The slide features decorative blue lines in the corners. In the top-left, three parallel lines form an L-shape. In the bottom-left, three parallel lines extend horizontally and then diagonally. In the bottom-right, three parallel lines extend diagonally upwards.

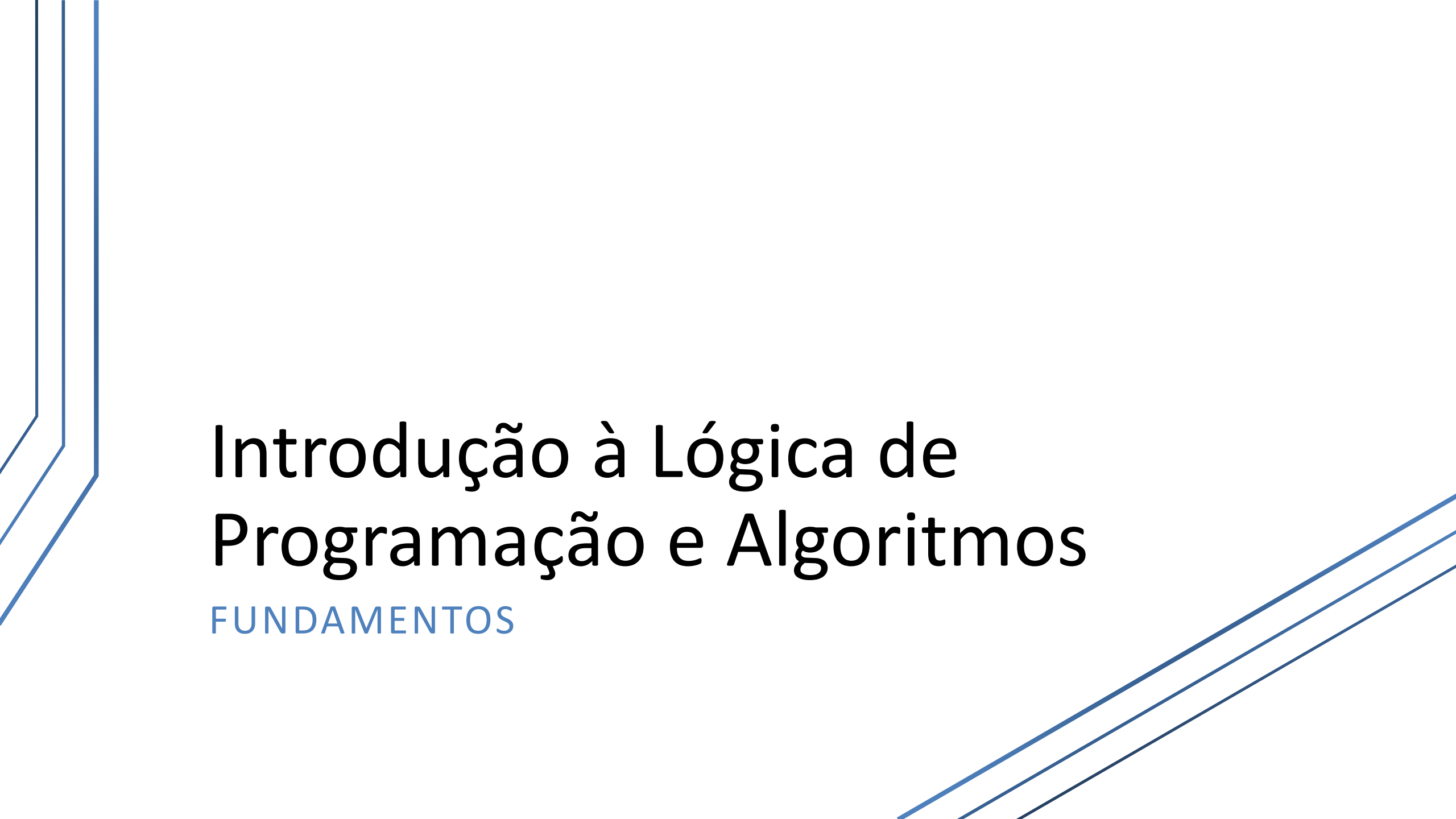
Módulo 1 - Introdução à Lógica de Programação e Algoritmos

Roteiro

- Fundamentos
- Variáveis e Atribuição
- Comandos de Entrada e Saída
- Operações Aritméticas
- Operações Relacionais, expressões lógicas e condicionais
- Estruturas de repetição
- Linguagem de Programação JavaScript

Objetivos

- Desenvolver o pensamento lógico;
- Entender como são feitos os sistemas de computador;
- Construir algoritmos com os elementos básicos de programação.



Introdução à Lógica de Programação e Algoritmos

FUNDAMENTOS

Roteiro

- Organização de um computador
- Lógica de Programação
- Algoritmo



Objetivos

- Compreender os conceitos de lógica de programação e de algoritmos.

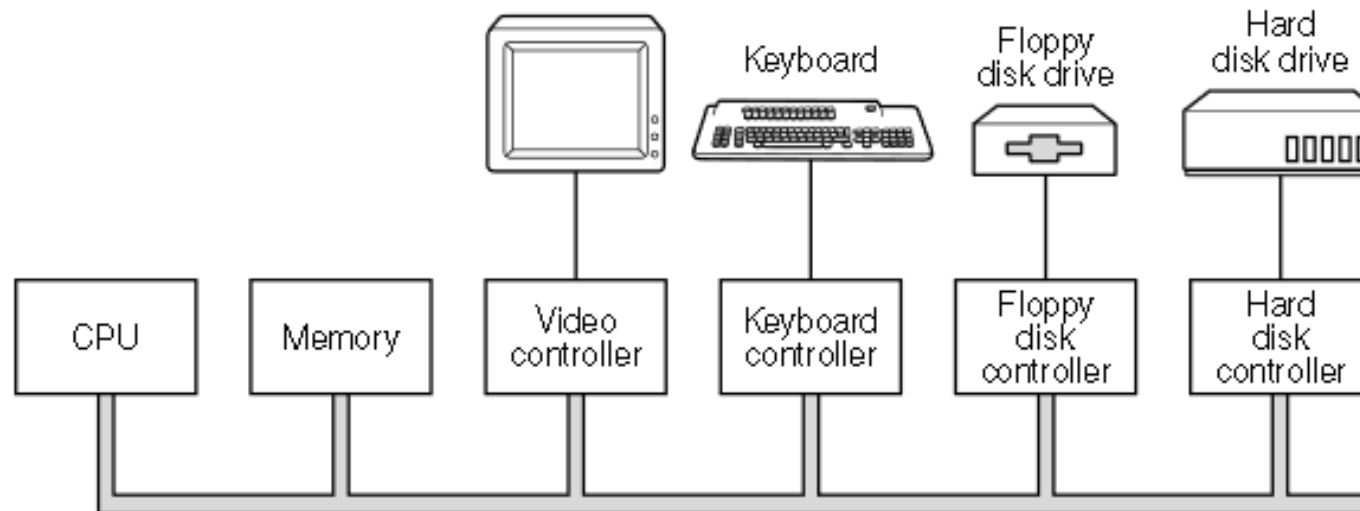
O que é um computador?

- Computador: o que computa, calculador, calculista (dicionário Houaiss).
- Os primeiros “computadores” eram humanos que calculavam tabelas de logaritmos ou trajetórias pra canhões.
- Um computador é uma máquina que, a partir de uma entrada, realiza um número muito grande de cálculos matemáticos e lógicos, gerando uma saída.

