

# Residência de Software

## Lógica de Programação

---

# Quem sou eu?

Nome: Felipe Sanches

Formação: Análise de Sistemas

Idade: 37 anos

18 anos no mercado de TI

Hobbie – Viajar e assistir series



# Agora é com vocês

---



QUEM SÃO VOCÊS?



O QUE ESPERAM  
DO CURSO?



QUAL A SUA  
FAMILIARIDADE  
COM TECNOLOGIA?



PORQUE ESSE  
CURSO?

# Algumas dicas importantes

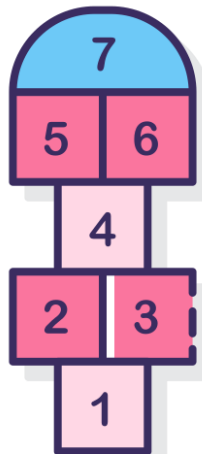
- Serão 3 avaliações: Prova, trabalhos e conceito;
- Prova individual, trabalho em grupo e avaliação individual que levará em conta as características esperadas;
- Microfones “mutados”;
- Quando quiser falar levante a mão (na interface, ok? );
- Não podemos abonar uma falta (minutos contam);
- Chegou atrasado, precisa sair mais cedo, AVISE!
- Precisou faltar e tem justificativa ou atestado.

# Conteúdo programático

- Algoritmo (Portugol)
- Lógica booleana (E, OU, NÃO)
- Árvore de decisão
- Estruturas de Laço
- Conceito de recursividade
- Estrutura de dados (Vetor, Matriz, Fila, Pilha...)
- Conceito de variável e constante
- Estatística Básica • Regra de três
- Introdução a armazenamento de dados
- Git e GitHub

# Algoritmo

- Sequência **finita de passos** que levam à execução de uma tarefa
- Algo muito comum no nosso dia a dia, sendo de TI ou não =D



# Exercício 1

---

- Vamos fazer um algoritmo para fazer um bolo
- Os ingredientes são definidos antes;
- Modo de preparo depois;



# Algoritmo fazer bolo

```
FazerBolo(){  
    //Ingredientes  
    Acucar = 2 xícaras;  
    Farinha = 2,5  
    xícaras; Ovos = 4;  
    Oleo = 0,5 xícaras;  
    Fermento = 1 colher;  
    FormaDeBolo = nada;  
  
    //Modo de preparo  
    //Misturar os ingredientes “até homogeneizar”  
    formaDeBolo = Acucar + Farinha + Ovos + Oleo + Fermento ;  
    ColocarFormaNoForno(formaDeBolo);  
    Aguardar assar por 1 hora  
}
```





# 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!

{Portugol  Studio}

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

# Estrutura inicial de um código em Portugol

programa

```
{  
    /* Declaração de variáveis, estruturas e outras funções */  
  
    funcao inicio ()  
    {  
        /*Execução da função início*/  
    }  
}
```

# 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



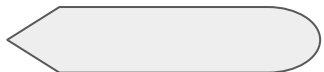
Indica o sentido do fluxo de dados



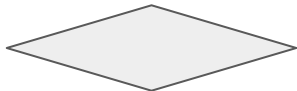
Indica cálculos e atribuições de valores



Representa a entrada de dados



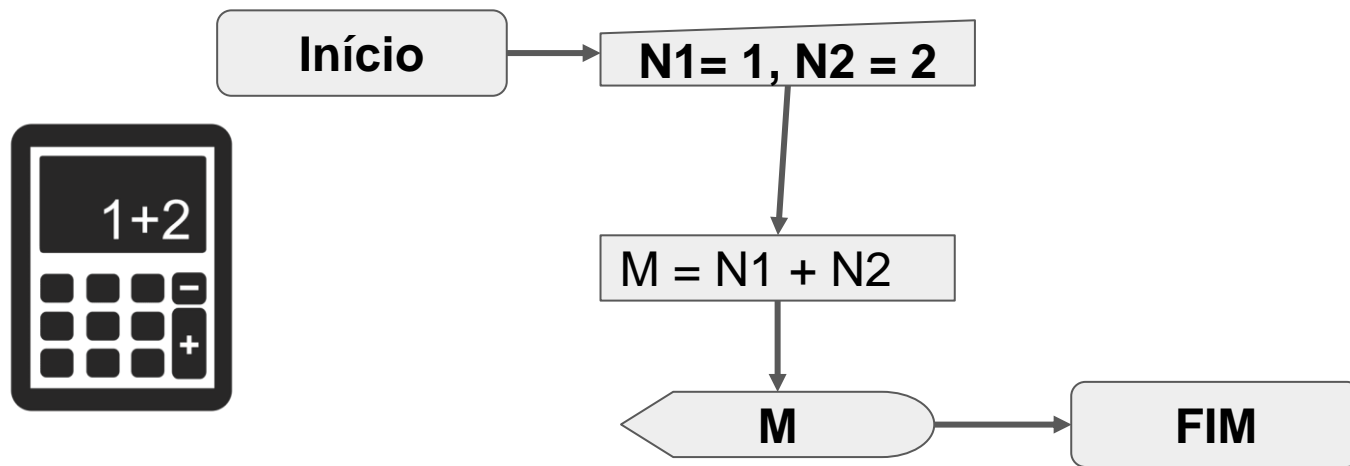
Representa a saída de dados



Tomada de decisão

# Fluxograma

Exemplo : Soma de 2 números



# O que iremos aprender :

- Operações de entrada e saída
- O que são variáveis e constantes
- Desvios condicionais ( se e senão )
- Operadores Lógicos ( E, OU ... )
- Laços de repetição ( enquanto, para ... até )
- Estruturas de dados ( Vetores, Matrizes, Filas e Pilhas )
- Subrotinas ( Funções )
  - Recursividade
  - Bibliotecas



