

Introdução Lógica de Programação

Com VisualG



VisualG

Sumário

Introdução -----	03
Introdução ao VisualG - Algoritmo -----	03
Tipo de Dados - Fases de um Algoritmo -----	04 - 05
Resumo -----	06
Variáveis de Memória em Programas -----	07 - 09
Operadores Aritméticos -----	09
Concatenação -----	10
Tela de entrada do VisualG -----	11
Entrada Processamento e Saída -----	12 - 30
Fluxograma -----	13 - 16
Botão passo a passo VisualG - Formatação de números -----	21
Operadores Relacionais - Operadores Lógicos -----	31 - 32
Estruturas Condicionais -----	33 - 40
Estrutura Condicional Simples -----	33 - 34
Estrutura Condicional Composta -----	35 - 36
Estrutura Condicional Aninhada -----	37 - 38
Estrutura Condicional de Escolha (Escolha Caso) -----	39 - 40
Estrutura de Repetição -----	41 - 49
Estrutura de Repetição Enquanto -----	41 - 44
Estrutura Repita -----	44 - 46
Estrutura de Repetição PARA -----	47 - 49
Aula Prática: Funções no VisualG -----	49 - 53
Aula Prática sobre Array e Vetor em Visualg -----	54 - 56
Aula Prática de Matrizes em Visualg -----	57 - 58

Introdução Lógica de Programação

Introdução ao Visualg

O Visualg é um ambiente de desenvolvimento que permite a execução e simulação de algoritmos escritos em uma linguagem chamada Portugol. Ele é amplamente utilizado em ambientes educacionais para ensinar lógica de programação.

A lógica na programação de computadores é essencial para o desenvolvimento de softwares eficientes e funcionais. Ela envolve o uso de raciocínio lógico para resolver problemas e criar soluções que podem ser implementadas em código.

Download VISUALG: <https://sourceforge.net/projects/visualg30/>

Algoritmo

Um algoritmo é um conjunto finito de instruções ou passos definidos que resolvem um problema ou realizam uma tarefa específica. Ele serve como um roteiro para o programador, orientando o desenvolvimento do código.

Fases de um Algoritmo

1. **Definição do Problema:** Entender claramente o problema que precisa ser resolvido.
2. **Planejamento:** Elaborar um plano para criar o algoritmo, muitas vezes usando diagramas de fluxo ou pseudocódigo.
3. **Desenvolvimento:** Escrever o algoritmo em um formato compreensível e detalhado.
4. **Teste:** Executar o algoritmo com diferentes dados de entrada para garantir que funcione conforme esperado.
5. **Depuração:** Identificar e corrigir quaisquer erros encontrados durante os testes.

Tipos de Dados

Os tipos de dados são categorias de dados que informam ao compilador ou interpretador como o programador pretende usar os dados. Os tipos de dados comuns incluem:

- A. **Inteiros:** Números inteiros, positivos ou negativos.
- B. **Número Real:** Números com casas decimais.
- C. **Caractere:** Letras, números ou símbolos.
- D. **Booleano:** Valores verdadeiros ou falsos.
- E. **Cadeia de Caracteres:** Sequências de caracteres, como palavras ou frases.

Fases de um Algoritmo

Um algoritmo é uma sequência de passos finitos e bem definidos que visam solucionar um problema ou realizar uma tarefa. Em geral, um algoritmo pode ser dividido em três fases principais: **entrada de dados**, **processamento de dados** e **saída de dados**. A seguir, vamos explorar cada uma dessas etapas em detalhes.

Entrada de Dados

A fase de entrada de dados é o ponto inicial do algoritmo. Nesta etapa, são coletadas todas as informações necessárias para que o algoritmo possa realizar seu processamento. A qualidade e a precisão dos dados de entrada são cruciais para o sucesso do algoritmo. Exemplos de entrada de dados incluem:

- **Usuário:** Informações digitadas por um usuário em um formulário.
- **Sensores:** Dados coletados por dispositivos de medição.
- **Arquivos:** Dados lidos de arquivos de texto ou bancos de dados.

Processamento de Dados

Na fase de processamento, o algoritmo executa operações sobre os dados de entrada para transformá-los em informações úteis. Este é o núcleo do algoritmo, onde a lógica e as regras definidas são aplicadas. Durante o processamento, podem ocorrer operações como:

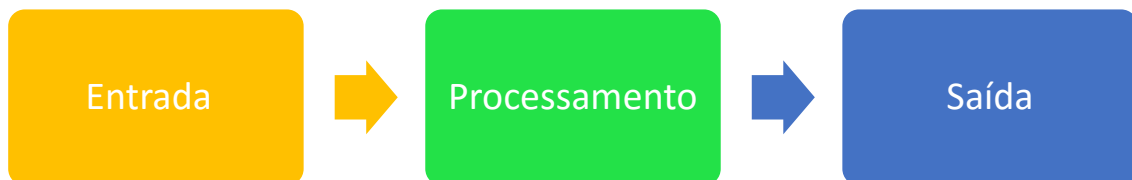
- **Cálculos Matemáticos:** Realizar operações aritméticas.
- **Classificação:** Organizar dados em uma ordem específica.
- **Filtragem:** Selecionar apenas os dados que atendem a determinados critérios.
- **Transformação:** Converter dados de um formato para outro.

Saída de Dados

A fase final de um algoritmo é a saída de dados, onde os resultados do processamento são apresentados de forma compreensível. A saída pode ser direta, como uma resposta exibida na tela, ou indireta, como o armazenamento dos resultados em um arquivo. Exemplos de saídas incluem:

- **Exibição em Tela:** Resultados mostrados ao usuário.
- **Impressão:** Informações impressas em papel.
- **Armazenamento:** Dados salvos em um banco de dados ou arquivo.

Essas três fases são essenciais para o funcionamento eficaz de qualquer algoritmo e garantem que o problema proposto seja resolvido de maneira eficiente e precisa.



Resumo

Entrada	Processamento	Saída
8	Somar 8 + 2	10
2		

Numérico

Inteiro
(-100, 10, 0)

Real
(1.25, 56.79, -50.35)

Caractere

Letras, Dígitos, Espaço
em Branco, Símbolos
Especiais

Lógico

Verdadeiro
Sim

Falso
Não

Variáveis de Memória em Programas

As variáveis de memória desempenham um papel crucial no funcionamento dos programas de computador. Elas são utilizadas para armazenar dados temporários enquanto um programa está sendo executado.

O Que São Variáveis de Memória?

Variáveis de memória são locais na memória do computador onde os dados são armazenados para serem utilizados por um programa. Cada variável tem um nome, um tipo de dado associado (como inteiro, caractere ou booleano), e um valor que pode ser alterado durante a execução do programa.

Declaração de Variáveis de Entrada

A declaração de variáveis de entrada é o processo de definir quais dados o algoritmo precisará receber para executar sua função. Essas variáveis geralmente são definidas no início do algoritmo, e seu tipo (como inteiro, ponto flutuante, texto, etc.) deve ser especificado.

Exemplos

Declaração de Variáveis de Entrada

Entrada:

Inteiro : idade

caractere : nome

real : altura

Declaração e Inicialização de Variáveis

A declaração é a fase na qual se define uma variável; já a inicialização é quando se atribui um valor inicial a ela. A inicialização pode ocorrer no momento da declaração ou posteriormente no algoritmo.