Hardware e dispositivos

A linguagem nativa do computador é codificada numericamente, de forma binária (utilizando zeros e uns).

- Bit → Pode assumir valores 0 ou 1.
- Byte → Agrupamento de 8 bits.
- Letras e símbolos são representados por números.



Organização básica de um ambiente computacional

- Computadores realizam tarefas complexas por meio de um número enorme de atividades simples.
- Para gerenciar a complexidade das soluções, existe uma hierarquia de funções, onde cada uma apresenta uma interface mais simples.

Programa de Aplicações		
Shell	Compiladores	Editores
Sistema Operacional		
Hardware		

Programando computadores

- Como usuários do computadores, interagimos com os **programas de aplicações**.
- Neste curso iremos construir novos **programas de aplicação**.

Programa de Aplicações		
Shell	Compiladores	Editores
Sistema Operacional		
Hardware		

Introdução à Lógica de Programação

- Lógica: É a arte de pensar corretamente. A lógica visa colocar ordem no pensamento.
 - Utilizamos a lógica em nosso dia a dia. Por exemplo:
 - ✓ Sei que sou mais velho que João.
 - ✓ Sei que João é mais velho que José.
 - ✓ Então, concluo que eu sou mais velho que José.
 - Sejam os seguintes fatos:
 - ✓ Todos os filhos de João são mais altos do que Maria.
 - ✓ Antônio é filho de João.
 - ✓ Então, podemos concluir que?

Introdução à Lógica de Programação

- Lógica de programação: é a **técnica** de colocar **ordem nos pensamentos** para atingir determinado **objetivo**.
- Sequência Lógica: são os passos executados até **atingir o objetivo** ou a solução de um problema.
- Instruções: é um conjunto de regras definidas para a realização de algo. É o que indica a um computador **uma ação elementar a executar**.

Algoritmos

- O objetivo fundamental de toda programação é construir algoritmos.
- Algoritmos é **uma sequência de passos** que levam à execução de uma tarefa.
- Podemos pensar em **algoritmo como uma receita**.
- A **criação de um algoritmo** é um exercício de **criatividade** (conhecimento) e **experiência** (técnica e prática)

Algoritmos

Problema: Trocar uma lâmpada.

Sequência de passos para a solução:

1. Pegue uma escada;
2. Posicione a escada embaixo da lâmpada;
3. Pegue uma lâmpada nova;
4. Suba na escada;
5. Retire a lâmpada velha;
6. Coloque a lâmpada nova.

Esta solução é apenas uma das muitas soluções possíveis para o problema apresentado.

Algoritmos

Problema: Somar dois números e multiplicar o resultado pelo primeiro número.

Sequência de passos para a solução:

1. Receba o primeiro número;
2. Receba o segundo número;
3. Some o primeiro com o segundo número;
4. Multiplique o primeiro número pelo resultado da soma obtido no passo 3;
5. Mostre o resultado do cálculo.

Algoritmos

Problema: Calcular a média aritmética dos alunos. Os alunos realizarão quatro provas: P1, P2, P3 e P4.

Sequência de passos para a solução:

Algoritmos

- Pseudocódigo;
- Diagrama de blocos ou fluxograma;

Algoritmos - Pseudocódigo

- São independentes das linguagens de programação;
- Devem ser fácil de se interpretar e fácil de codificar;
- Devem ser o intermediário entre a linguagem falada e a linguagem de programação (Java, Javascript e Python).

Algoritmos - Pseudocódigo

Regras para construção:

- Usar somente um verbo por frase;
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática;
- Usar frases curtas e simples;
- Ser objetivo;
- Procurar usar palavras que não tenham duplo sentido.

Algoritmos - Pseudocódigo

Ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais:

- Entrada: São os dados de entrada do algoritmo;
- Processamento: São os procedimentos utilizados para chegar ao resultado;
- Saída: São os dados já processados.



Algoritmos - Pseudocódigo

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

1. Receba código da peça;
2. Receba valor da peça;
3. Receba Quantidade de peças;
4. Calcule o valor total (Quantidade * Valor da peça);
5. Mostre o código da peça e seu valor total.

Algoritmos - Pseudocódigo

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

1. Receba código da peça;
 2. Receba valor da peça;
 3. Receba Quantidade de peças;
 4. Calcule o valor total (Quantidade * Valor da peça);
 5. Mostre o código da peça e seu valor total.
- } Entrada
- } Processamento
- } Saída

Algoritmos – Exercício

1. Suponha que você tenha uma caixa cheia de bolas. Nessa caixa existem bolas azuis 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 azuis na “caixa 2” e as bolas vermelhas na “caixa 3”.

Algoritmos – Diagrama de Blocos ou Fluxograma

- E uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento;
- Eles permitem visualizar os caminhos (fluxos) e as etapas de processamento de dados possíveis e, dentro destas, os passos para a resolução do problema.
- A principal função é facilitar a visualização de um algoritmo, já que um "desenho" muitas vezes substitui, com vantagem, várias palavras.

Algoritmos – Diagrama de Blocos ou Fluxograma

- Principais símbolos



TERMINAL

Indica o INÍCIO ou FIM do processamento



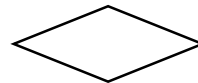
PROCESSAMENTO

Processamento em geral



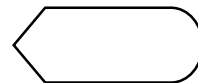
ENTRADA DE DADOS MANUAL

Indica entrada de dados através do teclado



DECISÃO

Indica uma decisão no fluxo do programa

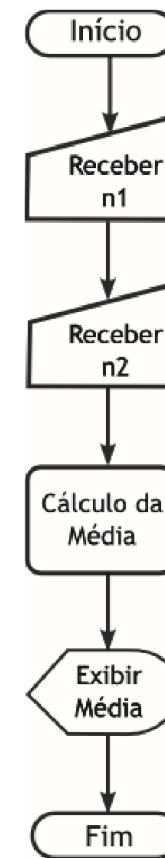


EXIBIR

Mostra informações ou resultados

Algoritmos – Diagrama de Blocos ou Fluxograma

- Problema: Calcular a média de dois números.
- Sequência de passos para a solução:
 1. Receber primeiro número (n1);
 2. Receber segundo número (n2);
 3. Cálculo da Média ($(n1 + n2) / 2$);
 4. Exibir Média.

