Nivelamento

Lógica de Programação

Residência em TIC/Software Serratec 2021.2

Sejam Bem-Vindos!

Objetivos do nivelamento

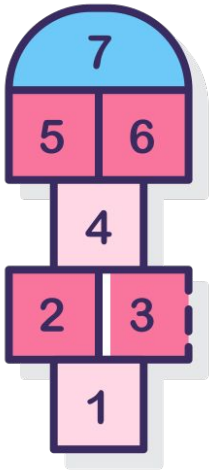
- Introdução na construção de Algoritmos computacionais
- Nivelar o conhecimento
- Desenvolver capacidades e habilidades necessárias para o aproveitamento do curso
- Estimular o trabalho em equipe

O que vamos aprender?

- Algoritmos
- Variáveis e constantes
- Lógica booleana (E, OU, NOT)
- Árvore de decisão
- Estruturas de Laço
- Funções (Sub-Rotinas)
- Recursividade
- Estrutura de dados (Vetor, Matriz, Fila, Pilha)
- Introdução ao Versionamento de código com GIT

Algoritmo

- Sequência finita de passos que levam à execução de uma tarefa
- Algoritmos são muito comuns no nosso dia a dia, sendo de TI ou não



Aquele bolinho da vovó

- **Dinâmica 1** : Em grupos, vocês devem escrever em um arquivo de texto as etapas de uma receita de bolo.
- Dica : Lembre-se que uma receita geralmente é separada em ingredientes e modo de preparo

Como ficou a receita? Primeira versão

```
Receita da vovó {
        // Ingredientes
        Açúcar = 2 xícaras;
        Farinha = 2,5 xícaras;
        Ovos = 4;
        Óleo = 0,5 xícaras;
        Fermento = 1 colher;

        //Modo de preparo
        Misturar os ingredientes
        Colocar no forno
        Assar por 1 hora
}
```

Como ficou a receita? Versão com a turma

Receita da vovó {

// Ingredientes

Açúcar = 2 xícaras;

Farinha = 2,5 xícaras;

Manteiga = 4 colheres

Ovos = 4;

Leite = 0,5 L;

Fermento = 1 colher chá;

Chocolate em pó = 5 colheres de sopa

}

//Modo de preparo

- Pré aquecer o forno a 220ºC
- Pegar Liquidificador
- Inserir Ovos
- Inserir Manteiga
- Inserir Açucar
- Ligar o Liquidificador e Bater até ficar homogêneo
- Desligar o Liquidificador
- Inserir Leite
- Farinha
- Chocolate em pó
- Ligar o Liquidificador e bater até ficar homogêneo
- Desligar liquidificador
- Inserir fermento
- Bater a mistura a mão
- Pegar Forma

Algoritmo

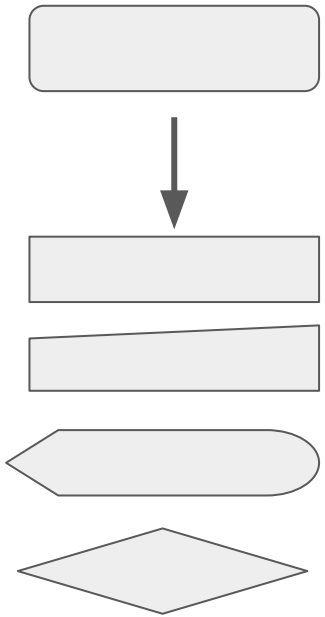
```
Algoritmo AtravessarRua
    Olhar para a direita
    Olhar para a esquerda
    Se estiver vindo carro
        Não Atravesse
    Senão
        Atravesse
    Fim-Se
Fim-Algoritmo
```

```
Algoritmo AtravessarRua
    Olhar para a esquerda
    Olhar para a direita
    Se não estiver vindo carro
        Atravesse
    Senão
        Não Atravesse
    Fim-Se
Fim-Algoritmo
```

```
Algoritmo AtravessarRua
    Atravesse
    Se estiver vindo carro
        Olhar para direita
    Senão
        Olhar para esquerda
    Fim-Se
    Não Atravesse
Fim-Algoritmo
```

Fluxograma - Outra forma de representação

- É uma forma universal de representação, pois se utiliza de figuras geométricas para ilustrar passos a serem seguidos para a resolução de problemas



Indica o início ou fim do algoritmo

Indica o sentido do fluxo de dados

Indica cálculos e atribuições de valores

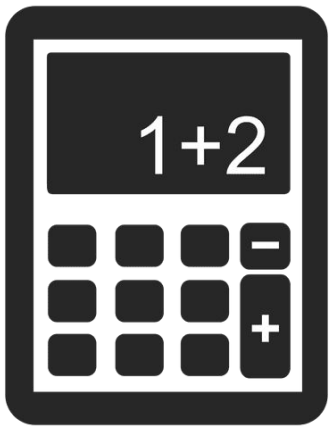
Representa a entrada de dados

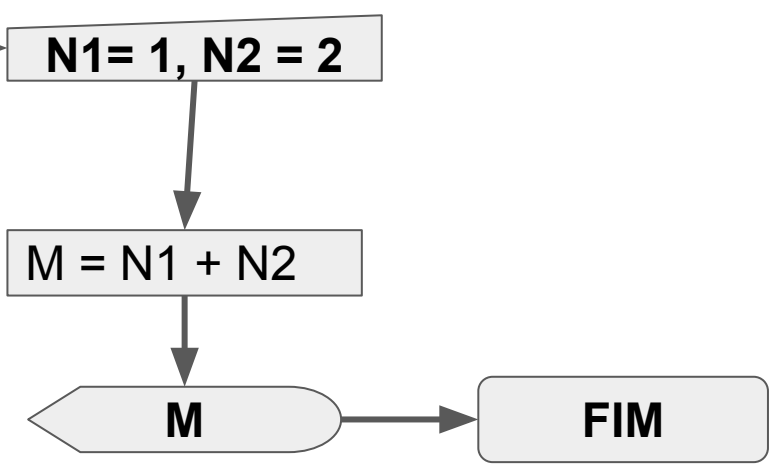
Representa a saída de dados

Tomada de decisão

Fluxograma

Exemplo : Soma de 2 números





Programa

- **Algoritmo** escrito em uma **linguagem** de programação

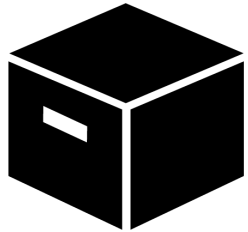
Linguagem natural

Linguagem de máquina

Programador

Linguagem de programação

Compilador



E qual linguagem usaremos neste curso?

- Neste nivelamento o foco é entender os princípios da programação. Assim, utilizaremos a ferramenta Portugol Studio, que possui uma linguagem própria que aproxima a linguagem de programação ao português!
- Ferramenta que podemos criar algoritmos com uma Linguagem baseada em pseudocódigo

O que é Pseudocódigo?

- É uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples de tal forma que possa ser entendida por qualquer pessoa sem necessidade de conhecer a sintaxe de nenhuma linguagem de programação.
- O objetivo é trabalhar a lógica para a construção de algoritmos
- Iniciar o desenvolvimento de códigos que se aproxime do padrão visto em linguagens como C#, JAVA, Javascript, Python

Estrutura inicial de um código em Portugol

```
programa
{
	/* Declaração de variáveis, estruturas e outras funções */

	funcao inicio ()
	{
		/*Execução da função início*/
	}
}
```

Estrutura inicial de um código em Portugol

```
programa
{
        // Declaração de variáveis, estruturas e outras funções

        inteiro numero1 = 1
        inteiro numero2 = 2

        funcao inicio ()
        {
                escreva(numero1 + numero2)
                … //define outros passos
        }
}
```

Recapitulando

Já aprendemos…

- Algoritmo
- O que é um Programa
- Representações de Algoritmos
  - Escrita natural
  - Fluxograma
- A Ferramenta que vamos utilizar
- Pseudocódigo

Vamos instalar o Portugol Studio?

http://lite.acad.univali.br/portugol/

Vamos explorar o Portugol Studio

Nosso primeiro programa: Olá mundo!