Mas antes, vamos explorar o Portugal Studio

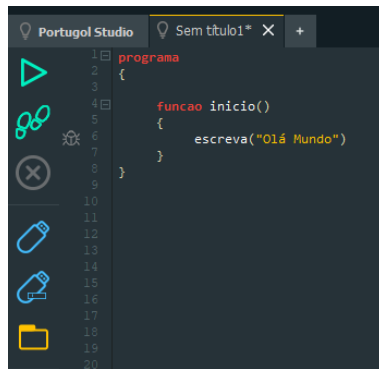


# Nosso primeiro programa: Olá mundo!

Execute no Portugol Studio o código : Olá Mundo

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

<HELLO WORLD />

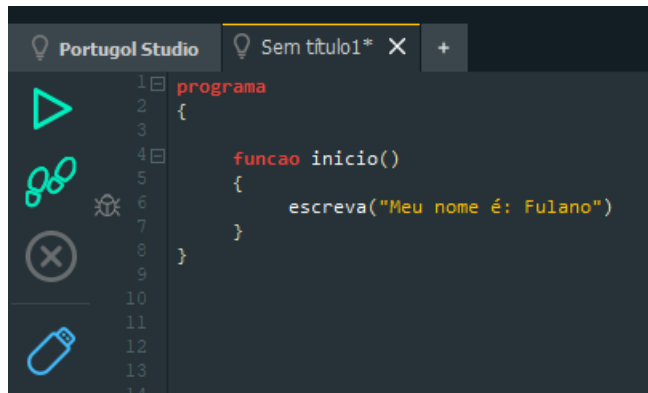


```
1 programa
2 {
3
4     funcao inicio()
5     {
6         escreva("Olá Mundo")
7     }
8 }
9
10
11
12
13
14
15
16
17
18
19
20
```



# Identificando-se : Qual é o seu nome?

Execute os seguintes programas - **'Meu nome é: Fulano'**



```
1 programa
2 {
3
4     funcao inicio()
5     {
6         escreva("Meu nome é: Fulano")
7     }
8 }
9
10
11
12
13
14
```



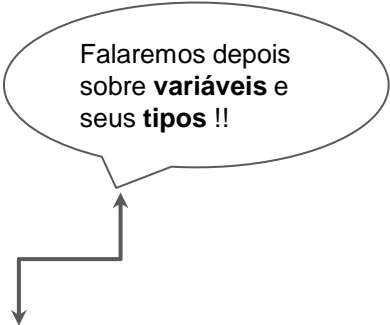
Meu nome é  
Fulano

# Operações de entrada e saída

Fazendo a leitura do nome a partir do teclado.

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 : ", "\n", nome )  
    }  
}
```



Falaremos depois  
sobre **variáveis** e  
seus **tipos** !!

**cadeia** nome /\*cadeia se refere ao **tipo** da **variável** que é uma **cadeia de**

# Operações de 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 = Maria
```

```
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**.

# Voltando ao programa anterior...

Fazendo a leitura do nome a partir do teclado.

```
programa
```

```
{
```

```
    funcao inicio ()
```

```
{
```

```
    caracteres*/
```

**cadeia** nome /\*cadeia se refere ao **tipo** da **variável** que é uma **cadeia de**

```
    escreva("Digite seu nome: ")
```

```
    leia(nome)
```

```
    escreva("Seu nome é : ", "\n", nome )
```

```
}
```

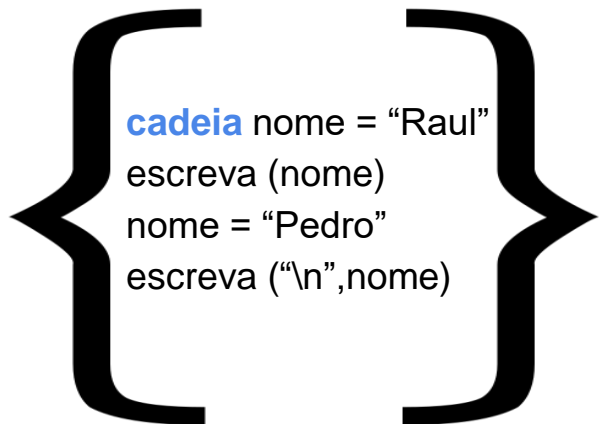
```
}
```

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 ("\n",nome)
```

O que será  
impresso na tela?

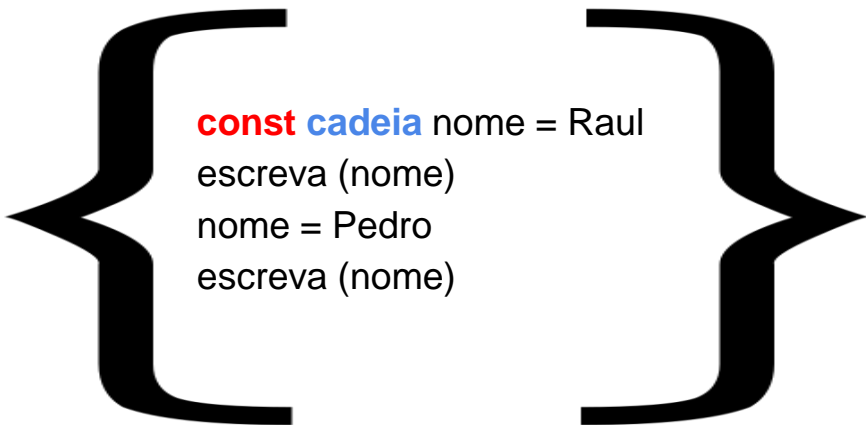


# Variáveis e constantes

- 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

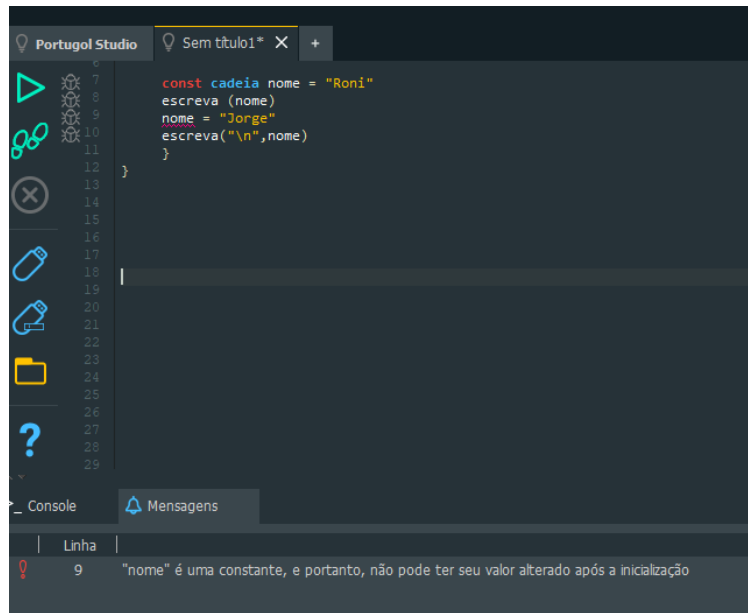
# Variáveis e constantes

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



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

O que será impresso na tela?



```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29
```

```
const cadeia nome = "Roni"  
escreva (nome)  
nome = "Jorge"  
escreva("\n", nome)  
}
```

