



C U R S O S

Byte

www.cursosbyte.com.br



@cursosbyte

Cursos Byte Treinamentos Ltda

CNPJ: 12.156.417/0001-27

Dados do Aluno

Nome: _____

Número da matrícula: _____

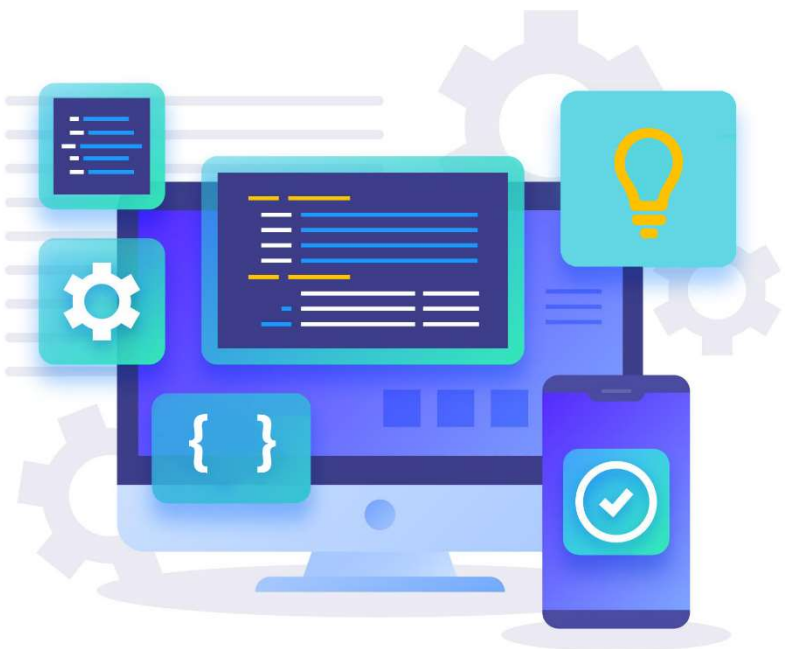
Endereço: _____

Bairro: _____

Cidade: _____

Telefone: _____

Anotações Gerais: _____



Lógica de Programação V2

Apresentação da Lógica de Programação

O curso de Lógica de Programação visa introduzir o aluno ao pensamento lógico, etapa importante para iniciar o aprendizado de uma linguagem de programação de computadores. Hoje em dia, independentemente da linguagem a ser aprendida, será necessário utilizar os conceitos básicos de lógica de programação.

O curso de Lógica de Programação irá preparar o aluno a entender os princípios básicos de uma linguagem de programação, como Entrada, Processamento e Saída, a trabalhar com estudo de variáveis, análise de códigos através de Breakpoints, operações relacionais, estruturas de decisão e de repetição.

Depois de concluído o curso, você já terá conhecimentos de algoritmos e conseguirá aplicar os conhecimentos adquiridos em uma linguagem de programação.

Marcas Registradas:

Todas as marcas e nomes de produtos apresentados nesta apostila são de responsabilidade de seus respectivos proprietários, não estando a editora associada a nenhum fornecedor ou produto apresentado nesta apostila.



Método CGD® - Todos os direitos reservados.

Protegidos pela Lei 5988 de 14/12/1973.

Nenhuma parte desta apostila poderá ser copiada sem prévia autorização.

O Método CGD é um produto da Editora CGD.

[illegible]

01 – CONHECENDO O PORTUGOL STUDIO	7
• CONHECENDO O PORTUGOL STUDIO	7
• O QUE É SOFTWARE?.....	8
• TIPOS DE SOFTWARES DE COMPUTADOR	8
• LINGUAGEM PSEUDOCÓDIGO.....	8
• SINTAXE DO PROGRAMA PORTUGOL STUDIO	9
• COMO ESCREVER UM SOFTWARE.....	9
• O QUE É ALGORITMO?	10
02 – ESCRREVENDO ALGORITMOS.....	11
• ALGORITMO BÁSICO	11
03 – COMANDO ESCREVA E LEIA	14
• COMANDO ESCREVA E LEIA	14
• COMANDO ESCREVA: CONCATENAR.....	16
04 – USANDO VARIÁVEIS E O INSPETOR DE VARIÁVEIS.....	17
• UTILIZANDO VARIÁVEIS E O QUADRO DE VARIÁVEIS	17
• INSPETOR DE VARIÁVEIS	19
• VARIÁVEIS: TIPOS DE DADOS	21
• CONFIRA AS VARIÁVEIS DISPONÍVEIS NO PORTUGOL STUDIOS	21
05 – CALCULAR MÉDIA ARITMÉTICA	22
• MÉDIA ARITMÉTICA SIMPLES	22
06 - ESTRUTURAS SEQUENCIAIS / DECISÃO CONDICIONAL SIMPLES SE	24
• ESTRUTURAS SEQUENCIAIS.....	24
• ESTRUTURAS DE DECISÃO.....	25
• ESTRUTURAS DE DECISÃO: CONDICIONAL SIMPLES - SE.....	26
07 – ESTRUTURAS DE DECISÃO CONDICIONAL COMPOSTA SE- SENAO E OPERADORES.....	27
• ESTRUTURA DE EXEMPLO	27
• OPERADORES RELACIONAIS.....	28
08 – PRATICANDO A CONDICIONAL SE	30
• PRATICANDO	30
09 – ESTRUTURAS DE DECISÃO: CASO	33
• EXPLICAÇÃO	33

10 – OPERADORES LÓGICOS (OR / AND)	35
• OPERAÇÕES LÓGICAS.....	35
• OPERADOR LÓGICO DO TIPO OR	36
• OPERADOR LÓGICO DO TIPO AND	38
11 – PRATICANDO E CONHECENDO O OPERADOR DO TIPO NOT	39
• PRATICANDO ALGORITMOS	39
• OPERADOR LÓGICO DO TIPO NOT	41
12 – ESTRUTURAS DE REPETIÇÃO / ENQUANTO	43
• O QUE É UMA ESTRUTURA DE REPETIÇÃO?	43
• ESTRUTURA DE REPETIÇÃO ENQUANTO...FAÇA	45
13 – PRATICANDO A ESTRUTURA DE REPETIÇÃO ENQUANTO	48
• PRATICANDO A ESTRUTURA DE REPETIÇÃO / ENQUANTO... FAÇA	48
• BIBLIOTECA	52
14 – ESTRUTURA DE REPETIÇÃO / PARA-ATÉ-FAÇA.....	55
• RELEBRANDO ESTRUTURAS DE REPETIÇÃO	55
• ESTRUTURA PARA-ATÉ-FAÇA.....	55
• DIFERENÇAS ENTRE ESTRUTURAS DE REPETIÇÃO.....	57

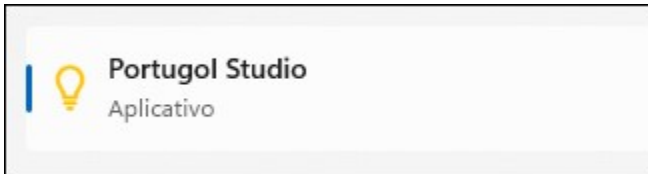


CGD

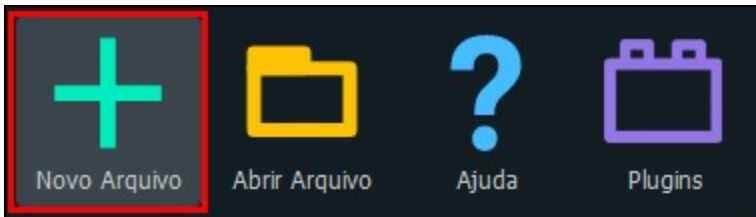
01 – Conhecendo o Portugol Studio

• Conhecendo o Portugol Studio

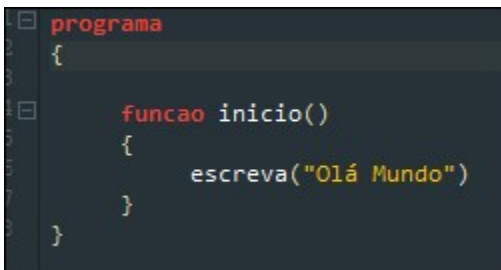
- O Portugol Studio é uma ferramenta para aprender programação, voltada para as pessoas que falam o idioma português. Possui uma sintaxe fácil baseada em C e PHP, diversos exemplos e materiais de apoio à aprendizagem. Também possibilita a criação de jogos e outras aplicações.
- Acesse seu menu iniciar do Windows
- Pesquise pelo programa "Portugol Studio"



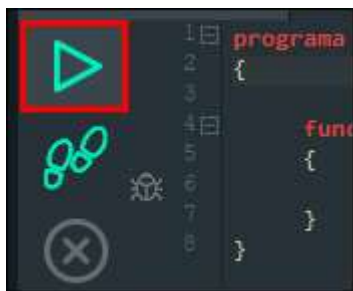
- Para criar um novo "Algoritmo", clique em:



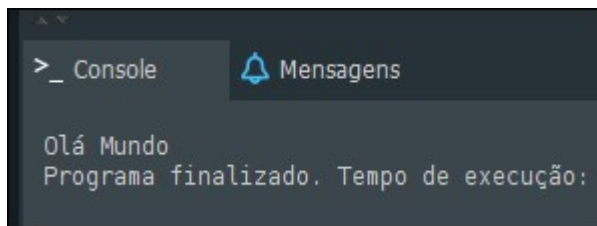
- Ao iniciar um novo arquivo o programa já apresenta uma estrutura básica



- Para executar as ações de um algoritmo você deve utilizar o botão "Executar"



- Assim o programa fará a apresentação do programa



• O Que É Software?

- **Software** é uma sequência de instruções a serem seguidas e/ou executadas, na manipulação, redirecionamento ou modificação de um dado/informação ou acontecimento. **Software** também é o nome dado ao comportamento exibido por essa sequência de instruções quando executada em um computador ou máquina semelhante, além de um produto desenvolvido pela Engenharia de software, e inclui não só o programa de computador propriamente dito, mas também manuais e especificações.

• Tipos de Softwares de Computador

- Software de sistema que inclui o firmware
- Software aplicativo

• Linguagem Pseudocódigo

- Pseudocódigo é uma forma genérica de escrever um algoritmo, no curso será utilizado o programa Portugol Studio
- Portugol: O programa Portugol Studio utiliza a linguagem "Portugol" que é uma linguagem chamada de pseudocódigo, isto é, não é uma linguagem de programação. É uma linguagem para o aprendizado de programação.

- A linguagem que o Portugol Studio interpreta é bem simples: é uma versão portuguesa dos pseudocódigos largamente utilizados nos livros de introdução à programação, conhecida como "Portugol".