Dinâmica 2 : Execute no Portugol Studio o código : Olá Mundo

- O que esse código faz?
- Quais dificuldades vocês tiveram em entender este trecho de código?

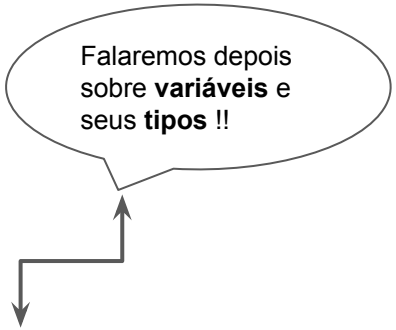
Vamos executar mais alguns programas

- ‘Número digitado’
- ‘Seu Nome’

- O que esses códigos fazem?
- A partir deles, escreva um novo programa que recebe seu nome e escreve o seu nome na tela

Operações de entrada e saída

Ficou parecido com a solução abaixo?




```
programa
{
	funcao inicio ()
	{
		cadeia nome   /*cadeia se refere ao tipo da variável que é uma cadeia de caracteres*/

		escreva("Digite seu nome: ")
		leia(nome)
		escreva("Seu nome é : ", nome ,"\n")
	}
}
```

Por que **entrada** e **saída**?

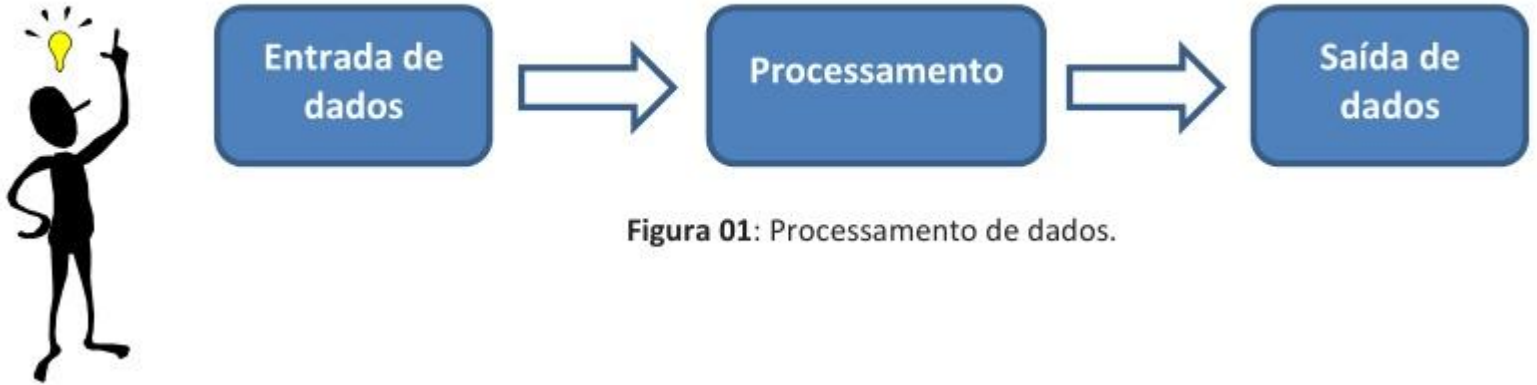


Figura 01: Processamento de dados.

Por que **entrada** e **saída**?

Quando escrevemos :

```
cadeia nome
leia(nome)
```

leia é uma operação de **entrada** que permite que o que escrevemos no teclado seja **lido e armazenado** na variável “**nome”.** Logo estamos **entrando** com uma informação no programada durante sua execução.

Por que **entrada** e **saída**?

Quando escrevemos :

```
cadeia nome = Raul
escreva("Meu nome é: ", nome)
```

escreva é uma operação de **saída** que permite que a informação escrita entre seus parênteses “**()**” seja apresentado na tela do computador, logo como é uma informação de apresentação, entendemos como uma informação de **saída.**

Por que **entrada** e **saída**?

Quando escrevemos :

```
cadeia nome = Raul
escreva("Meu nome é: ", nome)
```

escreva é uma operação de **saída** que permite que a informação escrita entre seus parênteses **"()"** seja apresentado na tela do computador, logo como é uma informação de apresentação, entendemos como uma informação de **saída.**

Dúvidas?

Comandos de **entrada** e **saída** de dados no Portugol

cadeia nome

leia(nome) - Permite a entrada de dados

escreva(nome) - Permite a saída de dados

Voltando ao programa anterior...

```
programa
{
        funcao inicio ()
        {
                cadeia nome   /*cadeia se refere ao tipo da variável que é uma cadeia de caracteres*/

                escreva("Digite seu nome: ")
                leia(nome)
                escreva("Seu nome é : ", nome ,"\n")
        }
}
```

Ficou faltando entendermos o que são as **variáveis!**

Variáveis e constantes

- **Variáveis** e **constantes** representam uma posição na memória, onde pode ser armazenado um **único dado** (valor)
- Possuem **tipo**, **nome** e um **valor**
- A diferença entre variáveis e constantes é que enquanto o **valor da variável pode mudar** durante a execução do programa o **valor da constante não**.

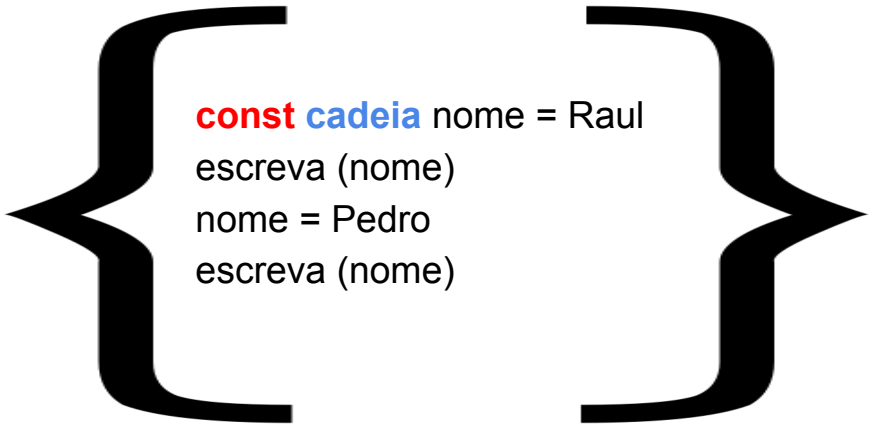
```
cadeia nome = “Raul”
escreva (nome)
nome = “Pedro”
escreva (nome)
```

Variáveis e constantes

- Em algumas linguagens (incluindo Portugol) as variáveis podem ser **tipadas**, ou seja, aceitam apenas valores referentes ao seu tipo, representado antes do nome da variável:
  - Tipos de variáveis na linguagem do Portugol Studio
    - inteiro : Número inteiros -> 1 ; 2 ; 3
    - real : Números de ponto flutuante -> 1.1 ; 3.14 ; 10.3
    - cadeia : Cadeia de caracteres -> “Adoro estudar programação”
    - caracter : Apenas um caractere -> ‘A’, ‘1’
    - logico : Caractere booleano : verdadeiro, falso

Variáveis e constantes

- Finalmente, para declarar uma **constante** basta colocar o indicador **const** antes da declaração da constante



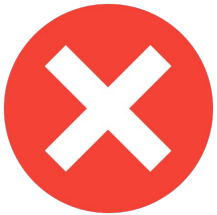
O que será impresso na tela?

Variáveis e constantes

- Finalmente, para declarar uma constante basta colocar o indicador **const** antes da declaração da constante

```
const cadeia nome = Raul
escreva (nome)
nome = Pedro
escreva (nome)
```

O que será impresso na tela?



ERRO!! Pois não podemos alterar o valor de uma constante =(

Até aqui, como estamos?

- Já aprendemos:
  - Valores
    - Como nos organizamos como turma
    - Valores em trabalho em equipe e desenvolvimento de software
  - Conteúdo
    - O que é um algoritmo
    - O que é um programa
    - Qual ferramenta utilizaremos
    - Operações de entrada e saída
    - O que são variáveis e constantes

Vamos fixar este conteúdo juntos

O que mais precisamos aprender :

- Desvios condicionais (se e senão)
- Operadores lógicos (E, OU …)
- Laços de repetição (enquanto)
- Estruturas de dados (Vetores, Matrizes, Filas e Pilhas)
- Subrotinas (Funções)
  - Recursividade
  - Bibliotecas

Com menos teoria e mais prática pois programar é treinar bastante!! =D

Lembram do exemplo de escrever e imprimir?