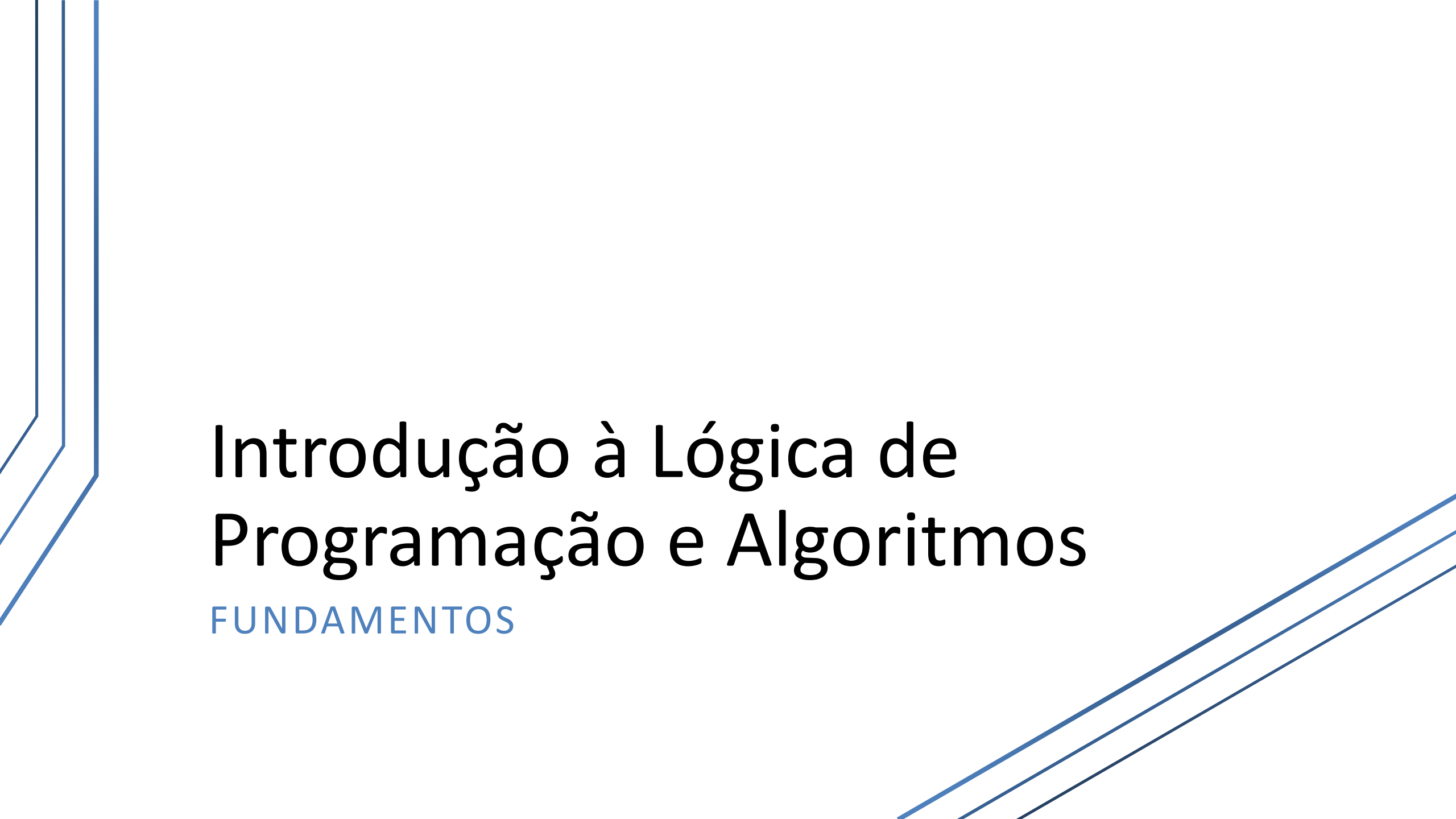


Algoritmos – Exercício

1. Construa um algoritmo para pagamento de comissão de vendedores de peças, levando-se em consideração que a comissão será de 5% do total da venda e que você tem os seguintes dados:

- Identificação do vendedor;
- Código da peça;
- Preço unitário da peça;
- Quantidade vendida;

2. Construa o diagrama de blocos do algoritmo desenvolvido no exercício anterior



Introdução à Lógica de Programação e Algoritmos

VARIÁVEIS E ATRIBUIÇÃO



Roteiro

- Estrutura básica de um programa
- Bloco de comentários
- Variáveis
- Constantes

Objetivos

- Entender a necessidade de se utilizar uma linguagem formal para construir algoritmos a serem interpretados por computadores.
- Compreender os conceitos de variáveis, constantes e tipos de dados.
- Conhecer os conceitos de comandos de atribuição, entrada e saída de dados.

Estrutura básica de um programa

- Utilizado para especificar o início e o fim do algoritmo em Portugol.

Sintaxe:

```
programa
```

```
{
```

```
    funcao inicio()
```

```
    {
```

```
        ...
```

```
    }
```

```
}
```



Código do Programa

Bloco de comentários

- Utilizado para explicar um determinado trecho do algoritmo, para torna-lo mais claro, facilitar o entendimento para outras pessoas.

Sintaxe:

programa

{

 // Função inicial do programa. Este é um comentário de apenas uma linha

funcao inicio()

 {

 /* Este é um bloco de comentário.

 Pode ocupar várias

 linhas */

 }

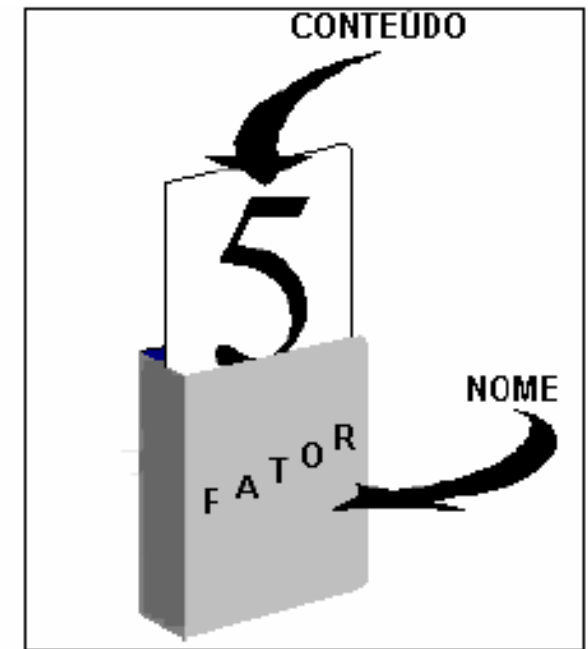
}

Variáveis

- Uma variável é um **espaço da memória do computador** que “*reservamos*” para guardar informações (dados).
- O conteúdo de uma variável pode ser alterado, consultado ou apagado quantas vezes forem necessárias durante o algoritmo. Mas, **ao alterar o conteúdo da variável, a informação anterior é perdida**, ou seja, **sempre "vale" a última informação armazenada na variável**.
- Embora uma variável possa conter valores diferentes, **ela só pode armazenar um valor a cada instante**.

Variáveis

- Toda variável é caracterizada por um **nome**, que a identifica em um algoritmo, e por um **tipo**, que determina o que pode ser armazenado naquela variável.
- Uma variável pode ser vista como uma caixa com um rótulo (**nome**) colado nela, que em um dado momento guarda um determinado **valor**.



Variáveis

- O nome dado à variável deve deixar claro o objetivo da mesma; devemos utilizar nomes sugestivos.
- Regras para nomear variáveis:

REGRA	EXEMPLO
Inicie sempre por um caractere alfabético, nunca por um número.	Nome (correto); 1nome (errado)
Não utilize acentos ou caracteres especiais como “ , () / * ; +	Nome(M); N*B; Salário
Não coloque espaços em branco ou hífen entre nomes.	salario bruto; salario-bruto
Utilize, se necessário, <i>underline</i> (_)	salario_bruto
Crie suas variáveis com nomes sugestivos.	Se vai guardar salário de funcionários, dê à variável o nome salario.

Declaração de Variáveis

- A forma de solicitar ao computador que reserve memória é chamada de **declaração de variáveis**.
- **Declaração de variáveis** corresponde a reserva de locais na memória rotulada com o **nome da variável** (identificador) e cujo **valor** (conteúdo) será interpretado conforme o **tipo declarado**.
- As variáveis só podem armazenar valores de um mesmo tipo.