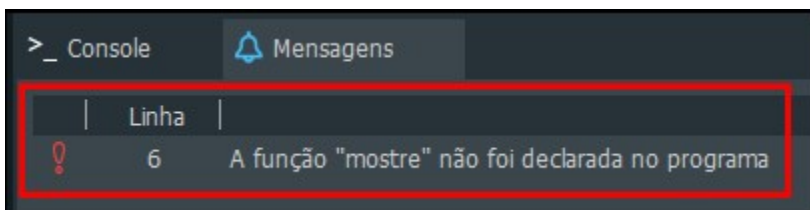
• Sintaxe do Programa Portugol Studio

- A sintaxe de uma linguagem de programação é a forma como os seus comandos devem ser escritos.
- Por exemplo, para escrever uma mensagem na tela, foi utilizado o comando escreva

- Exemplo:

```
escreva("Seu arquivo foi enviado para a análise")
```

- Sabemos então, que o comando chamado escreva, escreve mensagens na tela do usuário
- Um erro de sintaxe ocorre se uma sintaxe incorreta for utilizada como por exemplo a substituição da palavra "escreva" para a palavra "mostre", o comando correto é o "escreva", assim, ao executar o programa um problema será apresentado. Exemplo:



- Além deste erro, mais simples de resolver podendo trocar a sintaxe para a forma correta de escrever é possível identificar erros na estrutura do algoritmo onde se tem uma

• Como Escrever Um Software

- Normalmente, programas de computador são escritos em linguagens de programação, pois estas foram projetadas para aproximar-se das linguagens usadas por seres humanos. Atualmente existe uma quantidade muito grande de linguagens de programação, dentre elas, as mais populares no momento são **C#, C++, Java, Swift, PHP**, dentre outras.
- Um software precisa reproduzir a ideia gerada por um ser humano, ou seja, sua execução vai depender de um conjunto de informações ordenadas, para que ele possa ter um resultado final.

• O Que É Algoritmo?

- Um algoritmo é uma sequência de instruções bem definidas, cada uma das quais pode ser executada mecanicamente num período de tempo fi-nito. Ele representa um conjunto de regras para a solução de um proble-ma.
- No exemplo da receita abaixo, a correta execução das instruções vai resultar na preparação do bolo para o consumo.



• Exemplo:

INGREDIENTES

1 ovo pequeno
4 colheres (sopa) de leite
3 colheres (sopa) de óleo
2 colheres (sopa) rasas de chocolate em pó
4 colheres (sopa) rasas de farinha de trigo
4 colheres (sopa) rasas de açúcar
1 colher (café) rasa
de fermento em pó

MODO DE PREPARO

1 - Na própria caneca onde irá consumir, coloque o ovo e bata bem com um garfo;
2 - Coloque o óleo, o açúcar, o leite e o chocolate e bata mais;
3 - Coloque a farinha de trigo e o fermento e misture delicadamente até encorpar;
4 - Leve ao forno micro-ondas por 3 minutos em potência alta.

- Para um programa de computador, o efeito será o mesmo. O programa deve possuir um algoritmo ordenado, para que o computador processe as informações na ordem especificada e possa apresentar o resultado proposto pelo seu programa.
- Veja agora um exemplo de algoritmo para ser implementado em um programa de computador:

Início

Receber o valor do primeiro número

Receber o valor do segundo número

Calcular a soma dos dois números

Mostrar o resultado da soma na tela

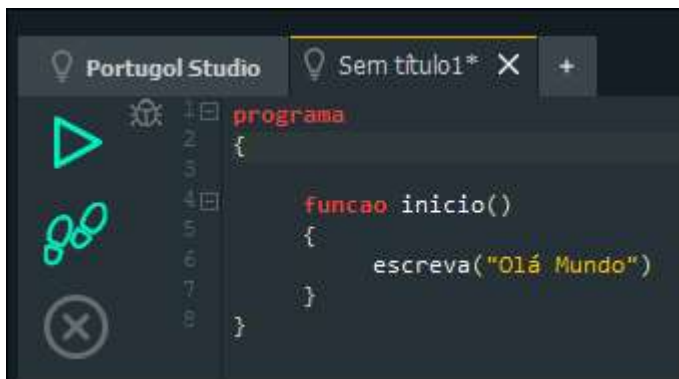
Final

- No exemplo:
- 1. O programa irá receber 2 números.
- 2. Em seguida, vai ocorrer um processamento: que é a soma dos números recebidos.
- 3. E, por final, será apresentado na tela o resultado da soma dos números.
- Seguindo as instruções fornecidas pelo algoritmo, o programa (software) vai apresentar o resultado desejado na tela.

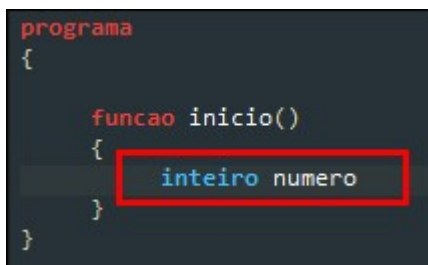
02 – Escrevendo Algoritmos

• Algoritmo Básico

- O primeiro passo antes de digitar o algoritmo no Portugol Studio é estender o problema que foi proposto
- O Portugol Studio está com a estrutura básica criada



- Comece declarando as variáveis do seu algoritmo



- No Portugol Studio, você verá uma variedade de variáveis disponíveis onde cada uma é utilizada para uma função específica.
- A variável "inteiro" é utilizada para armazenar números inteiros (sem casas decimais), no caso da utilização de casas decimais é utilizado por exemplo a variável "real"
- Quando o usuário precisa "entrar com dados", estes dados devem ser gravados em algum local, em nosso exemplo, ficará na memória do computador. Para isso, existem as chamadas "variáveis". Uma variável é um espaço criado na memória do computador, onde é possível gravar e ler os dados.
- Para emitir mensagens ao usuário do programa, utiliza-se a sintaxe "escreva", a frente dela digite os parenteses e aspas, conforme no exemplo:



```
funcao inicio()
{
    inteiro numero
    inteiro resultado

    escreva("Entre com o número desejado")
}
```

- Após emitir a mensagem o programa precisa armazenar o valor digitado pelo usuário no algoritmo, para isto é usado a sintaxe "leia" e o nome da variável entre parênteses a frente da sintaxe. Assim:

```
escreva("Entre com o número desejado")
leia(numero) ←
```

- Criando as variáveis e armazenando os valores delas você pode criar qualquer regra a partir disto, seja uma estrutura mais complexa ou um cálculo simples, como a adição do número digitado pelo usuário somado a outro número qualquer, que você programe, neste exemplo o "40". Veja:

```
escreva("Entre com o número desejado")
leia(numero)
resultado = numero + 40
```

- Programando uma nova ação como essa, o próximo passo é exibir o resultado ao usuário, para isto, você pode criar uma mensagem que detalhe esse resultado e em seguida, exiba a sintaxe "escreva" com a variável (resultado) junto a ela. Exemplo:

```
escreva("Entre com o número desejado")
leia(numero)
resultado = numero + 40
escreva("O resultado informado acrescentado o valor de 40 é:")
escreva(resultado)
```

- Para testar os ajustes do algoritmo, use o botão "Executar"

03 – Comando Escreva e Leia

• Comando Escreva e Leia

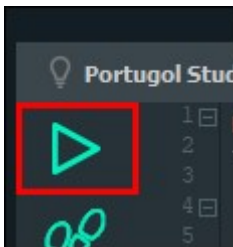
- O comando "escreva" é utilizado para mostrar a "saída" de algum processamento. Essa saída pode ser uma mensagem para o usuário, como pode ser o resultado de algum cálculo efetuado pelo programa. Para utilizar o comando "escreva" não é obrigatório utilizar variáveis.
- Neste exemplo é possível conferir um algoritmo sem variáveis:

```
programa
{
    funcao inicio()
    {
        escreva("O comando escreva é usado ")
        escreva("para escrever um resultado ")
        escreva("ou uma mensagem para o usuário ")
        escreva("ele também pode escrever o conteúdo de uma variável ")
    }
}
```

- O comando escreva precisa das aspas para indicar o início e o final do texto

```
escreva("O comando escreva é usado ")
```

- Além das aspas, o comando escreva também precisa dos parênteses, para indicar o que será escrito
- Clique sobre o botão "Executar" para conferir os textos



- O texto será exibido de maneira extensa, com um texto ao lado do outro sem quebras de linha. Confira:



- Para quebrar uma linha a cada texto, podemos utilizar o comando "\n". O comando "\n" acrescenta uma quebra de linha ao final da saída.
- Confira o código com as modificações para as quebras de linha:

```
funcao inicio()
{
    escreva("O comando escreva é usado\n")
    escreva("para escrever um resultado\n")
    escreva("ou uma mensagem para o usuário\n")
    escreva("ele também pode escrever o conteúdo de uma variável ")
}
```

- No exemplo acima, veja que na última linha não foi adicionando o comando "\n" pois não existem outras linhas abaixo dela, assim, não é necessário haver uma quebra.
- Caso você queira dar um espaço extra sem texto, apenas com uma quebra utilize a sintaxe com o comando:

escreva("\n")

- Ao executar o algoritmo com as quebras as linhas são exibidas uma abaixo das outras. Exemplo:

A screenshot of a console window with a dark background. At the top, there are two tabs: 'Console' and 'Mensagens'. The 'Console' tab is active, showing the output of the code: 'O comando escreva é usado', 'para escrever um resultado', 'ou uma mensagem para o usuário', 'ele também pode escrever o conteúdo de uma variável', and 'Programa finalizado. Tempo de execução: 45 milissegundos'. The text is displayed on five separate lines, with the last line being a comment in a lighter color.

- Como o algoritmo possui apenas comandos do tipo "escreva" o programa trabalha com mensagens de saída, ele não armazena nenhuma informação.
- As linhas na cor cinza são comentários. São usados para criar explicações no código do algoritmo.


```
escreva("O comando escreva é usado para escrever na tela\n")  
  
// exemplo do comando escreva para mostrar o valor de uma variável  
escreva(nome)
```

- Os comentários são importantes, principalmente em seu desenvolvimento inicial para que as ações fiquem claras em sua leitura do algoritmo, elas auxiliam também na explicação para outros usuários que tenham acesso ao seu código.