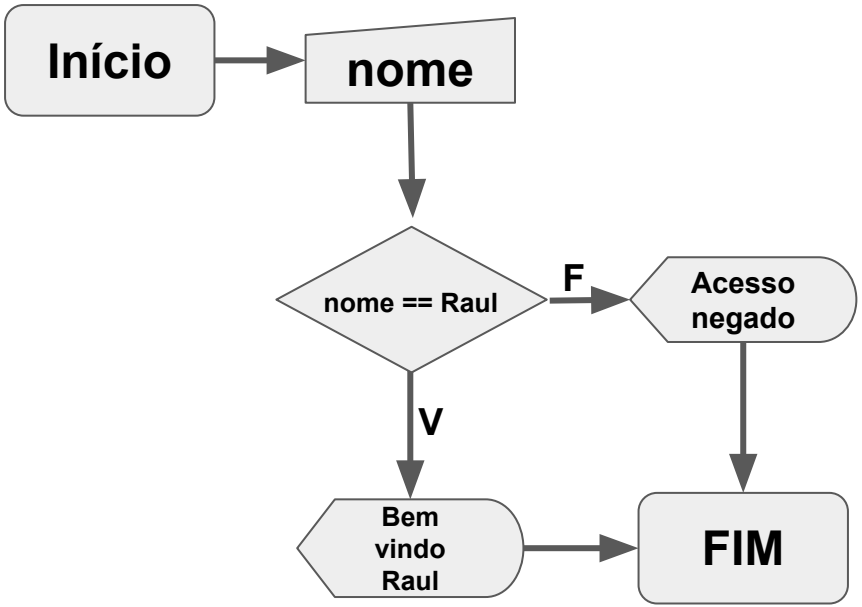
```
programa
{
        funcao inicio ()
        {
                cadeia nome
                escreva("Digite seu nome: ")
                leia(nome)
                escreva("Seu nome é : ", nome ,"\n")
        }
}
```

Vamos transformar este programa para ser capaz de **validar** o nome do usuário e conceder acesso ao sistema!
Apenas o usuário cadastrado poderá entrar no sistema.

Lembram do exemplo de escrever e imprimir?

Para resolver este problema podemos usar os condicionais **se** e **senao** (**if** e **else** do inglês)**.**

```
programa
{
        funcao inicio () {
                cadeia nome
                escreva("Digite seu nome de usuário: ")
                leia(nome)
                se(nome == “Raul”) {
                        escreva(“Bem vindo ", nome ,“\n”)
                }
                senao {
                        escreva(“Acesso negado!!! \n”)
                }
        }
}
```



Se… então … senão

- Como vimos, podemos utilizar as cláusulas **se** e **senão** para direcionar a execução de nosso código. A estrutura consiste em basicamente :

```
se (condição) {
    // Execute uma parte de código
}
senao {
    // Execute outra parte de código
}
```

Se… então … senão

- Como vimos, podemos utilizar as cláusulas **se** e **senão** para direcionar a execução de nosso código. A estrutura consiste em basicamente :

```
se (condição) {
    // Execute uma parte de código
}
senao {
    // Execute outra parte de código
}
```

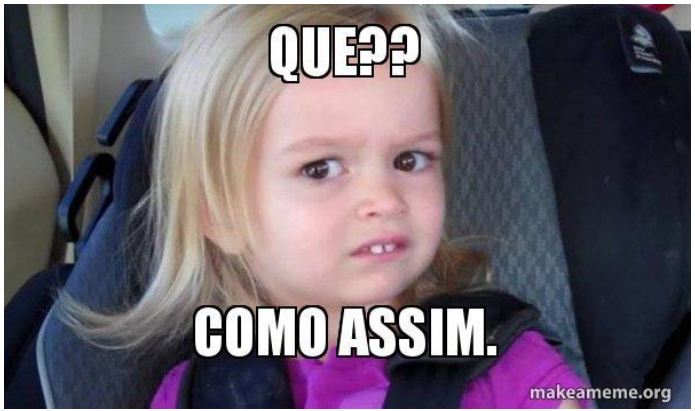
Será que apenas o nome de usuário é suficiente?

- Para validarmos corretamente um usuário precisamos também verificarmos se sua senha está correta. Assim precisamos validar o **nome** de usuário **E** sua **senha.**

```
programa
{
	funcao inicio () {
		cadeia nome
		escreva("Digite seu nome de usuário: ")
		leia(nome)
		se(nome == “Raul” e senha == “MinhaSenha”) {   /*Note o operador lógico E para verificar o usuário E senha*/
			escreva(“Bem vindo ", nome ,“\n”)
		}
		senao {
			escreva(“Acesso negado!!! \n”)
		}
	}
}
```

A Lógica de George Boole

- George Boole foi um matemático e filósofo do século XIX, que defendeu a ideia de que o raciocínio humano poderia ser expresso em termos matemáticos, por meio da lógica formal desenvolvida pelos gregos. Por meio desse raciocínio originou-se a Álgebra de Boole.
- A álgebra de Boole ou álgebra Booleana é embasado na lógica binária.



Lógica booleana

- A Lógica binária possui duas representatividades, “falso” e “verdadeiro” ou “0” e “1” padrão conhecido por máquinas e consequentemente os computadores.
- Em relação aos seus operadores, são definidos AND, OR, NOT, ou seja, E, OU e NÃO, onde (“E”) é a conjunção, (“OU”), a disjunção e (NÃO), a negação.
- Vamos analisar melhor esses operadores...

Lógica booleana

- Conjunção(“E”) - Somente se as duas representatividades forem verdadeiras, a resposta será verdadeira.

A	B	A (“E”)B
Verdade	Verdade	Verdade
Verdade	Falso	Falso
Falso	Verdade	Falso
Falso	Falso	Falso

A	B	A (“E”)B
1	1	1
1	0	0
0	1	0
0	0	0

Lógica booleana

- Disjunção(“OU”) - Se pelo menos uma de suas representatividades forem verdadeiras, a resposta será verdadeira

A	B	A(“OU”)B
Verdade	Verdade	Verdade
Verdade	Falso	Verdade
Falso	Verdade	Verdade
Falso	Falso	Falso

A	B	A(“OU”)B
1	1	1
1	0	1
0	1	1
0	0	0

Lógica booleana

Negação (“NÃO”) - Quando uma representatividade for verdadeira, a resposta será falsa, e quando uma representatividade for falsa, a resposta será verdadeira

A	NÃO A
Verdadeiro	Falso
Falso	Verdadeiro

A	NÃO A
1	0
0	1

B	NÃO B
Verdadeiro	Falso
Falso	Verdadeiro

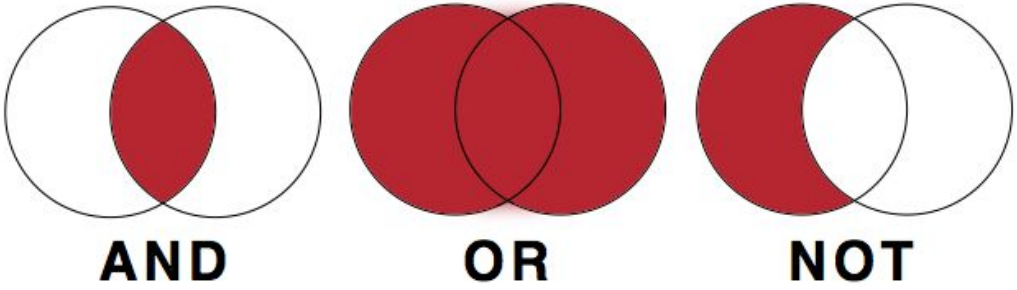
B	NÃO B
1	0
0	1

Operadores Lógicos

- Podemos usar os operadores lógicos E , OU e NÃO (!) para melhorar ainda mais nossas condições.
  - Entendo melhor os resultados dos operadores lógicos:
    - Verdadeiro **E** Verdadeiro = Verdadeiro
    - Verdadeiro **E** Falso = Falso
    - Falso **E** Falso = Falso
    - Verdadeiro **OU** Falso = Verdadeiro
    - Falso **OU** Falso = Falso
    - **!**Verdadeiro = Falso
    - **!**Falso = Verdadeiro
    - == (igual)
    - != (diferente, ou seja **não** igual)

