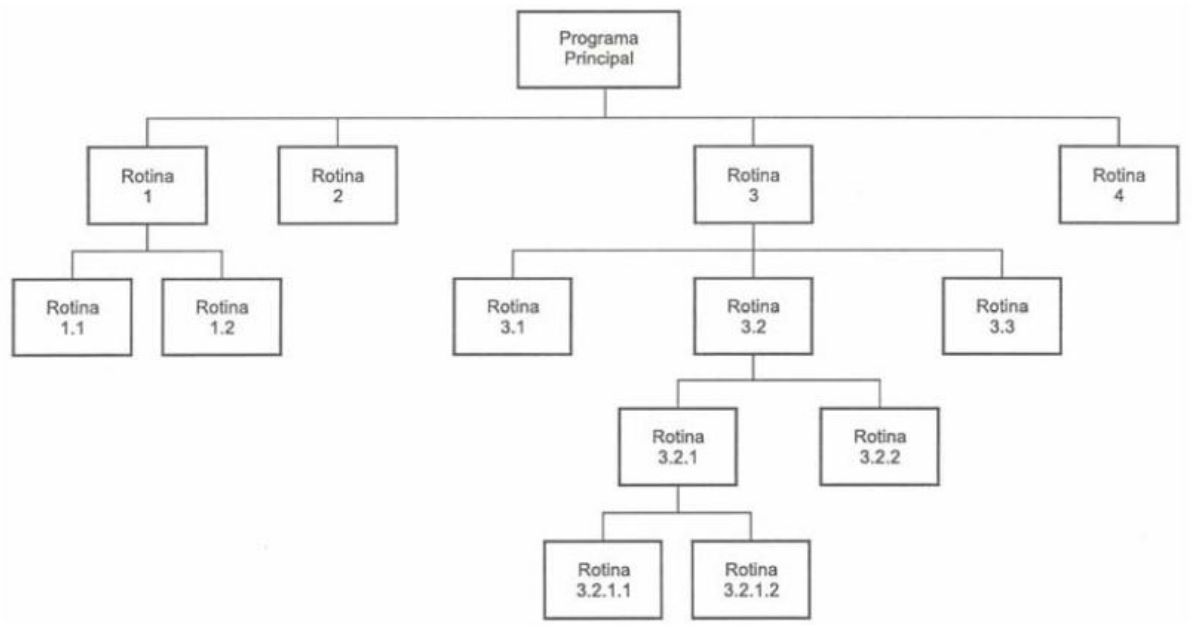


Figura 36: programa utilizando o método Top-Down

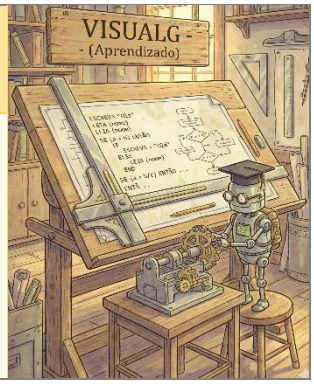
A partir deste momento, de posse de todos os conceitos abordados na presente apostila, o aluno pode recuperar os exercícios propostos e convertê-los para Python. Lembrando que algoritmos não se aprende lendo ou copiando os já realizados, mas, sim, elaborando os próprios algoritmos.

LISTA DE EXERCÍCIOS - CAPÍTULO 10

10.1 Faça um diagrama que retrate um sistema de controle de alunos da EsPCEx, abordando áreas como controle de notas e fatos observados.

10.2 Escolha um sistema qualquer, computacional ou não, e faça um diagrama que represente sua modularização.

Anexo A - VisualG



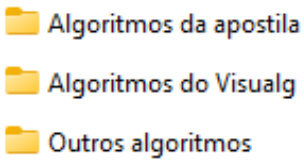
- *Tutorial de instalação do interpretador específico para pseudocódigo.*
- *Manual de uso: áreas da janela do programa, funcionalidades de menus e ícones, depuração (debugging) e breakpoints.*
- *Conjunto de comandos internos do pseudocódigo.*
- *Referências externas.*

Um algoritmo em pseudocódigo é uma forma de resolução de um problema, um esboço menos refinado do que o que será efetivamente implementado em alguma linguagem de programação. Não foi desenvolvido com o objetivo de ser efetivamente executado, entretanto, existem ferramentas que podem testar a sintaxe e executar tal tipo de código. Em nossa Disciplina utilizamos uma sintaxe de pseudocódigo compatível com o software Visualg, que é uma possibilidade do aluno testar seu código. O software Visualg foi desenvolvido para Windows. Existem alternativas como o site https://visualg.com.br/ ou o app Pseudocode para Android, mas eles não apresentam os mesmos recursos e compatibilidade com a sintaxe adotada na disciplina. Importante ressaltar que, para não prejudicar a dinâmica das aulas, os professores dão suporte ao uso de outros softwares.

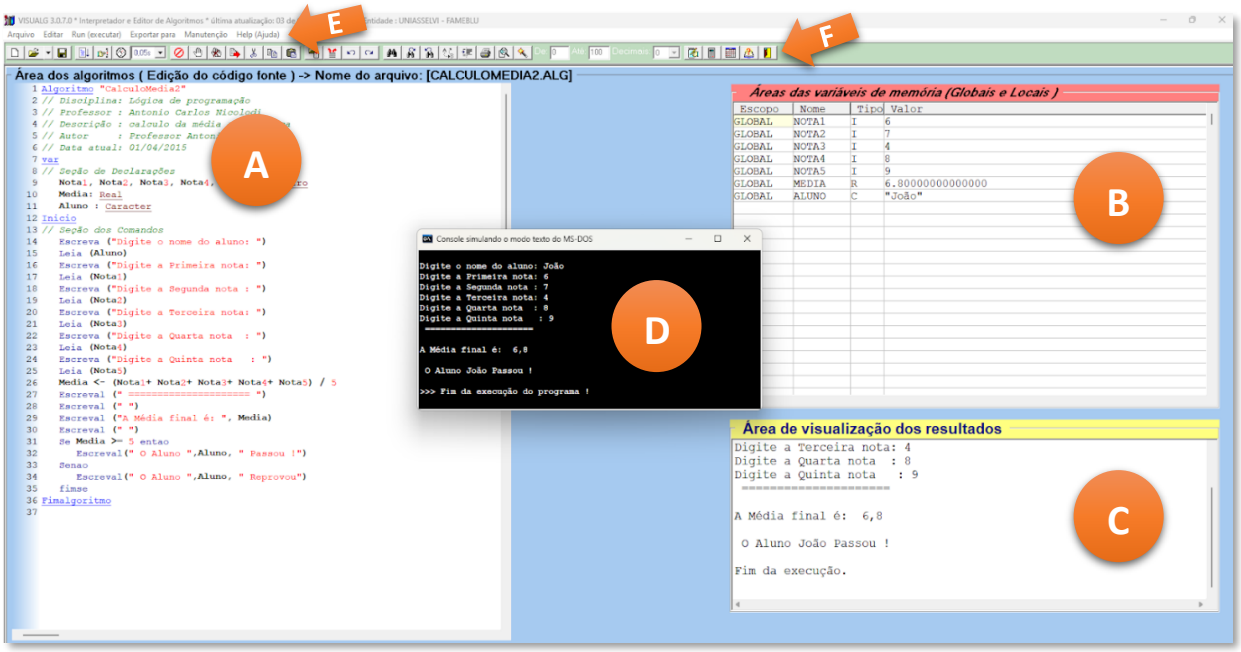
Instalação de Visualg para Windows

Baixe o arquivo .zip disponível no AVA ou no site de apoio da disciplina. Não é necessário instalação, bastando descompactar o arquivo para o local desejado. Depois procure o arquivo **visualg30.exe** dentro desta pasta e execute. É interessante criar um ícone relativo a este arquivo executável na sua área de trabalho ou barra de tarefas, para facilitar o acesso futuro.

Na pasta do Visualg (C:\visualg3.0.7\visualg3.0.7\), existe uma subpasta “Algoritmos”. Nesta pasta há as seguintes subpastas:



Visualg para Windows / aspecto geral e utilização

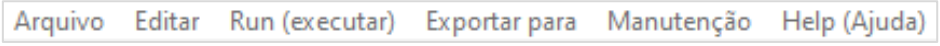


[A] Editor do pseudocódigo. Possui um recurso fundamental conhecido como "Syntax Highlighting" (Destaque de Sintaxe). Sua função não é apenas estética, mas semântica: ele utiliza cores e formatações distintas para informar visualmente ao programador a função de cada termo dentro do algoritmo. Isso previne erros de digitação antes mesmo da execução. No padrão do Visualg, as palavras reservadas (comandos que pertencem à linguagem, como Algoritmo, Var, Inicio, Se, FimSe) são exibidas automaticamente em azul e negrito. Se um aluno digitar "escreva" e a palavra não ficar azul, ele sabe imediatamente que cometeu um erro de grafia. Já os comentários (iniciados por //) são exibidos em verde, indicando que aquele trecho será ignorado pelo processador. Os textos aparecem em vermelho, facilitando a distinção entre o que é código e o que é dado.

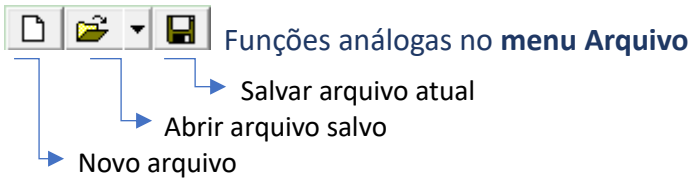
[B] A área de variáveis funciona como uma "janela transparente" para a memória RAM do computador durante a execução do algoritmo. Ela é uma tabela dinâmica que lista, em tempo real, todas as variáveis que foram declaradas no bloco Var. Esta área é dividida em colunas essenciais: o escopo (se a variável é global ou local), o nome da variável, o tipo de dado (Inteiro, Real, Caractere, Logico) e, o mais importante, o valor atual. Durante a execução passo a passo, essa área é vital para o ensino de lógica, pois permite que o aluno veja o momento exato em que uma variável muda de valor (por exemplo, x passando de 0 para 1 em um contador), concretizando o conceito abstrato de "atribuição".