- Uma variável do tipo cadeia é utilizada para armazenar dados de textos (letras, números e símbolos). Exemplo:

```
funcao inicio()  
{  
  cadeia nome
```

• Comando Escreva: Concatenar

- Em programação, concatenar é o nome dado para juntar termos. Veremos agora como concatenar duas ou mais variáveis.
- Para concatenar o texto com variáveis, é importante retirar as aspas do local onde serão exibidas as variáveis, e utilizar vírgulas para separar os conteúdos. Confira o exemplo:

```
escreva("Os dados informados são:", nome, idade, cidade)
```

- Explicação:



- Usando o recurso de concatenar (juntar), conseguimos juntar em uma única linha o conteúdo de várias variáveis

04 – Usando Variáveis e o Inspetor de Variáveis

• Utilizando Variáveis e o Quadro de Variáveis

- Com os valores armazenados nas variáveis, podemos efetuar o processamento da informação, e em seguida, desenvolver ao usuário o resultado de um cálculo ou de qualquer outra operação.
- Lembre-se:
escreva = utilizado para escrever uma mensagem
leia = utilizado para armazenar um valor dentro de uma variável
- Sempre que for utilizar uma variável no programa Portugol Studio, ela precisa ser declarada antes da execução do programa. Na maioria das linguagens de programação as variáveis também são declaradas antes da execução do programa.
- Confira o código de exemplo:

```

funcao inicio()
{
    inteiro salario
    inteiro comissao
    inteiro desconto
    inteiro resultado

    escreva("Informe o valor do salário:\n")
    leia(salario)

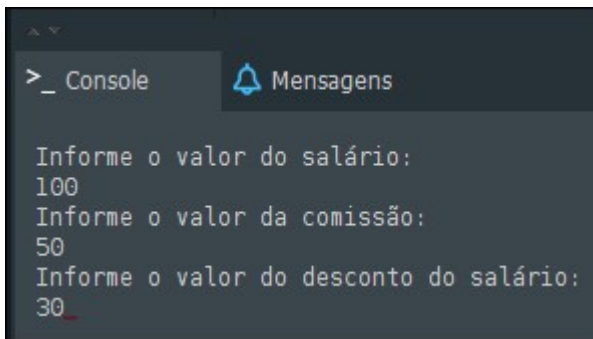
    escreva("Informe o valor da comissão:\n")
    leia(comissao)

    escreva("Informe o valor do desconto do salário:\n")
    leia(desconto)

    resultado = salario + comissao - desconto
    escreva("Salário final: ", resultado, "\n")
}

```

- Ao executar o algoritmo, se usar os dados indicados...



```

> _ Console
Informe o valor do salário:
100
Informe o valor da comissão:
50
Informe o valor do desconto do salário:
30

```

- O resultado será apresentado com o valor de "120"
- Valor referente a variável "resultado"
- A variável resultado, armazena o resultado de uma operação matemática

```

resultado = salario + comissao - desconto
escreva("Salário final: ", resultado, "\n")

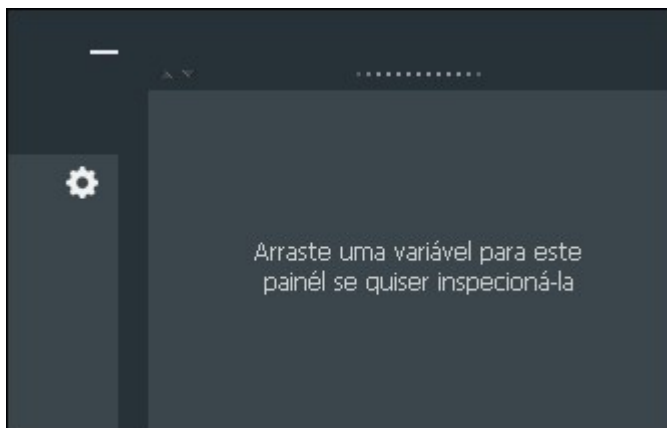
```

- Explicação: O programa executou da seguinte forma; Resultado é igual a: **salario + comissão – desconto**

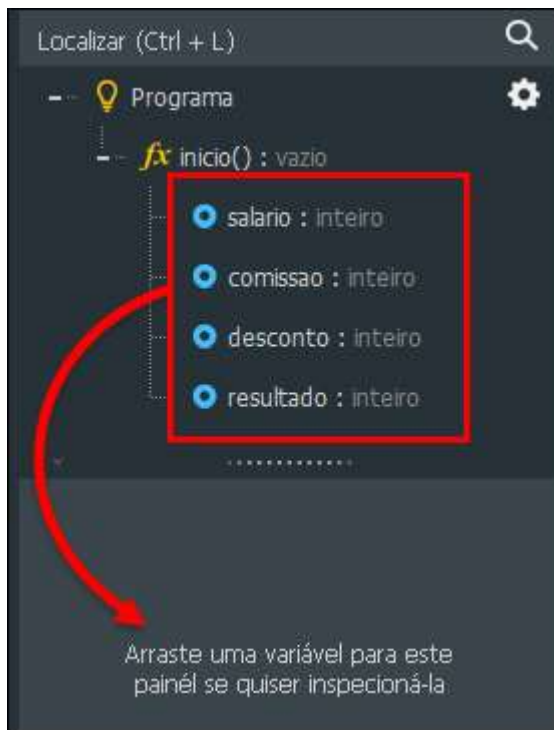


- **Inspetor de Variáveis**

- O inspetor de variáveis fica localizado do lado inferior direito do programa



- Para que seja possível a visualização das variáveis, localize a árvore estrutural, clique sobre as variáveis e arraste para dentro do inspetor
- Confira um exemplo de algumas variáveis que são arrastadas até o inspetor de variáveis:



- As variáveis são exibidas no painel:



- Através do inspetor de variáveis, podemos acompanhar o valor de cada variável sendo preenchida durante a execução

- Isso é muito útil quando queremos identificar possíveis problemas na estrutura de um algoritmo, pois quando se executa o algoritmo passo-a-passo, conseguimos identificar o comportamento de cada variável

• **Variáveis: Tipos de Dados**

- Quando estamos prevendo a execução de um programa, temos de prever também qual o tipo de dados a variável vai receber. Por exemplo: se você deseja que o usuário informe o "nome da pessoa", a variável nome deverá ser do tipo cadeia, não poderá ser do tipo inteiro.
- No caso de dados tratando de salário ou valores com casas decimais, é utilizado variáveis do tipo "real". O tipo de dados "real" aceita os valores decimais, separados por vírgula.
- Existem centenas de linguagens de programação. Cada uma delas possui uma estrutura para o "tipo de dados" das variáveis. Então, quando você começar a estudar uma determinada linguagem de programação, é interessante procurar pelos tipos de dados aceitos naquela linguagem. Isso é uma variação que vai existir de linguagem para linguagem.



• **Confira As Variáveis Disponíveis No Portugol Studios**

- **Inteiro:** Define variáveis numéricas do tipo inteiro, ou seja, sem casas decimais
- **Real:** Define variáveis numéricas do tipo real, ou seja, com casas decimais
- **Cadeia:** Define variáveis do tipo string, ou seja, cadeia de caracteres
- **Logico:** Define variáveis do tipo booleano, ou seja, com valor VERDADEIRO ou FALSO
- Uma maneira de saber qual tipo de dados utilizar para cálculos, é seguir essa tabela...

Operação	1º operando	2º operando	Resulta em
Adição +	Inteiro	Inteiro	Inteiro
	Real	Real	Real
	Real	Inteiro	Real
	Inteiro	Real	Real

Subtração -	Inteiro	Inteiro	Inteiro
	Real	Real	Real
	Real	Inteiro	Real
	Inteiro	Real	Real

Multiplicação *	Inteiro	Inteiro	Inteiro
	Real	Real	Real
	Real	Inteiro	Real
	Inteiro	Real	Real

Divisão real /	Inteiro	Inteiro	Real
	Real	Real	Real
	Real	Inteiro	Real
	Inteiro	Real	Real

Divisão inteira /	Inteiro	Inteiro	Inteiro

- Nessa tabela são indicados os tipos de dados retornados quando se realizam operações aritméticas. Você deve adequar o seu programa para "prever" qual tipo de dado o usuário poderá fornecer.

05 – Calcular Média Aritmética

• Média Aritmética Simples

- A média aritmética simples é a mais utilizada no nosso dia a dia. No caso das notas ela é obtida dividindo-se a soma de todas as notas pelo número, notas (avaliações)

- **Exemplo:** Um aluno tirou as notas 5, 7, 9 e 10 em quatro provas.
- A sua média será: **(5 + 7 + 9 + 10) / 4 = 7.75**
- Temos de dividir a soma das notas, pelo número de notas somadas.
- Confira o exemplo de um algoritmo programado para calcular médias:

```
funcao inicio()
{
    real n1
    real n2
    real n3
    real n4

    real soma
    real media

    escreva("Digite a primeira nota: ")
    leia(n1)

    escreva("Digite a segunda nota: ")
    leia(n2)

    escreva("Digite a terceira nota: ")
    leia(n3)

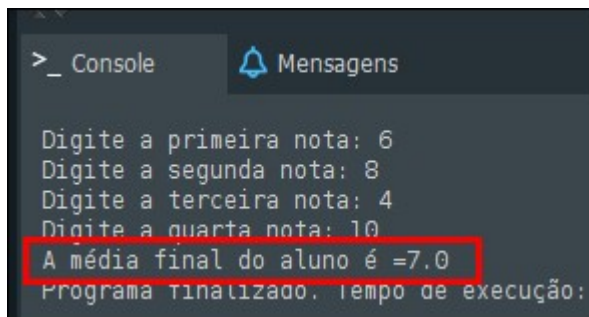
    escreva("Digite a quarta nota: ")
    leia(n4)

    soma = (n1+n2+n3+n4)
}
```

- Ao utilizar o sinal "=" estamos indicando uma atribuição. Logo, a variável soma receber a operação: (n1+n2+n3+n4)
- No exemplo acima, existem as variáveis e as mensagens solicitando os dados de entrada, que são lidos pela sintaxe "leia" e em seguida armazenados nas variáveis (n1, n2, n3 e n4)
- O resultado da soma, deve ser dividido pela quantidade de lançamentos (4), assim, será encontrado o valor da média. Confira o código:

```
soma = (n1+n2+n3+n4)
media = (soma/4)
escreva("A média final do aluno é =", media)
```


- Confira o exemplo de lançamento de dados:



```
> _ Console Mensagens

Digite a primeira nota: 6
Digite a segunda nota: 8
Digite a terceira nota: 4
Digite a quarta nota: 10
A média final do aluno é =7.0
Programa finalizado. tempo de execução:
```

- Nestes lançamentos temos os valores: 6, 8, 4 e 10 (Logo, se somarmos os valores encontramos o resultado 28 e $28 / 4 = 7$).
- É importante lembrar que se as notas tiverem casas decimais deve-se utilizar a variável do tipo "real".

06 - Estruturas Sequenciais / Decisão Condicional Simples SE

• Estruturas Sequenciais

- Em ciência da computação, uma estrutura sequencial é uma estrutura de desvio do fluxo de controle presente em linguagens de programação, que realiza um conjunto predeterminado de comandos de forma sequencial, de cima para baixo, na ordem em que foram declarados.

Origem: Wikipédia, a enciclopédia livre.