Exemplos

Declaração e Inicialização

inteiro contador = 0

real preco = 19.99

caractere mensagem = "Bem-vindo!"

Atribuição de Valores

As variáveis podem ter seus valores alterados durante a execução do programa por meio de operações de atribuição. No VisualG usamos o operador de atribuição (<-).

Exemplo:

Idade <- 30 - A variável idade recebe ou passa a valer o valor de 30.

Resumo

Variável
A <- 5
B <- 4
C <- A + B

Onde temos: variável A recebe o valor de 5, do tipo inteiro, variável B recebe o valor de 4 do tipo inteiro. E então as variáveis vão ser armazenada cada uma na sua respectiva célula. E por fim a variável C, recebe a soma de A + B.

Regras para Nomeação de Variáveis

- Utilizar letras, números e Underscore ou sublinhado.
- Não utilizar caracteres especiais, acentos ou símbolos.
- O Primeiro caractere deve sempre ser uma letra, maiúsculas ou minúsculas.
- Os demais podem ser alfanuméricos (números, letras e _).
- Não podemos usar palavras reservadas da linguagem de programação.
- Não pode ter espaços em branco para o nome da variável.
- Para o nome composto de uma variável, devemos escrever tudo junto, ou utilizar o underscore _ .
- Escolher sempre nomes significativos e intuitivos para as variáveis.

Nomes Válidos
salario
Resultado_Soma
numValor
Nome_completo

Operadores Aritméticos

Operadores Aritméticos	A <- 7	Sinal	B <- 2	Resultado
Adição	A	+	B	9
Subtração	A	-	B	5
Multiplicação	A	*	B	14
Divisão	A	/	B	3.5
Divisão Inteira	A	\	B	3
Módulo	A	%	B	1
Potência	A	^	B	49

Ordem de Precedência

Parênteses	()
Exponencial	\wedge
Multiplicação / Divisão	$*$ /
Adição / Subtração	$+$ -

Concatenação

Concatenar significa juntar em uma sequência lógica.

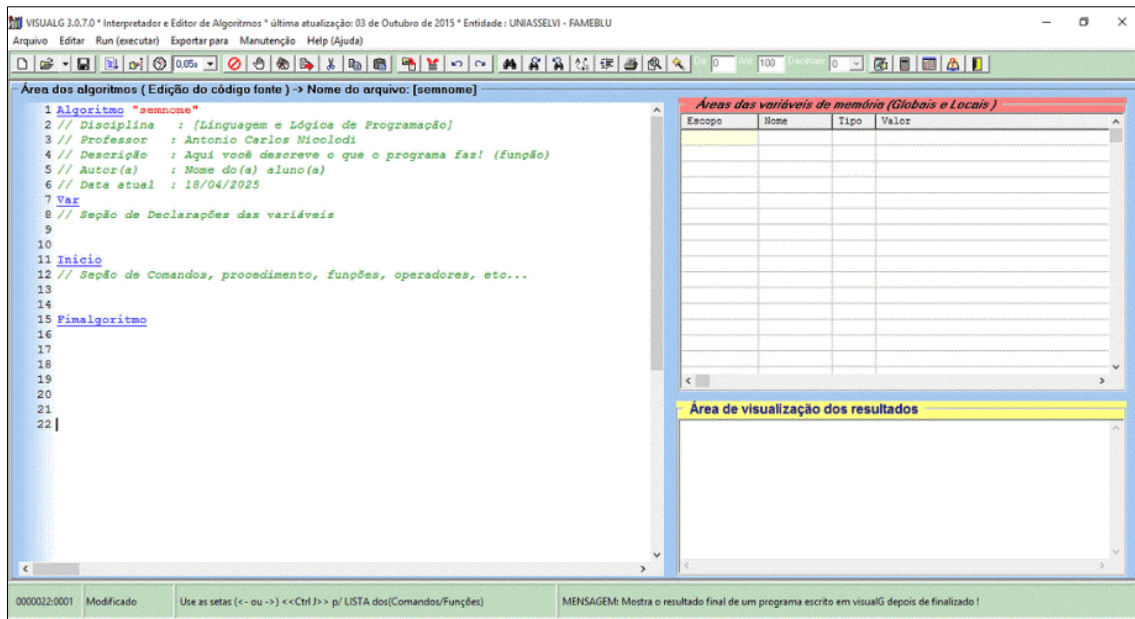
Exemplo

Escreva ("A soma é: ", soma)

Temos então a junção ou concatenação do texto "A soma é: " com a variável soma, separado por vírgula.

Exercícios

Tela de entrada do VisualG



Na esquerda temos a Área dos algoritmos, onde vamos digitar os códigos, a direita temos a Área das variáveis de memória (Globais e Locais). Logo abaixo temos a Área de visualização dos resultados. Acima temos a guia de comandos, e também o botão executar.

Exercício 01. Escreva um programa que vai imprimir na tela a mensagem: “Meu Primeiro Código! ”.

Entrada	Saída
Não possui entrada	“Meu Primeiro Código!”

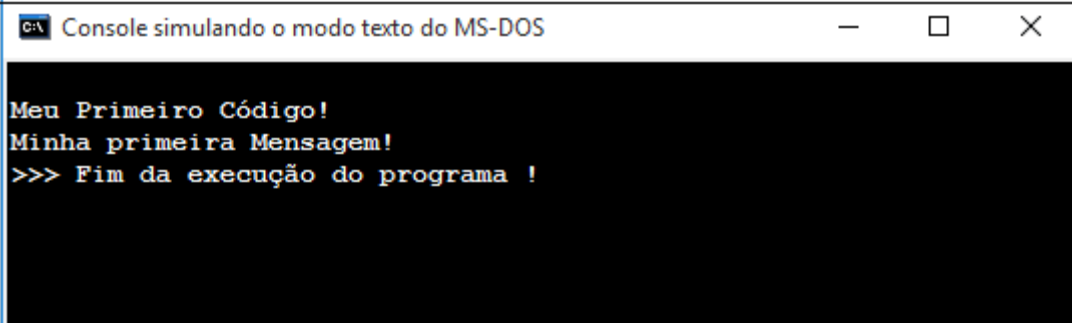
Resolução

```
algoritmo "primeiroCodigo"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Escreva um programa que vai imprimir na tela
// a mensagem: "Meu Primeiro Código! ".
// Autor(a) : Rogerio Curtio
// Data atual : 21/04/2025

Var

inicio
    // Comando de saída de Dados
    Escreval("Meu Primeiro Código!")
    Escreva("Minha primeira Mensagem!")
Fimalgoritmo
```

Ao executar o programa temos o resultado:



```
Meu Primeiro Código!
Minha primeira Mensagem!
>>> Fim da execução do programa !
```