Mais sobre operadores lógicos

- Na maioria das linguagens os operadores E , OU, e NÃO são representados por &&, || e ! , respectivamente.
  - Então :
    - E == && == AND
    - OU == || == OR
    - NAO == ! == NOT



Um pouco mais sobre operadores lógicos

- A negação (!) pode ser utilizada na comparação de igual para negar uma igualdade
  - Exemplo:
    - 1 == 1 (um igual a 1)
    - 1 != 2 (um não igual a 2 || um diferente de 2)
  - Outro exemplo :
    -
```
se(nome != “Fulano”) {  /**/
        escreva(“Você não é o Fulano \n”)
}
senao {
        escreva(“Olá Fulano!!! \n”)
}
```

Senao se ...

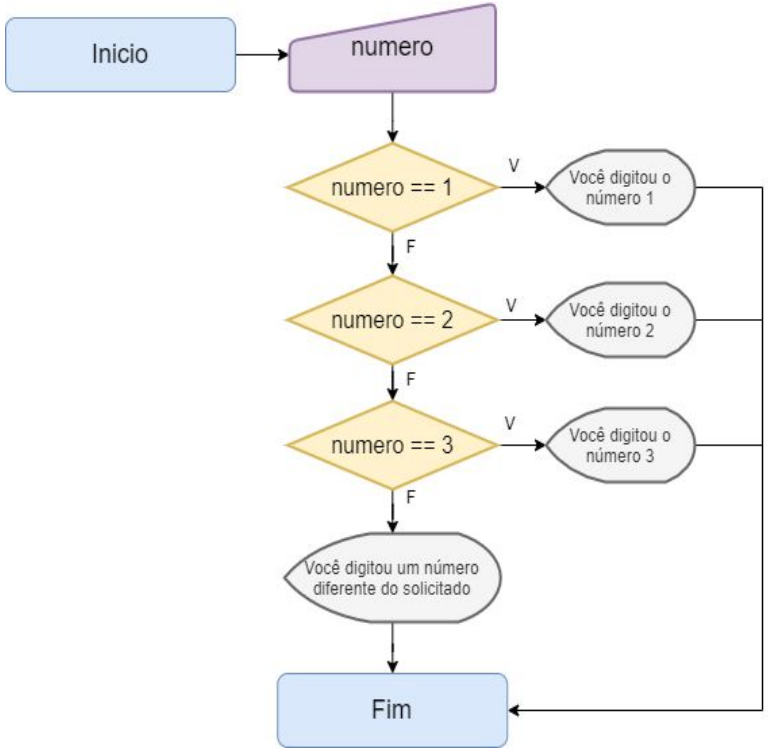
Explicar o senão/se

Estrutura condicional de escolha de casos

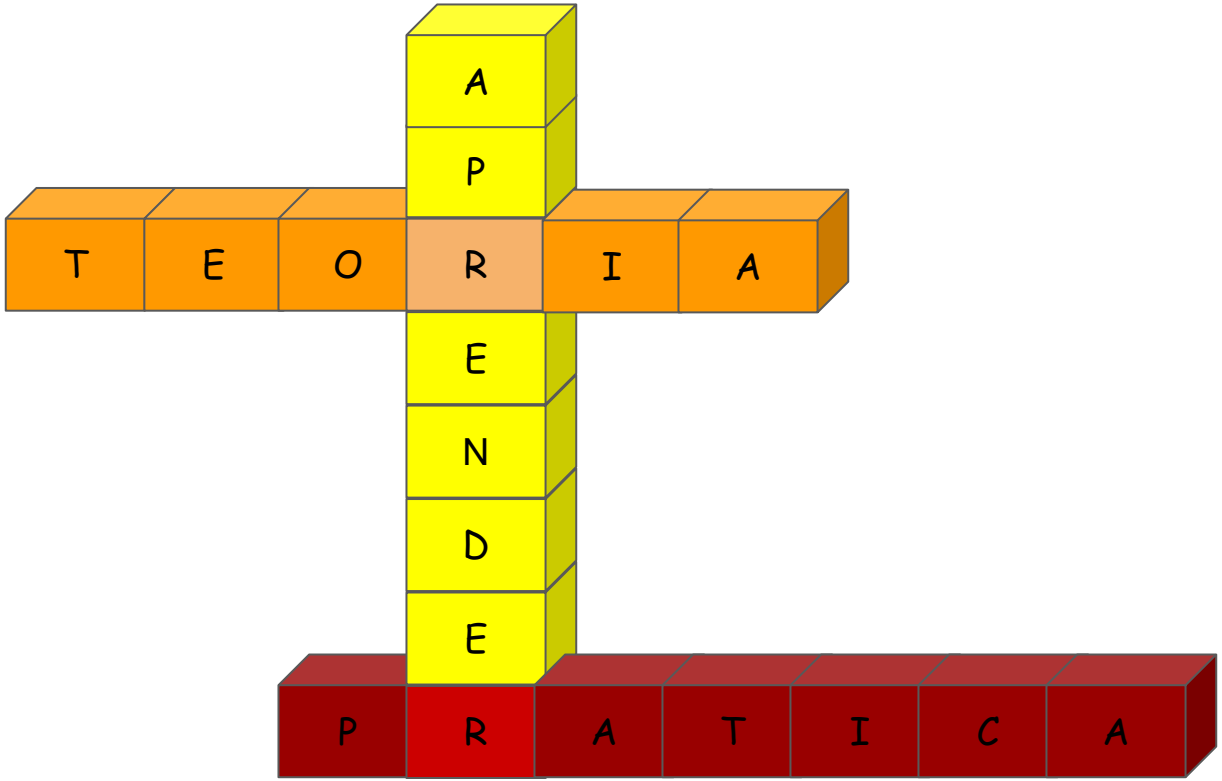
- Uma outra opção de estrutura condicional é a estrutura de seleção de casos onde a partir de uma série de casos (opções) uma deve ser escolhida
- É menos popular que o se senão, porém, pode ser bastante útil quando é preciso testar a variável diversas vezes de uma maneira simples
- Porém possui a limitação de conseguir realizar apenas testes de igualdade
- Os comandos necessário para a execução da estrutura no Portugol é: escolha, caso, pare e caso contrario
- Na maioria das linguagens de programação encontramos essa estrutura com a seguinte sintaxe: escolha == switch, caso == case, pare == break e caso contrario == default

Estrutura condicional de escolha de casos

```
inteiro numero = 0
escreva("Digite um número entre 1 e 3\n")
leia(numero)
escolha(numero){
        caso 1:
                escreva("Você digitou o número ", numero)
                pare
        caso 2:
                escreva("Você digitou o número ", numero)
                pare
        caso 3:
                escreva("Você digitou o número ", numero)
                pare
        caso contrario:
                escreva("Você digitou um número diferente do solicitado")
                pare
}
```



Praticar é fundamental



Retrospectiva : O que já aprendemos?

- Até aqui, já vimos
  - Conteúdo
    - O que é um algoritmo
    - O que é um programa
    - Qual ferramenta utilizaremos
    - Operações de entrada e saída
    - O que são variáveis e constantes
    - Desvios condicionais (se e senão)
    - Operadores lógicos (E, OU …)
    - Estrutura condicional de seleção de casos (escolha caso)

Retrospectiva : O que iremos aprender?

