POTÊNCIA FEMININA

GLOSSÁRIO

# Lógica de Programação

Aprendendo conceitos teóricos e práticos necessários para desenvolver um programa e resolver problemas.

## Consultas rápidas

Utilize este documento para consultar termos utilizados de forma recorrente no curso e facilitar o seu aprendizado.

Lembre-se de conversar com suas colegas, vizinhas e familiares sobre o que está aprendendo no Potência Feminina.

## Colabore

Ouviu algum termo no curso e gostaria de vê-lo aqui no glossário? Converse a pessoa que está facilitando este curso em sua ONG ou envie uma mensagem para equipe do Potência Feminina.

Agradecemos a sua ajuda 

**Lógica de programação:** distribuição sistêmica de ações, organizando o fluxo de tarefas que um software irá executar para desempenhar a função para o qual ele foi desenvolvido.

**Algoritmo:** sequencia lógica de passos que um programa executa para fazer determinada tarefa. Uma receita de bolo ou um passo a passo para plantio são exemplos de algoritmos.

**Software:** conjunto de componentes lógicos de um computador ou sistema de processamento de dados; programa, rotina ou conjunto de instruções que controlam o funcionamento de um computador (parte intangível de um computador).

**Hardware:** componentes físicos que compõem um computador ou qualquer outro dispositivo eletrônico (parte tangível de um computador).

**Linguagem de Programação:** é um conjunto de regras sintáticas e semânticas (ou seja, regras de escrita), símbolos e códigos usadas usados para orientar a escrita de estruturas (algoritmos) no desenvolvimento de software.

**Linguagem de Programação de Baixo Nível:** são voltadas para o entendimento da máquina, para o hardware. Por isso, elas têm uma sintaxe mais complexa e normalmente utilizam a "telinha preta" do computador, sem criação de uma interface. Exemplos de linguagens de baixo nível são *Assembly* e *Linguagem C*.

**Linguagem de Programação de Alto Nível:** as linguagens de alto nível abstraem conceitos voltados para a máquina e sintetizam comandos. Utilizam termos mais próximos utilizados por nós humanos. Exemplos de linguagens de alto nível são *JavaScript* e *Python*.

## Estrutura de um Programa em Portugol

**Inicialização:** no portugol, utilizamos a palavra *programa* acompanhada de um *par de chaves {}*. Tudo o que escrevermos dentro destas chaves será executado como parte do programa.

```
1 programa {
2     funcao inicio() {
3         //vamos aprender logica
4     }
5 }
6
```

**Comentários:** os comentários são partes do código que não serão executadas pelo computador. São apenas anotações. Para inserir anotações no seu código em Portugol, inserir um par de barras // antes a frase, ou um par de barra e asterisco /**/ quando o comentário ocupar mais de uma linha.

**Variáveis:** posições onde armazenaremos os dados na memória. As variáveis possuem *tipo do dado, nome* e o *valor* que essa variável terá.

```
1  programa {
2      funcao inicio() {
3          //Numericos
4          inteiro numero = 6
5          real valor = 6.3
6          //Literal
7          caracter status = 'S'
8          cadeia nome ='Ana'
9          //Logico
10         logico venda = verdadeiro
11     }
12 }
13
```

**Tipos de Dados:** basicamente, possuímos três tipos de dados:

- **Numérico:** números - inteiro e real (o separador dos números reais no portugol é o ponto).
- **Literal:** letras - caracter (apenas uma letra) e cadeia (duas ou mais letras).
- **Lógico:** binário - verdadeiro ou falso.

**Constante:** valor que não poderá ser alterado durante a execução do programa. Para declaração de uma constante no código, utilizamos o termos *const* acompanhado do *tipo de dado* que esta constante guardará, e *nome* que daremos à constante, em *letras maiúsculas*. Como não podemos utilizar espaço no nome das constantes e variáveis, caso seja necessário separar palvras na nomeclatura, utilizamos o *underline* _.

**Função:** uma função é um conjunto de instruções para execução de uma ação em específico, uma sub-rotina. Escrevendo funções, evitamos que o código fique todo repetido e seguimos a sequencia lógica para resolução de ´problemas. Possuímos dois tipos básicos de funções:

- **Entrada e Saída de Dados:** escreva(), leia() e limpa()
- **Tarefa:** criadas por quem está escrevendo o código

**Função escreva():** função de saída de dados que irá exibir na tela as informações desejadas.

```
programa {
    funcao inicio() {
        inteiro numero = 12
        real valor = 12.50

        escreva("Primeira linha \nSegunda linha \n")

    }
}
```

Utilizando *\n* para fazer a *quebra de linha*.

```
programa {
    funcao inicio() {
        inteiro numero = 12
        real valor = 12.50

        escreva("Rua XYZ, numero ", numero, ", bairro ABC")

    }
}
```

Buscando informação de uma variável ou constante.