Console

Mensagens

9 "nome" é uma constante, e portanto, não pode ter seu valor alterado após a inicialização



# Exemplo Variáveis

inteiro idade

real peso

real altura

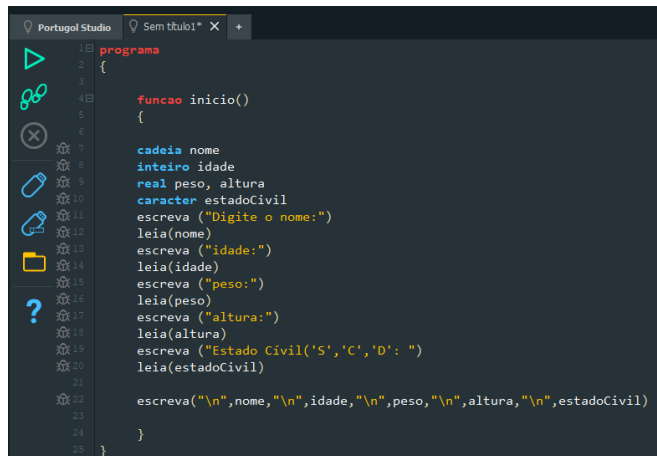
idade = 10

## Memória

|        |    |
|--------|----|
| idade  | 10 |
| peso   |    |
| altura |    |

## Exercício:

Faça um programa para ler o nome, idade, peso, altura e estado cívil sendo do tipo caracter de uma pessoa e exibir os dados na tela.



```
1 programa
2 {
3
4     funcao inicio()
5     {
6
7         cadeia nome
8         inteiro idade
9         real peso, altura
10        caracter estadoCivil
11        escreva ("Digite o nome:")
12        leia(nome)
13        escreva ("idade:")
14        leia(idade)
15        escreva ("peso:")
16        leia(peso)
17        escreva ("altura:")
18        leia(altura)
19        escreva ("Estado Civil('S','C','D'): ")
20        leia(estadoCivil)
21
22        escreva("\n",nome,"\n",idade,"\n",peso,"\n",altura,"\n",estadoCivil)
23    }
24 }
```

# Operadores Matemáticos

+ soma

- subtração

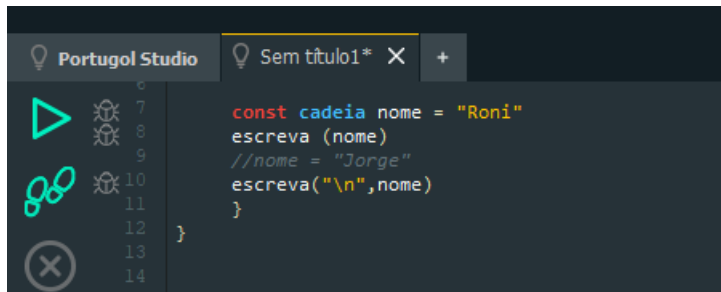
\* multiplicação

/ divisão

% resto da divisão

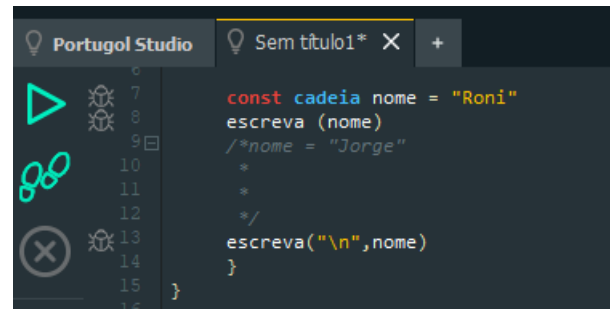
# Comentários

- Usado para ignorar uma parte do código ou para inserir informações sobre um determinado trecho de código



The screenshot shows the Portugol Studio interface with a file named 'Sem título1\*'. The code is written in a dark theme. On the left, there are icons for execution (play), error (bug), and a close button (X). The code is as follows:

```
6  
7 const cadeia nome = "Roni"  
8 escreva (nome)  
9 //nome = "Jorge"  
10 escreva("\n",nome)  
11 }  
12  
13  
14
```



The screenshot shows the Portugol Studio interface with a file named 'Sem título1\*'. The code is written in a dark theme. On the left, there are icons for execution (play), error (bug), and a close button (X). The code is as follows:

```
6  
7 const cadeia nome = "Roni"  
8 escreva (nome)  
9 /*nome = "Jorge"  
10 *  
11 *  
12 */  
13 escreva("\n",nome)  
14 }  
15  
16
```

# Teclas de Atalho

Pressione F11 para visualizar as teclas de atalho



|   | Ação  | Atalho            |
|---|---|-------------------|
| 📁 | Abrir um arquivo                                    | Ctrl+O            |
| ❓ | Ajuda   | F1                |
| 📄 | Ativar/desativar a centralização de código fonte    | Shift+Pause/Break |
| 📄 | Crear novo arquivo                                  | Ctrl+N            |
| 🔍 | Depurar programa atual                              | Shift+F5          |
| 🔍 | Depurar próxima instrução do programa atual         | Shift+F9          |
| 📋 | Dicas de Interface                                  | F3                |
| ▶ | Executar programa atual                             | Shift+F6          |
| 🏠 | Exibir tela inicial                                 | Alt+Home          |
| 🔍 | Expandir/restaurar o tamanho do editor              | Shift+Esc         |
| ✕ | Fechar a aba atual                                  | Ctrl+Q            |
| ✕ | Fechar todas as abas                                | Ctrl+Shift+Q      |
| ⏏ | Interromper a execução/depuração do programa atual  | Shift+F7          |
| 🔍 | Pesquisar e/ou substituir um texto no arquivo atual | Ctrl+F            |
| 💾 | Salvar o arquivo atual                              | Ctrl+S            |
| 💾 | Salvar uma cópia do arquivo atual                   | Ctrl+Shift+S      |
| ⏪ | Selecionar a aba anterior (à esquerda)              | Alt+Esquerda      |
| ⏩ | Selecionar a próxima aba (à direita)                | Alt+Direita       |

Preenchimento Automático - CTRL + Espaço

# Exercícios

- 1) Leia dois valores pelo teclado e imprima a soma.
- 2) Construir um algoritmo que leia um número e exiba na tela o seu sucessor e antecessor.
- 3) Construa um algoritmo que leia o nome de um aluno, disciplinas, duas notas e exiba na tela a média/
- 4) Faça um programa com duas variáveis ano\_nascimento que receberá o ano que você nasceu e outra variável com o nome ano\_futuro que deverá ser atribuído o valor 2035. Criar uma variável com o nome resultado para calcular a diferença. No final escreva na tela qual será a sua idade em 2035.