- Laços de repetição (enquanto, para … faça)
- Subrotinas (Funções)
  - Recursividade
  - Bibliotecas
- Estruturas de dados (Vetores, Matrizes, Filas e Pilhas)

Laços de repetição

Estrutura de repetição com teste no início

- Nesse tipo de estrutura uma condição é testada antes de o ciclo de repetição começar. Caso a condição seja verdadeira, o ciclo inicia-se e só para quando a condição **NÃO** for mais verdadeira.
- No Portugol essa estrutura utiliza a palavra **enquanto**

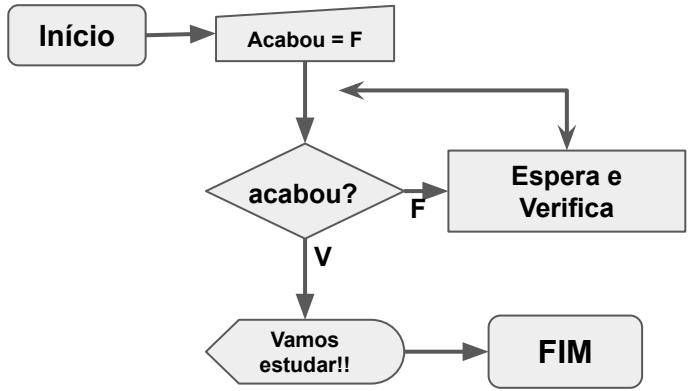
```
inteiro n = 0
enquanto(n < 10){
        escreva("Olá Mundo\n")
        n = n + 1
}
```

Laços de repetição

- Podemos usar **laços de repetição** para sabermos se podemos sair de casa ou não?

```
programa
{
        funcao inicio () {
                logico acabou_coronavirus = falso
                enquanto (acabou_coronavirus == falso){
                        acabou_coronavirus = verifica_pandemia()
                        espera(1 dia)
                }// fim enquanto
                escreva("Vamos para a Residência de software!!")
        }// fim inicio
}// fim programa
```

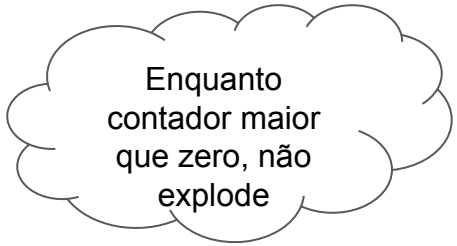
Note que o programa ainda está incompleto pois precisamos programar como verificar a pandemia



Outro exemplo

- Podemos colocar condições dentro da estrutura **enquanto**

```
programa
{
        funcao inicio() {
                inteiro contador = 10
                enquanto (contador > 0)
                {
                        limpa()
                        escreva ("Detonação em: ", contador)
                        contador = contador - 1
                        aguarde(1000) // Aguarda 1000 milisegundos (1 segundo)
                }
                limpa()
                escreva ("Booom!\n")
        }
}
```



Laços de repetição

Estrutura de repetição com teste no final

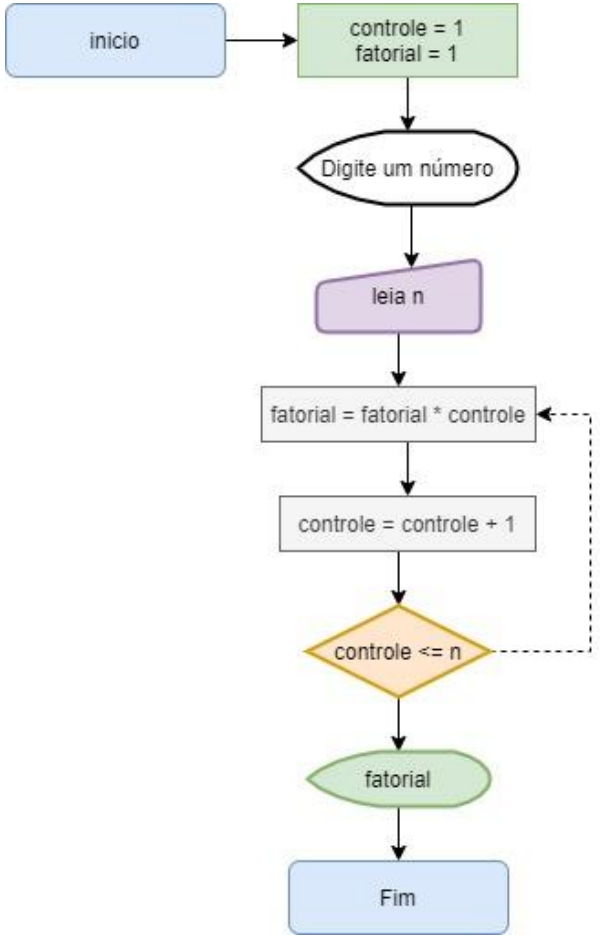
- Este tipo de estrutura de repetição é caracterizado por fazer o teste de controle no final do bloco de comandos. Nessa estrutura tem-se um bloco que inicia com o comando **faca** e termina com o teste **enquanto.**

```
inteiro n = 0
faca{
        escreva("Olá Mundo\n")
        n = n + 1
}enquanto(n < 10)
```

Laços de repetição

Estrutura de repetição com teste no final

```
funcao inicio(){
        inteiro controle, n, fatorial
        controle = 1
        fatorial = 1
        escreva("Digite um número para calcular o fatorial: ")
        leia(n)
        faca{
                fatorial = fatorial * controle
                controle = controle + 1
        }enquanto(controle <= n)
        escreva(fatorial)
}
```



Laços de repetição

Teste no início ou teste no final?

- A diferença entre as estruturas com teste no início e teste no final é que, nas estruturas com o teste no início, o bloco de comandos pode repetir **0 ou n vezes.** Já na estrutura com o teste no final o bloco de comandos irá repetir **1 ou n vezes**, ou seja, nessa estrutura os comandos serão executados pelo menos uma vez.
- As estruturas de repetição com teste no início ou no final muitas vezes terão o mesmo resultado; nesse caso, cabe ao programador escolher qual estrutura usar.

Laços de repetição

Estrutura de repetição com variável de controle

- Essa estrutura deve ser utilizada sempre que se conhece o início e o final das repetições, ou seja, sempre que se sabe quantas vezes o bloco irá se repetir antes de a execução acontecer
- Nessa estrutura precisamos de uma variável para **controlar as repetições**, um valor para marcar o **início da contagem**, outro valor para marcar o **final da contagem de repetições** e ainda podemos controlar como ocorre essa contagem se será de 1 em 1, de 2 em 2, de 3 em 3, e assim por diante.

Laços de repetição

Estrutura de repetição com variável de controle

```
programa
{
	funcao inicio()
	{
		inteiro contador
		para(contador = 1; contador <= 10; contador++){
			escreva("Olá mundo!!!\n")
		}
	}
}
```

Além do **enquanto,** temos o **para… até … faça**

- Imagine que queremos saber a tabuada de um número.
  - Quais são os requisitos?
    - Escolher um número
    - Multiplicar o número escolhido por 1 até 10
- Então **para** 1 **até** 10 **multiplique** o número escolhido.

Como fica o código??

Tabuada usando laços de repetição

```
programa
{
	funcao inicio()
	{
		inteiro numero, resultado, contador

		escreva("Informe um número para ver sua tabuada: ")
		leia(numero)

		limpa()

		para (contador = 1; contador <= 10; contador++)
		{
			resultado = numero * contador
			escreva (numero, " X ", contador, " = ", resultado , "\n")
		}
	}
}
```

Note que ao usar o "para" temos uma estrutura facilitada para intervalos de repetição

Laços de repetição

Exemplo no Portugol algoritmo distância percorrida carro

Sobre laços de repetição

- Se uma ação se repete em um algoritmo, em vez de escrevê-la várias vezes, em certos casos podemos resumir anotando uma vez só e solicitando que ela se repita, usando umas das **estruturas de repetição.**
- Podemos pedir que uma ação (ou um conjunto de ações) seja executada um número definido ou indefinido de vezes, ou enquanto um estado permanecer ou até que um estado seja atingido.
- Fora do Portugol, essas estruturas são denominadas do inglês , **while** (enquanto) **Do While** (faca enquanto) **for** (para)

Voltando ao caso do coronavírus

- Lembra do código que verificava se já podíamos retornar às aulas?

```
programa
{
        funcao inicio () {
                logico acabou_coronavirus = falso
                enquanto (acabou_coronavirus == falso){
                        acabou_coronavirus = verifica_pandemia()
                }
                escreva("Vamos para a Residencia de software!!")
        }
}
```