Declaração de Variáveis

Uma variável pode ser de um dos seguintes tipos:

- **inteiro:** Declararemos variáveis do tipo numérico inteiro quando precisarmos armazenar valores inteiros, positivos ou negativos (0, 1, 5, 7, -10, -5, ...).
- **real:** Declararemos variáveis do tipo numérico real para armazenar valores reais; em outras palavras, valores com ponto decimal (5.7, 3.2, -8.5).
- **caracter:** Declararemos variáveis do tipo literal caractere para armazenar um único caractere, que pode ser uma letra ou um símbolo.
- **cadeia:** Declararemos variáveis do tipo literal cadeia para armazenar uma sequência de caracteres, ou seja, uma palavra, uma mensagem, um nome.
- **lógico:** Declararemos variáveis do tipo lógico para armazenar valores lógicos. O valor de variáveis desse tipo será sempre VERDADEIRO ou FALSO.

Declaração de Variáveis

Sintaxe:

```
tipo_da_variável nome_da_variável
```

nome_da_variável: As variáveis podem ser entendidas como sendo apelidos para as posições de memória.

tipo_da_variável: Precisamos informar o tipo de dados que armazenaremos na variável para que o computador saiba o tamanho do espaço de memória que reservará.

Exemplos:

```
caracter Sexo
```

```
inteiro numero_filhos
```

```
real Altura
```

```
cadeia Nome
```

```
logico aprovado
```

Constantes

- Constantes são informações (dados) que não variam com o tempo, ou seja, permanecem sempre com o mesmo conteúdo, é um valor fixo (invariável).
- As constantes são criadas obedecendo às mesmas regras de nomenclatura já vistas em variáveis.
- É uma variável com valor pré-definido que não pode ser modificado por nenhuma função de um programa.

Declaração de Constante

Sintaxe:

```
const tipo_da_constante nome_da_constante = valor_da_constante
```

const: A palavra reservada **const** é utilizada para indicar ao programa que o identificado refere-se a uma constante.

nome_da_constante: As constante podem ser entendidas como sendo apelidos para as posições de memória.

tipo_da_constante: Precisamos informar o tipo de dados que armazenaremos na constante para que o computador saiba o tamanho do espaço de memória que reservará.

Exemplo:

```
const real ACELERACAO_GRAVIDADE = 9.78
```


Atribuição

- A atribuição é a notação utilizada para colocar um valor em uma variável.
- Atribuir um valor de uma expressão a uma variável significa calcular o valor da expressão e, então, copiar o resultado para a variável.
- O sinal de igual “ = ” ou “ <- ” são utilizados como operador de atribuição (lê-se “recebe”).
- Erros comuns nas atribuições:
 - Incompatibilidades de tipos - Tentar atribuir um tipo de dado a uma variável de outro tipo;
 - Perda de Precisão - Atribuir valores reais a variáveis do tipo inteiro, perdendo a parte decimal do número;
 - Atribuição de valor às variáveis não declaradas.

Atribuição

Sintaxe:

```
identificador = expressão
```

identificador: Nome da variável ou constante a ser utilizada.

nome_da_constante: Valor ou expressão a ser armazenado (podemos ter expressões aritméticas e expressões lógicas).

Exemplo:

```
const real PI = 3.1415
```

```
inteiro nota
```

```
nota = 10
```

Atribuição - Exercícios


Assinale os comandos de atribuição realizados corretamente:

- a) ☐ cadeia SEXO = 'F'
- b) ☐ inteiro ALTURA = 1.80
- c) ☐ real SALÁRIO = 3000.00
- d) ☐ cadeia = "NOME"

Atribuição - Exercícios

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

- | | |
|---------------------------------------|---------------------------------------|
| a) () valor = 2.5 tipo= real | f) () valor = -10.35 tipo= real |
| b) () valor = 'F' tipo= inteiro | g) () valor = 38 tipo= real |
| c) () valor = -2 tipo= inteiro | h) () valor = 'Jose' tipo= cadeia |
| d) () valor = 'M' tipo= caractere | i) () valor = 135 tipo= inteiro |
| e) () valor = 5 tipo= cadeia | |



Introdução à Lógica de Programação e Algoritmos

COMANDO DE ENTRADA E SAÍDA



Roteiro

- Saída de Dados
- Entrada de Dados



Objetivos

- Conhecer os comandos de entrada e saída de dados.

Comando de Entrada e Saída

- Comandos de Entrada e saída são instruções que nos permitem, ou receber dados (Entrada) ou informar dados (Saída).
- O padrão de Entrada para os computadores básicos é o teclado, é dele que nós informamos o que o programa nos solicita.
- O padrão de Saída é o monitor, todas as informações mostradas ao usuário serão passadas por este dispositivo de saída.

Comando de Saída

O comando de saída de dados exibe no monitor valores de constantes, variáveis ou expressões.

Sintaxe:

```
escreva(expressão)
```

escreva(): Função responsável por escrever no monitor uma mensagem para o usuário.

expressão: Indica o que será escrito no monitor. É normalmente composta por um texto fixo seguido por uma vírgula e um nome de variável.

Exemplo:

```
escreva("Olá Mundo!")
```

Comando de Entrada

O comando de entrada de dados será responsável pela leitura e armazenamento desses dados na variável que indicarmos.

Sintaxe:

```
leia(nome_da_variável)
```

leia(): Função responsável por ler o que o usuário digitou e armazenar o valor na variável indicada.

nome_da_variável: Nome da variável utilizada para armazenar o valor digitado.

Exemplo:

```
inteiro nota
```

```
leia(nota)
```

Comando de Entrada e Saída

Primeiro exemplo de algoritmo completo em Portugol:

```
programa
{
    funcao inicio ()
    {
        /* criar a variável nome do tipo cadeia */
        cadeia nome
        /* solicitar que o usuário digite seu nome */
        escreva("Digite seu nome: ")
        /* ler para a variável nome o valor digitado pelo usuário */
        leia(nome)
        /* Imprimir na tela a mensagem "Bom dia" acompanhada pelo
        valor digitado pelo usuário */
        escreva("Bom dia ", nome)
    }
}
```

Comando de Entrada e Saída - Exercícios

1) Faça um algoritmo que solicite que o usuário digite seu nome e a seguir solicite que seja digitada sua idade. Depois que o usuário digitar o nome e a idade, o programa deve exibir na tela duas mensagens: uma com o nome e outra com a idade do usuário.

Suponha que o usuário seja o Pedro e tenha 32 anos. Assim, após a digitação dos dados, seu programa deve exibir as seguintes mensagens:

“Seu nome é Pedro”

“Você tem 32 anos”.


Comando de Entrada e Saída - Exercícios

- 2) Faça um algoritmo que solicite que leia um número inteiro e mostre uma mensagem indicando se este número é par ou ímpar, e se é positivo ou negativo.
- 3) Faça um algoritmo que solicite que leia quatro números inteiros e encontre a média aritmética simples entre as que correspondem a números pares. Lembre-se que não pode haver divisão por zero.
- 4) Faça um algoritmo que solicite que leia 4 notas, calcule a média e escreva:
 - Reprovado (média < 5)
 - Recuperação (média ≥ 5 e < 7)
 - Aprovado (média ≥ 7)

Atribuição - Exercícios

Assinale os comandos de atribuição realizados corretamente:

- a) ☐ cadeia SEXO = 'F'
- b) ☐ inteiro ALTURA = 1.80
- c) ☐ real SALÁRIO = 3000.00
- d) ☐ cadeia = "NOME"



Introdução à Lógica de Programação e Algoritmos

OPERAÇÕES ARITMÉTICAS



Roteiro

- Operadores aritméticos
- Expressões aritméticas



Objetivos

- Conhecer os operadores aritméticos e a ordem de precedência.

Operadores Aritméticos

- Os operadores aritméticos são símbolos que representam operações aritméticas, ou seja, as operações matemáticas básicas.