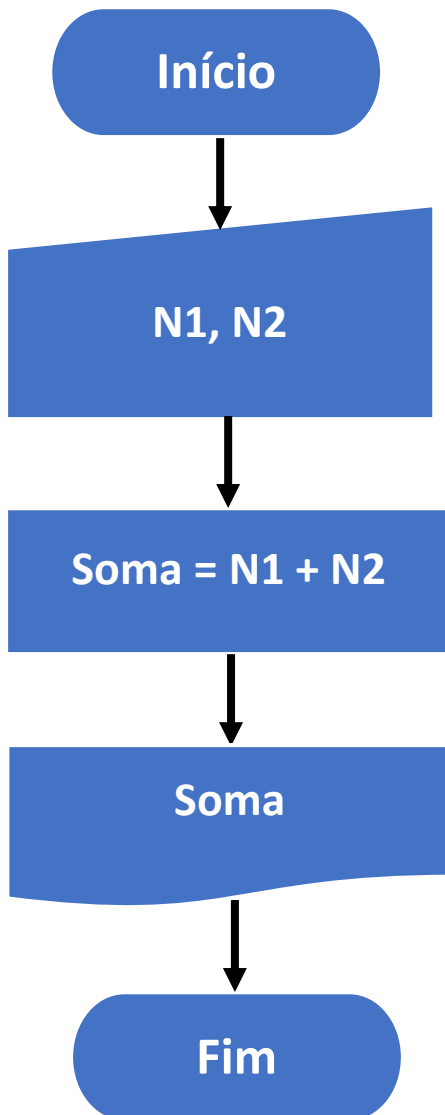
O comando Escreva - Escreve ou exibe a mensagem na mesma linha.

O comando Escreval - Escreve ou exibe a mensagem na linha abaixo ou pula uma linha.

Fluxograma

Um fluxograma é uma linguagem semi-gráfica que pode ser utilizada para a descrição de algoritmos.

Exemplo



Exemplo de algoritmo

- Descrição: Algoritmo que lê dois números inteiros e mostra a soma entre os números.
- Sinalização: Início
- Entrada de dados: N1, N2
- Processamento: (Instrução) $Soma = N1 + N2$
- Saída de dados: (Imprimir) Soma
- Sinalização: de Fim

Vamos criar um exemplo simples de um fluxograma que ilustra o processo de verificar se um número é par ou ímpar.

Descrição do Algoritmo

- Entrada: Um número inteiro.
- Processamento:
 - Dividir o número por 2.
 - Verificar o resto da divisão.
- Saída: Mensagem indicando se o número é par ou ímpar.

Estrutura do Fluxograma

- Início: Representado por uma elipse.
- Entrada de Número: Representada por um paralelogramo.
- Divisão por 2 e Verificação do Resto: Representada por um retângulo.
- Decisão: Verificar se o resto é igual a zero, representada por um losango.
 - Sim: Seguir para a saída "Número é par".
 - Não: Seguir para a saída "Número é ímpar".

Exercício 02. Desenvolva um algoritmo que leia dois números inteiros e mostra a soma entre os números.

Entrada	Processamento	Saída
10	$10 + 2$	Resultado da soma: 12
2		
- 20	$- 20 + 5$	Resultado da soma: - 15
5		

Resolução

```
Algoritmo "somaNumerica"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Algoritmo que lê dois números inteiros e mostra
// a soma entre os números.
// Autor(a) : Rogerio Curtio
// Data atual : 19/04/2025
Var
    N1, N2, Soma : Inteiro

Inicio
    // Entrada de Dados
    Escreva("Digite o Primeiro Número: ")
    Leia(N1)
    Escreva("Digite o Segundo Número: ")
    leia(N2)

    // Processamento
    Soma <- N1 + N2

    // Saída de Dados
    Escreva("Resultado da soma:", Soma)
Fimalgoritmo
```


Exercício 03. Fazer um programa para apresentar a soma, subtração, multiplicação e divisão de dois números.

Entrada	Processamento	Saída
20	$A + B$	Soma = 30
10	$A - B$	Subtração = 10
	$A * B$	Multiplicação = 200
	A / B	Divisão = 2

Resolução

```
Algoritmo "operacaoBasica"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : programa para apresentar a soma, subtração,
// multiplicação e divisão de dois números.
// Autor(a) : Rogerio Curtio
// Data atual : 21/04/2025
Var
    n1, n2, soma, subtracao, multiplicacao : Inteiro
    divisao : Real

Inicio
    // Entrada de Dados
    Escreva("Informe o Primeiro Número: ")
    Leia(n1)
    Escreva("Informe o Segundo Número: ")
    Leia(n2)

    // Processamento
    soma <- n1 + n2
    subtracao <- n1 - n2
    multiplicacao <- n1 * n2
    divisao <- n1 / n2

    // Saída de Dados
    Escreval("Soma = ", soma)
    Escreval("Subtração = ", subtracao)
    Escreval("Multiplicação = ", multiplicacao)
    Escreval("Divisão = ", divisao)
Finalgoritmo
```

Exercício 04. Fazer um programa para apresentar a soma, subtração, multiplicação de três números inteiros.

Entrada	Processamento	Saída
25	$A + B$	Soma = 45
8	$A - B$	Subtração = 5
12	$A * B$	Multiplicação = 2400
20	$A + B$	Soma = 14
- 8	$A - B$	Subtração = 26
2	$A * B$	Multiplicação = - 320

Resolução

```

Algoritmo "operacaoNumerica"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : - Fazer um programa para apresentar a soma,
// subtração, multiplicação de três números inteiros.
Var
    N1, N2, N3 : Inteiro
    soma, subtracao, multiplicacao : Inteiro
Inicio
    // Entrada de Dados
    Escreva("Informe o Primeiro Número: ")
    Leia(N1)
    Escreva("Informe o Segundo Número: ")
    Leia(N2)
    Escreva("Informe o Terceiro Número: ")
    Leia(N3)

    // Processamento
    soma <- N1 + N2 + N3
    subtracao <- N1 - N2 - N3
    multiplicacao <- N1 * N2 * N3

    // Saída de Dados
    Escreval("Soma =", soma)
    Escreval("Subtração =", subtracao)
    Escreval("Multiplicação =", multiplicacao)
Fimalgoritmo
  
```

Exercício 05. Fazer um programa para apresentar a soma, subtração, multiplicação e divisão de dois números reais.

Entrada	Processamento	Saída
14.5	A + B	Soma = 18.5
4	A - B	Subtração = 10.5
	A * B	Multiplicação = 58
	A / B	Divisão = 3.625
- 8	A + B	Soma = - 5.5
2.5	A - B	Subtração = - 10.5
	A * B	Multiplicação = - 20
	A / B	Divisão = 3.2

Resolução

```

Algoritmo "operacaoNumeros"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : - Fazer um programa para apresentar a soma,
// subtração, multiplicação e divisão de dois números reais.

Var
    n1, n2 : Real
    soma, subtracao, multiplicacao, divisao : Real

Inicio
    // Entrada de Dados
    Escreva("Digite o Primeiro Número: ")
    Leia(n1)
    Escreva("Digite o Segundo Número: ")
    Leia(n2)

    // Processamento
    soma <- n1 + n2
    subtracao <- n1 - n2
    multiplicacao <- n1 * n2
    divisao <- n1 / n2

    // Saída
    Escreval("Soma =", soma)
    Escreval("Subtração =", subtracao)
    Escreval("Multiplicação =", multiplicacao)
    Escreval("Divisão =", divisao)
Fimalgoritmo

```

Exercício 06. Fazer um programa para apresentar a divisão de dois números reais.