## Exercícios

5) Uma empresa paga R\$10.00 por hora normal trabalhada e R\$ 15.00 por hora extra. Escreva um algoritmo que leia o total de horas normais e o total de horas extras trabalhadas por um empregado em um ano e calcule o salário anual deste trabalhador.

*Exemplo : Entrada : Digite o número de horas trabalhadas no ano : 1760 Digite o número de horas extras trabalhadas no ano : 400*

*Saída : Seu salário anual é de : R\$ 23600*

6) Escreva um programa que receba a temperatura em Celsius e retorne o valor em Fahrenheit

*Exemplo : Entrada: Digite o valor da temperatura em Celsius : 10*

*Saída: 10 graus Celsius é o mesmo que 50 graus Fahrenheit*

7) Criar um algoritmo em português estruturado que leia dois números inteiros e imprima a seguinte saída: Dividendo, Divisor, Quociente e Resto

8) Escreva um programa que diga se o número é par ou ímpar. Digite o número X por um número Y e retorna o resto da divisão

Exemplo :

Entrada: Digite um número : 2

Saída: O número 2 é par

```
programa
{
    funcao inicio()
    {
        inteiro n1, n2, quociente, resto
        escreva ("digite primeiro número: ")
        leia(n1)
        escreva ("digite o segundo número: ")
        leia(n2)
        quociente = n1 / n2
        resto = n1 % n2
        escreva ("o dividendo é: ", n1, "\n")
        escreva ("o divisor é: ", n2, "\n")
        escreva ("quociente é: ", quociente, "\n")
        escreva ("o resto é: ", resto)
    }
}
```

# 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



# Aula 2





# 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")  
    }  
}
```

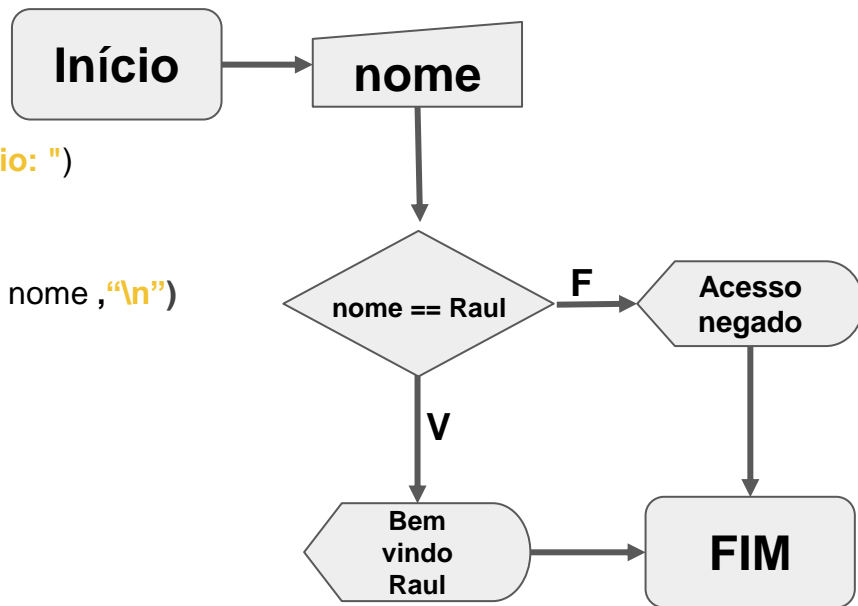
Vamos transformar este programa para ser capaz de **validar** o nome do usuário e conceder acesso ao sistema!  
Apenas o usuário cadastrado poderá entrar no sistema.

# 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  
}
```



# 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")
        }
    }
}
```

# Operadores

## Operadores

Maior >

Menor <

Maior ou Igual >=

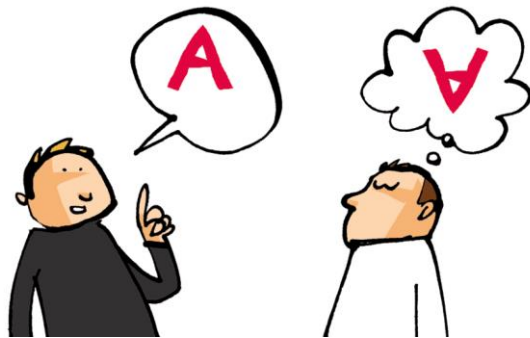
Menor ou Igual <=

Igual ==

Diferente !=

# Operadores Lógicos

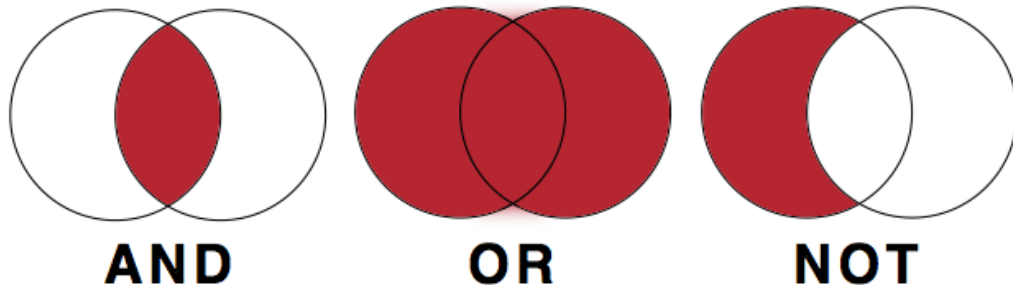
- Podemos usar os operadores lógicos E , OU e NÃO (!) para melhorar ainda mais nossas condições.
  - Entendo melhor os resultados dos operadores lógicos:
    - Verdadeiro **E** Verdadeiro = Verdadeiro
    - Verdadeiro **E** Falso = Falso
    - Falso **E** Falso = Falso
    - Verdadeiro **OU** Falso =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



Entendido?

# 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")  
}
```