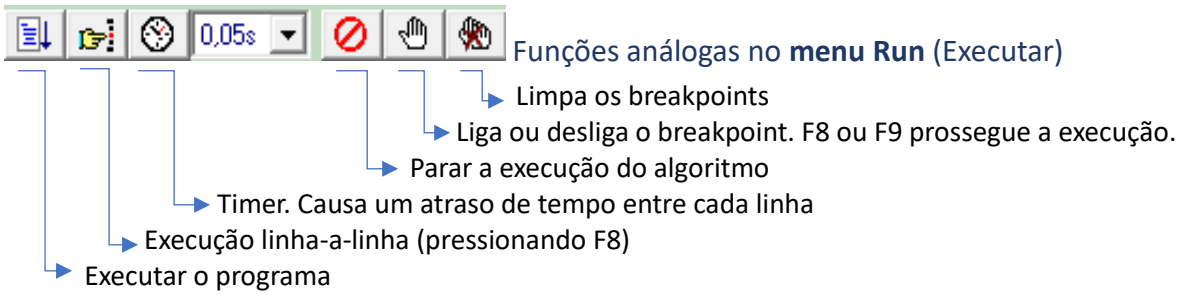
[C][D] Área de visualização dos resultados. Esta área simula o console ou terminal do sistema operacional. É a interface de entrada e saída (I/O) entre o algoritmo e o usuário. Quando o algoritmo encontra um comando Leia, o cursor pisca nesta área aguardando que o usuário digite um valor e pressione *Enter*. E, tudo o que o comando Escreva ou Escreval processa é "impresso" nesta tela. Esta mesma informação estará em uma janela que sobrepõe o programa, no momento da execução (pequena tela preta no centro da imagem).

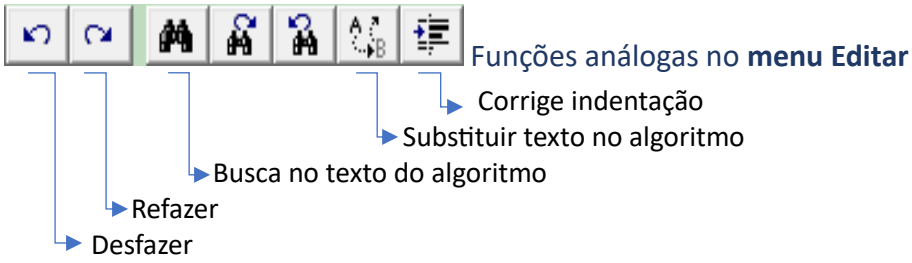
[E] Menus do programa.



[F] Ícones do Programa.







Breakpoint (Ponto de Parada)

O breakpoint é uma ferramenta de controle de fluxo utilizada para interromper intencionalmente a execução automática do programa em uma linha específica.

Marcação: o usuário marca um breakpoint clicando na margem cinza ao lado do número da linha ou pressionando a tecla F5. Uma linha vermelha aparece, sinalizando a marcação.

O que provoca: quando o interpretador do Visualg está executando o código (geralmente com F9), ele verifica antes de cada linha se existe um breakpoint. Se encontrar, ele "congela" o estado do programa imediatamente antes de executar aquela linha.

Dinâmica: ao atingir o breakpoint, o programa não termina; ele entra em modo de pausa. O controle retorna para o programador, que pode então analisar a área de variáveis naquele instante exato para verificar se os dados estão corretos. A partir dali, pode-se optar por continuar linha a linha (F8) ou retomar a velocidade normal (F9) até o próximo ponto de parada.

O Processo de Debugging (Depuração)

Debugging, ou depuração, é o processo metódico de encontrar e reduzir defeitos (bugs) em um programa. No Visualg, esse processo é facilitado visualmente e simula o que é conhecido academicamente como "Teste de Mesa". O processo se dá através da execução controlada, geralmente utilizando a tecla F8 (Passo a Passo). Ao ativar esse modo, uma barra azul destaca a linha que está sendo executada naquele momento. O programador observa a barra azul avançar, prevê o que deveria acontecer, e então olha para a Área de Variáveis para confirmar se o

resultado real bate com a previsão. Se a variável receber um valor inesperado ou se a barra azul não tomar um caminho previsto dentro do código, o programador localizou o erro lógico.

Como Sair de um Looping Infinito

Um looping infinito ocorre quando a condição de parada de uma estrutura de repetição (como Enquanto ou Repita) nunca é satisfeita, fazendo o programa rodar indefinidamente e, muitas vezes, travar a interface.

No Visualg, para interromper esse ciclo forçosamente, utiliza-se o comando de "Parar". Isso pode ser feito através do menu Run(rodar) > Parar, ou pelo atalho de teclado Ctrl + F2. Essa ação mata o processo de execução imediatamente, liberando o editor para que o usuário possa corrigir a lógica defeituosa no código.

Erro em tempo de execução

Erros em tempo de execução (Runtime Errors) são falhas que ocorrem enquanto o programa está sendo executado, e não durante a escrita ou compilação do código. Diferente dos erros de sintaxe (quando se escreve um comando errado, como *escrev* em vez de *escreva*, e o Visualg avisa antes de rodar), o erro de tempo de execução acontece quando o comando está escrito corretamente, mas o computador não consegue executá-lo devido a uma operação inválida ou dados incorretos. Aqui estão os casos mais comuns no Visualg e como resolvê-los:

1) Divisão por Zero

Ocorre quando você tenta dividir um número por uma variável cujo valor é 0. Matematicamente, isso não existe, e o programa trava. Exemplo do erro (esquerda) e como tratar o problema (à direita):

```
x <- 10
y <- 0
z <- x / y
```

```
se y <> 0 entao
    z <- x / y
senão
    escreval("Erro: Não é possível dividir por zero.")
fimse
```

De forma análoga, uma raiz quadrada de número negativo seria um erro matemático que causaria um erro em tempo de execução.

2) Incompatibilidade de Tipos (Entrada de Dados)

Ocorre muito no comando leia. O programa espera um número, mas o usuário digita letras ou texto. Exemplo do erro: Você declarou idade : inteiro e, na hora do leia(idade), o usuário digita "vinte" (por extenso) ou "20 anos". O Visualg não consegue converter esse texto para número. Para minimizar o problema emita uma instrução clara, para orientar o usuário (ex: "Digite apenas números"). Há técnicas mais avançadas para tratar esse tipo de erro em linguagens mais modernas, como o uso de try/catch.

3) *Índice de Vetor Fora dos Limites*

Quando há a tentativa de acessar uma posição em vetor fora do limite. Por exemplo, em um vetor de 5 posições, há a tentativa de se acessar a posição 6 ou a posição 0. À direita, um exemplo do erro.

```
var
    nomes: vetor[1..5] de caractere
inicio
    escreva(nomes[6])
    (...)
fimAlgoritmo.
```

4) *Loop Infinito (Travamento)*

Quando há a utilização dos comandos *enquanto* ou *repita*, sem a atualização da variável de controle, o programa nunca sairá do laço e parece ter "travado" ou congelado. Exemplo do erro:

```
contador <- 1
enquanto contador <= 10 faca
    escreva("Olá")
    // Esqueceu de colocar contador <- contador + 1
fimenquanto
```

Como resolver: certifique-se sempre de que, dentro do bloco de repetição, existe um comando que fará a condição se tornar falsa em algum momento, como um incremento do contador, ou a entrada de um valor pelo usuário que será avaliada pela estrutura de repetição).

Observação: há de se distinguir erros de execução com erros de lógica. O primeiro provocará a interrupção prematura do algoritmo, pois o computador não consegue executar determinado comando. No erro de lógica o programa pode até chegar ao fim, apresentar um resultado, mas incorreto. Por exemplo, no cálculo da média de dois alunos, com nota1 + nota2 / 2 (sem parênteses). O computador executa perfeitamente (divide a nota2 por 2 e soma com a nota1), mas o resultado matemático está errado por falta de precedência. A utilização de *breakpoints* e do processo de *debugging* podem ser valiosos aliados para se identificar e resolver erros de lógica em um algoritmo.

Referência das funções internas

Funções matemáticas

Função	Descrição	Exemplo	Resultado
abs(x)	Valor absoluto (módulo).	y <- abs(-5)	y será 5
raizq(x)	Raiz quadrada de x.	y <- raizq(9)	y será 3
pi	Retorna o valor de Pi	Escreva(pi)	3.14159...
quad(x)	Eleva x ao quadrado.	y <- quad(4)	y será 16
exp(b, e)	Exponenciação (base b, expoente e).	y <- exp(2, 3)	y será 8 (2^3)
int(x)	Retorna apenas a parte inteira de um número real (trunca).	y <- int(3.9)	y será 3
log(x)	Logaritmo na base 10.	y <- log(100)	y será 2
sen(x)	Seno (ângulo em graus*).	y <- sen(30)	y será 0.5
cos(x)	Cosseno (ângulo em graus*).	y <- cos(60)	y será 0.5
tan(x)	Tangente (ângulo em graus*).	y <- tan(45)	y será 1.0

Funções de manipulação de texto

Função	Descrição	Exemplo de uso	Resultado
compr(s)	Retorna o tamanho (número de caracteres) da string s.	t <- compr("Brasil")	t será 6
copia(s, p, n)	Retorna uma parte da string s, começando na posição p e pegando n caracteres.	t <- copia("Futebol", 1, 4)	t será "Fute"
maiusc(s)	Converte as letras da string s para maiúsculas.	t <- maiusc("Bom dia")	t será "BOM DIA"
minusc(s)	Converte todas as letras da string s para minúsculas.	t <- minusc("VisualG")	t será "visualg"
pos(sub, s)	Procura a substring sub dentro de s e retorna a posição inicial. Retorna 0 se não encontrar.	p <- pos("a", "Casa")	p será 2
asc(s)	Retorna o código ASCII correspondente ao primeiro caractere da string s.	cod <- asc("A")	cod será 65
carac(n)	Retorna o caractere correspondente ao código ASCII n.	t <- carac(65)	t será "A"
numpcarac(n)	Converte um número n (inteiro ou real) para texto (string).	t <- numpcarac(10.5)	t será "10.5"
caracpnum(s)	Converte a string s para um valor numérico (inteiro ou real). Útil para cálculos.	n <- caracpnum("50") + 10	n será 60

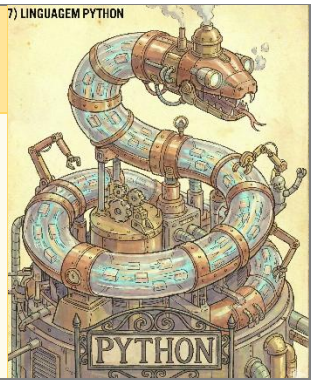
Comandos de Sistema e Controle

Comandos que controlam o ambiente de execução ou o fluxo do console.