Entrada	Processamento	Saída
20	A / B	Resultado = 4
5		
-10.0	A / B	Resultado = - 2
5.0		

Resolução

```
Algoritmo "divisaoNumerica"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Fazer um programa para apresentar a divisão
// de dois números reais.
Var
    num1, num2, resposta : Real

Inicio

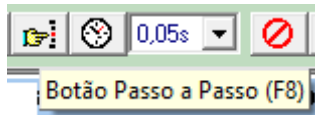
    // Entrada de dados
    Escreva("Digite o Primeiro Número: ")
    Leia(num1)
    Escreva("Digite o Segundo Número: ")
    Leia(num2)

    // Processamento
    resposta <- num1 / num2

    // Saída de Dados
    Escreva("Resultado =", resposta)
Finalgoritmo
```

Botão passo a passo no VisualG

Podemos usar o botão passo a passo, para acompanhar o passo do algoritmo. Também podemos executar com timer para ver a execução do passo a passo e verificar se estiver tudo ok.



```
1 Algoritmo "divisaoNumerica"
2 // Disciplina : [Linguagem e Lógica de Programação]
3 // Descrição : Fazer um programa para apresentar a divisão
4 // de dois números reais.
5 Var
6     num1, num2, resposta : Real
7
8
```

Formatação de números

Significa especificar o número de espaços no qual se deseja escrever um determinado valor. Escreva (x:5) escreve o valor da variável x em 5 espaços, alinhado à direita.

Escreva (y:6:2) escreve seu valor em 6 espaços colocando 2 casas decimais para números reais. Temos então para 6 caracteres onde dois caracteres são casas decimais.

```
Algoritmo "ExemploFormatacao"
Var
    x, y : Real
Inicio
    x <- 17.8
    y <- 1.893

    // 4 caracteres e 2 casas decimais
    Escreval(x:4:2)
    // 5 caracteres e 4 casas decimais
    Escreval(y:5:4)

Fimalgoritmo
```

Exercício 07. Fazer um programa que calcule a área e o comprimento de uma circunferência.

Fórmulas $A = \pi r^2$ $c = 2\pi r$

Entrada	Processamento	Saída
$r = 1$	$A = \pi r^2$	Área = 3.1415
	$c = 2\pi r$	Comprimento = 6.283
$r = 2,5$	$A = \pi r^2$	Área = 19.634375
	$c = 2\pi r$	Comprimento = 15.7075

Resolução

```

Algoritmo "algoritmoArea"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Fazer um programa que calcule a área e o
// comprimento de uma circunferência.
// Formulas A = pi * r² c = 2 * pi * r
// Autor(a) : Rogerio Curtio
// Data atual : 25/04/2025
Var
    A, C, raio : Real

Inicio
    // pi 3,1415
    // Entrada de Dados
    Escreva("Informar o valor do Raio: ")
    Leia(raio)

    // Processamento
    A <- 3.1415 * raio ^ 2
    C <- 2 * 3.1415 * raio

    // Saída
    Escreval("Area =", A)
    Escreval("Comprimento =", C)
Fimalgoritmo
    
```

Exercício 08. Elaborar um programa para calcular o volume de uma esfera.

$$\text{Volume} = \frac{4}{3} \pi r^3$$

Entrada	Processamento	Saída
r = 2	$\frac{4}{3} \pi r^3$	Volume = 33.510293
r = 3,5	$\frac{4}{3} \pi r^3$	Volume = 179.594228

Resolução

```
Algoritmo "volumeEsfera"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Elaborar um programa para calcular
// o volume de uma esfera. volume = 4/3 * pi * r³
// Autor(a) : Rogerio Curtio
// Data atual : 25/04/2025
Var
    volume, raio : Real

Inicio
    // Contante pi 3.14159
    Escreva("Valor do Raio: ")
    Leia(raio)

    // Processamento
    volume <- 4/3 * 3.14159 * raio ^ 3

    // Saida
    Escreval("Volume = ", volume:4:6)

Fimalgoritmo
```

Exercício 9. Escreva um programa para calcular e apresentar a área de um trapézio. O programa deve ler os valores correspondentes a base maior (B), a base menor (b) e a altura (h) do trapézio.

Fórmula $A = \frac{B+b}{2} h$

Entrada	Processamento	Saída
B = 8	$A = (B + b) / 2 * h$	Área = 12
b = 4		
h = 2		
B = 14	$A = (B + b) / 2 * h$	Área = 44
b = 8		
h = 4		

Resolução

```

Algoritmo "areaTrapezio"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Escreva um programa para calcular e apresentar
// a área de um trapézio. Fórmula  $A = (B + b) / 2 * h$ 
// Autor(a) : Rogerio Curtio
// Data atual : 25/04/2025
Var
    base_maior, base_menor, area, altura: Real
Inicio
    // Entrada de Dados
    Escreva("Valor da base maior: ")
    Leia(base_maior)
    Escreva("Valor da base menor: ")
    Leia(base_menor)
    Escreva("Valor da Altura: ")
    Leia(altura)

    // Processamento
    area <- ((base_maior + base_menor) / 2) * altura

    // Saida de Dados
    Escreval("Área =", area)
Fimalgoritmo
    
```


Exercício 10. Efetuar a leitura de uma temperatura medida em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula para conversão das temperaturas é $F = (9 * C + 160) / 5$ (ou $F = (C * 9/5) + 32$), sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

Entrada	Processamento	Saída
C = 20	$F = (9 * C + 160) / 5$	F = 68
C = 22.5		F = 72.5

Resolução

```

Algoritmo "temperatura"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Efetuar a leitura de uma temperatura
// medida em graus Celsius e apresentá-la convertida
// em graus Fahrenheit. A fórmula para conversão das
// temperaturas é  $F = (9 * C + 160) / 5$  (ou  $F = (C * 9/5) + 32$ )
// sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.
// Autor(a) : Rogerio Curtio
// Data atual : 26/04/2025
Var
    F, C : Real

Inicio
    // Entrada de dados
    Escreva("Graus Celsius: ")
    Leia(C)

    // Processamento
    F <- (9 * C + 160) / 5

    // Saída
    Escreval("Graus Fahrenheit:", F)
Fimalgoritmo
  
```

Exercício 11. Efetuar a leitura de uma temperatura medida em graus Fahrenheit e apresentá-la convertida em graus Celsius. A fórmula para conversão das temperaturas é Fórmula: $C = 5/9 * (F - 32)$, sendo C a temperatura em Celsius e F a temperatura em Fahrenheit.

Entrada	Processamento	Saída
F = - 24	$C = 5 / 9 * (F - 32)$	C = - 31.111
F = 45.4		C = 7.444

Resolução

```

Algoritmo "leituraTemperatura"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Efetuar a leitura de uma temperatura
// medida em graus Fahrenheit e apresentá-la
// convertida em graus Celsius. A fórmula para
// conversão das temperaturas é Fórmula:  $C = 5/9 * (F - 32)$ ,
// sendo C a temperatura em Celsius e F a temperatura em Fahrenheit.
// Autor(a) : Rogerio Curtio
// Data atual : 26/04/2025
Var
    C, F : Real

Inicio

    // entrada de dados
    Escreva("Temperatura em Fahrenheit: ")
    Leia(F)

    // Processamento
    C <- 5/9 * (F - 32)

    // Saída
    Escreva("Graus Celsius = ", C:4:3)

Fimalgoritmo

```

Exercício 12. Escreva um programa que leia os valores de A, B e C de uma equação do segundo grau e mostre o valor de Delta.