## Caso (Condicional)

- Utilizar somente inteiros e caracteres
- Não pode utilizar operadores lógicos
- Só utilizar o operador relacional de igualdade

Escolha (variável)

```
caso 1
    //instruções
pare
caso 2
    //instruções
pare
caso 3
    //instruções
pare
caso contrário
    //instruções
```

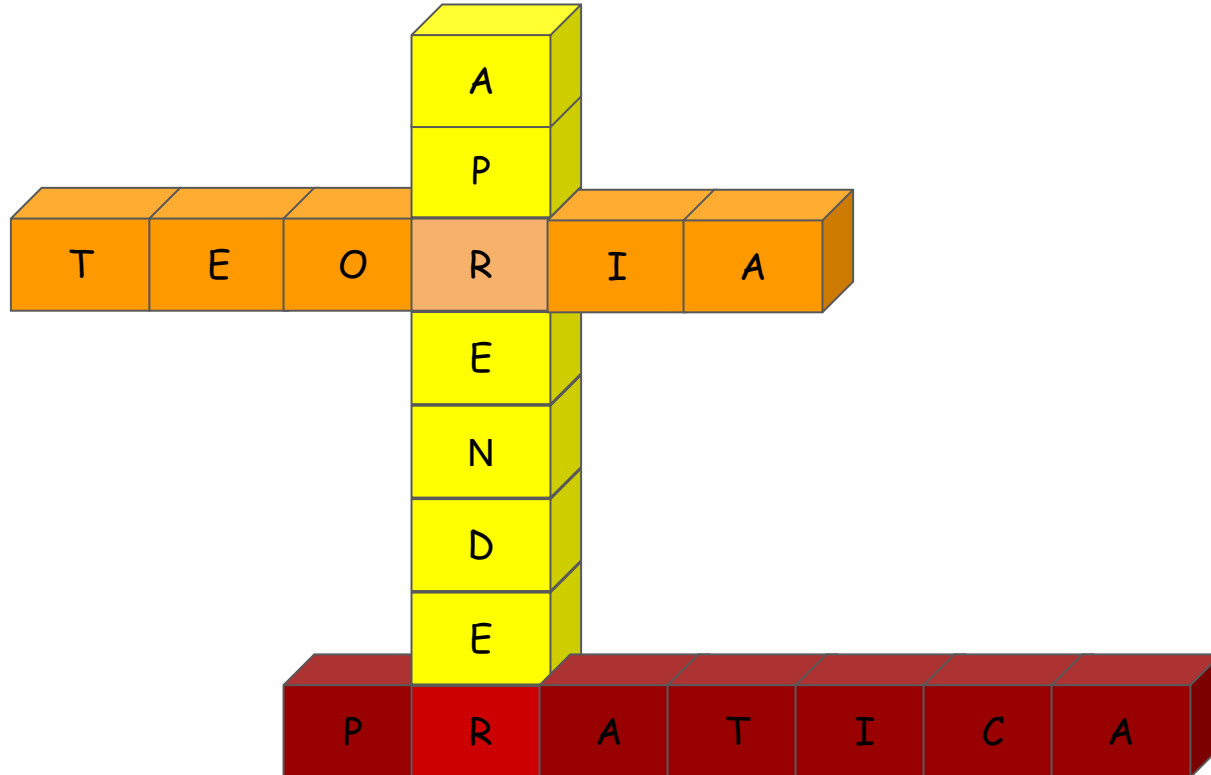
```
funcao inicio()
{
    inteiro diaSemana

    escreva("Digite o número do dia da semana (1-7)")
    leia(diaSemana)

    escolha(diaSemana){
        caso 1:
            escreva("Domingo")
        pare

        caso 2:
            escreva("Segunda-Feira")
        pare
        caso 3:
            escreva("Terça-Feira")
        pare
        caso 4:
            escreva("Quarta-Feira")
        pare
        caso 5:
            escreva("Quinta-Feira")
        pare
        caso 6:
            escreva("Sexta-Feira")
        pare
        caso 7:
            escreva("Sábado")
        pare
        caso contrario:
            escreva("Dia inválido")
        pare
    }
}
```

# Praticar é fundamental



## Exercícios

1) Leia um número e retorne como resposta se ele é positivo, negativo ou zero.

Criar um algoritmo que receba quatro notas e calcule a média. Se a média for maior que 7 deverá ser exibida a mensagem aprovado caso contrário deverá ser exibida a mensagem reprovado.

2) Escreva um programa que encontre o valor máximo entre 2 números

**Exemplo :**

**Entrada: Digite um número: 2 Digite outro número: 1**

**Saída: O número 2 é maior que o número 1**

3) Escreva um programa que funcione como uma calculadora simples de soma (+), subtração(-), multiplicação(\*) e divisão(/)

**Exemplo:**

**Entrada: 10 \* 2**

**Saída esperada: 10 \* 2 = 20**

4) Implemente um programa tomador de decisão que considera as seguintes opções para determinar se o usuário usará a fila preferencial ou a fila comum.

O usuário usa a fila preferencial caso :

- Possui mais de 60 anos : Usa fila preferencial
- É deficiente físico : Usa fila preferencial
- É mulher gestante : Usa fila preferencial

O programa recebe como entrada a Idade, Sexo e a condição especial do usuário, se houver.

Exemplo de entrada: 22homem deficiente

Saída esperada: Fila preferencial

## Exercícios

Faça um programa para que leia a idade e o nome de um jogador de futebol.

Categorias:

De 10-17: categorias de base

18-40: profissional

acima de 40: master

abaixo de 10: escolinha

A resposta deverá ser conforme exemplo abaixo:

Entrada:

nome: João

idade: 30

Categoria: Profissional

A padaria Hotpão vende uma certa quantidade de pães franceses e uma quantidade de broas a cada dia. Cada pãozinho custa R\$ 0,50 e a broa custa R\$ 5,00. Ao final do dia, o dono quer saber quanto arrecadou com a venda dos pães e broas (juntos), e quanto deve guardar numa conta de poupança (10% do total arrecadado). Você foi contratado para fazer os cálculos para o dono. Com base nestes fatos, faça um algoritmo para ler as quantidades de pães e de broas, e depois calcular os dados solicitados.