Comando	Descrição	Ex. de Uso	Efeito Prático
limpatela	Apaga todo o texto exibido no console (tela de saída).	limpatela	A tela fica em branco, pronta para novas mensagens.
timer <t>	Interrompe a execução (pausa) por <t> milissegundos.	timer 2000	O programa "congela" por 2 segundos antes de continuar.
pausa	Antigo comando de espera. Sem parâmetros, aguarda um "Enter".	pausa	Aguarda o usuário teclar Enter para continuar.
aleatorio on	Ativa o modo de sorteio.	aleatorio on	O computador simula os valores de entrada
aleatorio off	Desativa o modo de sorteio, voltando a leitura para o teclado.	aleatorio off	O comando leia volta a aguardar a digitação do usuário.
aleatorio x, y	Define o intervalo (mín. e máx.) para a geração de números aleatórios.	aleatorio 1, 60	Os números sorteados estarão entre 1 e 60.
cronometro on	Inicia a contagem de tempo de execução	cronometro on	Começa a medir quanto tempo o algoritmo leva para rodar.
cronometro off	Para a contagem de tempo.	cronometro off	Finaliza a medição de tempo.

Anexo B – Python

- *Tutorial de instalação e uso de um interpretador específico para a Linguagem Python.*
- *Demonstra a conversão de pseudocódigo para Python.*
- *Apresenta as funções internas da linguagem.*
- *Apresenta funções específicas para tratamento de listas.*



Introdução

Ao longo desta disciplina, focamos na construção do raciocínio lógico e na estruturação de algoritmos em pseudocódigo (Portugol), utilizando o Visualg. O objetivo foi permitir que o aluno se preocupasse com a solução do problema, sem se distrair com regras complexas de sintaxe.

Dominada a lógica de programação, com das estruturas básicas ensinadas, manipulação de expressões, dentre outros conhecimentos, chegou o momento de aplicar em uma ferramenta de mercado. Este anexo apresenta a transição do Portugol para o Python[11].

Razões para a escolha da linguagem Python:

- É a linguagem utilizada na sequência desta disciplina na AMAN;
- é uma possibilidade de linguagem que poderá ser empregada na atividade integradora, no segundo semestre do ano letivo;
- pela sua eficiência, poder, versatilidade (utilizada para diversas aplicações), legibilidade (fácil compreensão) e fácil execução em qualquer sistema operacional.

A lógica mantém-se a mesma, alterando-se apenas a sintaxe e o idioma. Recomenda-se o uso deste guia como referência de tradução. Compreender a lógica é o passo fundamental para programar; o Python atua apenas como o idioma para a interação com a máquina.

Instalação e uso do interpretador Python

Embora a linguagem Python possa ser utilizada em diversos ambientes de desenvolvimento, esta disciplina adota oficialmente o interpretador nativo. Esta ferramenta já se encontra instalada e configurada em todos os computadores dos laboratórios e será a base para o suporte oferecido durante as aulas.

Existem outros programas como PyCharm, Visual Studio ou Thonny; aplicativos para celular/tablets ou sites da Internet onde é possível escrever e testar programas em Python. Como a linguagem Python é padrão, terá o mesmo funcionamento em qualquer um dos editores. Assim, os alunos podem utilizar outros editores de sua predileção, mas não será possível dar suporte ao uso destes programas durante as aulas.

O instalador pode ser obtido no site oficial python.org/downloads, diretamente no AVA (Ambiente Virtual de Aprendizagem) ou no site de apoio da disciplina. Há versões paras Windows, Linux e Mac.

Download Python

Após instalado execute o programa **IDLE**. A partir daí há dois modos de uso: logo ao abrir se utilizar o Shell, uma tela interativa para testes rápidos (Figura 37) ou, para criar programas ir no menu *File* > *New File*, o que abrirá o editor de texto (Figura 38).

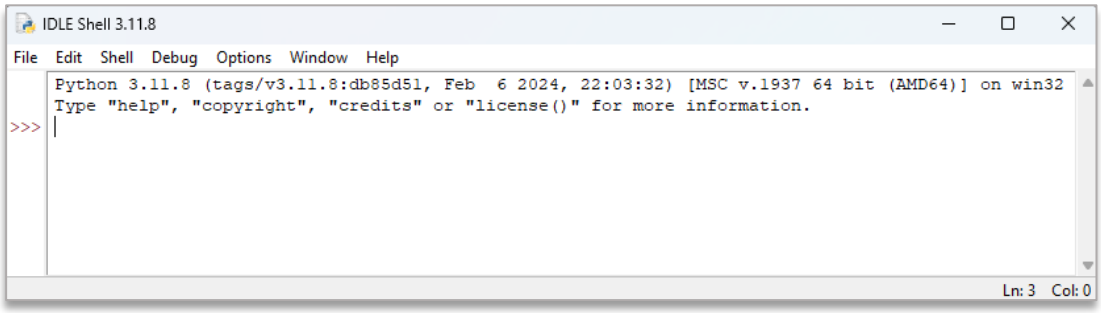


Figura 37: Tela do IDLE

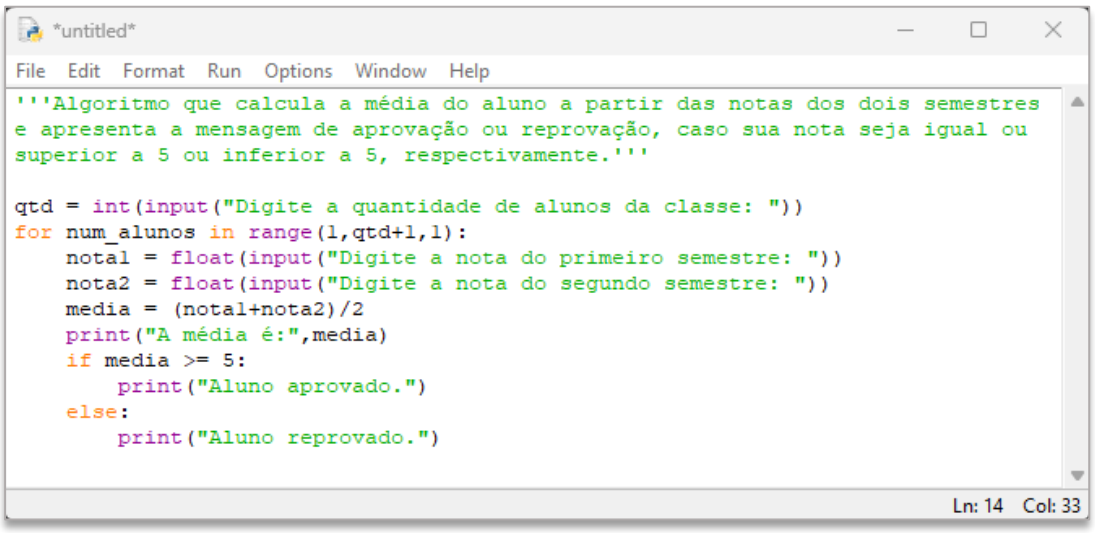
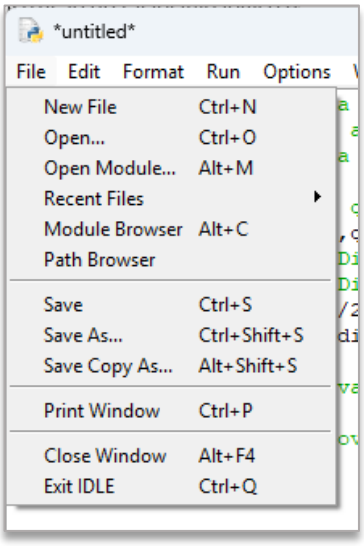


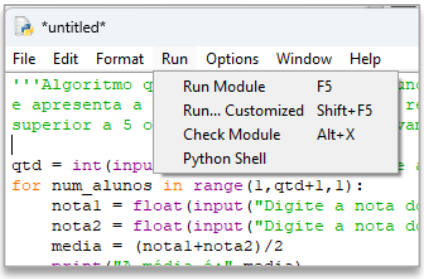
Figura 38: Editor de texto nativo do Python

O editor é minimalista. Possui coloração de sintaxe, que distingue comentários e strings (em verde), palavras reservadas (laranja), funções nativas (roxo) e variáveis/operadores (em preto). Este recurso facilita ao programador observar a estrutura geral do programa e se precaver de erros. Por exemplo, se digitou a função “input” como “iput”, esta não assumirá a cor roxa indicando o erro.

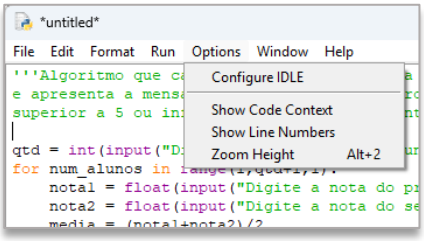
Principais recursos disponíveis nos menus



No menu **File** é possível iniciar um novo arquivo de código (New File), abrir um arquivo salvo anteriormente (Open...), Salvar as alterações realizadas no arquivo atual (Save), Salvar o arquivo com outro nome (Save as...).



No menu **Run** é possível executar o arquivo atual no editor (Run Module). Ao tentar executar um arquivo ainda não salvo, o editor força primeiro a salvar o arquivo, antes da execução.



No menu **Options** é possível configurar uma série de atributos visuais do editor (Configure IDLE). Ali é possível escolher a fonte do editor, o tamanho da fonte, as cores adotadas no texto do editor, dentre outras customizações.

Características gerais:

- **Python diferencia maiúsculas de minúsculas**, seja nos nomes de variáveis, comandos e outros componentes de sua sintaxe. Assim, uma variável nomeada como “Alunos” não pode ser referenciada como “alunos”. Um comando “print” não pode ser escrito como “Print”, pois gerará um erro.
- Ele é **menos “verboso”**: não tem cláusulas como “Algoritmo”, “Var”, “Início”, “Fim”
- **A indentação tem função sintática**, representando blocos de código vinculados a estruturas como condicionais e laços de repetição. O alinhamento visual do código define onde um bloco começa e termina. A não observância da tabulação em um código Python certamente levará a um erro de execução.
- **Tipagem Dinâmica**: no Python não é necessário declarar o tipo da variável (inteiro, real, booleano) explicitamente; a linguagem define isso automaticamente ao se atribuir um valor pela primeira vez.

Nas partes 1 a 6 serão abordadas as estruturas correspondentes entre pseudocódigo e Python.