Ficou faltando programarmos como verificaremos se o coronavírus já está contido

Voltando ao caso do coronavírus

- Podemos escrever a execução da subrotina (ou função, ou método) abaixo do programa início. A lógica é semelhante à função **inicio**

```
programa
{
	funcao inicio () {
		logico acabou_coronavirus = falso
		inteiro dias_parados = 0
		enquanto (acabou_coronavirus == falso){
			acabou_coronavirus = verifica_pandemia(dias_parados)
			dias_parados ++
		}
		escreva("Vamos para a Residencia de software!!")
	}
	funcao logico verifica_pandemia(inteiro dias_parados){
		se(dias_parados>15){
			retorne verdadeiro
		 }
		retorne falso
	}
}
```

Laços de repetição

- Os Laços de repetição com teste no início ou no fim enquanto e faca enquanto devem ser utilizados quando não se conhece de antemão o número de vezes que determinado bloco de comandos deve ser executado. Já as estruturas de repetição com variável de controle para ate faça devem ser usadas sempre que se sabe quantas repetições serão feitas, em outras palavras, somente quando se conhece o início e o final das repetições

Funções

- Definição : Sequência de instruções executadas somente quando chamadas por um programa em execução
  - Devem executar **uma tarefa** específica
  - Um programa **pode conter diversas funções**, além da função principal **início()** , que é **obrigatória**
  - As funções executam **somente** quando chamadas à partir da função inicio()
  - Após a execução, o fluxo retorna ao ponto **imediatamente após** o da chamada da função
  - Uma função pode (ou não) **retornar um valor** ao bloco que a chamou
  - Uma função pode (ou não) **necessitar de um ou mais argumentos** ao ser chamada

Vamos olhar uma etapa de cada vez …

Estrutura de uma Função

```
funcao nomeDaFuncao(){
        escreva("Olá Mundo!")
}

funcao inteiro somar(inteiro n1, inteiro n2){
        //Corpo da função
        inteiro resutado = n1 + n2
        retorne resultado
}
```

Mais alguns exemplos - Repetição de código

```
programa {
        funcao inicio(){
                inteiro i
                para(i=0; i<20; i++)
                        escreva(“*”)
                escreva(“\n”)
                escreva(“Numeros entre 1 e 5\n”)
                para(i=0; i<10; i++)
                        escreva(“*”)
                escreva(“\n”)
                para(i=1; i<=5; i++)
                        escreva(i,“\n”)
                para(i=0;i<20;i++)
                        escreva(“*”)
                escreva(“\n”)
        }
}
```

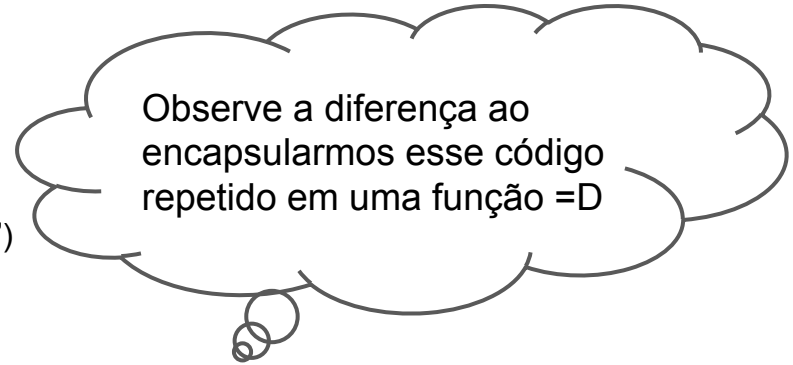
Note o código repetido. Se tivermos que consertar, teremos que fazer o mesmo ajuste várias vezes

Saída:

```
**********
Numeros entre 1 e 5
**********
1
2
3
4
5
**********
```

Mais alguns exemplos - Repetição de código

```
programa {
        funcao inicio(){
                inteiro i
                escreve_linha()
                escreva("Numeros entre 1 e 5\n")
                escreve_linha()
                para(i=1; i<=5; i++)
                        escreva(i,"\n")
                escreve_linha()
        }
        funcao escreve_linha(){
                para(i=0; i<20; i++)
                        escreva("*")
                escreva("\n")
        }
}
```



Saída:

```
**********
Numeros entre 1 e 5
**********
1
2
3
4
5
**********
```

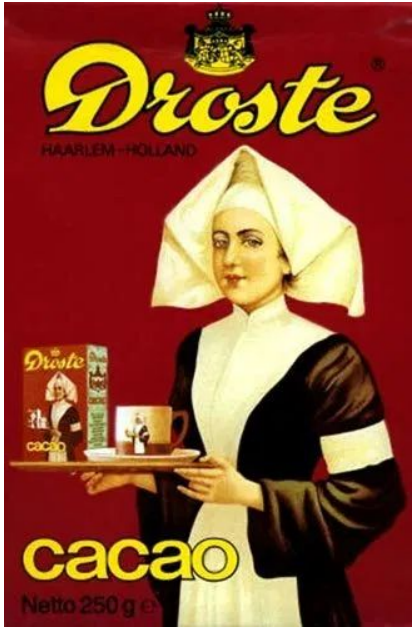
Mais alguns exemplos - Recursividade

- Podemos também fazer a função chamar ela mesma para resolvermos problemas chamados **recursivos**.

Mas o que é recursão?

Recursividade

- Recursão é a definição de algo a partir dele mesmo



Visualmente, recursão pode ser comparada ao efeito Droste!

Recursividade

- Em Matemática e Ciência da Computação, uma classe de métodos tem comportamento recursivo quando eles podem ser definidos por duas propriedades:
  - Um caso base simples (ou vários casos)
  - Um conjunto de regras que reduz todos os outros casos para o caso base

Exemplo : Fatorial de um número inteiro positivo!!

5 * (fatorial de 4)
5! = 5 * 4 * 3 * 2 * 1

Fatorial Recursivo

```
programa {
    funcao inteiro fatorial(inteiro n){
        se(n == 0){
            retorne 1
        } senao {
            retorne  n * fatorial( n - 1 )
        }

    }
}
```

Execução : 4 fatorial

```
fatorial(4) -> 4 * 3* 2 *1
n = 4
retorne 4 * fatorial(3)
                n = 3
                retorne 3 * fatorial(2)
                                n = 2
                                retorne 2 * fatorial(1)
                                                n = 1
                                                retorne 1 * fatorial(0)
                                                                retorne 1
```

Passos para escrever uma função recursiva

1. Escreva um protótipo da função recursiva
2. Escreva um comentário que descreve o que a função deve fazer
3. Determine o caso base (pode haver mais de um) e a solução desse caso
4. Determine qual é o problema menor do que o atual a ser resolvido
5. Use a solução do problema menor para resolver o problema maior.

Voltando ao coronavírus parte 3 ...

- Voltando ao código da quarentena do coronavírus

```
programa
{
        funcao inicio () {
                logico acabou_coronavirus = falso
                enquanto (acabou_coronavirus == falso){
                        acabou_coronavirus = verifica_pandemia()
                }
                escreva("Vamos para a Residencia de software!!")
        }
}
```

Além do método/função/subrotina verifica_pandemia , temos mais alguma outra função?

Funções de bibliotecas

- Nós vimos várias funções como **escreva(), leia(), limpa().**
- Estas funções são métodos padrões já disponíveis em qualquer programa do PortugolStudio. Além dessas funções, podemos adicionar outras funções através da importação de bibliotecas.