Equação $ax^2 + bx + c = 0$ Delta = $b^2 - 4ac$

Entrada	Processamento	Saída
A = 2	delta = $b^2 - 4ac$	delta = 256
B = 8		
C = - 24		

Resolução

```

Algoritmo "calculoDelta"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Equação  $ax^2 + bx + c = 0$     Delta =  $b^2 - 4ac$ 
// Autor(a) : Rogerio Curtio
// Data atual : 29/04/2025
Var
  A, B, C: Inteiro
  delta: Real

Inicio

  // Entrada de Dados
  Escreva("Informe o valor de A: ")
  Leia(A)
  Escreva("Informe o valor de B: ")
  Leia(B)
  Escreva("Informe o valor de C: ")
  Leia(C)

  // Processamento
  delta <- (B ^ 2) - 4 * A * C

  // Saída
  Escreval("ax² =", A)
  Escreval("bx =", B)
  Escreval("c =", C)
  Escreva("Resultado Delta =", delta)

Fimalgoritmo
  
```

Exercício 13. Fazer um programa que calcula a média entre duas notas e mostra o nome da disciplina e o resultado da média.

Entrada	Processamento	Saída
Disciplina = Geografia	Nota 1 + Nota2 / 2	Primeira Nota = 7.5
Disciplina = Português		Segunda Nota = 8.00

Resolução

```

Algoritmo "mediaNota"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Média entre duas notas e mostrar o resultado.
// Autor(a) : Rogerio Curtio
// Data atual : 29/04/2025
Var
    dcp: Caractere
    nota1, nota2, media : Real

Inicio

    // Entrada de Dados
    Escreva("Informe a Disciplina: ")
    Leia(dcp)
    Escreva("Informe a Primeira Nota: ")
    Leia(nota1)
    Escreva("Informe a Segunda Nota: ")
    Leia(nota2)

    // Processamento
    media <- (nota1 + nota2) / 2

    // Saída
    Escreva("Disciplina: ", dcp, " - Média: ", media:2:2)

Fimalgoritmo
  
```

Exercício 14. Desenvolva um programa que mostra o valor de A, B, C e calcula a soma entre eles.

A = 8 B = 14 C = 20

Entrada	Processamento	Saída
A	A + B + C	42
B		
C		

Resolução

```
Algoritmo "somaABC"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Mostrar o valor de A, B, C e calcula a soma entre eles.
// Autor(a) : Rogerio Curtio
// Data atual : 29/04/2025
Var
    A, B, C, Soma: Inteiro

Inicio

    // Entrada de Dados
    A <- 8
    B <- 14
    C <- 20

    // Processamento
    Soma <- A + B + C

    // Saída
    Escreval("A =", A)
    Escreval("B =", B)
    Escreval("C =", C)
    Escreva("Soma =", soma)

Fimalgoritmo
```

Exercício 15. Desenvolva um programa que leia o nome e o preço de um produto, calcule e mostre o seu preço promocional, com 10% de desconto.

Desconto = preço * 10% Promocional = preço - Desconto

Entrada	Processamento	Saída
Produto: Celular	Desconto = preço * 10%	20.00
Preço = R\$ 200.00	Promocional = preço - Desconto	R\$ 180.00

Resolução

```

Algoritmo "produtoPromocional"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : programa que leia o nome e o preço de um produto, calcule e mostre
o seu preço promocional, com 10% de desconto.
// Autor(a) : Rogerio Curtio
// Data atual : 29/04/2025
Var
    nomeProd : Caractere
    preco, desconto, precPromoc : Real
Inicio
    // Entrada de Dados
    Escreva("Informe o Produto: ")
    Leia(nomeProd)
    Escreva("Informe o Preço R$: ")
    Leia(preco)
    // Processamento
    desconto <- (preco * 10) / 100
    precPromoc <- (preco - desconto)

    // Saída
    Escreval("Produto: ", nomeProd)
    Escreval("Preço R$ ", preco:4:2)
    Escreval("Preço Promocional de 10% - R$ ", precPromoc:4:2)
Fimalgoritmo

```

Operadores Relacionais

Operador	Descrição
=	igual
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
<>	Diferente

Operadores Lógicos

A	B	A E B
V	V	V
V	F	F
F	V	F
F	F	F

A	B	A OU B
V	V	V
V	F	V
F	V	V
F	F	F

A	Não A
V	F
F	V

Exercício 16. Mostrar o resultado das operações lógicas abaixo:

A = 4	resultA = (A = B)
B = 12	resultB = (A < B)
C = 20	resultC = (A < B) OU (A >= C)
	resultD = (A < B) E (A >= C)

Entrada	Processamento	Saída
A = 4	resultA <- (A = B)	Falso
B = 12	resultB <- (A < B)	Verdadeiro
C = 20	resultC <- (A < B) OU (A >= C)	Verdadeiro
	resultD <- (A < B) E (A >= C)	Falso

Algoritmo "testeLogico"

// Disciplina : [Linguagem e Lógica de Programação]

// Professor : Antonio Carlos Nicolodi

// Descrição : Mostrar o resultado das operações lógicas

// Autor(a) : Rogerio Curtio

// Data atual : 01/05/2025

Var

A, B, C : Inteiro

resultadoA, resultadoB, resultadoC, resultadoD : Logico

Inicio

A <- 4

B <- 12

C <- 20

resultadoA <- (A = B)

resultadoB <- (A < B)

resultadoC <- (A < B) OU (A >= C)

resultadoD <- (A < B) E (A >= C)

Escreval(resultadoA)

Escreval(resultadoB)

Escreval(resultadoC)

Escreval(resultadoD)

Fimalgoritmo

Estruturas Condicionais

A compreensão de algoritmos e das estruturas condicionais é essencial para a programação. Estruturas condicionais permitem que o programa tome decisões com base em condições dadas, tornando-o mais dinâmico e adaptável. Vamos explorar as diferentes estruturas condicionais utilizadas em algoritmos.

Estrutura Condicional Simples

A estrutura condicional simples é a mais básica das estruturas condicionais. Ela verifica se uma condição é verdadeira, e, se for, executa um bloco de código. Caso a condição não seja verdadeira, o programa continua sem executar o bloco de código.