Parte 1: Variáveis e expressões

VARIÁVEIS

As regras de nomeação de variáveis seguem as mesmas do pseudocódigo. Não podem ser utilizadas palavras reservadas do Python. Estas são palavras que a linguagem já "tomou para si" para definir sua estrutura e sintaxe, portanto, **não podem ser usadas como nomes de variáveis** ou funções. Algumas estão listadas a seguir:

False	in	print	break
True	while	input	with
if	continue	def	try
else	not	return	except
elif	and	import	finally
for	or		

A atribuição de valores em Python utiliza o símbolo "=" (igual), no lugar do "<-" (menor+hífen).

```
<variável> = <valor>
```

Assim como no pseudocódigo, não existem variáveis em Python, mas convencionou-se utilizar variáveis para este propósito, escritas totalmente em letras maiúsculas. Exemplo:

```
PI = 3.14159
```

Os tipos de dados em Python, comparado aos tipos disponíveis em pseudocódigo, estão expressos na Tabela 16.

Tabela 16: Tipos de dados em Python

GRUPO	TIPO DE DADOS Python	Pseudo.	DESCRIÇÃO	EXEMPLOS
NUMÉRICOS	*int*	Inteiro	Números inteiros	5, 75, -30, 100
	float	real	Números reais (ponto flutuante). Um ponto divide a parte inteira da parte fracionária, podendo estar na notação científica.	4.7 88.987 0.456, 3e2
	complex	---	Números complexos (*a* + *bj*) *a* e *b* são float e *j* = raiz quadrada de -1	2.5 + 4.3j
LITERAIS	*str*	caractere	Conjunto de caracteres ou um único caractere escrito entre aspas simples ('...') ou duplas ("...").	"frase" 'palavra' "letra"
BOOLEANOS	*bool*	lógico	Dados com dois valores possíveis: verdadeiro ou falso	*True* *False*

No programa "AnexoB_01", estão vários exemplos de uso de variáveis em Python. Sobre este programa cabem várias observações:

As linhas 1,2,7,13 e 17 são comentários, e não têm qualquer ação, servem somente para documentação do código.

Como já falado, em Python há uma tipagem dinâmica. Não é necessário declarar o tipo da variável (inteiro, real, booleano) explicitamente; a linguagem define isso automaticamente ao se atribuir um valor pela primeira vez, como na linha 3 do código, a variável "idade" recebe o valor 25 e já é tipada como do tipo int (integer). Da mesma forma, na linha 9, "preco_produto" recebe o valor 88.987, portanto é automaticamente criada a variável sendo do tipo float (floating point, análogo ao real do pseudocódigo). A parte fracionária deve ser separada por ponto.

Na linha 14 há um exemplo de variável do tipo string (texto). O texto atribuído deve ser delimitado por aspas simples (' ') ou duplas (" ").

Por último, na linha 18, há atribuição de um valor booleano (lógico) à variável "aprovado". As palavras **True** e **False** indicam se tratar de um valor booleano, e devem ser escritas exatamente desta forma, com a primeira letra em maiúsculo.

```python
# Python   AnexoB_01   Atribuição de valores em variáveis
# 1. NÚMEROS INTEIROS (int)
idade = 25
temperatura = -30
qt_alunos = 100

# 2. NÚMEROS REAIS / PONTO FLUTUANTE (float)
nota = 8.5
preco_produto = 88.987
taxa_juros = 0.456
notacao_cientifica = 3e2  # 3 vezes 10 elevado a 2 (300.0)

# 3. LITERAIS / STRINGS (str)
nome = "Roberto"
disciplina = 'Lógica de Programação'

# 4. BOOLEANOS (bol)
aprovado = True
disciplina = 'Lógica de Programação'
```

OPERADORES ARITMÉTICOS, RELACIONAIS E LÓGICOS

Vale em Python a mesma precedência observada sobre tipos de operadores em Pseudocódigo. Assim, operadores aritméticos tem maior precedência e serão resolvidos primeiro em uma expressão, seguidos dos relacionais e por últimos os lógicos.

Cada grupo de operadores tem sua precedência, detalhada na Tabela 17, Tabela 18 e Tabela 19. No caso de operadores com a mesma precedência, resolve na ordem em que se apresentar, da esquerda para a direita.

Vale lembrar que, em uma expressão que contenha operadores relacionais ou lógicos, o resultado final sempre será um valor booleano (True ou False).

Tabela 17: operadores aritméticos em Python

Operação	Operador		Ordem	Descrição
	Pseudo.	Python		
Exponenciação	^	**	1	Eleva o operando da esquerda (base) ao operando da direita (expoente). Ex.: *2**3=8*
Divisão	/	/	2	Divide o operando da esquerda (dividendo) pelo operando da direita (divisor). Ex.: *6/3=2*
Divisão inteira	\	//		Retorna a parte inteira da divisão. Ex.: *9//2=4*
Multiplicação	*	*		Realiza a multiplicação dos operandos
Módulo	%	%		Retorna o resto da divisão de inteiros. Ex.: *9%2 = 1*
Adição	+	+	3	Quando os operandos forem do tipo numérico, realiza a soma deles. Se do tipo literal, executa a concatenação dos operandos
Subtração	-	-		Subtrai o operando da direita do operando da esquerda. Ex.: *8 – 7 = 1*

Tabela 18: operadores relacionais em Python

Comparação	Operador		Ordem	Descrição
	Pseudo.	Python		
Menor	<	<	1	Compara se o operando da esquerda é menor do que o operando da direita. Ex.: *a < b*.
Maior	>	>		Compara se o operando da esquerda é maior do que o operando da direita. Ex.: *7 > 5*.
Menor ou igual	<=	<=		Compara se o operando da esquerda é menor ou igual ao operando da direita. Ex.: *a <= b*.
Maior ou igual	>=	>=		Compara se o operando da esquerda é maior ou igual ao operando da direita. Ex.: *7 >= 5*.
Igualdade	**=**	**==**	2	Compara se dois valores são iguais. Ex.: *a==b*.
Desigualdade	**< >**	**!=**		Compara se dois valores são diferentes. Ex.: *c!=d*.

Tabela 19: operadores lógicos em Python

Operação	Operador		Ordem	Descrição
	Pseudocódigo	Python		
Negação	*NAO*	*not*	1	Inverte o valor lógico do operando examinado. Ex.: *not True = False*; *not False = True*.
Conjunção	*E*	*and*	2	Retorna verdadeiro se os dois operandos forem verdadeiros
Disjunção	*OU*	*or*	3	Retorna verdadeiro se, pelo menos, um dos operandos for verdadeiro.
Disjunção exclusiva	*EOU*	*xor*		Retorna verdadeiro se somente um dos operandos for verdadeiro.

Exemplo, em Python, com o uso de todos os tipos de operadores:

```
Expressão original   10 + 5 * 2 > 15  and  2 != 0
                              ↓
1º (Aritmética *):   10  +  10   > 15  and  2 != 0
                           ↓
2º (Aritmética +):        20   >  15  and  2 != 0
                              ↓          ↓
3º (Relacional):           True    and   True
                                   ↓
4º (Lógico):                     True
```

Parte 2: Comandos de entrada, saída e comentários

COMANDO DE SAÍDA

O comando print é o comando de saída similar, em Python, ao comando leia, em pseudocódigo. Também aceita múltiplos argumentos, separados por vírgula, que serão exibidos na tela do dispositivo.

A forma geral do comando é:

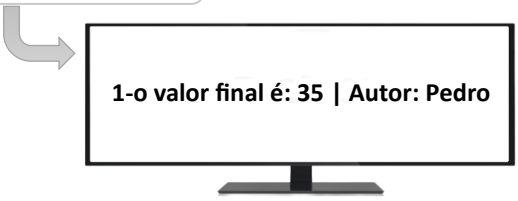
```
print(<argumento1>,<argumento2>, ... , <argumento_n>)
```

Os argumentos podem ser:

- valores numéricos (int ou float): serão exibidos como digitados.
- Literais (string): entre aspas delimitando o texto, será exibido como texto.
- Variáveis: será exibido o valor armazenado pela variável.
- Expressões: será calculado o valor, considerando as precedências e os valores de variáveis e valores que a componham, sendo exibido o valor final.

Um exemplo de input com múltiplos argumentos (considere a=3, b=5, nome="Pedro"):

```
print(1, "- o valor final é:", 10*a+b, " | Autor: ", nome)
```



O comando input pode ter argumentos literal que contenha alguns caracteres de controle:

\n – mudança de linha (enter)
\t – tabulação horizontal
\' – caractere aspas simples
\" – caractere aspas (")
\\ – caractere barra invertida

```
print("parte 1 \n parte 2")
print("\t Texto tabulado")
```

Há também a possibilidade de se utilizar as F-Strings (Formatated Strings), uma forma mais moderna e recomendada de combinar texto e variáveis. Usa-se um f antes das aspas e chaves {} para as variáveis.

```
print(f"O aluno {nome} tirou {nota}")
```

COMANDO DE ENTRADA

Em pseudocódigo, sempre existia o uso em conjunto de uma mensagem, através do comando escreva, e da leitura de um valor para uma variável, através do comando leia.

```
Escreva(“Digite a idade”)
Leia(idade)
```

Em Python, o comando input faz os dois papéis, emite uma mensagem (um parâmetro passado para o comando, entre aspas), de orientação sobre o que deve ser digitado, e abre um prompt para receber um valor via teclado (a função do leia, anterior). A forma geral de um comando input:

```
<variável> = <funcaoConversão>(input(<mensagem>))
```

Exemplo de um comando input para a entrada da idade do usuário:

```
idade = int(input(“Informe a idade:”))
```

A ordem de execução do comando de atribuição do exemplo anterior é:

- input(“Digite a idade:”)

  Uma mensagem é exibida, seguida de um cursor que permite a entrada de um valor, via teclado, pelo usuário

- int(…)

  Recebe o valor digitado pelo comando input e transforma para o tipo inteiro. Caso seja digitado um valor não numérico, será apresentado um erro e o programa será abortado. Para evitar esse tipo de erro, existem técnicas de validação mais avançadas que não serão tratadas neste momento.

- idade = (…)

  O sinal de igualdade no Python é o equivalente à seta do pseudocódigo (“<-”). O valor obtido de um comando input, uma expressão resolvida ou um literal, à direita do sinal de igualdade, será atribuído à variável à esquerda deste sinal.