- As estruturas sequenciais são comandos que são executados imperativamente, do início ao final do código. Não há desvio no código, como por exemplo, uma condição.
- Exemplo básico de uma condição: Pense num programa que calcula o preço da entrada no cinema... Se a idade é maior ou igual a 18 anos, então, um preço será oferecido. Caso contrário, se a idade for menor que 18, um outro preço será oferecido.
- Quando um programa não possui desvios (condições) e nem repetições, então dizemos que o programa é uma estrutura sequencial. O código será sempre executado da mesma maneira, imperativamente, na sequência como foi escrito.



- **Estruturas de Decisão**

- As estruturas de decisão permitem que um programa tome uma decisão no código, isto é, de acordo com uma condição. O programa faz o teste com o resultado de uma expressão lógica. De acordo com esse resultado, o programa sabe qual o caminho deve seguir.
- Veja uma representação gráfica de uma estrutura de decisão do tipo SE



- Um exemplo de estrutura de decisão do tipo SE, é lembrar-se da função SE utilizada no programa Excel. No Excel utilizamos a função SE para “testar uma condição”, e, dependendo do resultado da condição, apresentar um determinado valor.
- **Nota:** As Estruturas de Decisão também podem ser chamadas de Estruturas de Seleção

• Estruturas de Decisão: Condicional Simples - SE

- Confira um exemplo de estrutura SE simples:

```
nome = "Pedro"

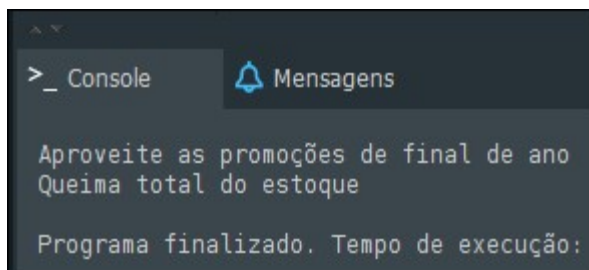
se (nome == "Pedro"){
    escreva("Olá Pedro!!! vem-vindo de volta a nossa loja\n")
}

escreva("Aproveite as promoções de final de ano\n")
escreva("Queima total do estoque\n")
```

- No exemplo acima, é exibida a condicional indicando se o “nome” for igual à “Pedro”, então escreva a mensagem e no topo do algoritmo é possível ver a variável “nome” atribuída ao dado “Pedro”.
- Assim ao executar o programa a variável “nome” já utiliza o dado “Pedro” e assim a condição torna-se verdadeira, exibindo a mensagem
- Se você mudar o nome da atribuição para outro qualquer ao executar o programa não exibirá a mensagem, dará como encerrado.
- Exemplo:

```
nome = "Maria"

se (nome == "Pedro"){
    escreva("Olá Pedro!!! vem-vindo de volta a nossa loja\n")
}
```



```
> _ Console  Mensagens

Aproveite as promoções de final de ano
Queima total do estoque

Programa finalizado. Tempo de execução:
```

- Confira a mensagem:

- O programa exibe apenas os outros comandos de mensagens após a estrutura SE, mas a mensagem "Olá Pedro!!!..." não é exibida

```
nome = "Pedro"

se (nome == "Pedro"){
    escreva("Olá Pedro!!! vem-vindo de volta a nossa loja\n")
}

escreva("Aproveite as promoções de final de ano\n")
escreva("Queima total do estoque\n")
```

- Outra opção é alterar a atribuição da variável nome, removendo ela, você pode criar uma linha para que o algoritmo escreva uma frase solicitando um nome de entrada e em seguida ele leia a variável "nome". Veja:

```
escreva("Informe o nome da pessoa ")
leia(nome)
```

- Dessa forma o programa dependerá da sua entrada para que ele execute a estrutura SE, se o nome for "Pedro" a mensagem é exibida, caso contrário, não.

07 – Estruturas De Decisão Condicional Composta SE-SENAO E Operadores

• Estrutura de Exemplo

- **Exemplo:** Se o estado informado for "RJ", vai oferecer uma mensagem; senão for RJ, então vai oferecer uma segunda mensagem. Confira o código:

```
se (estado == "RJ"){
    escreva("Para você do Rio, desconto de 15%. Só hoje!")
}
senao
    escreva("A venda está autorizada apenas para o RJ")
```

- Dessa maneira você pode atribuir a variável estado com um dado fixo, como RJ ou não, porém, nas estruturas de decisão é mais recomendado a

adição de uma interação com o usuário, como a adição de uma mensagem solicitando o estado desejado e assim ler a variável "estado" para que apresente a estrutura.

- A estrutura "SENAO" é descrita abaixo de uma estrutura "SE" como condição contrária e ela não necessita abrir e fechar "{ }". Confira:

```
se (estado == "RJ"){  
    escreva("Para você do Rio, desconto de 15%. Só hoje!")  
}  
senao  
    escreva("A venda está autorizada apenas para o RJ")  
}
```

- A representação gráfica da condição SE composta:



- Nesse tipo de decisão, o programa tem 2 caminhos a tomar. Tudo vai depender do teste Lógico, que neste exemplo é se o "estado = RJ". SE o resultado do teste for VERDADEIRO, vai em um caminho. SENAO, SE o resultado for FALSO, vai pelo outro caminho.

• Operadores Relacionais

- Através dos operadores relacionais é possível realizar as comparações entre duas variáveis. Por exemplo, quando comparamos se o estado era igual a RJ, temos 2 variáveis sendo utilizadas. A primeira é a variável "estado", a segunda é ao que será comparado, ou seja, testamos se era "RJ".

- Quando efetuamos comparações sempre iremos ter um retorno dessa comparação. Esse retorno será “Verdadeiro” ou “Falso”, não existe outra possibilidade de retorno
- Essa é a lista de operadores relacionais:

Lista de operadores relacionais	
Símbolo	Significado
=	Igualdade
<	Menor
>	Maior
<=	Menor ou igual
>=	Maior ou igual
<>	Diferente

- Confira o exemplo de um algoritmo com a estrutura SE utilizando o operador “Maior”

```
se (numero > 18){
    escreva("O aluno é maior de idade")
}
senao
    escreva("O aluno é menor de idade")
```

- Dessa forma, se o usuário digitar um número maior que 18, então será exibido a mensagem “O aluno é maior de idade”, SENAO, exibe a mensagem “O aluno é menor de idade”.
- Outra opção para este exemplo é usar o operador “Maior ou igual” já que se o usuário digitar 18 o teste daria como FALHO já que 18 não é maior que 18, assim exibiria a mensagem de menor de idade.
- Com o ajuste o código ficaria assim:

```

se (numero >= 18){
    escreva("O aluno é maior de idade")
}
senao
    escreva("O aluno é menor de idade")

```

- Dessa forma com o usuário digitando qualquer número MAIOR ou IGUAL a 18 exibe a mensagem "O aluno é maior de idade".



08 – Praticando A Condicional SE

• Praticando

- Declare variáveis para seu algoritmo, use o tipo "real"

```

funcao inicio()
{
    real a
    real b
}

```

- Para testes com operadores relacionais geralmente se utiliza variáveis "reais" ou inteiras se o número for apenas números inteiros.
- Solicite os dados de entrada ao usuário através de mensagens como essas, e em seguida, leia as variáveis "a" e "b".

```
escreva("Entre com o primeiro número ")
leia(a)

escreva("Entre com o segundo número ")
leia(b)
```

- Veja um exemplo de estrutura SE composta com outra estrutura dentro de uma já existente.

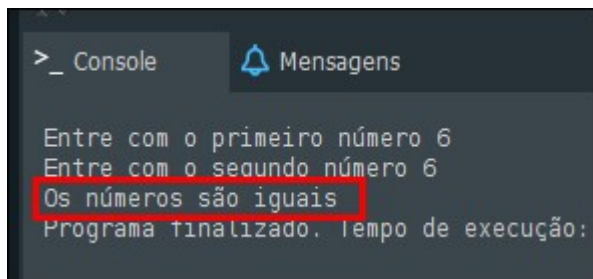
```
se (a == b){
    escreva("Os números são iguais")
}
senao

    se (a > b){
        escreva("O primeiro número é o maior")
    }
    senao
        escreva("O segundo número é o maior")
```

- Nesse exemplo, foi utilizado o "senao" para criar outra estrutura SE, fazendo com que o programa faça uma ação se o teste não for verdadeira na primeira instância, ou seja, se os números não forem iguais como testado na primeira linha

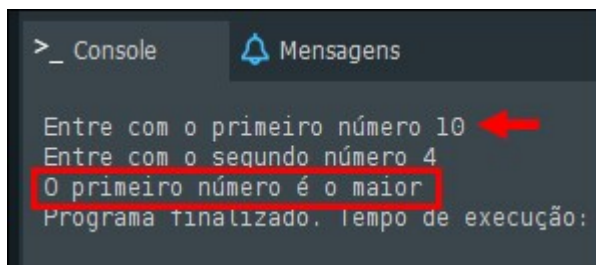


- Assim o programa exibida a mensagem “O primeiro número é maior” se o usuário digitar o primeiro número maior que o segundo ($a > b$)
- Senao, exibe “O segundo número é o maior”
- Lembre-se as estruturas do “senao” não precisam abrir e fechar as chaves “{ }”
- Neste exemplo, digitando os valores iguais no programa, veja o resultado:



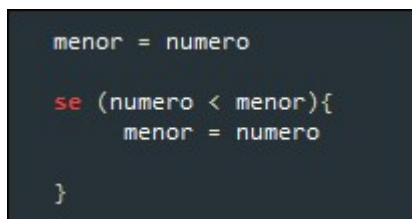
```
>_ Console Mensagens
Entre com o primeiro número 6
Entre com o segundo número 6
Os números são iguais
Programa finalizado. tempo de execução:
```

- Se você digitar o primeiro número como 10 e o segundo 4 por exemplo o programa deve exibir o resultado:



```
>_ Console Mensagens
Entre com o primeiro número 10
Entre com o segundo número 4
O primeiro número é o maior
Programa finalizado. tempo de execução:
```

- É possível também comparar variáveis na estrutura SE, assim se um valor digitado for maior, menor ou igual a de outra variável uma ação pode ser iniciada. Confira a estrutura:



```
menor = numero

se (numero < menor){
    menor = numero
}
```

- Neste exemplo temos a atribuição da variável menor para a variável numero, assim, inicia a estrutura SE, onde se o numero for MENOR que a variável menor, então a variável menor se torna igual a variável numero

- Em seguida, veja a estrutura de comparações com 3 números diferentes que podem ser digitados pelo usuário:

```
    escreva("Informar o segundo número: ")
    leia(numero)

    se (numero < menor){
        menor = numero
    }

    escreva("Informar o terceiro número: ")
    leia(numero)

    se (numero < menor){
        menor = numero
    }
    escreva("O menor número digitado foi: ", menor)
}
```