Exemplo em pseudocódigo:

SE (condição) ENTÃO

 // bloco de código a ser executado

FIM_SE

Exercício 17. Utilizar a estrutura condicional simples e verificar se a idade é maior ou igual a 18. Resolução:

```
Algoritmo "maiorIdade"
// Disciplina : [Linguagem e Lógica de Programação]
// Entrada de Dados: Utilize o comando LEIA para capturar a idade
// Processamento: verificar se a idade é maior ou igual a 18
// Saída : utilizar o comando Escreva para imprimir na tela a verificação.
Var
    idade : Inteiro
Inicio
    // Entrada
    Escreva("Digite a idade: ")
    Leia(idade)
    // Condicional Simples
    Se(idade >= 18)Entao
        Escreva("Você é maior de idade.")
    FimSe
Fimalgoritmo
```

Exercício 18. Utilizar a estrutura condicional simples e verificar se a condição é verdadeira

Resolução

```
Algoritmo "valorLogico"  
// Disciplina : [Linguagem e Lógica de Programação]  
// Professor : Antonio Carlos Nicolodi  
// Descrição : Utilizar estrutura condicional simples  
// e verificar se a condição é verdadeira  
// Autor(a) : Rogerio Curtio  
// Data atual : 01/05/2025  
Var  
    A, B, C : Inteiro  
  
Inicio  
    // Entrada  
    A <- 8  
    B <- 12  
    C <- 20  
  
    // Processamento / Saida  
    Se(C >= A) ou (B < C)Entao  
        Escreva("Teste Verdadeiro")  
    FimSe  
  
Fimalgoritmo
```

Estrutura Condicional Composta

A estrutura condicional composta permite definir um bloco de código a ser executado caso a condição inicial não seja verdadeira. Isso é feito por meio do comando **SENÃO**.

Exemplo em pseudocódigo:

SE (condição) **ENTÃO**

 // bloco de código se a condição for verdadeira

SENÃO

 // bloco de código se a condição for falsa

FIM_SE

Exercício 19. Utilizar a estrutura condicional composta e verificar a idade e imprimir uma mensagem apropriada com base na condição.

Resolução

```
Algoritmo "verificaldade"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Utilizar estrutura condicional Composta
// e verificar a idade e imprimir uma mensagem apropriada com base na condição.
Var
    idade : Inteiro
Inicio
    // Entrada
    Escreva("Digite a idade: ")
    Leia(idade)

    // Processamento / Saída
    Se(idade >= 18)Entao
        Escreva("Você é maior de idade.")
    Senao
        Escreva("Você é menor de idade.")
    FimSe
Fimalgoritmo
```

Exercício 20. Utilizar a estrutura condicional composta e verificar se o número digitado é par ou ímpar.

```
Algoritmo "ParImpar"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Utilizar estrutura condicional composta se o número digitado é par ou
ímpar
// Autor(a) : Rogerio Curtio
// Data atual : 04/05/2025
Var
    N: Inteiro

Inicio
    Escreva("Digite um Número: ")
    Leia(N)
    Se(N MOD 2 = 0)Entao
        Escreva("Número:", N, " é PAR")
    Senao
        Escreva("Número:", N, " é ÍMPAR")
    FimSe
Fimalgoritmo
```

Estrutura Condicional Aninhada

Uma estrutura condicional aninhada ocorre quando uma instrução condicional está contida dentro de outra. Isso permite que decisões mais complexas sejam tomadas, pois o resultado de uma condição pode depender de outra. No Portugal, essas estruturas são geralmente implementadas usando as instruções `se`, `senão` e `fimse`.

Exemplo do Algoritmo

- Primeira Condição: A primeira condição avalia se a nota é maior ou igual a 7. Se for, o estudante é considerado "Aprovado".
- Aninhamento com `senão`: Caso a primeira condição não seja atendida, o algoritmo entra no bloco `senão`, onde uma nova condição é verificada: se a nota é maior ou igual a 5. Se esta condição for verdadeira, o estudante está em "Recuperação".
- Condição Final: Se nenhuma das condições anteriores for verdadeira, o algoritmo determina que o estudante está "Reprovado".

Vantagens das Condicionais Aninhadas

- ✓ Flexibilidade: Permite lidar com múltiplas condições de maneira organizada.
- ✓ Clareza: Ajuda a estruturar o código de forma que as decisões lógicas sejam mais claras para quem lê o algoritmo.
- ✓ Eficiência: Reduz o número de instruções necessárias, uma vez que cada condição é avaliada apenas conforme necessário.

Exercício 21. Determinar o estado de um estudante com base em sua nota

Resolução

```
Algoritmo "mediaAluno"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Determinar o estado de um estudante com base em sua nota
// Autor(a) : Rogerio Curtio
// Data atual : 02/05/2025
Var
    nota1, nota2, media: Real
    situacao : Caractere

Inicio
    // Entrada
    Escreva("Informe a Primeira Nota: ")
    Leia(nota1)
    Escreva("Informe a Segunda Nota: ")
    Leia(nota2)

    // Processamento
    media <- (nota1 + nota2) / 2
    Se(media >= 7)Entao
        situacao <- "Aprovado(a)"
    Senao
        Se(media >= 5)Entao
            situacao <- "Recuperação"
        Senao
            situacao <- "Reprovado"
        FimSe
    FimSe

    // Saída
    Escreval(situacao, " Com Média = ", media:4:2)
Fimalgoritmo
```

Estrutura Condicional de Escolha (Escolha Caso)

A estrutura de escolha, também conhecida como Escolha Caso, é utilizada quando há múltiplas condições e múltiplos blocos de código que podem ser executados. É uma alternativa mais organizada e eficiente do que utilizar múltiplos SE e SENÃO.

Exemplo em pseudocódigo:

ESCOLHA (variável)

CASO valor1:

// bloco de código

CASO valor2:

// bloco de código

CASO valorN:

// bloco de código

CASO_CONTRÁRIO:

// bloco de código se nenhum caso for verdadeiro

FIM_ESCOLHA

Exercício 22. Utilizar a estrutura Escolha Caso, e imprimir a operação desejada.

Resolução

```
Algoritmo "imprimeMensagem"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Utilizar a estrutura Escolha Caso, e imprimir o Resultado
// Autor(a) : Rogerio Curtio
// Data atual : 03/05/2025
Var
    Valor : Inteiro
    resultado : Caractere

Inicio
    Escreval("**** Operações Básicas ****")
    Escreval()
    Escreval("[1] Soma")
    Escreval("[2] Subtração")
    Escreval("[3] Multiplicação")
    Escreval("[4] Divisão")
    Escreva("Escolha a Operação: [1] a [4]: ")
    Leia(Valor)
    Escolha(valor)
        Caso 1
            resultado <- "Soma = A + B"
        Caso 2
            resultado <- "Subtração = A - B"
        Caso 3
            resultado <- "Multiplicação = A x B"
        Caso 4
            resultado <- "Divisão = A / B"
        OutroCaso
            resultado <- "[0] Opção Inválida!"
    FimEscolha
    Escreval("Resultado: ", resultado)

Fimalgoritmo
```


Estrutura de Repetição Enquanto

As estruturas de repetição são fundamentais na programação, permitindo que um bloco de código seja executado várias vezes, dependendo de uma condição específica. Entre as estruturas de repetição mais utilizadas, está o laço ENQUANTO (ou while em inglês).

O que é a Estrutura ENQUANTO?

A estrutura ENQUANTO é um tipo de loop que continua a executar um bloco de comandos enquanto uma condição especificada for verdadeira. Esse tipo de loop é ideal quando não se sabe antecipadamente quantas vezes o bloco de comandos precisará ser executado, mas sim que ele deve continuar enquanto uma certa condição permanecer válida.