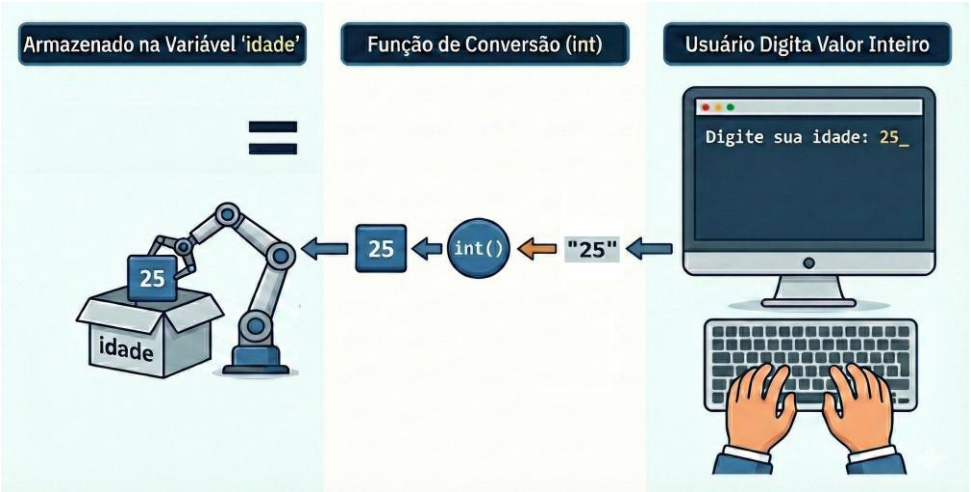


Figura 39: ordem de execução de um comando input

A informação passada pelo comando input normalmente é atribuída à uma variável. Quando não utilizada a função de conversão de tipo, o Python assume que o tipo é str (string, ou seja, texto/caractere). Assim, a variável que está sendo definida neste momento será do tipo texto, mesmo que o valor digitado seja numérico. Exemplos:

```
nome = input("Informe o nome:")
idade = input("Informe a idade:")
```

Como não foi usada a função de conversão de tipo, tanto nome quanto idade serão variáveis do tipo str (string ou texto). No caso de idade, mesmo que digitado 10, o valor armazenado será "10", como se texto fosse e não um valor numérico.

Outros exemplos de uso do input com função de conversão:

```
preco = float(input("Digite o preço:"))
aprovado = bool(input("Informe True ou False para aprovação:")
nome = str(input("Digite o nome do aluno:"))        # desnecessário
```

Nestes exemplos foram utilizadas funções de conversão para os tipos float (real), bool (lógico) e str(caractere/texto).

COMENTÁRIOS

São informações inseridas nos algoritmos que servirão apenas como explicações e orientações sobre o algoritmo ou parte dele. O conteúdo desses comentários não é levado em consideração no momento da execução do algoritmo. Comentários em Python:

```
# comentário de linha
```

```
''' comentário de bloco
comentário de bloco
comentário de bloco '''
```

Comparação de pseudocódigo x Python:

```
1  Algoritmo "Area_Circulo"
2  Var
3     area, r: real
4     PI : real
5  Inicio
6     PI <- 3.141593
7     Escreva("Digite o valor do raio:")
8     Leia (r)
9     area ← (PI * r * r)
10    Escreva ("Valor da área:",area)
11 FimAlgoritmo
```

```
1  # Área do Círculo
2  PI = 3.141593
3  r = input("Digite o valor do raio:")
4  area = PI * r * r
5  print("Valor da área: ", area)
6
```

Parte 3: Controle de Fluxo de Instruções

CONDICIONAL SIMPLES – forma geral:

```
if <condição> :
    <instrução/bloco de instruções>
```

A seguir a comparação entre condicionais simples, em pseudocódigo e Python:

```
1  Algoritmo "Uso de condicional simples"
2  // Algoritmo verifica a aprovação
3  Var
4     nota: real
5  Inicio
6     Escreva ("Digite a nota do aluno: ")
7     Leia (nota)
8     Se nota >= 5 então
9           Escreva ("Parabéns! Aprovado.")
10    Fimse
11 FimAlgoritmo
```

```
1  # Programa checa a aprovação de um aluno
2  nota = int(input("Digite a nota do aluno:"))
3  If nota >= 5 :
4     print("Prabéns! Aprovado.")
5
```

CONDICIONAL COMPOSTA – forma geral:

```
if <condição> :
|    <instrução/bloco de instruções>
else :
|    <instrução/bloco de instruções>
```

A seguir a comparação entre condicionais compostas, em pseudocódigo e Python:

```
1  Algoritmo "Uso de condicional composta"
2  // Algoritmo verifica a aprovação
3  Var
4     nota: real
5  Inicio
6     Escreva ("Digite a nota do aluno: ")
7     Leia (nota)
8     Se nota >= 5 então
9           Escreva ("Parabéns! Aprovado.")
10    Senão
11          Escreva ("Reprovado.")
12    FimSe
13 FimAlgoritmo
```

```
1  # Programa checa a aprovação de um aluno
2  nota = int(input("Digite a nota do aluno:"))
3  if nota >= 5 :
4  |    print("Parabéns! Aprovado.")
5  else:
6  |    print("Reprovado.")
7
```

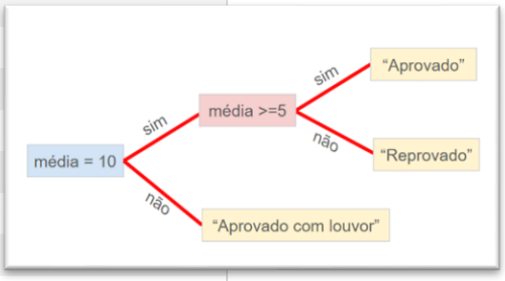
A barra "|" no código é somente para efeito didático, indicando o alinhamento de blocos de código pelo uso de tabulação, fundamental em Python.

CONDICIONAL ANINHADA – forma geral:

```
if <condição 1> :
    <instrução/bloco de instruções A>
else :
    if <condição 2> :
        <instrução/bloco de instruções B>
    else :
        <instrução/bloco de instruções C>
```

A seguir a comparação entre condicionais aninhadas, em pseudocódigo e Python:

```
1  Algoritmo "Média aluno com uso de condicional aninhada"
2  /*Algoritmo que demonstra condicional aninhada
3  Var
4     nota1, nota2, media: real
5  Inicio
6     Escreva ("Digite a nota do primeiro semestre: ")
7     Leia (nota1)
8     Escreva ("Digite a nota do segundo semestre: ")
9     Leia (nota2)
10    media ← (nota1 + nota2)/2
11    Escreva ("A média é: ",media)
12    Se media = 10 então
13       Escreva ("Sensacional!!!! Você foi aprovado com louvor.")
14    Senão
15       Se media >= 5 E media < 10 então
16          Escreva ("Parabéns!!!! Você foi aprovado.")
17       Senão
18          Escreva ("Sinto muito, mas você foi reprovado.")
19       Fimse
20    Fimse
21 FimAlgoritmo
```




```
1  # Algoritmo com Condicional Aninhada
2  nota1 = float(input("Digite a nota do primeiro semestre: "))
3  nota2 = float(input("Digite a nota do segundo semestre: "))
4  media = (nota1 + nota2) / 2
5  print("A média é: ", media)
6  if media == 10:
7      print("Sensacional!!!! Você foi aprovado com louvor.")
8  else:
9      # Observe o recuo: Este bloco inteiro está dentro do else anterior
10     if media >= 5 and media < 10:
11         print("Parabéns!!!! Você foi aprovado.")
12     else:
13         print("Sinto muito, mas você foi reprovado.")
```

Há uma outra forma de implementar uma condicional composta, sem utilização de condicionais aninhadas, com o uso de “elif”. A forma geral é:

```
if (condição1):
    <instrução/bloco de instruções 1>
elif (condição2):
    <instrução/bloco de instruções 2>
elif (condição3):
    <instrução/bloco de instruções 3>
else:
    <instrução/bloco de instruções padrão>
```

SELEÇÃO DE MÚLTIPLA ESCOLHA: forma geral:

```
match <Expressão/variável de escolha>:
    case <valor 1>:
        <instruções referentes ao caso valor 1>
    case <valor 2>:
        <instruções referentes ao caso valor 2>
    case _:
        <instruções referentes a outros valores>
```

```
Algoritmo “Escolha de Disciplina optativa”
// Algoritmo que oferece opções de disciplina
Var
    opcao: inteiro
Inicio
    Escreva (“Selecione a opção: “)
    Escreva (“1-Álgebra; 2-Filosofia; 3-Pesquisa“)
    Leia (opcao)
    Escolha opcao
    caso 1
        Escreva (“Você optou por Álgebra”)
    caso 2
        Escreva (“Você optou por Filosofia”)
    caso 3
        Escreva (“Você optou por Pesquisa”)
    outro caso
        Escreva (“Opção inválida”)
    FimEscolha
FimAlgoritmo
```

```
# Algoritmo que oferece opções disciplina
print("Selecione a opção:”)
print(“1-Álgebra; 2-Filosofia; 3-Pesquisa")
opcao = int(input("Digite sua escolha: "))
match opcao:
    case 1:
        print("Você optou por Álgebra")
    case 2:
        print("Você optou por Filosofia")
    case 3:
        print("Você optou por Pesquisa")
    case _:
        print("Opção inválida”)
```

python

Parte 4: Laços de Repetição

Figura 40: tipos de laço de repetição

LAÇO DE REPETIÇÃO CONTADO

Sintaxe geral:

```
for <variável> in range(<início>, <fim>,<passo>):
    <instrução/bloco de instruções A>
```

Algumas observações da estrutura **for**, em Python:

- Análoga à estrutura **para** em pseudocódigo.
- Não é necessário declarar a <variável> previamente.
- O valor de fim é exclusivo, isto é, se definir 5, o laço se repetirá de 1 até 4 (fim-1).
- O valor do <passo> é opcional. Se não declarado assume o valor 1.

```
1  Algoritmo "Media de 30 alunos"
2  // Algo.que calcula a média dos alunos
3  // e apresenta a msg de resultado.
4  Var
5     nota1, nota2, media: real
6     alunos: inteiro
7  Inicio
8     Para alunos de 1 até 30 passo 1 faça
9           Escreva ("Nota do 1º semestre: ")
10          Leia (nota1)
11          Escreva ("Nota do 2º semestre: ")
12          Leia (nota2)
13          media ← (nota1 + nota2)/2
14          Escreva ("A média é: ",media)
15          Se media >= 5 então
16                Escreva ("Aluno aprovado.")
17          Senão
18                Escreva ("Aluno reprovado.")
19          Fimse
20    Fimpara
21 FimAlgoritmo
```

```
1  # Programa Média de 30 alunos
2  for num_alunos in range(1, 31):
3     nota1 = float(input("Nota do 1º semestre: "))
4     nota2 = float(input("Nota do 2º semestre: "))
5     media = (nota1 + nota2) / 2
6     print("A média é: ", media)
7     if media >= 5:
8        print("Aluno aprovado.")
9     else:
10       print("Aluno reprovado.")
```