OPERADOR	OPERAÇÃO MATEMÁTICA
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão

Expressões Aritméticas

- Os operadores aritméticos são utilizados para formar expressões aritméticas.
- As expressões aritméticas são formadas por operadores aritméticos que agem sobre operandos.
- Os operandos podem ser variáveis ou constantes do tipo numérico, ou seja, inteiros ou reais.

Exemplo:

Nota / 2

$x * 2 + y / 2$

Expressões Aritméticas

- Para resolver expressões aritméticas formadas por mais de um operador, deve-se utilizar uma ordem de precedência (ordem dos cálculos).

PRIORIDADE	OPERADOR	OPERAÇÃO
1º	()	Parêntesis
2º	* / %	multiplicação, divisão, resto da divisão
3º	+ -	Adição, Subtração

- Resolvemos primeiro as expressões contidas nos parênteses mais internos, seguindo a ordem de precedência entre operadores, passando depois para os parênteses mais externos

Expressões Aritméticas

Exemplo de expressão:

$$\text{nota1} + (\text{nota2} + \text{nota3}) / 2$$

Primeiro somamos nota2 a nota3. O resultado é dividido por 2 e só depois somamos com nota1.

Operações Aritméticas

Algoritmo para ler e multiplicar dois inteiros e exibir o resultado:

```
01. programa {
02.     funcao inicio() {
03.         inteiro num1, num2, mult
04.         escreva("Digite o primeiro número: ")
05.         leia(num1)
06.         escreva("Digite o segundo número: ")
07.         leia(num2)
08.         mult = num1 * num2
09.         escreva("O resultado da multiplicação é: ", mult)
10.     }
11. }
```


Operações Aritméticas

Vamos entender todas as linhas do nosso algoritmo:

01. Início do programa.
02. Executa a função `inicio()`.
03. Declaração das três variáveis do tipo `inteiro` necessárias ao programa.
04. O comando `escreva` exibirá na tela uma mensagem que solicita a digitação do primeiro número.
05. O primeiro número digitado será lido e armazenado na variável `num1`.
06. O comando `escreva` exibirá na tela uma mensagem que solicita a digitação do segundo número.
07. O segundo número digitado será lido e armazenado na variável `num2`.
08. A variável `mult` receberá o resultado da multiplicação de `num1` pelo `num2`.
09. O comando `escreva` exibirá na tela uma mensagem com o resultado da multiplicação.
10. Encerra a função `inicio()`.
11. Encerra o programa.

Operações Aritméticas - Exercícios

- 1) Resolva as seguintes expressões aritméticas considerando $A=2$, $B=5$ e $C=10$:
 - a) $A+B*C/A$
 - b) $B+C\%A*(B-A/2)$
 - c) $(B+C)\%2+A*(B+(C*4))$
- 2) Faça um algoritmo que leia um número inteiro e imprima seu antecessor e seu sucessor.
- 3) Faça um algoritmo que leia dois números reais e imprima a soma e a média aritmética desses números.
- 4) Faça um algoritmo que receba como entrada as medidas dos dois catetos de um triângulo retângulo e calcule e exiba a medida da hipotenusa e a área do triângulo.



Introdução à Lógica de Programação e Algoritmos

OPERAÇÕES RELACIONAIS, EXPRESSÕES
LÓGICAS E CONDICIONAIS

Roteiro

- Operadores relacionais
- Expressões lógicas
- Comandos condicionais
 - Escolha
 - Se
 - Se-Senão
 - Se-Senão-Se

Objetivos

- Conhecer os operadores relacionais.
- Conhecer os operadores lógicos e suas Tabelas-verdade.
- Compreender a ordem de precedência entre os operadores em expressões lógicas.
- Conhecer a formalização de uma estrutura de decisão no contexto de um algoritmo em Portugol.
- Construir algoritmos em Portugol com estruturas de decisão.

Operadores relacionais

- Os operadores relacionais são utilizados para comparar **cadeias de caracteres** e **números**. Os valores a serem comparados podem ser **caracteres** ou **variáveis**.
- Estes operadores sempre retornam **valores lógicos** (verdadeiro ou falso).

SÍMBOLO	DESCRIÇÃO
==	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Operadores relacionais

Se $A = 5$ e $B = 3$ então:

EXPRESSÕES	RESULTADO
$A == B$	
$A != B$	
$A > B$	
$A < B$	
$A >= B$	
$A <= B$	
$A >= (B * 3 - 4)$	

Operadores relacionais

Se $A = 5$ e $B = 3$ então:

EXPRESSÕES	RESULTADO
$A == B$	FALSO
$A != B$	VERDADEIRO
$A > B$	VERDADEIRO
$A < B$	FALSO
$A >= B$	VERDADEIRO
$A <= B$	FALSO
$A >= (B * 3 - 4)$	VERDADEIRO

Operadores lógicos

- Os operadores lógicos retornam **verdadeiro (V)** ou **falso (F)** de acordo com seus operandos.
- Os operadores lógicos também são conhecidos como **conectivos**, pois são utilizados para formar novas proposições a partir da junção de duas outras.

SÍMBOLO	DESCRIÇÃO
E	Verifica se ambos os operandos são verdadeiros. Se sim, ele retorna verdadeiro , senão, retorna um valor falso .
OU	Verifica se um dos valores testados é verdadeiro. Se sim, ele retorna verdadeiro , senão, retorna um valor falso .
NÃO	É usado para negar o valor de uma variável ou expressão

Operadores lógicos

Se $A = 5$, $B = 8$ e $C = 1$ então:

EXPRESSÕES			RESULTADO
$A == B$	E	$B > C$	
$A != B$	OU	$B < C$	
	NÃO	$A > B$	
$A < B$	E	$B > C$	
$A >= B$	OU	$B == C$	
	NÃO	$A <= B$	
$A == B$	OU	$B > C$	
$(A * C) <= B$	E	$(B \% A) > C$	

Operadores lógicos

Você deve ter notado, pelos exemplos anteriores:

- quando utilizamos o operador lógico **E**, o resultado só será verdadeiro se as duas condições relacionadas forem verdadeiras.
- para o operador **OU**, basta que uma das condições seja verdadeira para que o resultado seja verdadeiro.
- em consequência: com o operador **OU**, para que o resultado seja falso as duas condições devem ser falsas.

Operadores lógicos

Utilizamos as **Tabelas-verdade** para visualizar todas as opções possíveis ao combinar operadores lógicos.

Operador OU		
P	Q	P OU Q
V	V	V
V	F	V
F	V	V
F	F	F

Operador E		
P	Q	P E Q
V	V	V
V	F	F
F	V	F
F	F	F

Operador Não	
P	Não P
V	F
F	V

Expressões lógicas

- As expressões lógicas são expressões formadas a partir do uso de variáveis e constantes, operadores relacionais e operadores lógicos.
- As expressões lógicas são avaliadas e retornam sempre um valor lógico: **verdadeiro ou falso**.