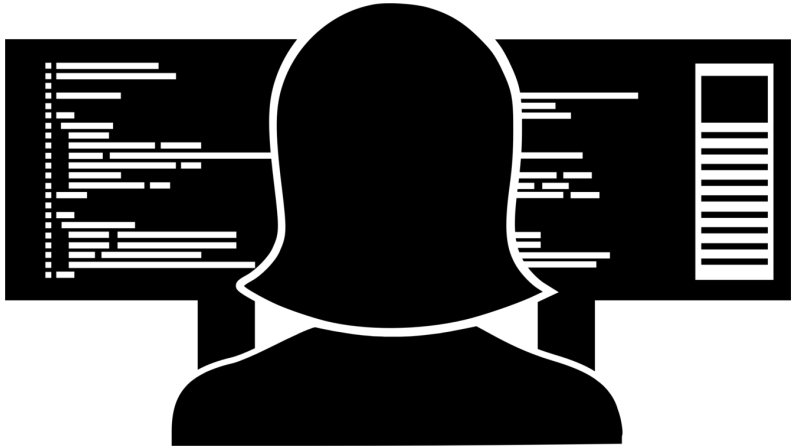
```
programa
{
	inclua biblioteca Matematica --> mat
	funcao inicio()
	{
		real numero = 4.0
		real raiz = mat.raiz(numero, 2.0) // Obtém a raíz quadrada do número
		escreva("A raíz quadrada de ", numero , " é: ", raiz, "\n")
	}
}
```

Retrospectiva : Como estamos?

- Até aqui, já vimos
  - Valores
    - Como nos organizamos como turma
    - Valores em trabalho em equipe e desenvolvimento de software
  - Conteúdo
    - O que é um algoritmo
    - O que é um programa
    - Qual ferramenta utilizaremos
    - Operações de entrada e saída
    - O que são variáveis e constantes
    - Desvios condicionais (se e senão)
    - Operadores lógicos (E, OU …)
    - Funções
      - Recursividade

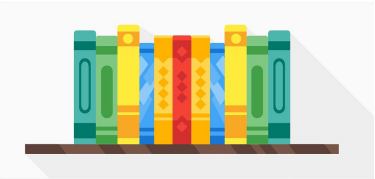
Retrospectiva : O que iremos aprender?

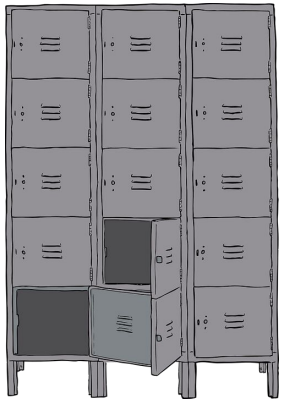
- Estruturas de dados (Vetores, Matrizes, Filas e Pilhas)



Estrutura de Dados

“Estrutura de dados é o ramo da computação que estuda os diversos mecanismos de organização de dados para atender aos diferentes requisitos de processamento.” - **RICARTE, IVAN LUIZ MARQUES (UNICAMP)**





Estruturas de dados : Conceitos

Uma estrutura de dados pode ser dividia em dois pilares fundamentais : **dado** e **estrutura**.

DADO

Dados são qualquer sequência de um ou mais símbolos que tenham significado por ato(s) específico(s) de interpretação.

ESTRUTURA

Elemento estrutural responsável por carregar as informações dentro de uma estrutura de software

Estruturas de dados : Conceitos

Uma estrutura de dados pode ser dividia em dois pilares fundamentais : **dado** e **estrutura**.

DADO

Tipos de dados :

- Inteiro
- Ponto flutuante
- Caractere
- Texto

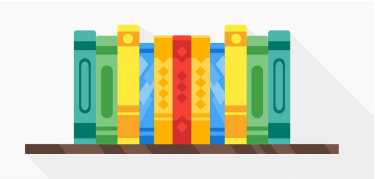
ESTRUTURA

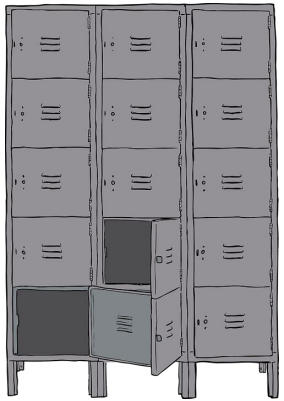
Tipos de estruturas:

- Vetores
- Pilhas
- Filas
- Listas

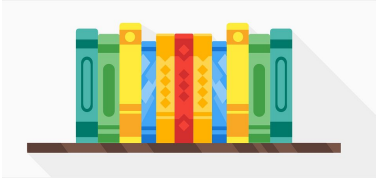
Principais tipos de estruturas de dados

- Vetores
  - Unidimensionais
  - Bidimensionais (Matrizes)
- Pilhas (não estudaremos agora)
- Filas (não estudaremos agora)

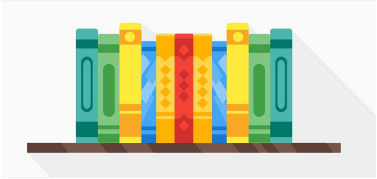




Vetores

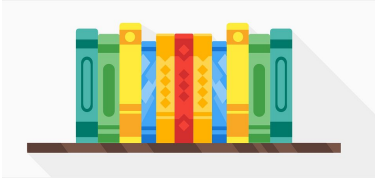


Vetores



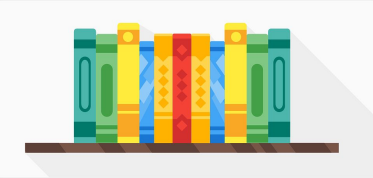
Tipo	Nome	Capacidade

Vetores



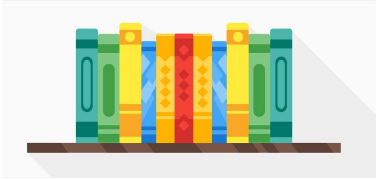
Tipo	Nome	Capacidade
livros		

Vetores



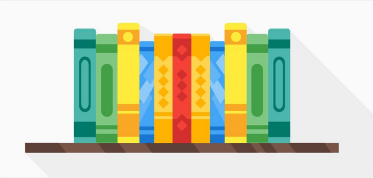
Tipo	Nome	Capacidade
livros	minhaEstante	

Vetores



Tipo	Nome	Capacidade
livros	minhaEstante	11

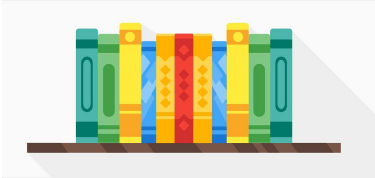
Vetores



Tipo	Nome	Capacidade
livros	minhaEstante	11

livros minhaEstante[11];

Vetores



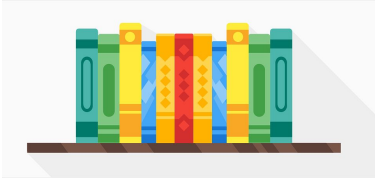
Tipo	Nome	Capacidade
livros	minhaEstante	11

livros minhaEstante[11]

1	26	22	100	2

Tipo	Nome	Capacidade

Vetores



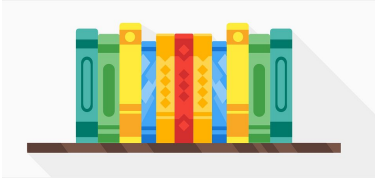
Tipo	Nome	Capacidade
livros	minhaEstante	11

livros minhaEstante[11]

1	26	22	100	2

Tipo	Nome	Capacidade
inteiro		

Vetores



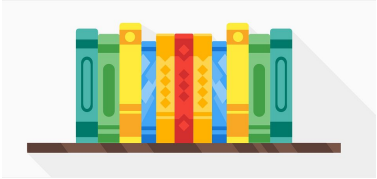
Tipo	Nome	Capacidade
livros	minhaEstante	11

livros minhaEstante[11]

1	26	22	100	2

Tipo	Nome	Capacidade
inteiro	meuVetor	

Vetores



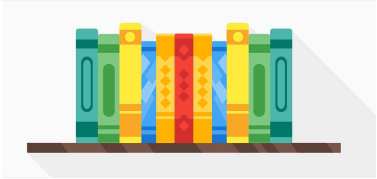
Tipo	Nome	Capacidade
livros	minhaEstante	11

livros minhaEstante[11];

1	26	22	100	2

Tipo	Nome	Capacidade
inteiro	meuVetor	5

Vetores



Tipo	Nome	Capacidade
livros	minhaEstante	11

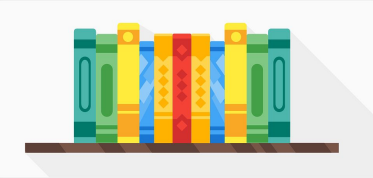
livros minhaEstante[11];