LAÇO DE REPETIÇÃO CONDICIONAL - WHILE

Sintaxe geral:

```
while <condição> :
|     <instrução/bloco de instruções>
```

Algumas observações da estrutura **for**, em Python:

- Análoga à estrutura **Enquanto**, em Pseudocódigo.
- Enquanto a <condição> for satisfeita, repete a instrução/bloco de instruções.
- Como já explicado, neste tipo de laço o valor avaliado na <condição> tem que sofrer alguma alteração dentro do laço. Se isso não ocorrer o programa entrará em *looping* infinito.

```
Algoritmo "Laço enquanto"
// Algoritmo que calcula e exibe a média
// de 2 semestres usando laço condicional.
Var
   nota1, nota2, media: real
   resp: inteiro
Inicio
   resp ← 1
   Enquanto resp = 1 faça
      Escreva ("Digite a nota do 1º sem: ")
      Leia (nota1)
      Escreva ("Digite a nota do 2º sem: ")
      Leia (nota2)
      media ← (nota1 + nota2)/2
      Escreva ("A média é: ",media)
      Escreva ("Digite (1) p/ continuar: ")
      Leia (resp)
   Fimenquanto
FimAlgoritmo
```

```
# Algoritmo "Laço enquanto" em Python
resp = 1
# Repete enquanto resp for igual a 1
while resp == 1:
   nota1 = float(input("Nota do 1º sem: "))
   nota2 = float(input("Nota do 2º sem: "))
   media = (nota1 + nota2) / 2
   print("A média é: ", media)
   resp = int(input("Digite 1 p/ continuar:"))
```

python

O laço condicional do tipo **Repita**, em pseudocódigo, não tem estrutura semelhante em Python. Entretanto, é possível adaptar a estrutura **while** para apresentar um comportamento análogo, como na seguinte forma geral:

```
while True :
|     <instrução/bloco de instruções>
|     if <condição> :
|        break
```

Este tipo de laço de repetição tem uma dinâmica diferente. O "while True:" no início faz com que o laço se repita indefinidamente, pois a "condição" foi substituída pelo valor True (verdadeiro). Com isso, a <instrução/bloco de instruções> será executada pelo menos uma vez. O ***if*** ao final avalia uma condição e, sendo verdadeira, executa o comando "**break**", que provoca a saída do laço, cessando as repetições. Esta é a diferença primordial entre os dois arranjos. Um

repete enquanto uma condição for verdadeira. O outro repete até que uma condição seja atingida (verdadeira).

```
1  Algoritmo "Uso de repita"
2  // Algoritmo que calcula e exibe a média de
3  // dois semestres usando < repita >.
4  Var
5     nota1, nota2, media: real
6     resp: inteiro
7  Inicio
8     Repita
9        Escreva ("Nota 1º semestre: ")
10       Leia (nota1)
11       Escreva ("Nota 2º semestre: ")
12       Leia (nota2)
13       media ← (nota1 + nota2)/2
14       Escreva ("A média é: ",media)
15       Escreva ("Digite 1 p/  continuar: ")
16       Leia (resp)
17    Até resp < > 1
18 FimAlgoritmo
```

```
1  # Programa que calcula a média de alunos
2  # Simula comportamento do "repita"
3  while True:
4      nota1 = float(input("Nota 1º semestre: "))
5      nota2 = float(input("Nota 2º semestre: "))
6      media = (nota1+nota2)/2
7      print("A média do aluno foi:",media)
8      resp = int(input("Digite 1 p/ continuar:"))
9      if resp != 1:
10         break
```

Parte 5: Funções e Procedimentos

A diferença entre procedimento e função reside no fato de que uma função sempre retorna um valor ao programa que a invocou, enquanto um procedimento não provê qualquer retorno. Tanto a função como o procedimento têm como sintaxe geral:

def <nome da função> (parâmetro1, parâmetro2, (...) , parâmetro$_n$) **:**
    <instrução/bloco de instruções>

Considerando o código de exemplo de uso de uma função em Python: um programa principal faz um chamado para uma função (linha 12), passando os parâmetros necessários à sua execução (no caso "vr_base" e "vr_expoente"). A função realiza algum processamento com base nos parâmetros passados e, em algum momento, no corpo da função, deverá existir uma cláusula **return** (linha 6), devolvendo o valor processado ao ponto no programa principal que chamou a função (novamente, linha 12). No programa exemplo, o valor retornado da função será atribuído à variável "valor_resultado".

```
1  # Programa Python com o uso de uma função
2
3  # Definição da função que calcula a potência
4  def calcula_potencia( base , expoente ):
5      resultado = base**expoente
6      return resultado
7
8  # Programa principal
9  valor_base = int(input("Digite o valor da base: "))
10 valor_expoente = int(input("Digite o valor do expoente: "))
11 #Chamada da função
12 valor_resultado = calcula_potencia(vr_base,vr_expoente)
13 #Apresentação do resultado da operação
14 print("O valor do resultado da operação é:",valor_resultado)
```

No procedimento, como não tem retorno de valor para o programa principal, não existe a cláusula **return**. No exemplo a seguir, verifique que é feita uma chamada ao procedimento (linha 12), mas que não existe atribuição de valor a uma variável, uma vez que não existe nenhum valor de retorno para tal. Considerando o código de exemplo, um programa principal faz um chamado para um procedimento(linha 12), passando a mensagem que será exibida como parâmetro, no caso uma *string* concatenada com o nome informado pelo usuário, atribuída à variável "nome".

```
1  # Programa Python com o uso de um procedimento
2
3  # Definição do procedimento que formata uma mensagem
4  def mostra_mensagem( mensagem ):
5        print( "\n-----------------------------------------------")
6        print( ">>", mensagem)
7        print( "-----------------------------------------------")
8
9  # Programa principal
10 nome = input("Digite o nome: ")
11 #Chamada da função
12 mostra_mensagem( 'Bem-vindo:'+nome )
```

Importação de bibliotecas de funções

A importação de bibliotecas em Python funciona como a adição de "caixas de ferramentas" especializadas ao núcleo básico da linguagem, permitindo que o programador realize tarefas complexas — como cálculos matemáticos avançados, geração de gráficos ou sorteios — sem precisar criar esses códigos do zero.

O grande benefício dessa prática é a eficiência e a segurança, seguindo o princípio de "não reinventar a roda": ao utilizar módulos que já foram amplamente testados e otimizados pela comunidade, se economiza tempo de desenvolvimento, reduz a chance de erros e mantém o programa mais legível.

No código abaixo estão duas formas de importar bibliotecas.

```
1  # Importando bibliotecas em Python
2
3  # Importa TODA a biblioteca de matemática
4  import math
5
6  raiz = math.sqrt(49)              # Preciso escrever 'math.' antes da função
7  fatorial = math.factorial(5)
8  print("A raiz de 49 é ", raiz)
9  print("O fatorial de 5 é ", fatorial)
10
11 # Importa só o gerador de inteiros e o escolhedor
12 from random import randint
13 dado = randint(1, 6)
14 print("O dado caiu em: ", dado)
```

Parte 6: Vetores e matrizes

A linguagem de programação Python, além das variáveis, possui diversas estruturas de dados, tais como: listas, tuplas, dicionários e conjuntos. A seguir será abordada uma das mais importantes estruturas de dados da linguagem de programação Python: a lista, similar ao vetor de pseudocódigo, mas muito mais poderosa. Trata-se de uma sequência de elementos, onde cada elemento possuí um índice para referenciá-lo, sendo possível a modificação de qualquer elemento dessa lista.

Diferentemente do que ocorre em algumas outras linguagens de programação, uma lista é uma estrutura heterogênea, pois pode ser constituída, concomitantemente, por elementos de qualquer tipo de dado: *int*, *float*, *string*, ou até outra lista.

Forma geral:

<*nome_variável*> = [*elemento1 , elemento2 , elemento3 , … , elemento*$_n$]

```
1  # Programa Python com o uso de listas
2  lista_inteiros = [1, 2, 3, 4, 5]
3  lista_reais = [6.7, 7.8, 10.0]
4  lista_textos = ["a", "professor", "aluno", "computação"]
5  listaX = [ ]          # Lista vazia (conjunto vazio)
6  matriz3x3 = [ [1,2,3] , [4,5,6] , [7,8,9] ]      # Lista bidimensional (matriz)
7
8  # Lista heterogênea (misturada) e aninhada:
9  # Contém um inteiro, uma string, outra lista (os meses) e uma variável de lista
10 minha_lista = [2022, "Python", ["janeiro", "fevereiro"], lista_reais]
11
12 # Referenciando elementos de uma lista (índice de 0 a n-1)
13 print( lista_inteiros[2] )
14 print( lista_reais[-1] )
15 print( minha_lista[2][0] )
16 print( matriz3x3[2][0] )
17
```

```
3
10.0
Janeiro
6
```

Existem uma série de funções e métodos nativos da linguagem Python para manipulação de listas. Considere as seguintes listas para aplicação na Tabela 20:

listaNumeros = [4, 3 ,1, 4, 5]
outrosNumeros = [88, 99]
listaNomes = [“Ana”,”João”,”Pedro”]
listaNomes2 = [“Carlos”, “César”]

Tabela 20: funções e métodos de manipulação de listas

Função	Ação	Resultado
len(listaNumeros)	Número de elementos na lista	5
sum(listaNumeros)	Soma os valores de uma lista	17
min(listaNumeros)	Obtém o menor valor na lista	1
max(listaNumeros)	Obtém o maior valor na lista	5
del listaNumeros[4]	Exclui o elemento de índice n	listaNumeros = [4, 3, 1, 4]
"Ana" in listaNomes	Verifica se o valor está na lista	True
todosNomes=listaNomes+listaNomes2	Consolida duas listas em uma	todosNomes = ["Ana","João","Pedro", "Carlos", "César"]
listaNomes.extend(listaNomes2)	Estende a primeira lista com o conteúdo da segunda lista	todosNomes = ["Ana","João","Pedro", "Carlos", "César"]
listaNumeros.append(9)	Adiciona um elemento ao final da lista	listaNumeros = [4, 3, 1, 4, **9**]
listaNumeros.extend(outrosNumeros)	Estende a primeira lista com o conteúdo da segunda lista	listaNumeros = [4, 3, 1, 4, 9, **88**, **99**]
listaNumeros.insert(2, 9)	Insere, na posição indicada pelo índice, um novo elemento	listaNumeros = [4, 3, 9, 1, 4, 9, 88, 99]
listaNumeros.remove(9)	Remove o primeiro elemento com um determinado valor	listaNumeros = [4, 3, 1, 4, 9, 88, 99] . Sacou o 1º nove da lista anterior
listaNomes.index("João")	retorna o índice do valor buscado	1 (lembre-se que o índice inicia em 0)
listaNumeros.pop(1)	Retira o elemento de um determinado índice e mostra o valor sacado	listaNumeros = [4, 1, 4, 9, 88, 99] e mostra o valor sacado: 3
listaNumeros.count(4)	Retorna a quantidade de vezes que um valor aparece na lista	retorna 2 (número de vezes que o 4 é encontrado na lista)
listaNumeros.sort()	Coloca os elementos da lista em ordem crescente	listaNumeros = [1, 4, 4, 9, 88, 99]
listaNumeros.reverse()	Inverte a ordem dos elementos de uma lista	listaNumeros = [99, 88, 9, 4, 4, 1]
listaNumeros[1:3]	Apresenta um subconjunto de elementos da lista, delimitados por índices de início e fim (-1)	[88, 9] entre os índices 1 ao 2 (3 excluído)
listaNumeros.clear()	Elimina os elementos de uma lista	listaNumeros = []

Considere, em cada resultado, que as listas podem ter sido modificadas nas ações anteriores.