09 – Estruturas de Decisão: CASO

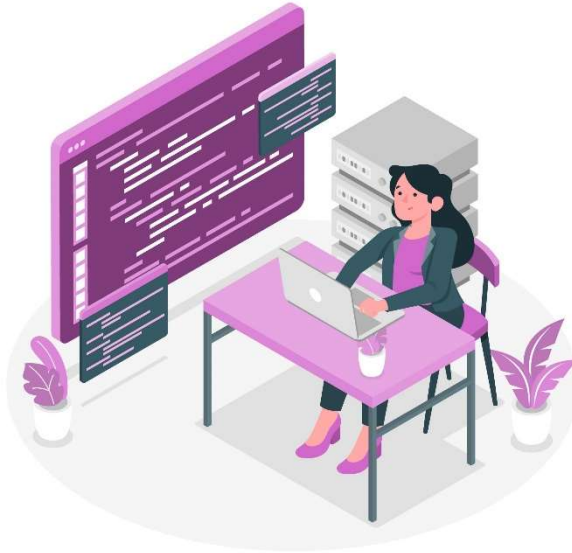
- **Explicação**

- Utilizando a estrutura CASO, podemos determinar vários caminhos, que será escolhido através de uma expressão. Por exemplo: podemos determinar que o programa escolhe qual caminho seguir, dependendo de um número informado, o usuário poderia ter 5 ou "n" caminhos diferentes.
- A estrutura CASO também pode ser chamada de "escolha". Confira o exemplo:

```
caso 9:
    escreva ("Aprovado\n")
pare
caso 10:
    escreva ("Aprovado\n")
pare

caso contrario:
    escreva ("Nota inválida.\n")
```

- Essa estrutura deve ser utilizada quando o código tiver muitas decisões a serem tomadas, isto é, quando o código oferecer muitas opções de caminhos. Por exemplo, imagine que cada mês informado poderia efetuar um cálculo de desconto diferente, então usaríamos o "CASO" para testar os meses e decidir qual cálculo será apresentado.



- Tudo que é feito com a estrutura "CASO", também pode ser feita com a condicional SE, porém, cada SE tem que ficar aninhado dentro de outra condicional SE, tornando o código muito extenso e de difícil compreensão
- Confira outro exemplo de algoritmo com a estrutura CASO

```
escolha (diasemana)
{
    caso 1:
        escreva ("Hoje é Domingo")
    pare
    caso 2:
        escreva ("Hoje é Segunda-feira")
    pare
    caso contrario:
        escreva ("Outros dias!")
    pare
}
```

- Veja que a estrutura se inicia com "CASO" e termina com "PARE" e ainda há a opção "CASO CONTRARIO" que valida uma ação oposta a do CASO.
- Assim como na maior parte dos comandos é necessário iniciar as chaves "{}". Exemplo:

```
→ escolha (diasemana)
{
    caso 1:
        escreva ("Hoje é Domingo")
    pare
    caso 2:
        escreva ("Hoje é Segunda-feira")
    pare

    caso contrario:
        escreva ("Outros dias!")
    pare
→ }
```

- Assim como na estrutura do "senao", no "caso contrario" não é preciso iniciar e fechar as chaves. Exemplo:

```
caso contrario:
    escreva ("Outros dias!")
pare
```

10 – Operadores Lógicos (OR / AND)

• Operações Lógicas

- São operações efetuadas com os valores booleanos (true ou false, [Verdadeiro ou Falso]). Esse tipo de operação é utilizado nas condicionais SE, e em estruturas de repetição.
- Os operadores lógicos que iremos aprender são os seguintes:
- **AND**, **OR** e o operador **NOT**

- **Operador Lógico do Tipo OR**

- Um operador lógico do tipo OR realiza o teste simultâneo de duas condições. Através do teste, ele devolve "Verdadeiro" se pelo menos um dos elementos for verdadeiro. O operador lógico OR também pode ser chamado de OU.
- Observe a tabela verdade:

Tabela verdade do operador OR

1º valor	Operador	2º valor	Resultado
Verdadeiro	OR	Verdadeiro	Verdadeiro
Verdadeiro	OR	Falso	Verdadeiro
Falso	OR	Verdadeiro	Verdadeiro
Falso	OR	Falso	Falso

- Uma expressão OR é verdadeira se uma das condições for verdadeira
- Confira o exemplo do algoritmo utilizando o operador OR (OU)

```
se (cor == "branco" ou cor == "preto"){
    escreva("O carro tem pintura básica")
}
senao
    escreva("Cor inválida")
```

- No exemplo acima o programa compara se a cor é branco ou preto e assim escreve a mensagem "O carro tem a pintura básica".
- Se a cor for outra qualquer, informa "Cor inválida"
- O operador OU é usado dentro de uma condição, veja que ele fica dentro dos parentes da estrutura SE.

```
se (cor == "branco" ou cor == "preto"){
    escreva("O carro tem pintura básica")
}
senao
    escreva("Cor inválida")
```

Lembre-se: a condição precisa que um operando OU outro sejam verdadeiro, ele já considera o resultado como VERDADEIRO.

- Confira este outro exemplo:

```
se (idade == 18 ou nome == "Bruno"){  
    escreva("Acesso autorizado")  
}  
senao  
    escreva("Acesso negado ao sistema")
```

- Se a entrada do usuário for "18" ou "Bruno" a mensagem "Acesso autorizado" já é emitida no programa, assim, se você digitar por exemplo "18" e o nome "Maria" a resposta ainda sim será de autorizado já que o programa entende que apenas uma das condições é necessária que seja verdadeira para que a mensagem seja exibida.
- Se a idade não for "18" e o nome também não for "Bruno", então é exibido a mensagem "Acesso negado ao sistema".
- Lembre-se: Você não precisa necessariamente criar um código com o operador "=", além dele existem outros que podem ser utilizados em outras estruturas, como a "OU" que está praticando. Operadores:

=, <, >, <=, >=, <>

- Utilize o operador OR quando quiser criar condições onde a situação permita a escolha de "um OU outro" caso

É importante lembrar que as condições podem ser mais de duas, você pode criar um algoritmo que teste se a cor é branco ou preto ou azul ou vermelho, basta incrementar a linha SE.



- **Operador Lógico do Tipo AND**

- Um operador lógico do tipo AND realiza o teste simultâneo de duas condições. Através do teste, ele devolve "Verdadeiro", somente se ambas forem verdadeiras.
- O operador lógico AND também pode ser chamado de E
- Observe a tabela verdade:

Tabela verdade do operador AND			
1º valor	Operador	2º valor	Resultado
Verdadeiro	AND	Verdadeiro	Verdadeiro
Verdadeiro	AND	Falso	Falso
Falso	AND	Verdadeiro	Falso
Falso	AND	Falso	Falso

- Confira o exemplo de algoritmo com a estrutura "AND" (E)

```
se (usuario == "sistema" e senha == "123"){
    escreva("acesso autorizado")
}
senao
    escreva("acesso inválido - fim do programa!")
```

- No exemplo acima o programa compara se o usuário é igual ao dado "sistema" e se a senha é "123", neste caso, ambos precisam ser verdadeiros se o usuário digitar "sistema" e a senha "1010" o resultado será "Acesso inválido – fim do programa!" já que irá cair sobre a estrutura "senao".
- O operador E é usado dentro de uma condição, veja que ele fica dentro dos parentes da estrutura SE.

```
se (usuario == "sistema" e senha == "123"){
    escreva("acesso autorizado")
}
senao
    escreva("acesso inválido - fim do programa!")
```

11 – Praticando E Conhecendo O Operador Do Tipo NOT

• Praticando Algoritmos

- Confira o seguinte algoritmo:

```
preco_produto = 200.00

escreva("Informe o primeiro cupom\n")
leia(cupom1)

escreva("Informe o segundo cupom\n")
leia(cupom2)

se (cupom1 == "CP10" e cupom2 == "CP20"){
    desconto = 0.20
    desconto = preco_produto * desconto
    escreva("Produto com desconto máximo:\n", preco_produto - desconto)
}
senao se (cupom1 == "CP10" ou cupom2 == "CP20"){
    desconto = 0.10
    desconto = preco_produto * desconto
    escreva("Produto com desconto médio:\n", preco_produto - desconto)
}
senao
    escreva("Produto sem desconto:\n", preco_produto)
```

- Na primeira parte do algoritmo você pode ver uma atribuição da variável "preco_produto" com o valor 200.00, essa é uma variável do tipo "real"
- Em seguida, existem as mensagens que solicitam os dados para armazenamento das variáveis "cupom1" e "cupom2"

```
preco_produto = 200.00

escreva("Informe o primeiro cupom\n")
leia(cupom1)

escreva("Informe o segundo cupom\n")
leia(cupom2)
```

- A primeira condicional dele, testa se o cupom1 possui o valor CP10 E se o cupom2 possui o valor CP20

```
se (cupom1 == "CP10" e cupom2 == "CP20"){
```


- Caso o resultado seja VERDADEIRO (duas condições verdadeiras), o desconto será de 20%

```
desconto = 0.20
```

- Senão, se somente o primeiro cupom for CP10 OU se somente o segundo cupom for CP20, então...

```
senao se (cupom1 == "CP10" ou cupom2 == "CP20"){
```

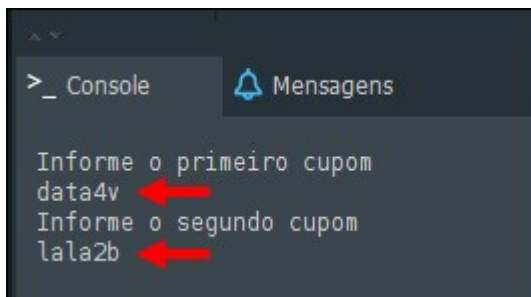
- ... o desconto será de 10%

```
}
senao se (cupom1 == "CP10" ou cupom2 == "CP20"){
    desconto = 0.10
}
```

- Caso contrário, o cliente não ganha nenhum desconto

```
}
senao
    escreva("Produto sem desconto:\n", preco_produto)
```

- Dessa forma, se o usuário usar um código que não existe na programação o resultado deverá exibir o preço real do produto, junto com a mensagem "Produto sem desconto". Confira o exemplo:



```
> _ Console Mensagens

Informe o primeiro cupom
data4v
Informe o segundo cupom
lala2b
```

- Porém, nessa linha do código de exemplo há um erro:

```
senao se (cupom1 == "CP10" ou cupom2 == "CP20"){
```

- Da forma como está construída a condicional SE, só vai fortalecer o desconto de 10%, se o primeiro cupom for o CP10 OU se o segundo cupom for o CP20, porém nesse caso, o usuário também pode inverter a ordem, e colocar o CP20 por primeiro
- Confira o código com o ajuste:

```
senao se (cupom1 == "CP10" ou cupom1 == "CP20" ou cupom2 == "CP10" ou cupom2 == "CP20"){
    desconto = 0.10
    desconto = preco_produto * desconto
    escreva("Produto com desconto médio:\n", preco_produto - desconto)
}
```

senao se (cupom1 == "CP10" ou cupom 1 == "CP20" ou cupom 2 == "CP10" ou cupom2 == "CP20")

- Agora a condicional está testando todos os valores possíveis para o cupom1 e o cupom2
- É importante ajustar também a estrutura "SE", confira:

```
se (cupom1 == "CP10" e cupom2 == "CP20" ou cupom1 == "CP20" e cupom2 == "CP10"){
    desconto = 0.20
    desconto = preco_produto * desconto
    escreva("Produto com desconto máximo:\n", preco_produto - desconto)
}
```

se (cupom 1 == "CP10" e cupom 2 == "CP20" ou cupom1 == "CP20" e cupom2 == "CP10")

Assim não importa a ordem em que o usuário informe os cupons, o programa está reconhecendo se o usuário possui ambos os cupons (operador lógico do tipo **"AND" (e)** ou se o usuário possui pelo menos um dos cupons, ou um ou outro... (operador lógico do tipo **"OR" (ou)** ou se o usuário não possuir nenhum cupom válido, apresentando a mensagem de produto sem desconto.