0	1	2	3	4
1	26	22	100	2

Tipo	Nome	Capacidade
int	meuVetor	5

inteiro meuVetor[5];

Vetores



Tipo	Nome	Capacidade
livros	minhaEstante	11

livros minhaEstante[11];

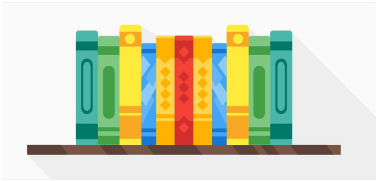
0	1	2	3	4
30	26	22	100	40

Tipo	Nome	Capacidade
int	meuVetor	5

inteiro meuVetor[5];

- meuVetor[0] = 30;
- meuVetor[4] = 40;

Vetores



Tipo	Nome	Capacidade
livros	minhaEstante	11

livros minhaEstante[11]

0	1	2	3	4
30	26	**50**	100	40

Tipo	Nome	Capacidade
int	meuVetor	5

inteiro meuVetor[5]

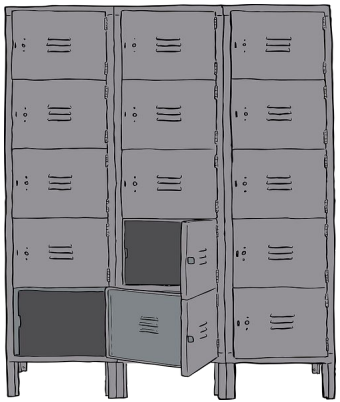
- meuVetor[0] = 30
- meuVetor[4] = 40
- meuVetor[2] = 50

Vetores

É uma das estruturas de dados mais simples e mais utilizadas dentre todas.
Principais características:

- Indexação com início em 0 (zero)
- Adição e pesquisa de novos elementos de forma aleatória
- Acesso aos elementos através de índices
- Possuem tamanho finito de elementos
- Carregam dados de tipos específicos
- Podem possuir uma ou mais dimensões

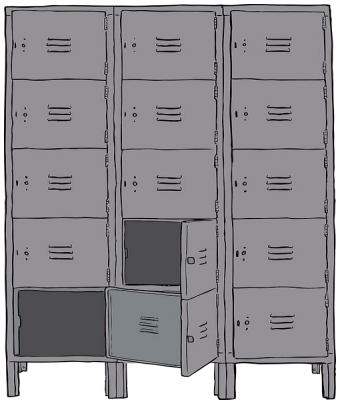
Matrizes



Tipo	Nome	Capacidade
mochila	meuArmario	[5][3]

mochila meuArmario[5][3]

Matrizes

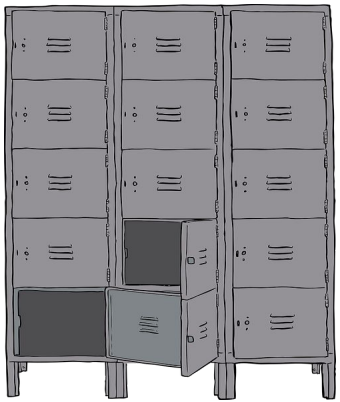


Tipo	Nome	Capacidade
mochila	meuArmario	[5][3]

mochila meuArmario[5][3];

	[0]	[1]	[2]
[0]	1.3	1.5	1.6
[1]	1.2	1.7	2.2
[2]	100.5	75.6	2.95

Matrizes



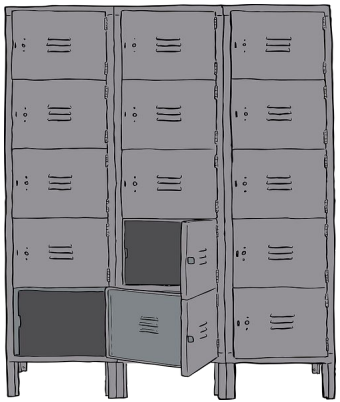
Tipo	Nome	Capacidade
mochila	meuArmario	[5][3]

mochila meuArmario[5][3];

	[0]	[1]	[2]
[0]	1.3	1.5	1.6
[1]	1.2	1.7	2.2
[2]	100.5	75.6	2.95

Tipo	Nome	Capacidade

Matrizes



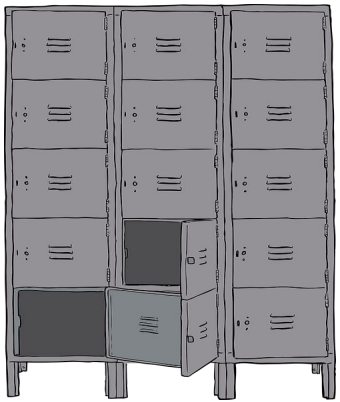
Tipo	Nome	Capacidade
mochila	meuArmario	[5][3]

mochila meuArmario[5][3];

	[0]	[1]	[2]
[0]	1.3	1.5	1.6
[1]	1.2	1.7	2.2
[2]	100.5	75.6	2.95

Tipo	Nome	Capacidade
real		

Matrizes



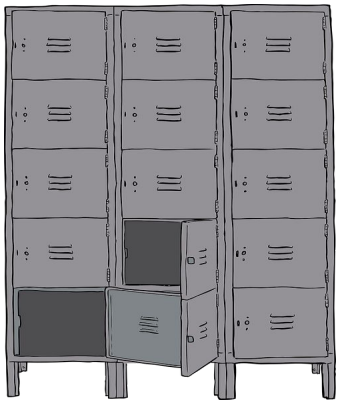
Tipo	Nome	Capacidade
mochila	meuArmario	[5][3]

mochila meuArmario[5][3];

	[0]	[1]	[2]
[0]	1.3	1.5	1.6
[1]	1.2	1.7	2.2
[2]	100.5	75.6	2.95

Tipo	Nome	Capacidade
real	minhaMatriz	

Matrizes



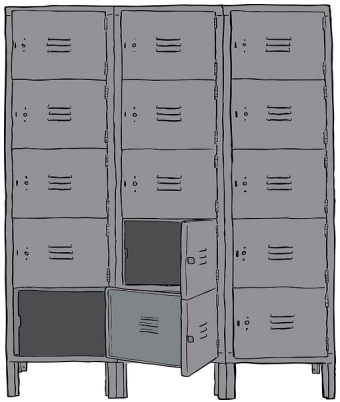
Tipo	Nome	Capacidade
mochila	meuArmario	[5][3]

mochila meuArmario[5][3];

	[0]	[1]	[2]
[0]	1.3	1.5	1.6
[1]	1.2	1.7	2.2
[2]	100.5	75.6	2.95

Tipo	Nome	Capacidade
real	minhaMatriz	[3][3]

Matrizes



Tipo	Nome	Capacidade
mochila	meuArmario	[5][3]

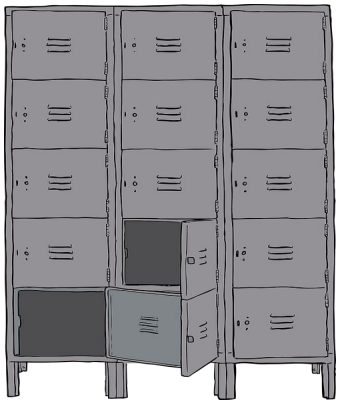
mochila meuArmario[5][3]

	[0]	[1]	[2]
[0]	1.3	1.5	1.6
[1]	1.2	1.7	2.2
[2]	100.5	75.6	2.95

Tipo	Nome	Capacidade
real	minhaMatriz	[3][3]

real minhaMatriz[3][3]

Matrizes



Tipo	Nome	Capacidade
mochila	meuArmario	[5][3]

mochila meuArmario[5][3]

minhaMatriz[1][2] = 5.0

	[0]	[1]	[2]
[0]	1.3	1.5	1.6
[1]	1.2	1.7	5.0
[2]	100.5	75.6	2.95

Tipo	Nome	Capacidade
real	minhaMatriz	[3][3]

real minhaMatriz[3][3]

A prática leva à perfeição