Ordem de precedência entre os operadores:

PRIORIDADE	OPERAÇÃO
1º	Operadores aritméticos
2 º	Operadores relacionais
3 º	Operador lógico NÃO
4 º	Operador lógico E
5 º	Operador lógico OU

Expressões lógicas

Se $A = 5$, $B = 8$ e $C = 3$ então:

EXPRESSÕES			RESULTADO
$A * C > B$	E	$B \% A < C$	
$B - A * 2 > (A + C) / 8$	OU	$A + C = B$	
NÃO $A > B$	E	$B \% C > C$	

Comando condicional

- Nos algoritmos encontramos situações onde um conjunto de instruções deve ser executado caso uma condição seja verdadeira.
 - Por exemplo: a seleção brasileira de futebol só participa de uma copa do mundo se for classificada nas eliminatórias.
- Um comando condicional é aquele que permite decidir se um determinado bloco de comandos deve ou não ser executado, a partir do resultado de uma expressão relacional ou lógica.
- Serão abordados os comandos:
 - escolha-caso
 - se
 - se-senao
 - se-senao-se

Comando condicional - escolha-caso

Sintaxe:

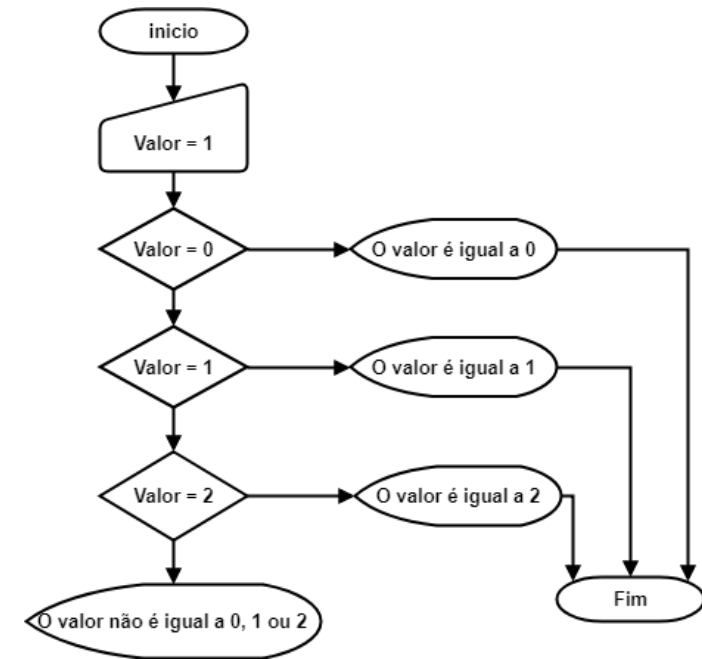
```
escolha (<teste>)  
{  
    caso <op-1>:  
        <bloco de comandos>  
    pare  
    ...  
    caso <op-n>:  
        <bloco de comandos>  
    pare  
    caso contrario:  
        <bloco de comandos>  
}
```

- Neste comando não é possível o uso de operadores lógicos, ele apenas trabalha com valores definidos. O **escolha** e o **caso** tem alguns casos testes, e se a instrução **pare** não for colocada ao fim de cada um destes testes, o comando executará todos casos existentes.

Comando condicional - escolha-caso

Exemplo:

```
programa
{
    funcao inicio()
    {
        inteiro valor=1
        escolha (valor)
        {
            caso 0:
            escreva ("o valor é igual a 0")
            pare
            caso 1:
            escreva ("o valor é igual a 1")
            pare
            caso 2:
            escreva ("o valor é igual a 2")
            pare
            caso contrario:
            escreva ("o valor não é igual a 0, 1 ou 2")
        }
    }
}
```



Comando condicional - se

Sintaxe:

```
se(<teste>)  
{  
    <bloco de comandos>  
}
```

- Se o **teste** lógico resultar **verdadeiro**, as instruções definidas dentro do desvio condicional serão executadas. Se o **teste** for **falso**, o algoritmo pulará o trecho e continuará sua execução a partir do ponto onde o desvio condicional foi finalizado.

Comando condicionais - escolha-caso

Exemplo:

```
programa
```

```
{
```

```
    funcao inicio()
```

```
    {
```

```
        inteiro num
```

```
        escreva ("Digite um número: ")
```

```
        leia (num)
```

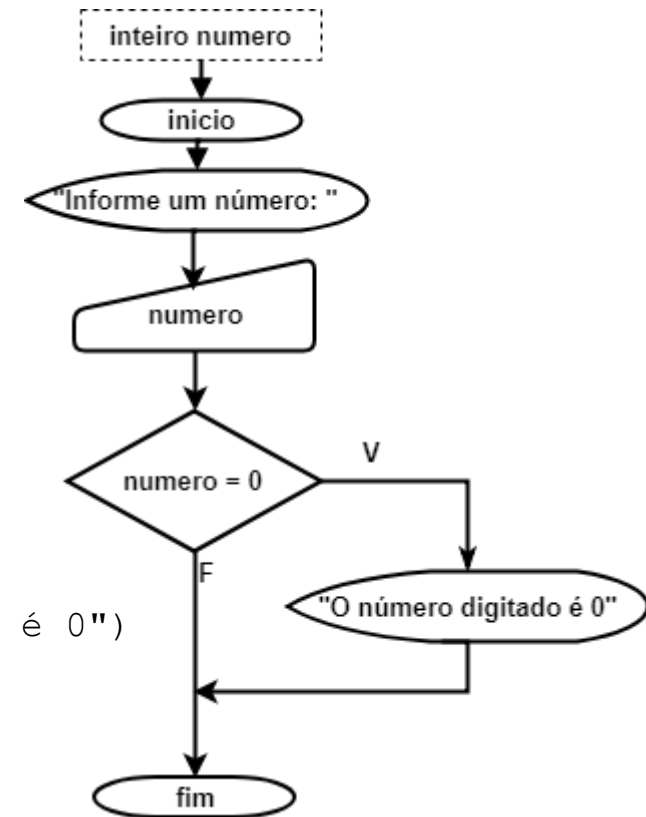
```
        se (num==0) {
```

```
            escreva ("O número digitado é 0")
```

```
        }
```

```
    }
```

```
}
```



Comando condicional – se-senão

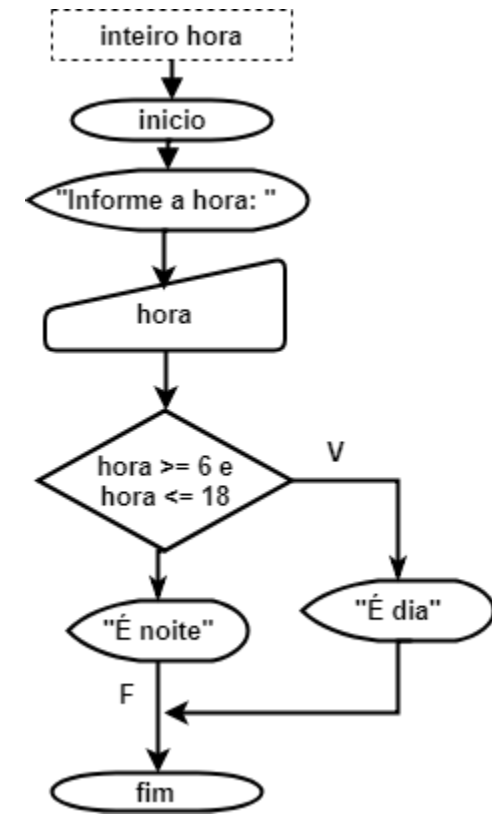
Sintaxe:

```
se (<teste>)  
{  
    <bloco de comandos> //se o desvio for verdadeiro  
}  
senao  
{  
    <bloco de comandos> //se o desvio for falso  
}
```

Comando condicional – se-senão

Exemplo:

```
programa
{
    funcao inicio()
    {
        inteiro hora
        escreva ("Digite a hora: ")
        leia (hora)
        se (hora >= 6 e hora <= 18) {
            escreva ("É dia")
        } senao {
            escreva ("É noite")
        }
    }
}
```