**Função escreva():** função de saída de dados que irá exibir na tela as informações desejadas.

```
programa {
    funcao inicio() {

        inteiro valorA = 5
        inteiro valorB = 3

        escreva("Resultado da soma é ", valorA + valorB)
    }
}
```

Exibindo na tela o resultado de uma operação matemática.

**Função leia():** função de entrada de dados que irá inserir os dados que o usuário informar dentro do programa.

```
programa {
    funcao inicio() {

        inteiro valorA = 5
        inteiro valorB = 3
        inteiro idade
        cadeia coordenadas


        escreva("Digite a sua idade: \n")
        leia(idade)

        escreva("\nDigite as suas coordenadas:\n")
        leia(coordenadas)

        escreva("\nCoordenadas: ", coordenadas, "\n", "Idade: ", idade)
    }
}
```

Solicitando informações ao usuários e as exibindo na tela.

**Função limpa():** função que limpa as informações do console (exibidas em tela).

```
programa {
    funcao inicio() {

        inteiro valorA = 5
        inteiro valorB = 3
        inteiro idade
        cadeia coordenadas


        escreva("Digite a sua idade: \n")
        leia(idade)

        limpa()

        escreva("\nDigite as suas coordenadas:\n")
        leia(coordenadas)

        limpa()

        escreva("\nCoordenadas: ", coordenadas, "\n", "Idade: ", idade)
    }
}
```

Exibindo em tela uma solicitação de cada vez, sem encher a tela do console de informações.

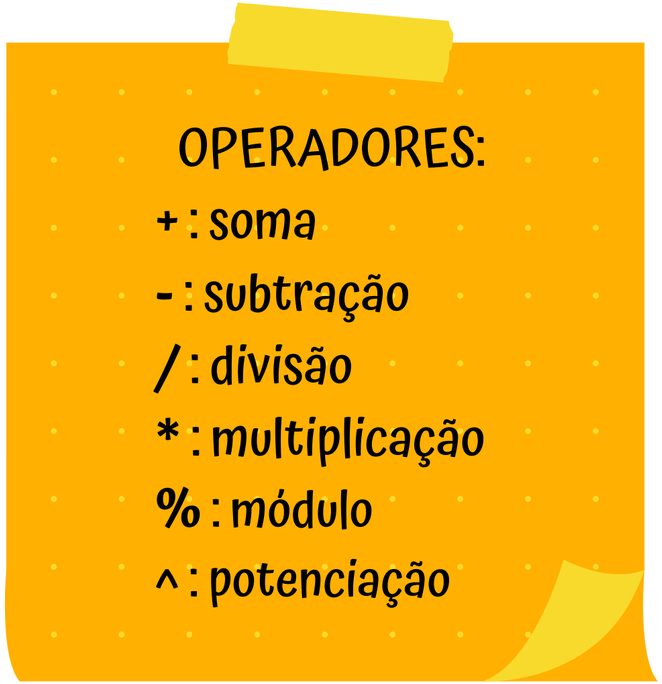
**Criando sua própria função:** Fora da função início, você pode criar uma função que realize uma ação específica. Para criá-la, você precisa colocar a palavra *funcao*, seguida do *tipo de dado* que ela vai retornar ao final de sua execução, e o *nome da função*. Após o nome, entre parênteses, você vai colocar os *atributos* necessários para que a função seja executada, sempre colocando o tipo de dado desse atributo. Após isso, abra e feche chaves {} e entre chaves, escreva o passo a passo que a função vai executar para realizar a ação que você precisa. O restante do código precisa seguir o padrão que aprendemos até aqui: *função início*, informar o *tipo de dado das variáveis* do código, *funções escreva, leia e limpa*, e *função retorne*, que vai trazer ao fim da execução da função que você criou, os dados que você quer/precisa.

```
programa {
    funcao inicio() {

        inteiro valorA, valorB

        escreva("Digite um valor para A: ")
        leia(valorA)

        escreva("\nDigite um valor para B: ")
        leia(valorB)

        limpa()

        soma_de_numeros(valorA,valorB)
        escreva("O resultado da soma é ", soma_de_numeros(valorA,valorB))
    }

    funcao inteiro soma_de_numeros (inteiro valorA, inteiro valorB) {
        retorne valorA + valorB
    }
}
```

**Obs:** Não esqueça que o nome da função não pode ter espaços ou caracteres especiais, como pontuações; utilize o underline para separar as palavras.