Um motorista deseja colocar no seu tanque X reais de gasolina. Escreva um algoritmo para ler o preço do litro da gasolina e o valor do pagamento, e exibir quantos litros ele conseguiu colocar no tanque.

Faça um algoritmo que leia um número e retorne como resposta se ele é par ou ímpar

## Exercícios

Calcule o IMC conforme tabela e fórmula abaixo:

$$\text{IMC} = \frac{\text{PESO}}{(\text{ALTURA})^2}$$

| IMC                   | Classificações        |
|-----------------------|-----------------------|
| Menor do que 18,5     | Abaixo do peso normal |
| 18,5 - 24,9           | Peso normal           |
| 25,0 - 29,9           | Excesso de peso       |
| 30,0 - 34,9           | Obesidade classe I    |
| 35,0 - 39,9           | Obesidade classe II   |
| Maior ou igual a 40,0 | Obesidade classe III  |

*Classificação segundo a OMS a partir do IMC*

Faça um algoritmo que leia a idade de uma pessoa e de acordo com a idade exiba a seguintes mensagens:

Menor que 16 anos - não pode votar

Entre 16 e 18 anos e maior que 70 anos - voto opcional

Entre 18 e 70 anos - voto obrigatório

# Retrospectiva : O que já aprendemos?

- 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 ... )



# Retrospectiva : O que iremos aprender?

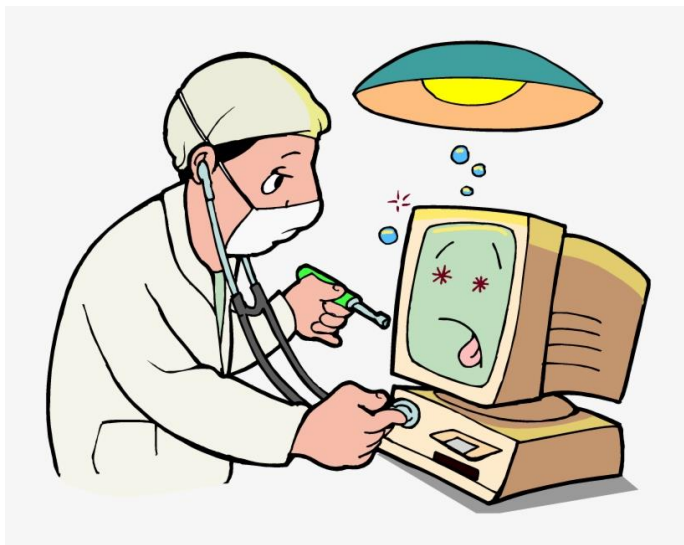
- Laços de repetição ( enquanto, para ... faça )
- Subrotinas ( Funções )
  - Recursividade
  - Bibliotecas
- Estruturas de dados ( Vetores, Matrizes, Filas e Pilhas )





# Enquanto isso...

- Devido à pandemia do coronavírus, não poderíamos começar nossas aulas da Residência de Software **enquanto** não fôssemos notificados. =(
- **Enquanto** isso, deveríamos ficar em casa aguardando novas notícias
- Como seria um programa de computador que representasse esse cenário?

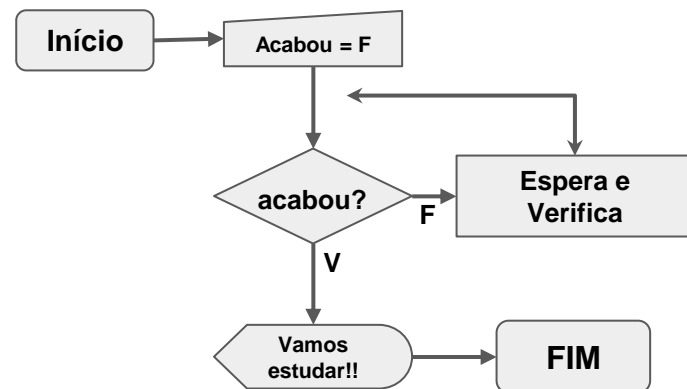


# Laços de repetição

- 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 Residencia de software!!")
    }// fim inicio
}// fim programa
```

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

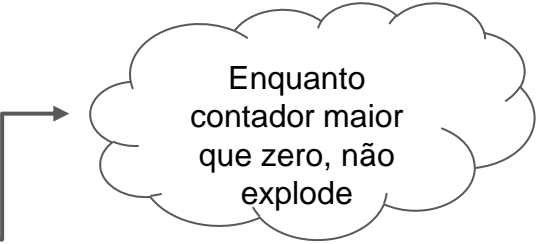


# Outro exemplo

- 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 segun  
    }  
    limpa()  
    escreva ("Booom!\n")  
  }  
}
```



Enquanto  
contador maior  
que zero, não  
explode



## Outro exemplo

Faça um programa usando o enquanto que escreva na tela números de 1 a 100.

```
programa
{
    funcao inicio()
    {
        inteiro numero
        numero = 1
        enquanto(numero<=100){
            escreva(numero + ",")
            numero++
        }
        escreva("Fim")
    }
}
```

Faça o mesmo exercício usando o para.

```
programa
{
    funcao inicio()
    {
        para(inteiro numero = 1; numero<=100;numero++){
            escreva(numero + ",")
        }
        escreva("Fim")
    }
}
```

## Exercício

Escrever um programa de computador que leia números inteiros e ao final, apresente a soma de todos os números lidos até que o valor digitado seja zero.

```
programa
{
    funcao inicio()
    {
        inteiro numero, total = 0
        escreva("Digite o número:")
        leia(numero)
        enquanto(numero != 0){
            total = total + numero
            escreva("Digite o número:")
            leia(numero)
        }
        escreva("Total:" + total)
    }
}
```

O Enquanto é muito utilizado quando precisamos fazer testes e não sabemos quantas vezes será realizado. No exemplo anterior precisamos fazer a leitura do número duas vezes porque a estrutura enquanto testa no início, neste caso podemos utilizar o **faça enquanto** que testa no final.

```
programa
{
    funcao inicio()
    {
        inteiro numero, total=0
        faca{
            escreva("Digite o número:")
            leia(numero)
            total = total + numero
        }
        enquanto(numero !=0)
            escreva("Total:" + total)
    }
}
```

# 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

O Para possui uma variável de controle, a qual podemos repetir um conjunto de instruções até um determinado número de vezes. A variável de controle é chamada de contador.