Parte 7: Erros em tempo de execução

Todos os programas apresentam erros em tempo de execução (*Runtime Errors*), que são falhas que ocorrem enquanto o programa está sendo executado, e não durante a escrita ou compilação do código. Em *Python*, assim como em qualquer outra linguagem, também ocorrem estes erros, sendo que os mais comuns estão listados na Tabela 21.

Tabela 21: erros mais comuns em tempo de execução, em Python

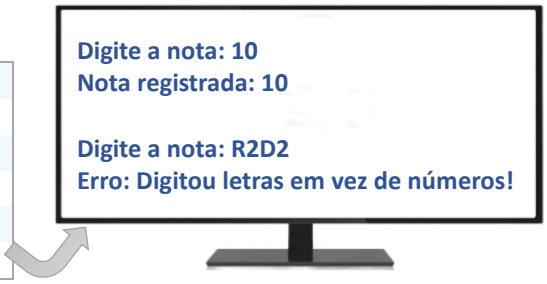
Erro / Descrição	Código	Mensagem
NameError *Ocorre quando você tenta usar uma variável ou função que não foi definida (ou cujo nome foi digitado errado).*	*# Início do programa* print(a)	*NameError: name 'a' is not defined*
TypeError *Acontece quando você tenta aplicar uma operação a um tipo de dado incompatível. Por exemplo, a entrada de dados não foi convertida, assumida como string (texto) e associada a uma variável. Na sequência, tenta-se adicionar um valor numérico a uma variável texto.*	*# Início do programa* idade = input(“Digite a idade) novaIdade = idade + 3	*TypeError: can only concatenate str (not "int") to str*
ValueError *Ocorre quando uma função, como int() ou float(), recebe um tipo correto (como uma string), mas o conteúdo dessa string não pode ser convertido. Por exemplo, o usuário digita letras quando o programa esperava números.*	*# Início do programa* idade = int(“Olá!”) *# Considere que o usuário* *# Digitou “R2D2” abaixo:* idade = int(input(“Digite a idade”))	*ValueError: invalid literal for int() with base 10: 'Olá!'* *ValueError: invalid literal for int() with base 10: 'R2D2'*
ZeroDivisionError *Tentativa de executar uma operação matematicamente impossível.* *Cenário: Calcular a média dividindo pela quantidade de alunos, mas a quantidade é 0.*	*# Início do programa* qtAlunos = 0 somaNotas = 15 media = somaNotas / qtAlunos	*ZeroDivisionError: division by zero*
IndexError *Ocorre ao tentar acessar um índice que não existe em uma lista, por exemplo. No exemplo ao lado, o índice 3 dá erro pois, embora existam 3 elementos na lista, ela é numerada como 0, 1 e 2.*	*# Início do programa* lista = ["Ana", "Bia", "Caio"] print(lista[3])	*IndexError: list index out of range*
UnexpectedIndex *Ocorre ao inserir* tabulações (indentação) em locais em que esse recurso não é necessário para delimitar um bloco. No exemplo, o print(idade) é precedido de espaços não necessários. A indentação é obrigatória para delimitar blocos dentro de comandos como ***if***, ***for*** ou ***def***.	*# Início do programa* idade = 10 *nome = “Pedro”* print(idade) print(nome)	*UnexpectedIndex*

Tratamento de Exceções

Em Python, é possível tratar um erro para que o programa não sofra uma interrupção abrupta. Usa-se a estrutura try...except (Tente... Exceto).

Exemplo usando o código de média anterior:

```
1  # Programa Python com o uso de Try...Catch
2  try:
3      nota1 = float(input("Digite a nota: "))
4      print("Nota registrada: ", nota1)
5  except ValueError:
6      print("Erro: Digitou letras em vez de números!")
```



Referências Bibliográficas

[1] G. Boole, “Title: The Mathematical Analysis of Logic Being an Essay Towards a Calculus of Deductive Reasoning”, 2011. [Online]. Disponível em: www.gutenberg.org

[2] Thiago Lima, “Portas Lógicas”, https://embarcados.com.br/poertas-logicas/. Acessado: 16 de dezembro de 2025. [Online]. Disponível em: https://embarcados.com.br/poertas-logicas/

[3] M. Karnaugh, “The Map Method for Synthesis of Combinational Logic Circuits”, *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 1953.

[4] Bruno Luvizotto Carli, *Algoritmos e Lógica de Programação com Python*, 1ª Edição. 2017. Acessado: 21 de dezembro de 2025. [Online]. Disponível em: https://github.com/brunolcarli/AlgoritmosELogicaDeProgramacaoComPython

[5] José Augusto N G Manzano e Jayr Figueiredo de Oliveira, *Algoritmos - Lógica para programação de computadores*. São Paulo: Editora Érika, 2005.

[6] Victorio Albani de Carvalho, “Lógica de Programação”, 2009, *CEAD / IFES*.

[7] T. H. . Cormen, C. Eric. Leiserson, R. L. . Rivest, e Clifford. Stein, *Introduction to algorithms*. MIT Press, 2009.

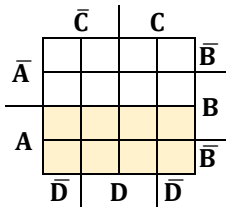
[8] R. Gotardo, *Linguagem de programação I*. Rio de Janeiro: SESES, 2015.

[9] Neilor Tonin, “Apostila de Algoritmos e Estrutura de Dados I”, março de 2006, *Universidade Regional Integrada do Alto Uruguai e das Missões*.

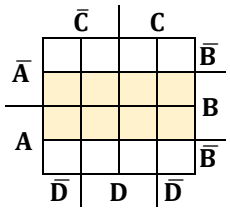
[10] J. C. dos Santos, “Curso Técnico em Automação Industrial: Algoritmo e linguagem de programação”, IFCE. Acessado: 11 de janeiro de 2026. [Online]. Disponível em: https://www.academia.edu/40444781/Técnico_em_Automação_Industrial_Algoritmo_e_Linguagem_de_programação_Professor_José_Ciro_dos_Santos

[11] A. B. Downey, *Pense em Python*. 2016.