Sintaxe Básica

A sintaxe básica de um loop ENQUANTO pode ser descrita da seguinte forma:

ENQUANTO condição_falsa

faça

bloco_de_comandos

FIM-ENQUANTO

Exercício 23. Utilizar a estrutura Enquanto, e imprimir os números de 0 a 20
Contagem Progressiva. Resolução:

```
Algoritmo "contagem_Progressiva"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Utilizar a estrutura Enquanto, e imprimir os números de 0 a
20
// Autor(a) : Rogerio Curtio
// Data atual : 04/05/2025
Var
    i : Inteiro

Inicio

    // Contagem Progressiva
    i <- 0
    Enquanto (i <= 20) Faça
        Escreva(i)
        i <- i + 1
    FimEnquanto

Fimalgoritmo
```

Explicação:

1. Inicialização: Começamos definindo o valor inicial da variável *i* como 0.
2. Condição: O loop continua enquanto *i* for menor ou igual a 20.
3. Bloco de Comandos: Dentro do loop, imprimimos o valor atual de *i* e, em seguida, incrementamos seu valor em 1.
4. Terminação: Quando contador se torna 21, a condição *i* <= 20 se torna falsa, e o loop termina.

Exercício 24. Utilizar a estrutura enquanto, e imprimir os números de 20 a 0
Contagem Regressiva. Resolução:

```
Algoritmo "contagem_Regressiva"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Utilizar a estrutura enquanto, e imprimir os números de 20 a
0
// Autor(a) : Rogerio Curtio
// Data atual : 04/05/2025
Var
    i : Inteiro

Inicio

    // Contagem Regressiva
    i <- 20
    Enquanto (i >= 0) Faça
        Escreva(i)
        i <- i - 1
    FimEnquanto

Fimalgoritmo
```

Cuidados ao Usar o Loop ENQUANTO

- **Condição Infinita:** É crucial garantir que a condição eventualmente se torne falsa, caso contrário, o loop pode se tornar infinito, causando problemas no programa.
- **Atualização da Condição:** Normalmente, a variável de controle dentro da condição deve ser atualizada dentro do bloco de comandos para evitar loops infinitos.

Vantagens e Desvantagens

Vantagens

- Versatilidade: Pode ser usado em situações em que o número de iterações não é conhecido antecipadamente.
- Leitura Condicional: A condição é verificada antes de cada execução do bloco de comandos, garantindo que o loop só execute quando necessário.

Desvantagens

- Risco de Loop Infinito: Se não for cuidadosamente implementado, pode resultar em loops infinitos.
- Complexidade: Em alguns casos, pode ser mais complexo de implementar do que outros tipos de loops, como o para (ou for).

Estrutura Repita

A estrutura "repita" é usada para executar um bloco de código repetidamente. A diferença principal entre "repita" e outras estruturas de repetição, como "enquanto", é que "repita" garante que o bloco de código seja executado pelo menos uma vez antes de verificar a condição.

Sintaxe Básica

Abaixo está a sintaxe básica de um laço "repita" em pseudocódigo:

repita

 // bloco de código

até que (condição)

Exercício 25. Utilizar a estrutura Repita, e imprimir os números de 1 a 40

Contagem Progressiva. Resolução:

```
Algoritmo "Contagem_Progressiva"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Utilizar a estrutura Repita, e imprimir os números de 1 a 40
// Autor(a) : Rogerio Curtio
// Data atual : 05/05/2025
Var
    i : Inteiro

Inicio
    // Inicializar
    i <- 1
    Repita
        Escreva(i)
        // incrementar
        i <- i + 1
    Ate i > 40
Fimalgoritmo
```

Explicação:

- Definimos uma variável **i** com o valor 1.
- O código dentro do laço escreve o valor atual de **i** e, em seguida, acrescenta-o.
- O laço continua até que **i** seja maior que 40.

Exercício 26. Utilizar a estrutura Repita, e imprimir os números de 20 a 1

Contagem Regressiva. Resolução:

```
Algoritmo "Contagem_Regressiva"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Utilizar a estrutura Repita, e imprimir os números de 20 a 1
// Autor(a) : Rogerio Curtio
// Data atual : 05/05/2025
Var
    i : Inteiro

Inicio
    // Inicializar
    i <- 20
    Repita
        Escreva(i)
        // Decrementar
        i <- i - 1
    Ate i < 1
Fimalgoritmo
```

Explicação:

- Definimos uma variável **i** com o valor 20.
- O código dentro do laço escreve o valor atual de **i** e, em seguida, decrementa-o.
- O laço continua até que **i** seja menor que 1.

Vantagens do "Repita"

- Execução Garantida: O bloco de código dentro do "repita" é sempre executado pelo menos uma vez.
- Simplicidade: Em situações onde queremos garantir pelo menos uma execução, "repita" é mais intuitivo.

Estruturas de Repetição PARA

As estruturas de repetição são essenciais na programação, permitindo a execução de um bloco de código várias vezes de maneira eficiente. Uma das estruturas mais comuns é o laço PARA (ou loop for), ideal quando o número de repetições é conhecido previamente.

O que é o Algoritmo PARA?

O algoritmo PARA é uma estrutura de repetição controlada por um contador. É usado quando o número de iterações é conhecido antes de iniciar o loop. Sua sintaxe é bastante intuitiva e fácil de usar.

Sintaxe Geral

A sintaxe do loop PARA pode variar entre linguagens de programação, mas geralmente segue um formato semelhante:

PARA i DE início ATÉ fim FAÇA

// bloco de código a ser executado

FIMPARA

início: determina o valor inicial do contador.

fim: determina o valor final do contador.

i: é a variável de controle, incrementada automaticamente a cada iteração.

Exercício 27. Utilizar a estrutura Para, e imprimir os números de 1 a 20

```
Algoritmo "ContagemProgressiva"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Utilizar a estrutura Para, e imprimir os números de 1 a 20
// Autor(a) : Rogerio Curtio
// Data atual : 05/05/2025
Var
    i: Inteiro
Inicio
    // Estrutura Para
    Para i De 1 Ate 20 Faça
        Escreva(i)
    FimPara
Fimalgoritmo
```

Este código resultará na impressão dos números de 1 até 20 com passo 1.

Exemplo com Incremento

Em algumas linguagens, é possível especificar um incremento diferente de 1:

PARA i DE 0 ATÉ 10 PASSO 2 FAÇA

IMPRIMIR(i)

FIMPARA

Exercício 28. Utilizar a estrutura Para, e imprimir os números de 0 a 100 com passo 5
Resolução

```
Algoritmo "Contagem"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Utilizar a estrutura Para, e imprimir os números de 0 a 100
com passo 5
// Autor(a) : Rogerio Curtio
// Data atual : 05/05/2025
Var
    i : Inteiro
Inicio
    Para i de 0 Até 100 Passo 5 Faça
        Escreva(i)
    FimPara
Fimalgoritmo
```

Este código resultará na impressão dos números de 0 até 100 com passo 5, ou de 5 em 5.

Aula Prática: Funções no VisualG

Bem-vindo à nossa aula prática sobre funções no VisualG! Vamos explorar como criar e utilizar funções nesse ambiente de programação, através de explicações teóricas e exercícios práticos.

O que é uma Função?

Uma função é um bloco de código separado que executa uma tarefa específica. Ela pode receber entradas, chamadas de parâmetros, e retornar um valor como resultado. Em VisualG, as funções ajudam a organizar e reutilizar o código de forma eficiente.