• Operador Lógico do tipo NOT

- Um operador lógico do tipo NOT é utilizado para realizar a negação de um valor. Ele não possui uma tabela verdade aplicável, pois é mais simples. Esse operador apenas faz a negação de um valor, por exemplo:

Operador A tem o valor "Verdadeiro" (true)

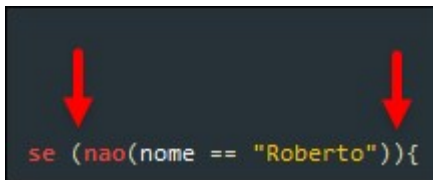
- Se aplicar o operador NOT... ficaria assim: Not Operador A = Falso (false)
- Ele sempre retorna o contrário, vejamos outro caso.
- A agora tem o valor "false". Se utilizarmos: NOT A, então A passa a ter o valor "true"



- Confira o algoritmo com o operador NOT (nao):

```
se (nao(nome == "Roberto")){  
    escreva("Condição verdadeira")  
}  
senao  
    escreva("Condição falsa")
```

- Neste exemplo o algoritmo testa se o nome digitado não é Roberto, se não for, então é verdadeiro e exibe a mensagem "Condição verdadeira", se o nome for "Roberto", exibe a mensagem "Condição falsa".
- É importante usar o operador NOT dentro dos parênteses de uma condição, como na estrutura SE, mostrada neste exemplo:



```
se (nao(nome == "Roberto")){
```

- Na programação o operador NOT é um recurso bastante utilizado. Por exemplo, para exibir a lista de clientes de uma loja, essa listagem só pode ser exibida caso o banco de dados NÃO esteja vazio. Se ele estiver vazio, nada será mostrado.
- Sempre que for necessário fazer a negação de um valor de uma variável, utilize o operador NOT (nao)

- Ele sempre vai inverter o teste, se era VERDADEIRO, será FALSO... e assim por diante.

12 – Estruturas de Repetição / Enquanto

• O Que é Uma Estrutura de Repetição?

- Estruturas de repetição são estruturas que proporcionam que um mesmo bloco de código do algoritmo seja processado "n" vezes, de acordo com a necessidade estipulada.
- Uma estrutura de repetição possui sempre uma condição de controle, isto é, uma espécie de teste lógico que verifica se o bloco de código deve ser repetido mais uma vez, ou o programa já pode encerrar a repetição
- Esse tipo de estrutura é muito utilizado em programação, para que o algoritmo possa efetuar repetições, sem a necessidade de linhas adicionais desnecessárias no programa
- Veja este exemplo de algoritmo:



```
funcao inicio()
{
    inteiro numero
    inteiro soma

    numero = 1
    soma = 5

    numero = numero + soma
    escreva(numero, "\n")
}
```

- O algoritmo possui 2 variáveis do tipo "inteiro". Quando o programa se inicia, elas recebem valores.

```
funcao inicio()
{
    inteiro numero
    inteiro soma

    numero = 1
    soma = 5
```

Recebem valores

- Em seguida acontece uma operação:

```
numero = numero + soma
escreva(numero, "\n")
```

- Esse algoritmo do exemplo é do tipo estrutura sequencial. Quantas vezes ele for executado, sempre vai retornar o valor "6", até que se modifique a sua estrutura. Ele não possui estruturas para decisão (condicional SE) e por isso sempre será o mesmo
- Nesse algoritmo, temos um valor para a variável numero (vale 1) inicialmente. Em seguida, ele executa uma operação de soma, usando o valor da variável soma (vale 5), logo $1 + 5 = 6$.
- Caso queira repetir a operação de somar o valor "5" ao conteúdo da variável numero. Essa repetição terá que ser feita por 5 vezes.
- Logo, ao somar 5 vezes o valor "5"... teremos um resultado final $= 1 + 5 + 5 + 5 + 5 + 5 = 26$

```
numero = 1
soma = 5

numero = numero + soma
escreva(numero, "\n")

numero = numero + soma
escreva(numero, "\n")

numero = numero + soma
escreva(numero, "\n")

numero = numero + soma
escreva(numero, "\n")

numero = numero + soma
escreva(numero, "\n")
```

- Confira o exemplo do código:

- O programa continua sendo uma estrutura sequencial, ou seja, ele sempre vai retornar o valor 26, pois não possui decisões no código.
- Imagine se você precisasse repetir esse cálculo por 200 vezes: teria de copiar/colar 200 vezes o mesmo bloco, o código do algoritmo ficaria muito extenso



• Estrutura de Repetição Enquanto...Faça

- Esta estrutura repete uma sequência de comandos enquanto uma determinada condição (especificada através de uma expressão lógica) for satisfeita. Enquanto a expressão lógica for true (verdadeira) o bloco de comandos será repetido e reiniciado ao início. O bloco só será interrompido, ou finalizado, quando a expressão lógica for false (falso).
- Confira um exemplo do algoritmo com a repetição sem a cópia dos blocos:

```
numero = 1
soma = 5
contador = 1

enquanto (contador < 10){
    escreva(contador)
    contador = contador + 1
}
```

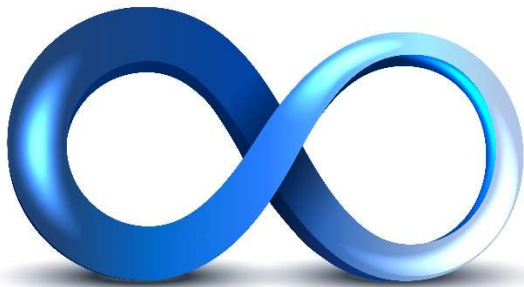
- Essa é a condição lógica da estrutura Enquanto...

```
enquanto (contador < 10) {
```

- O laço irá se repetir enquanto a variável contador for menor que 10 (<10)
- Essa é a linha mais importante da repetição:

```
    escreva(contador)
    contador = contador + 1 ←
```

- Ela é chamada linha de incremento, ela que faz com que em cada execução do laço, a variável "contador" some +1 em seu valor. Se não for colocado o incremento, o laço vai se repetir para sempre, sem nunca terminar



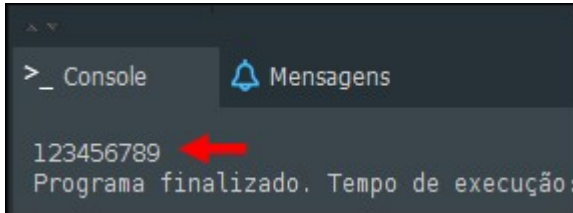
- Compare a diferença com o bloco anterior, sem a estrutura "enquanto":

```
funcao inicio()
{
    inteiro numero
    inteiro soma

    numero = 1
    soma = 5

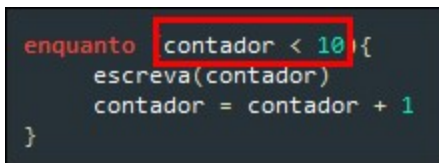
    numero = numero + soma
    escreva(numero, "\n")
}
```

- O resultado da execução do algoritmo com a estrutura enquanto será assim:



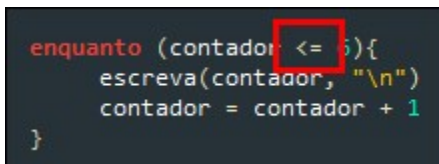
```
> _ Console Mensagens
123456789
Programa finalizado. Tempo de execução:
```

- O programa exibirá até o número 9, já que a condição testa se contador é menor que 10, e não menor ou igual, assim o número 10 não é exibido.



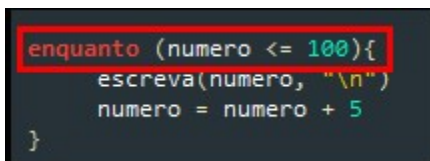
```
enquanto (contador < 10){
    escreva(contador)
    contador = contador + 1
}
```

- Para ajustar a estrutura para exibir o número "10" neste exemplo, basta deixar assim:



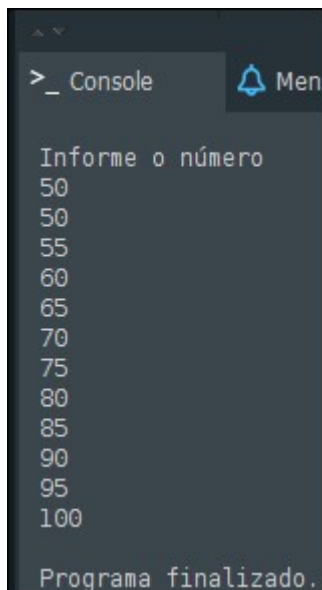
```
enquanto (contador <= 10){
    escreva(contador, "\n")
    contador = contador + 1
}
```

- Neste exemplo o incremento do laço (+1) está sendo de um em um, porém, você pode modificá-lo para que seja com outro valor e assim a repetição será baseada neste novo incremento até que o contador atinja seu limite, neste caso 10.
- Esse tipo de estrutura de repetição é muito boa para calcular resultados rápidos já que o algoritmo consegue exibir em poucos segundos até mesmo repetições em grande escala como até o número 100, 1000 ou o número que você desejar. Exemplo:



```
enquanto (numero <= 100){
    escreva(numero, "\n")
    numero = numero + 5
}
```


- Exemplo de exibição da execução com o incremento de 5 em 5, iniciando em 50:



```
> _ Console  Men.  
  
Informe o número  
50  
50  
55  
60  
65  
70  
75  
80  
85  
90  
95  
100  
  
Programa finalizado.
```

13 – Praticando A Estrutura De Repetição Enquanto

• Praticando A Estrutura de Repetição / Enquanto... Faça

- Sempre que precisar que um bloco de instruções seja repetido, até que se atinja uma determinada condição, deverá utilizar estruturas de repetição
- Será construído um algoritmo para simular uma caderneta de poupança. No algoritmo deverá receber:
 - **Valor do depósito a cada mês**
 - **Valor final que deseja atingir**
- As regras são as seguintes:
 - O primeiro mês não tem correções, pois o dinheiro só é corrigido após 30 dias.
 - A partir do segundo mês, o saldo que existir na poupança deverá ser atualizado com a seguinte função:

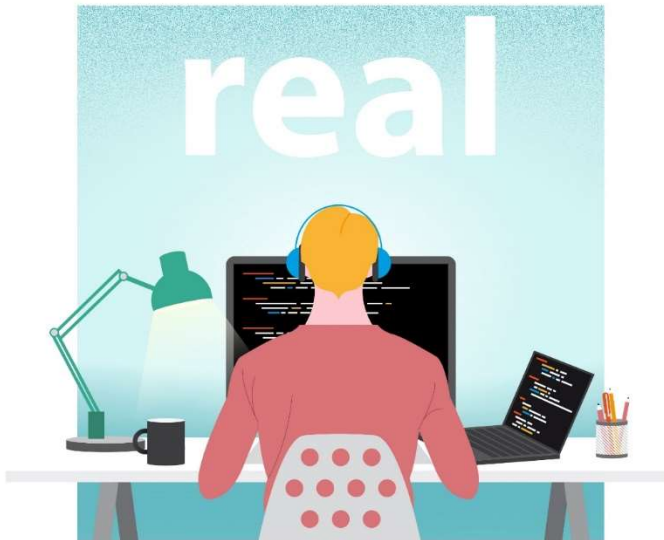
- ***saldo_atual + (taxa de rendimento * saldo_atual)***
- E depois somar com o valor depositado nesse mês
- Com isso, iremos simular o depósito de XX reais por mês, e o dinheiro rendendo de acordo com uma taxa estipulada
- O programa deverá mostrar em quantos meses o valor final será atingido, e qual será o total atingido
- O funcionamento do programa é assim:
 - Usuário irá informar o valor que vai depositar a cada mês (valor fixo), e quanto quer atingir. O programa vai informar em quanto tempo o montante será atingido, em meses
 - A estrutura de repetição é utilizada nesse exemplo para calcular o rendimento até que o valor desejado seja atingido
- As variáveis devem ser declaradas dessa forma, neste exemplo:

```
funcao inicio()
{
    real saldoatual
    real deposito
    real valoratingir
    real taxa

    inteiro mes
}
```

- Dicas sobre as variáveis:
 - **saldoatual:** variável que irá controlar quanto a poupança possui de saldo
 - **deposito:** valor a ser recebido do usuário, indica o valor que ele vai depositar a cada mês
 - **valoringir:** é o valor a ser atingido, o usuário informa a meta a ser atingida e a cada depósito será comparado se o valor já foi atingido
 - **taxa:** é a taxa de rendimento da poupança, nesse algoritmo ela será fixa para todos os meses
 - **mes:** ela vai controlar o número de meses necessários para atingir a meta de valor. A cada execução do laço, o mês será incrementado em +1 e assim ao final saberemos o tempo necessário para atingir a meta

- As variáveis que tratam de valor monetário estão declaradas com o tipo "real", pois podem conter valores com casa decimal. Já a variável mes é do tipo "inteiro" pois ela vai realizar a contagem total de meses, em número do tipo inteiro já é suficiente



- Confira outra etapa do algoritmo:

```
// Taxa média de rendimento da poupança (ao mês)
taxa = 0.6

// Declarando os valores das variáveis
saldoatual = 0.0
mes = 0

escreva("Informe o valor do depósito a cada mês\n")
leia(deposito)

escreva("Informe o valor que deseja atingir\n")
leia(valoratingir)
```

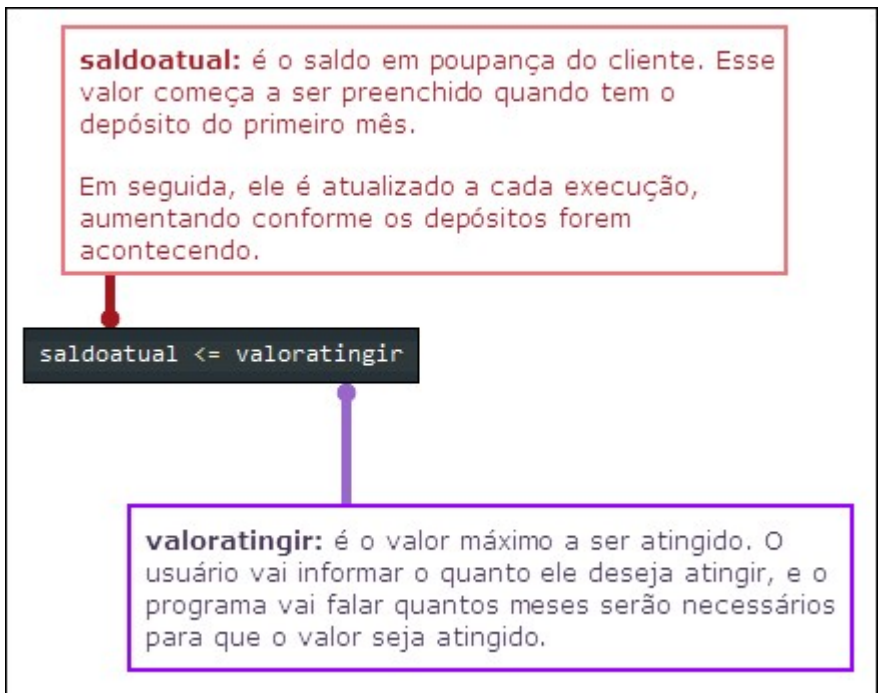
- Na linha acima, está sendo atribuído um valor para a variável taxa
- A taxa é usada como rendimento da poupança com base no mês de Janeiro de 2011

-

- A taxa será fixa, para todos os meses testados. O valor 0.6 representa menos de 1%, sendo 0,6%. A cada mês, o dinheiro rende 0,6%
- **ATENÇÃO:** No programa Portugol Studio, o separador decimal é o ponto
- Fique atento para digitar **valores decimais**. Exemplo: (0,6 = 0.6)
- Para o algoritmo de teste, veja a estrutura “enquanto” criada:

```
// Laço que se repete até que o valor desejado seja atingido
enquanto (saldoatual <= valoratingir){
```

- Analise o código:



- Como o objetivo do algoritmo é somar o rendimento mês a mês, é necessário adicionar o incremento aos meses com valor +1

```
// Laço que se repete até que o valor desejado seja atingido
enquanto (saldoatual <= valoratingir){
    // Incremento para contar os meses
    mes = mes + 1
```

- Em seguida é criada a estrutura SE:

```

mes = mes + 1

// Primeiro mês é um cálculo
se (mes == 1){
  saldoatual = deposito
}
// Segundo mês é outro cálculo
senao
  saldoatual = saldoatual + saldoatual * (taxa/100) + deposito

```

- Note que no primeiro mês o saldoatual recebe o valor do depósito. Isso acontece, pois a poupança começa a render 30 dias após o primeiro depósito.
- A partir do segundo mês (segunda execução do laço), ele já desvia o código para essas linhas:

```

// Segundo mês é outro cálculo
senao
  saldoatual = saldoatual + saldoatual * (taxa/100) + deposito

```

- Confira a explicação:

Aqui estamos **atualizando o saldo da poupança** do cliente.

A variável **saldoatual** <- recebe o valor que ela já tem
 + o cálculo da porcentagem de juros que é (**saldo atual * 0,6/100 (0,6%)**)
 + e depois soma com o valor que está sendo depositado agora.

- Por fim, é só exibir os resultados para o usuário, elabora a exibição de textos junto à variáveis. Exemplo:

```

senao
  saldoatual = saldoatual + saldoatual * (taxa/100) + deposito
}

// Apresentação do resultado na tela
escreva("*** Para atingir o valor ", valoratingir, " você levará ", mes, " meses ***\n")
escreva("*** Valor atingido: ", saldoatual, " ***\n")

```

- Não se esqueça de concatenar (,) para adicionar as variáveis junto ao texto

Biblioteca

- A biblioteca é um arquivo que contém códigos pré-programados e que pode ser importada para sua aplicação

- Uma biblioteca pode conter diversas funções diferentes que, ao importadas, ficarão disponíveis para uso em seu código
- Neste exemplo, utilizaremos a biblioteca "Matematica" para a formatação das casas decimais, isso será feito através da função "arredondar"
- Com a formatação das casas decimais você pode ajustar a exibição de um número decimal, deixando ele com mais ou menos casas decimais
- No algoritmo, a biblioteca deve ser adicionada antes da função início. Exemplo:

```
programa
{
    inclua biblioteca Matematica

    funcao inicio()
    {
        real saldoatual
```

- Após incluir a biblioteca, é necessário incluir a função, confira o exemplo da função "arredondar" já adicionada ao código:

```
// Apresentação do resultado na tela
escreva("*** Para atingir o valor ".valoratueir. " você levará ".mes, " mes
escreva("*** Valor atingido: ", Matematica.arredondar(saldoatual,2), " ***\n")
```

Matematica.arredondar(saldoatual, 2)

- *saldoatual* = nome da variável e 2 = ao número de casas decimais que será exibido na tela
- Confira um explicação mais detalhada:

```
na tela
valor ", 1 valorating 2 " você 3 ,ará 4 m
", Matematica.arredondar(saldoatual,2),
```

- (1) **Azul**: apresenta o nome da biblioteca (Matematica)
- (2) **Vermelho**: apresenta o nome da função existente na biblioteca (arredondar)
- (3) **Verde**: apresenta a variável (saldoatual)
- (4) **Roxo**: quantidade de casas decimais a serem exibidas

- Veja um exemplo de exibição do número decimal com duas casas, como programado no algoritmo:

```
Informe o valor que deseja atingir
7000
*** Para atingir o valor 7000.0 você levará
*** Valor atingido: 7032.52 ***
```

- Sem a adição da biblioteca e formatação do código o número poderia ser exibido de maneira mais extensa, dificultando o entendimento do usuário.
- Exemplo sem a formatação:

```
7000
*** Para atingir o valor 7000.0 você levará
*** Valor atingido: 7032.51910997492 ***
```



14 – Estrutura de Repetição / Para-Até-Faça

• Relebrando Estruturas de Repetição

- As estruturas de repetição proporcionam que um mesmo bloco de código do algoritmo, seja processado "n" vezes, de acordo com a necessidade estipulada.
- Uma estrutura de repetição possui sempre uma condição de controle, isto é, uma espécie de teste lógico que verifica se o bloco de código deve ser repetido mais uma vez, ou o programa já pode encerrar a repetição
- Esse tipo de estrutura é muito utilizado em programação, para que o algoritmo possa efetuar repetições, sem a necessidade de linhas adicionais desnecessárias no programa