Comando condicional – se-senao-se

Sintaxe:

```
se (<teste>)  
{  
    <bloco de comandos> //se o teste for verdadeiro  
}  
senao se (<teste-2>)  
{  
    <bloco de comandos> //se o teste-2 for verdadeiro  
}  
senao  
{  
    <bloco de comandos> //se o teste e o teste-2 forem falsos  
}
```

Comando condicional – se-senão-se

Exemplo de desempenho do aluno:


```
programa {  
    funcao inicio() {  
        real nota  
        leia(nota)  
        se(nota >= 9) {  
            escreva("O aluno foi um desempenho muito bom na prova")  
        } senao se (nota >= 7) {  
            escreva("O aluno teve um desempenho bom na prova")  
        } senao se (nota >= 6) {  
            escreva("O aluno teve um desempenho razoável na prova")  
        } senao {  
            escreva("O aluno teve um desempenho mau na prova")  
        }  
    }  
}
```

Comando condicional - Exercícios

- 1) Faça um algoritmo que leia a idade de um atleta e escreva na tela em que categoria ele se enquadra, seguindo o quadro abaixo:

Faixa Etária	Categoria
De 5 a 10 anos	Infantil
De 11 a 17 anos	Juvenil
De 18 a 30 anos	Profissional
Acima de 30 anos	Sênior

- 2) Escreva um algoritmo que leia um número inteiro e diga:
- Se ele é par ou ímpar. Dica: utilize o operador % (resto da divisão inteira).
 - Se ele é positivo, negativo ou nulo (zero).



Introdução à Lógica de Programação e Algoritmos

ESTRUTURAS DE REPETIÇÃO



Roteiro

- Estruturas de Repetição
 - Enquanto
 - Faça-Enquanto
 - Faça

Objetivos

- Compreender a utilização de estruturas de repetição em algoritmos.
- Conhecer dois tipos de estruturas de repetição em Portugol e avaliar quando utilizar cada uma.
- Construir algoritmos em Portugol com estruturas de repetição.

Estruturas de repetição

- Até agora, vimos como escrever programas capazes de executar comandos de forma linear e de tomar decisões com relação a executar ou não um bloco de comandos.
- No entanto, ao desenvolver nossos algoritmos, deparamos com situações nas quais precisamos repetir um bloco de comandos até que uma determinada condição ocorra.
- Nessas situações, utilizaremos os **comandos de repetição**, também conhecidos como **laços** ou **loops**.

Estruturas de repetição

- Um laço é uma técnica de programação onde um bloco de comandos se repete varias vezes até que uma determinada condição seja verdadeira.
- Estudaremos 3 tipos de laços: pré-testado, pós-testado e laço com variável de controle.
 - Enquanto
 - Faça-Enquanto
 - Faça

Estruturas de repetição - Enquanto

Sintaxe:

```
enquanto (<teste>)  
{  
    <bloco de comandos> //Executa enquanto o <teste> for verdadeiro  
}
```

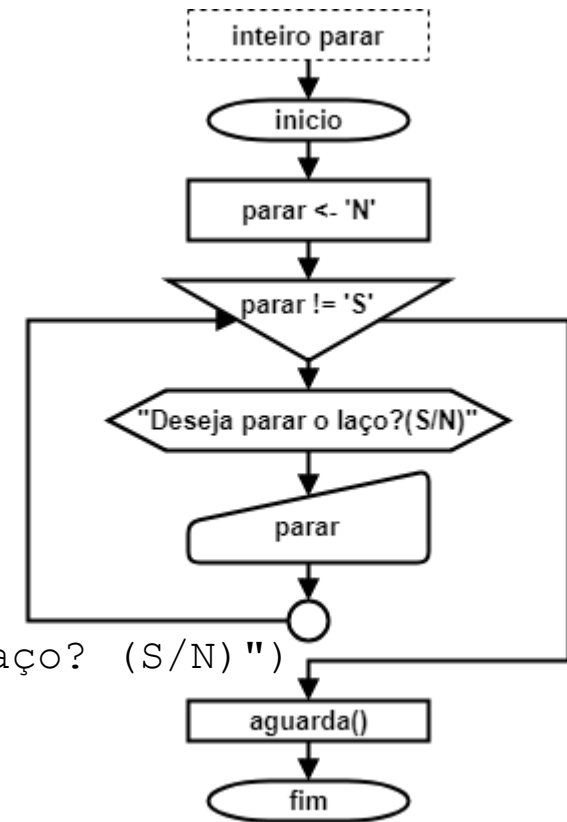
- A função do comando **enquanto** é executar o bloco de comandos enquanto uma determinada condição for verdadeira.
- Note que a condição é avaliada antes da execução do bloco de comando (pré-testado).

Estruturas de repetição - Enquanto

Exemplo de utilização do laço:

```
programa
{
    funcao inicio()
    {
        caracter parar
        parar = 'N'

        enquanto (parar != 'S')
        {
            escreva ("deseja parar o laço? (S/N) ")
            leia (parar)
        }
    }
}
```



Estruturas de repetição – Faça-Enquanto

Sintaxe:

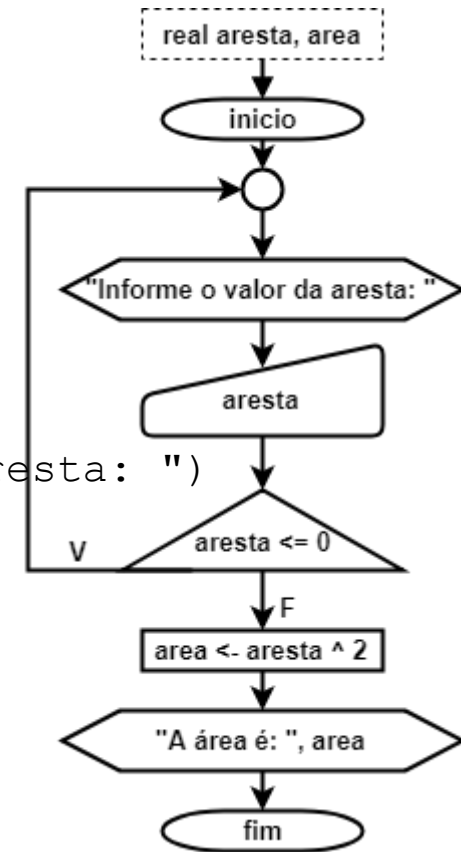
```
faça  
{  
    <bloco de comandos> // O bloco de comandos é executado pelo menos  
                        // uma vez e continua enquanto o <teste> for  
                        // verdadeiro  
} enquanto (<teste>)
```

- Este teste é bem parecido com o enquanto. A diferença está no teste que é realizado no final. O bloco de comando será executado pelo menos uma vez. O teste verifica o laço deve ser repetido ou não.
- Neste caso, a condição é avaliada após a execução do bloco de comando (pós-testado).

Estruturas de repetição - Faça-Enquanto

Exemplo de cálculo da área do quadrado:

```
programa
{
    funcao inicio()
    {
        real aresta, area
        faca
        {
            escreva ("Informe o valor da aresta: ")
            leia (aresta)
        } enquanto (aresta <= 0)
        area = aresta * aresta
        escreva("A área é: ", area)
    }
}
```



Estruturas de repetição – Para

Sintaxe:

```
para (inteiro i=0; i<8; i++)  
{  
    <bloco de comandos> // Bloco a ser executado enquanto a condição for  
                        // satisfeita  
}
```

- O laço de repetição com variável de controle facilita a construção de algoritmos com número definido de repetições, pois possui um contador (variável de controle) embutido no comando com o incremento automático.
- O laço com variável de controle possui três partes. A inicialização da variável contadora, a definição do valor final do contador e a definição do incremento. Estas três partes são escritas juntas, no início do laço.