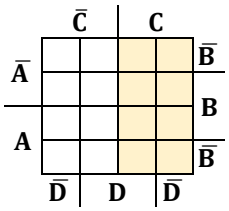
ENCARTE DO CAPÍTULO 1 – REGIÕES DO MAPA DE KARNAUGH DE 4 VARIÁVEIS



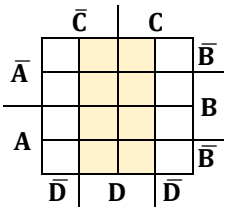
Região A



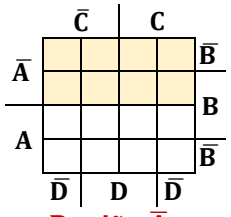
Região B



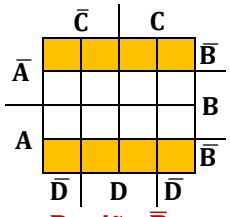
Região C



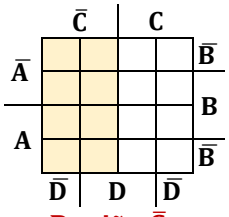
Região D



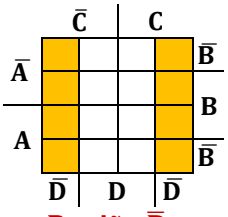
Região Ā



Região B̄



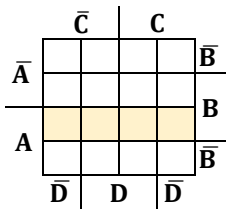
Região C̄



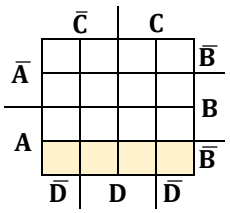
Região D̄

Agrupamentos contíguos

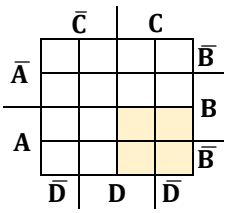
Agrupamentos desdobrados



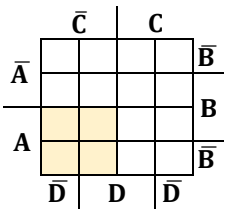
Região A. B



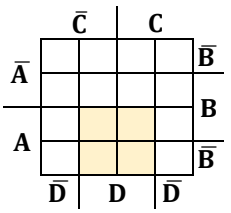
Região A. B̄



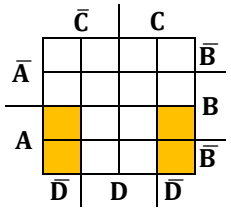
Região A. C



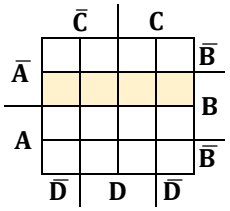
Região A. C̄



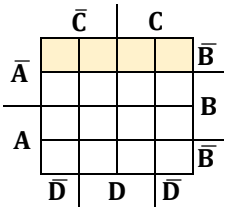
Região A. D



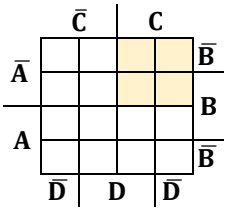
Região A. D̄



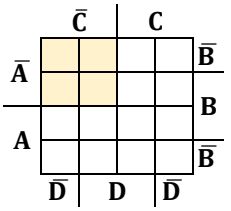
Região Ā. B



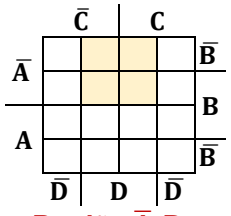
Região Ā. B̄



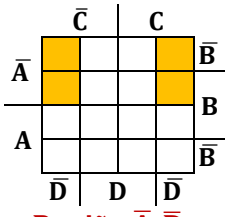
Região Ā. C



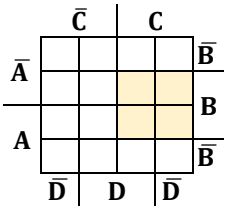
Região Ā. C̄



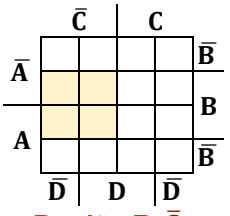
Região Ā. D



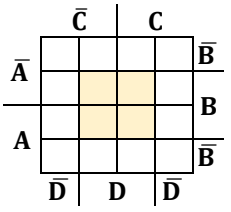
Região Ā. D̄



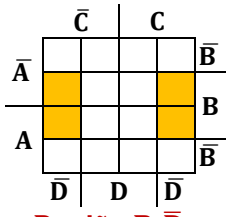
Região B. C



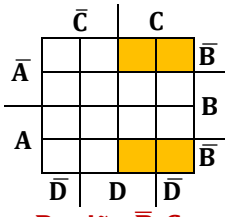
Região B. C̄



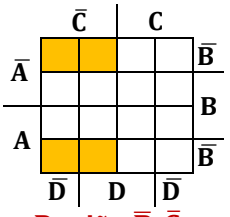
Região B. D



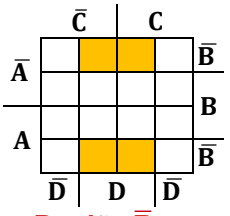
Região B. D̄



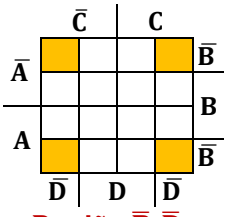
Região B̄. C



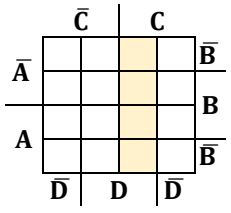
Região B̄. C̄



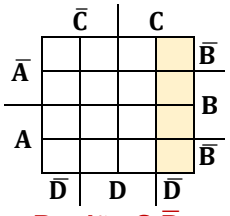
Região B̄.D



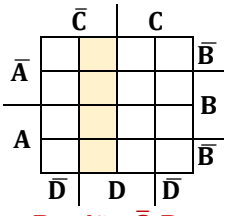
Região B̄. D̄



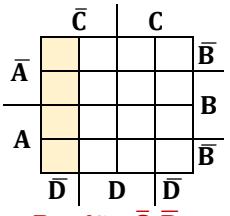
Região C. D



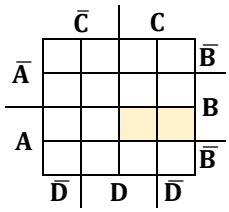
Região C. D̄



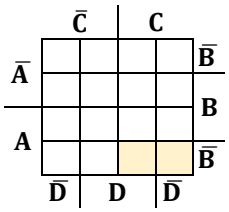
Região C̄. D



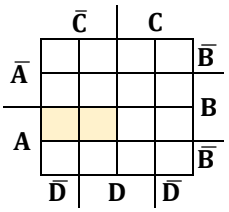
Região C̄. D̄



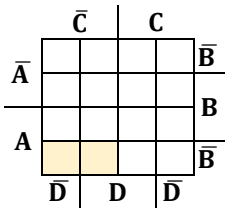
Região A. B. C



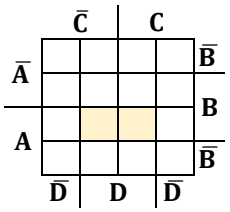
Região A. B̄. C



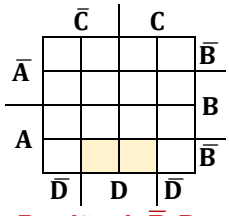
Região A. B. C̄



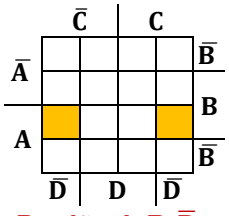
Região A. B̄. C̄



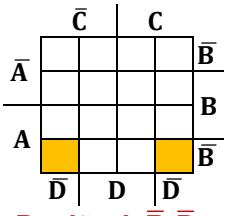
Região A. B. D



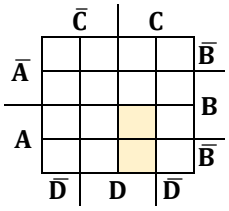
Região A. B̄. D



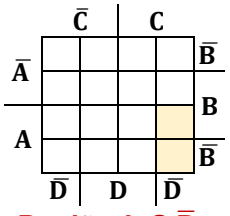
Região A. B. D̄



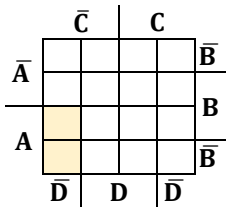
Região A. B̄. D̄



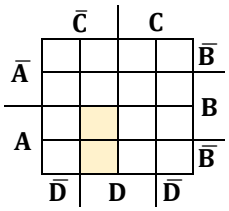
Região A. C. D



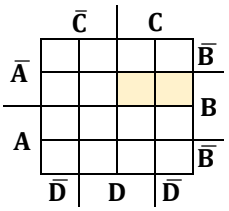
Região A. C. D̄



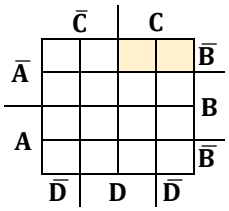
Região A. C̄. D̄



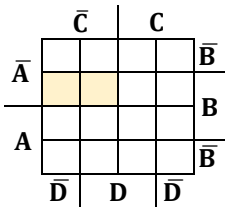
Região A. C̄. D



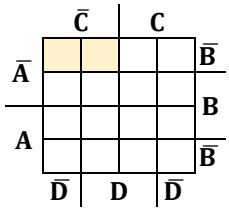
Região Ā. B. C



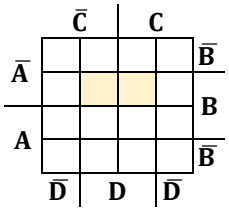
Região Ā. B̄. C



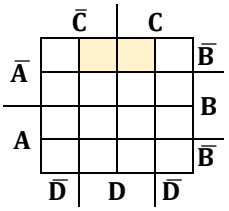
Região Ā. B. C̄



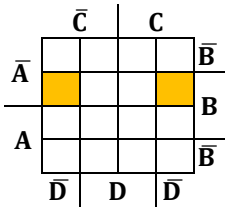
Região Ā. B̄. C̄



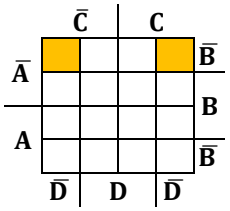
Região Ā. B. D



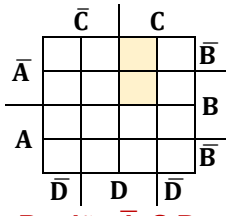
Região Ā. B̄. D



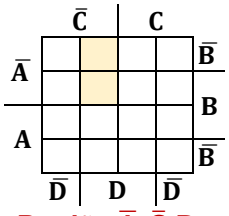
Região Ā. B. D̄



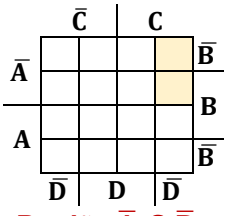
Região Ā. B̄. D̄



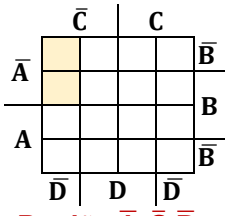
Região Ā. C. D



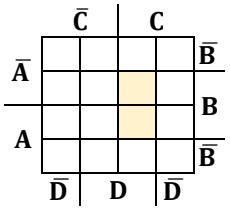
Região Ā. C̄. D



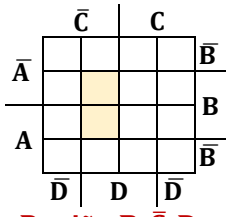
Região Ā. C. D̄



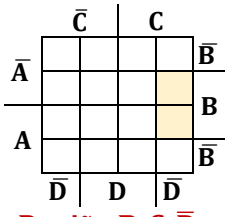
Região Ā. C̄. D̄



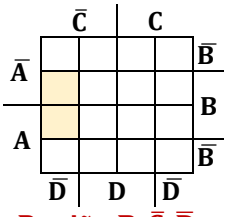
Região B. C. D



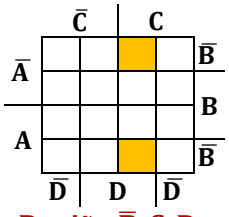
Região B. C̄. D



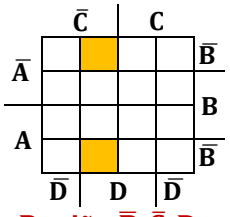
Região B. C. D̄



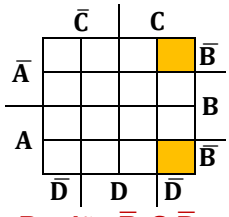
Região B. C̄. D̄



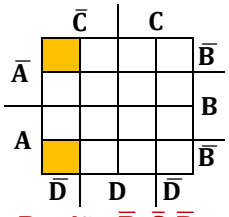
Região B̄. C. D



Região B̄. C̄. D



Região B̄. C. D̄



Região B̄. C̄. D̄

PARA CÉLULAS NÃO AGRUPADAS (ISOLADAS):

	C̄		C		
Ā	Ā B̄ C̄ D̄	Ā B̄ C̄ D	Ā B̄ C D	Ā B̄ C D̄	B̄
	Ā B C̄ D̄	Ā B C̄ D	Ā B C D	Ā B C D̄	B
A	A B C̄ D̄	A B C̄ D	A B C D	A B C D̄	
	A B̄ C̄ D̄	A B̄ C̄ D	A B̄ C D	A B̄ C D̄	B̄
	D̄	D		D̄	