**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





Leia a idade de uma determinada quantidade de pessoas que também deverá ser informada pelo usuário e diga no final quantos são de maior e menor idade.

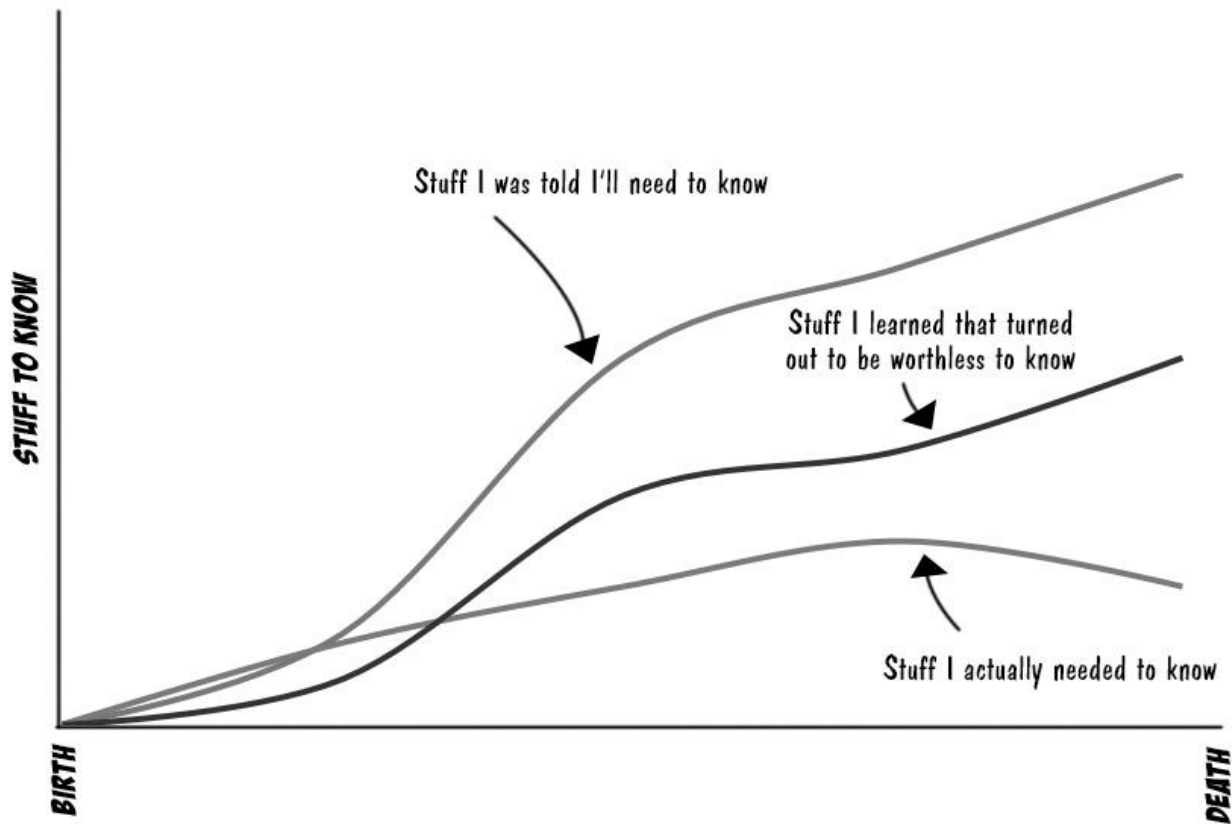
```
programa
{

    funcao inicio()
    {
        inteiro quantPessoas, idade, totalMaior18=0, totalMenor18=0
        escreva("Digite a quantidade de pessoas:")
        leia(quantPessoas)
        para(inteiro i=0; i<quantPessoas; i++){
            escreva("Digite a idade da pessoa:")
            leia(idade)
            se(idade >= 18){
                totalMaior18 ++
            }senao{
                totalMenor18 ++
            }
        }
        escreva("Total Maior de idade:" + totalMaior18, "\n")
        escreva("Total Menor de idade:" + totalMenor18)
    }
}
```

# 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 Portugal, essas estruturas são denominadas do inglês , **while** ( enquanto ) e **for** ( para )

# Aprender é um processo colaborativo



## Exercícios

Faça um programa que leia um número e apresente como resultado a multiplicação de 10 até 0.

Exemplo:  $3 \times 10 = 30$

$3 \times 9 = 27$

```
programa
{
    funcao inicio()
    {
        inteiro numero
        escreva("Digite o número:")
        leia(numero)
        para(inteiro i=10; i>=0;i=i-1){
            escreva("\n",numero,"x",i,"=",numero * i)
        }
    }
}
```

# 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 **início**

**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  
    }  
}
```

# 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 início()
  - 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 ...



# 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")  
    }  
}
```

Observe a diferença ao  
encapsularmos esse código  
repetido em uma função =D



Saída:

\*\*\*\*\*

Numeros entre 1 e 5

\*\*\*\*\*

1

2

3

4

5

\*\*\*\*\*

# Mais alguns exemplos - Recursividade

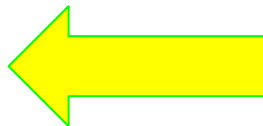
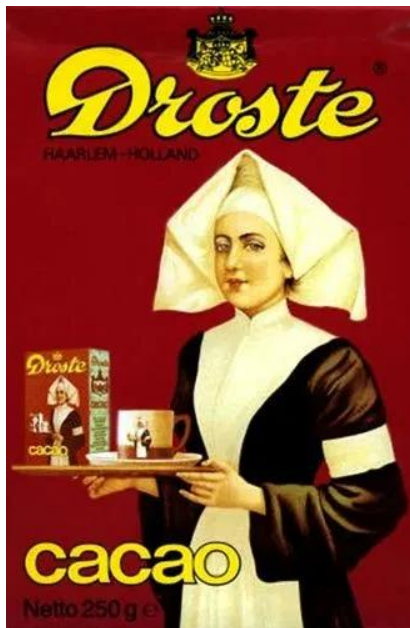
- Podemos também fazer a função chamar ela mesma para resolvermos problemas chamados **recursivos**.