Estruturas de repetição - Para

Exemplo da tabuada de 3:

```
programa
```

```
{
```

```
    funcao inicio()
```

```
    {
```

```
        inteiro tab
```

```
        para (inteiro c=1; c<=10; c++)
```

```
        {
```

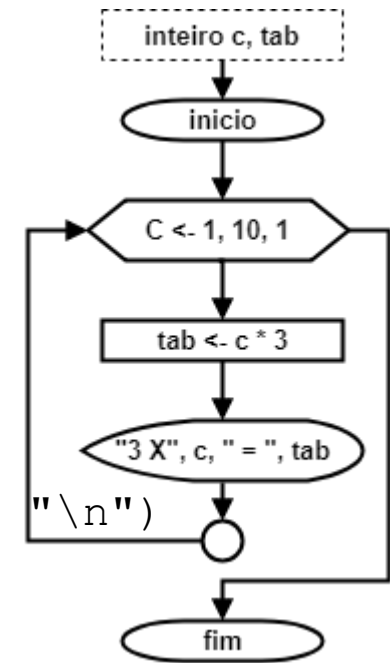
```
            tab = c * 3
```

```
            escreva ("3 x ", c, " = ", tab, "\n")
```

```
        }
```


```
    }
```

```
}
```



Estruturas de repetição - Exercícios

- 1) Faça um programa que calcule a média dos números digitados. O algoritmo deve ficar lendo números inteiros até que o número zero seja informado. Quando o número zero for informado, o algoritmo deve exibir na tela a quantidade de números digitados (contando inclusive com o zero digitado) e a média dos valores digitados.
- 2) Faça um programa que solicite a digitação da idade e do sexo de uma pessoa (o sexo deve ser F ou M) e depois pergunte se o usuário deseja informar uma nova pessoa. Esse processo deve se repetir até que o usuário informe que não deseja mais informar novas pessoas. Quando isso acontecer, o algoritmo deve imprimir na tela a quantidade de pessoas do sexo masculino informadas; a quantidade de pessoas do sexo feminino informadas; a média das idades informadas para pessoas de sexo masculino; e a média das idades informadas para pessoas de sexo feminino.



Introdução à Lógica de Programação e Algoritmos

PROCEDIMENTOS E FUNÇÕES



Roteiro

- Procedimentos e Funções
- Variáveis locais e globais

Objetivos

- Compreender a utilização de procedimentos e funções.
- Compreender o escopo de variáveis e avaliar quando utilizar cada uma.
- Construir algoritmos em Portugol com procedimentos e funções.

Procedimentos e Funções

- Procedimentos são estruturas que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado. Não retornam valor.

```
escreva ("escreva é um exemplo de procedimento. \n")
```

- Funções são procedimentos que retornam um único valor ao final de sua execução.

```
raiz_quadrada = Matematica.raiz (4, 2)
```

Por que utilizar procedimentos e funções

- Evitar que os blocos do programa fiquem grandes demais e, por consequência, mais difíceis de ler e entender.
- Separar o programa em partes que possam ser logicamente compreendidos de forma isolada.
- Permitir o reaproveitamento de código já construído (por você ou por outros programadores).
- Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, minimizando erros e facilitando alterações.
- Melhorar a interpretação visual de um programa.

Componentes de procedimentos e funções

- O seu protótipo, que inclui os parâmetros que são passados à função.
 - Os parâmetros se comportando como variáveis que são inicializadas quando o procedimento ou função é chamado.
 - Um procedimento ou função pode não ter parâmetros, para isso basta não informá-los.
- O corpo, que contém o bloco de código que resolve o problema proposto.
- Um possível valor de retorno.
- A declaração de um procedimento ou função deve vir sempre antes do seu uso.
- Os procedimento e funções só podem ser declaradas fora de outros procedimentos ou funções. Lembre-se que o corpo do programa principal (`inicio()`) é um procedimento.

Componentes de procedimentos e funções

Sintaxe:

```
funcao nome_do_procedimento (tipo_do_parametro nome_do_parâmetro)
{
    <bloco de comandos>
}
```

```
funcao tipo_do_retorno nome_da_funcao (tipo_do_parametro nome_do_parametro)
{
    <bloco de comandos>

    retorne <valor_do_retorno>
}
```

Procedimentos e Funções

Exemplo de criação e utilização de procedimento:

```
programa
{
    funcao escreva_cabecalho (cadeia nome_do_programa)
    {
        escreva ("-----+\n")
        escreva ("|", nome_do_programa, "\n")
        escreva ("-----+\n")
    }

    funcao inicio()
    {
        // invocando o procedimento escreva_cabecalho
        escreva_cabecalho ("Exemplo de utilização de procedimento")
    }
}
```

Procedimentos e Funções

```
programa {  
  
    funcao real media_ponderada ( inteiro m1, inteiro m2, inteiro m3 ) {  
        retorne (m1 * 2 + m2 * 3 + m3 * 8) / 13.0  
    }  
  
    funcao logico verifica_par ( inteiro num ) {  
        se (num % 2 != 0) {  
            retorne falso  
        }  
        retorne verdadeiro  
    }  
  
    funcao inicio()  
    {  
        inteiro num = 3  
  
        // invocando a função media_ponderada  
        escreva ( media_ponderada ( 4, 9, 8 ), "\n")  
  
        escreva ( num, " é par? ", verifica_par ( num ), , "\n")  
    }  
}
```

Variáveis Locais e Globais

- Uma variável é chamada **local** se ela foi declarada dentro de uma função. Nesse caso, ela existe somente dentro daquela função e após o término da execução da mesma, a variável deixa de existir.
- Uma variável é chamada **global** se ela for declarada fora de qualquer função. Essa variável é visível em todas as funções, ou seja, qualquer função pode alterá-la e ela existe durante toda a execução do programa.
- É possível declarar variáveis **locais** com o mesmo nome de variáveis **globais**. Nesta situação, a variável **local** sobrescreve (“esconde”) a variável **global**.
- As partes do código onde as variáveis podem ser acessadas é chamado de **escopo da variável**.

Variáveis Locais e Globais

```
programa {  
    // Constante e variável globais  
    const real PI = 3.1415  
    real raio  
  
    funcao real area_do_circulo ( real raio ) {  
        real area                // variável local  
        area = PI * (raio * raio)  
        retorne area  
    }  
  
    funcao real perimetro_do_circulo ( real raio ) {  
        real perimetro          // variável local  
        perimetro = 2 * PI * raio  
        retorne perimetro  
    }  
  
    funcao inicio() {  
        raio = 10.00  
  
        escreva ( "o circulo de raio ", raio, " possui\n")  
        escreva ( "\tperimetro de: ", perimetro_do_circulo (raio), "\n")  
        escreva ( "\tárea de: ", area_do_circulo(raio) )  
    }  
}
```

Procedimentos e Funções - Exercícios

- 1) Faça um programa para identificar se um número é primo. Lembre-se que número primo, é um número natural, maior que 1, apenas divisível por si próprio e por um.
- 2) Desafio - Faça um programa para somar os dígitos de um inteiro sem utilizar recursos da Biblioteca Texto, ou seja, trabalhe apenas com tipos numéricos.

Somar dígitos significa que dados um número qualquer, exemplo, 2019, devemos somar seus dígitos:

$$2 + 0 + 1 + 9 = 12$$