• Estrutura Para-Até-Faça

- Esta estrutura repete uma sequência de comandos de acordo com os parâmetros especificados. Os parâmetros vão orientar o "início" e o "final" da repetição. Por exemplo, repetir um bloco de comandos de 1 até 10 (10 vezes).
- Nessa estrutura o incremento é automático, isto é, não precisamos somar +1 no incremento, como na estrutura do "Enquanto...Faça"
- Confira o código de exemplo:

```
funcao inicio()
{
    inteiro j

    para (inteiro c=1; c<=10; c++)
    {
        j=c
        escreva(j, "\n")
    }
}
```


- O laço será executado com a variável "j" iniciando com o valor 1 e indo até o valor 10
O laço do tipo "Para-Até-Faça" faz o incremento sozinho, de 1 em 1
- O laço possui um início e um final, os comando que ficam no meio dele é que serão repetidos "n" vezes
- O laço se inicia em "para" e termina na chave que fecha a estrutura:

```
Início para (inteiro c=1; c<=10; c++)
{
    j=c
    escreva(j, "\n")
Final }
```

- Para este código de exemplo a execução do algoritmo resultaria nestes valores:

```
> _ Console
1
2
3
4
5
6
7
8
9
10
Programa finalizado.
```

- Nesse algoritmo, o laço se repete até que a variável j atinja o valor 10, assim o último valor exibido na lista de números é o 10.
- Modificando o código pode-se trabalhar com outros valores, a vantagem desta estrutura de repetição é também a rápida forma como o algoritmo calcula e exibe os resultados em tela

```
para (inteiro c=1; c<=100; c++)
{
    j=c
    escreva(j, "\n")
}
```

- Modificando o valor para "100" este seria o resultado:

```
>_ Console
91
92
93
94
95
96
97
98
99
100

Programa finalizado.
```

- O programa exibiria repetições de 1 até o número 100 de um em um.

• Diferenças Entre Estruturas de Repetição

- **Estrutura Enquanto...Faça:** Nessa estrutura devemos ter uma variável para controlar o incremento do laço a cada passagem dele, a variável do incremento deve ter seu valor atualizado. É uma estrutura geralmente utilizada para exibir registros de banco de dados na tela, ou para cálculos de valores, como foi exemplificado no algoritmo que calculava o tempo de retorno do investimento na poupança
- **Estrutura Para-Até-Faça:** Nesse outro tipo de estrutura de repetição, é definido o valor de início e de final da repetição. Nesse tipo o incremento é feito de maneira automática, através de uma variável do tipo ""inteiro"" que controla o avanço do laço. Essa estrutura é mais indicada quando você já sabe quantas vezes terá de repetir o código, por exemplo, calcular sempre algo que se repita por "3 vezes".

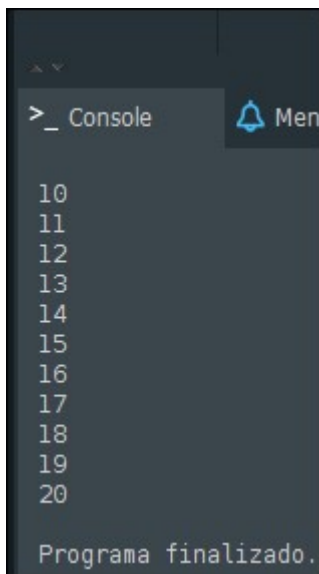
- Muitos algoritmos podem ser resolvidos da mesma maneira, pois em alguns casos é possível resolver um mesmo problema usando ambas as estruturas de repetição.



- Então não é correto dizer que se deve usar uma ou a outra repetição, a dica é utilizar "Enquanto...Faça" para repetições em que não se saiba quantas vezes será executado, e utilizar a estrutura "Para-Até-Faça" quando já se sabe quantas repetições serão executadas no laço
- Confira o exemplo de algoritmo:

```
para (inteiro c=10; c<=20; c++)  
{  
    j=c  
    escreva(j, "\n")  
}
```

- Ao executar o programa se tem o seguinte resultado:



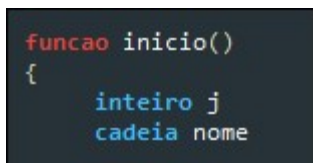
A screenshot of a console window with a dark background. At the top, there's a tab labeled "> _ Console" and a bell icon with the text "Men". Below the tab, the numbers 10 through 20 are listed vertically. At the bottom of the console, the text "Programa finalizado." is displayed.

```
> _ Console  Men

10
11
12
13
14
15
16
17
18
19
20

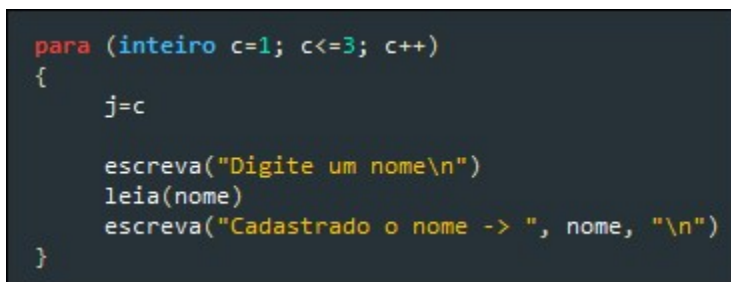
Programa finalizado.
```

- O laço se repetiu por 10 vezes, porém o seu valor foi 10 e ele repetiu até atingir 20
- Neste novo exemplo, será incrementada uma nova variável com a descrição nome e tipo "cadeia"



```
funcao inicio()
{
    inteiro j
    cadeia nome
```

- O código deve ser reajustado para o novo exemplo:



```
para (inteiro c=1; c<=3; c++)
{
    j=c

    escreva("Digite um nome\n")
    leia(nome)
    escreva("Cadastrado o nome -> ", nome, "\n")
}
```

- Note que o laço será repetido 3x

```
para (inteiro c=1; c<=3; c++)
{
    j=c
```

- Como está o programa irá armazenar 3 nomes sobre a variável "nome" e exibir assim:

```
Digite um nome
Ronaldo
Cadastrado o nome -> Ronaldo
Digite um nome
Maria
Cadastrado o nome -> Maria
Digite um nome
Fernanda
Cadastrado o nome -> Fernanda

Programa finalizado. Tempo de exe
```

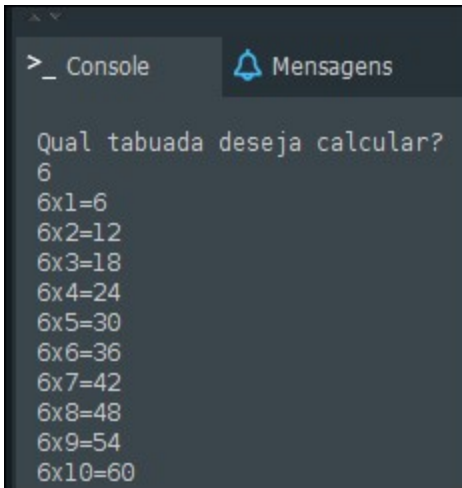
- Utilize a estrutura de repetição "Para-Até-Faça", quando você já tiver a certeza de quantas vezes um laço deve ser executado.
- **Por exemplo:** Se você precisa construir um algoritmo que executa um cálculo por 2 vezes seguidas, então você pode usar o Para-Até-Faça
- Para praticar a estrutura **Para-Até-Faça** iremos criar um algoritmo para calcular a "tabuada" de um número informado pelo usuário



- Para criar esse algoritmo, podemos utilizar apenas 2 variáveis: uma para armazenar qual número será calculado e a segunda para controlar o número de execuções do laço.
- Confira este outro exemplo de algoritmo:

```
para (inteiro c=1; c<=10; c++)  
{  
    j=c  
  
    escreva(numero, "x", j, "=", j * numero, "\n")  
}
```

- Com a estrutura para e as variáveis prontas, a execução do programa deve exibir a tabuada do 1 ao 10. Ao executar, veja o exemplo em que o usuário utiliza o número 6 como entrada:



```
>_ Console Mensagens  
  
Qual tabuada deseja calcular?  
6  
6x1=6  
6x2=12  
6x3=18  
6x4=24  
6x5=30  
6x6=36  
6x7=42  
6x8=48  
6x9=54  
6x10=60
```

- Fazendo uma análise mais detalhada desse trecho:

```
escreva(numero, "x", j, "=", j * numero, "\n")
```

- Essa linha utiliza o comando "escreva" para exibir o resultado da tabuada, tudo que está entre aspas é texto, ou seja, é informação fixa que aparece na tela.
- Tudo que está fora das aspas são nomes de variáveis. São informações contidas dentro de variáveis

- Aqui um exemplo de texto entre aspas... esses 2 textos criam o sinal de "=" e o "x"

```
escreva(numero, "x", j, "=", j * numero, "\n")
```

- E aqui as variáveis sendo escritas:

```
escreva(numero, "x", j, "=", j * numero, "\n")
```

número que o usuário digitou **Contador do laço (que vai de 1 até 10)** **j * numero -> resultado no cálculo da tabuada**

- Confira outro algoritmo de exemplo:

```
funcao inicio()
{
    para (inteiro j=1; j<=200; j++)
    {
        escreva ("Repetindo ", j, " vezes\n")
    }
}
```

- Como de costume, a variável j vai controlar as execuções da repetição
- O laço irá se repetir de 1 até 200, ou seja, 200 vezes

```
para (inteiro j=1; j<=200; j++)
{
```

- A cada execução do laço, escreve na tela o valor atua da variável j

```
escreva ("Repetindo ", j, " vezes\n")
```

- Na estrutura Para-Até-Faça o incremento por padrão é de 1. A cada execução, a variável automaticamente acrescenta +1 e assim o laço continua até que se atinja o parâmetro final, que é a instrução "Até". Essa é uma característica dessa estrutura de repetição



- No exemplo da estrutura Enquanto...Faça, o laço era executado enquanto uma condição fosse verdadeira. Nesse outro tipo de estrutura, na maioria dos casos, temos que ter uma variável controlando a execução, e essa variável geralmente é incrementado de forma manual, fazendo assim:

variavel = variavel + valor

- A estrutura repetição Para-Até-Faça possui uma instrução chamada Passo. Com essa instrução é possível determinar o quanto o laço irá avançar em cada "passagem" ou em cada "execução"
- Por padrão, o Passo é sempre +1, por isso esse laço já incrementa de forma automática. Alterando a instrução Passo podemos aumentar ou diminuir o número de execuções do laço
- Confira o exemplo do algoritmo com a instrução "Passo"

```
para (inteiro j=1; j<=200; j+=10)
{
    escreva ("Repetindo ", j, " vezes\n")
}
```

FIM DA APOSTILA

Método CGD ® - Todos os direitos reservados.

Protegidos pela Lei 5988 de 14/12/1973.

Nenhuma parte desta apostila poderá ser copiada sem prévia autorização.

[illegible]

[illegible]