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!!

$$5! = 5 * 4 * 3 * 2 * 1$$

5 \* (fatorial de 4)



# 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")  
    }  
}
```



Recordar é viver



# 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



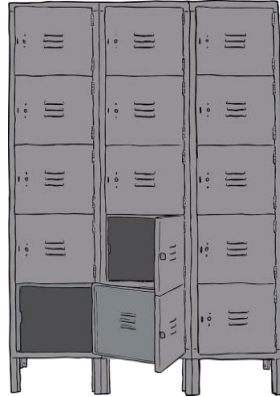
# Retrospectiva : O que iremos aprender?

- 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 dividida 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

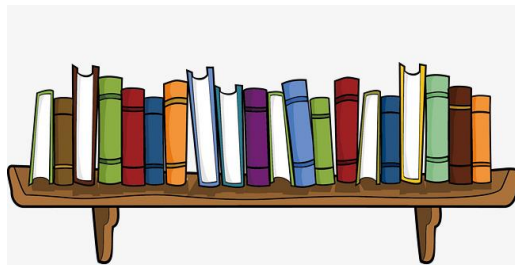
# Estruturas de dados : Conceitos

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

## DADO

Tipos de dados :

- Inteiro
- Ponto flutuante
- Caractere
- Texto



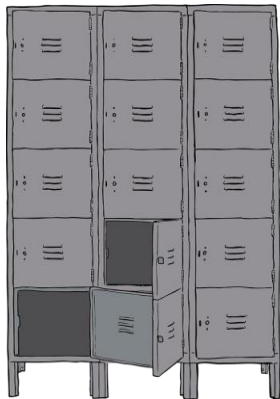
## ESTRUTURA

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 |

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

| 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 | 50 | 100 | 40 |

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

| Tipo | Nome     | Capacidade |
|------|----------|------------|
| int  | meuVetor | 5          |

inteiro meuVetor[5]

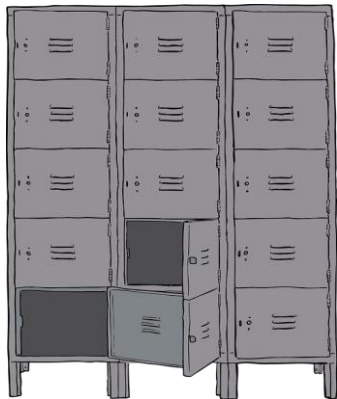
# Vetores

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

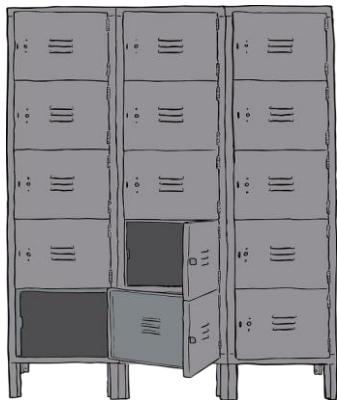
# Matrizes



| Tipo    | Nome       | Capacidade |
|---------|------------|------------|
| mochila | meuArmario | [5][3]     |

mochila meuArmario[5][3]

# Matrizes

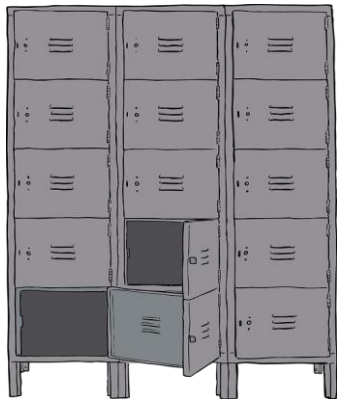


| Tipo    | Nome       | Capacidade |
|---------|------------|------------|
| mochila | meuArmario | [5][3]     |

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]     |

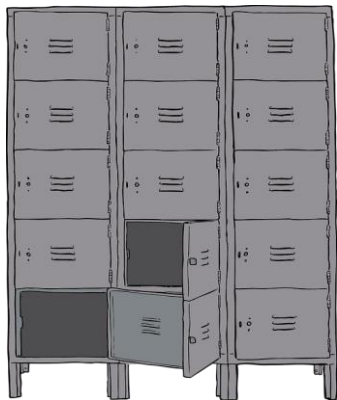
mochila meuArmario[5][3];

|     | [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



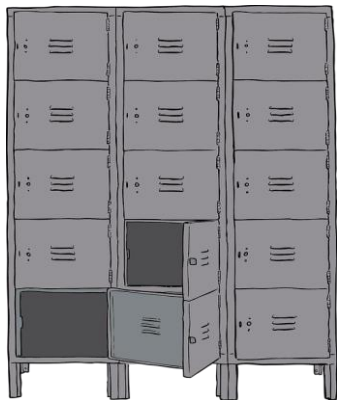
| Tipo    | Nome       | Capacidade |
|---------|------------|------------|
| mochila | meuArmario | [5][3]     |

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



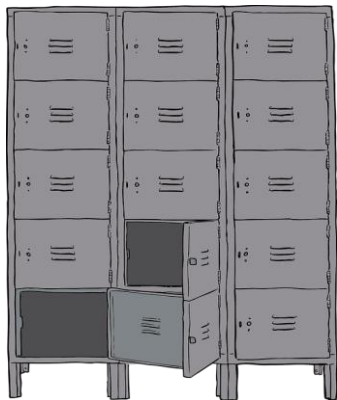
| Tipo    | Nome       | Capacidade |
|---------|------------|------------|
| mochila | meuArmario | [5][3]     |

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



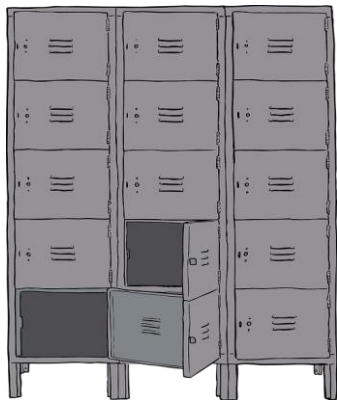
| Tipo    | Nome       | Capacidade |
|---------|------------|------------|
| mochila | meuArmario | [5][3]     |

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]     |

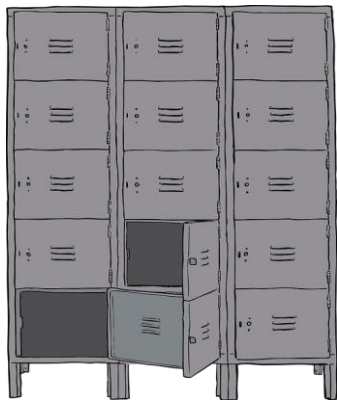
mochila meuArmario[5][3]

|     | [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**