Estrutura de uma Função no VisualG

No VisualG, a estrutura básica de uma função é a seguinte:

Função nomeDaFuncao(param1: Tipo; param2: Tipo): TipoRetorno

Início

// Corpo da função

Resultado <- valor

FimFunção

nomeDaFuncao: O nome que você dá à função.

param1, param2, ...: Os parâmetros que a função pode receber.

TipoRetorno: O tipo do valor que a função vai retornar.

Resultado: Palavra-chave usada para indicar o valor de retorno da função.

Exercício 29. Criar uma Função para Somar Dois Números

Descrição

Vamos criar uma função chamada soma que recebe dois números inteiros como parâmetros e retorna a soma deles.

Passo a Passo

1. Definição da Função: Vamos definir a função soma no VisualG.

Função soma(num1: Inteiro; num2: Inteiro): Inteiro

Início

Retorne A + B

FimFunção

Chamamos a função: Soma(N1, N2)

Uso da Função no Programa Principal:

```
Algoritmo "funcao_Soma"
// Disciplina : [Linguagem e Lógica de Programação]
// Descrição : Criar uma Função para Somar Dois Números
// Autor(a) : Rogerio Curtio
// Data atual : 06/05/2025
Var
    N1, N2, S : Inteiro
    funcao Somar(A, B: Inteiro): Inteiro
    Início
        Retorne A + B
    FimFuncao
Início
    Escreva("Informe o Primeiro Número: ")
    Leia(N1)
    Escreva("Informe o Segundo Número: ")
    Leia(N2)
    S <- Somar(N1, N2)
    Escreval("Soma:", N1, " +", N2, " =", S)
Fimalgoritmo
```

Exercício 30. Criar uma Função para retornar um valor par ou impar

Resolução

```
Algoritmo "funcao_Par_Impar"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Criar uma Função para retornar um valor par ou impar
// Autor(a) : Rogerio Curtio
// Data atual : 06/05/2025
Var
    val : Inteiro
    resultado : Caractere

Funcao parImpar(vl: Inteiro): Caractere
Inicio
    Se (vl%2 = 0) entao
        Retorne "PAR"
    senao
        Retorne "IMPAR"
    FimSe
FimFuncao

Inicio
    Escreva("Digite um Valor: ")
    Leia(val)
    resultado <- parImpar(val)
    Escreva ("O Valor", val, " é ", resultado)
Fimalgoritmo
```

Passo a Passo

Definição da Função: Vamos definir a função soma no VisualG.

Função parImpar (valor: Inteiro): Caractere

Início

Se (valor % 2 = 0) entao

Retorne "PAR"

senao

Retorne "IMPAR"

FimSe

FimFunção

Aula Prática sobre Array e Vetor em Visualg

Conceitos de Array e Vetor

Vetor

Um vetor é uma estrutura de dados que armazena uma sequência de elementos do mesmo tipo. No Visualg, um vetor é declarado especificando seu tipo e o número de elementos que ele pode conter.

Array

O termo "array" é frequentemente usado como sinônimo de vetor em contextos de programação. No entanto, em algumas linguagens, "array" pode se referir a estruturas multidimensionais.

Declaração de Vetores no Visualg

Para declarar um vetor no Visualg, usamos a seguinte sintaxe:

var

nomeVetor: vetor[1..n] de Tipo

- nomeVetor: Nome do vetor.
- n: Número de elementos que o vetor pode conter.
- Tipo: Tipo dos elementos no vetor (inteiro, real, caractere, etc.).

Exemplo de Declaração

var

numeros: vetor[1..5] de inteiro

Neste exemplo, declaramos um vetor chamado numeros que pode armazenar cinco números inteiros.

Exercício 31. Imprima na tela os valores armazenados do vetor

Resolução

```
Algoritmo "vetor_exemplo_01"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Imprima na tela os valores do vetor
// Autor(a) : Rogerio
// Data atual : 07/05/2025
Var
    vet : vetor[1..5] De Inteiro
    i : Inteiro

Inicio
    // Manipulando valores dos elementos manualmente
    vet[1] <- 50
    vet[2] <- 40
    vet[3] <- 12
    vet[4] <- 25
    vet[5] <- 200

    // Saída para o Vetor
    Para i De 1 Até 5 Faça
        Escreval(vet[i])
    FimPara
Fimalgoritmo
```

Exercício 32. Imprima na tela os valores do vetor de 5 valores inteiros

Resolução

```
Algoritmo "vetor_exemplo_02"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Imprimir os valores digitados do vetor na Tela
// Autor(a) : Rogerio
// Data atual : 07/05/2025
Var
    vetorA : vetor[0..4] de Inteiro
    i : Inteiro

Inicio
    // Entrada de Dados para o Vetor
    Para i de 0 Até 4 Faça
        Escreva("Digite um Valor: ")
        Leia(vetorA[i])
    FimPara

    // Saída de Dados para o Vetor
    Para i de 0 Até 4 Faça
        Escreva(vetorA[i])
    FimPara
Fimalgoritmo
```


Aula Prática de Matrizes em Visualg

O que é uma Matriz?

Uma matriz é uma estrutura de dados bidimensional que armazena elementos em linhas e colunas. No Visualg, podemos pensar em uma matriz como uma tabela, onde cada posição é identificada por dois índices: um para a linha e outro para a coluna.

Declaração de Matrizes

No Visualg, uma matriz é declarada especificando o tipo dos elementos e as dimensões (número de linhas e colunas). Veja o exemplo abaixo:

```
// Declaração de uma matriz 3x3 de inteiros
```

```
declare matriz: vetor[1..3, 1..3] de inteiro
```

Inicialização de Matrizes

Podemos inicializar uma matriz atribuindo valores individualmente a cada posição. Exemplo:

```
matriz[1, 1] <- 1
```

```
matriz[1, 2] <- 2
```

```
matriz[1, 3] <- 3
```

```
matriz[2, 1] <- 4
```

```
matriz[2, 2] <- 5
```

```
matriz[2, 3] <- 6
```

```
matriz[3, 1] <- 7
```

```
matriz[3, 2] <- 8
```

```
matriz[3, 3] <- 9
```

Exercício 33. Imprima na tela os valores do vetor de 5 valores inteiros

Resolução

```
Algoritmo "somaMatriz"
// Disciplina : [Linguagem e Lógica de Programação]
// Professor : Antonio Carlos Nicolodi
// Descrição : Crie um algoritmo que declare uma matriz 2x2 e some todos
os seus elementos, exibindo o resultado.
// Autor(a) : Rogerio
// Data atual : 11/05/2025
Var
    matriz: vetor[1..2, 1..2] de inteiro
    soma, i, j: inteiro

Inicio
    // Inicializando a matriz
    matriz[1, 1] <- 1
    matriz[1, 2] <- 2
    matriz[2, 1] <- 3
    matriz[2, 2] <- 4

    // Calculando a soma dos elementos
    soma <- 0
    para i de 1 ate 2 faca
        para j de 1 ate 2 faca
            soma <- soma + matriz[i, j]
        fimpara
    fimpara

    // Exibindo o resultado
    escreval("A soma dos elementos da matriz é: ", soma)
Fimalgoritmo
```