## Operadores

**operadores aritméticos :** conjunto de símbolos que represetam as funções básicas da matemática (soma, subtração, dividisão, multiplicação, potenciação e módulo).




**Obs:** No caso dos exemplos a seguir, o usuário poderia informar ao programa até mesmo um número com vírgula, pois o tipo de dado das variáveis é real.
As variáveis soma, sub, mult e div, nesse caso, são o resultado de uma operação aritimética entre outras duas variáveis (A e B) que foram informadas pelo usuário.

**soma:** soma simples de dois números.

```
programa {
    funcao inicio() {

        real a, b, soma

        escreva("Informe o valor de A: ")
        leia(a)

        escreva("\nInforme o valor de B: ")
        leia(b)

        soma = a + b

        limpa()

        escreva("O resultado de A + B é ", soma)
    }
}
```

**subtração:** subtração simples entre dois números.

```
programa {
    funcao inicio() {

        real a, b, sub

        escreva("Informe o valor de A: ")
        leia(a)

        escreva("\nInforme o valor de B: ")
        leia(b)

        sub = a - b

        limpa()

        escreva("O resultado de A - B é ", sub)
    }
}
```

**multiplicação e divisão:** multiplicação e divisão simples entre dois números.

```
programa {
    funcao inicio() {

        real a, b, mult, div

        escreva("Informe o valor de A: ")
        leia(a)

        escreva("\nInforme o valor de B: ")
        leia(b)

        mult = a * b
        div = a / b

        limpa()

        escreva("O resultado da multiplicação de A e B é ", mult)
        escreva("\nO resultado da divisão de A por B é ", div)

    }
}
```

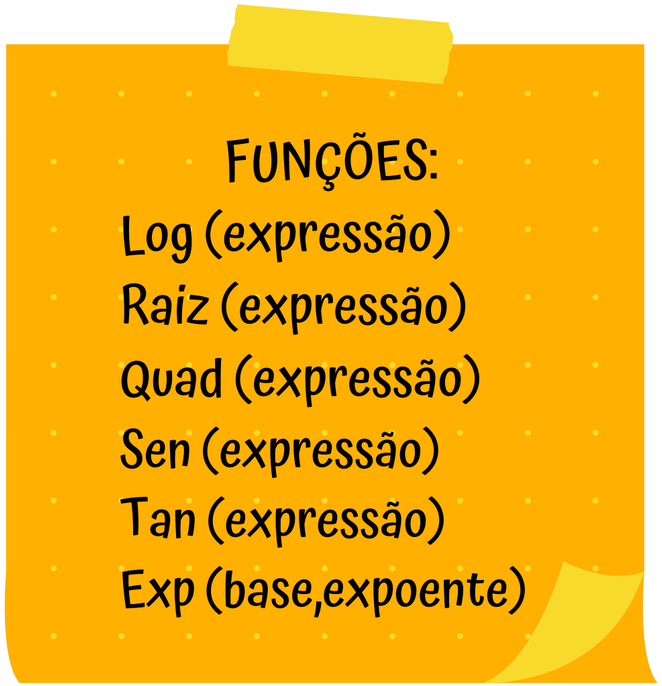
No exemplo, estamos multiplicando e dividindo os números das variáveis A e B, informados pelo usuário, utilizando o mesmo código.

**Expressões aritméticas:** são constituídas por operadores aritméticos, constantes e variáveis numéricas. O resultado é sempre um valor numérico.

```
programa {
    funcao inicio() {

        real a, b, c, p, exp

        a = 2 + 3 //soma
        b = 6 - 2 //subtração
        c = 5 * 5 //multiplicação
        p = 2 ^ 10 //potenciação
        exp = a + b + c + p // expressão aritmética

        escreva("Valor de A: ",a)
        escreva("Valor de B: ",b)
        escreva("Valor de C: ",c)
        escreva("Valor da Expressão: ",p)
    }
}
```

```
Valor de A: 5.0
Valor de B: 4.0
Valor de C: 25.0
Valor de P: 8.0
Valor da Expressão: 42.0
Programa finalizado.
```

**Funções aritméticas:** a maioria das linguagens de programação já possuem as funções aritméticas implementadas. Nesse caso, há um termo que quando escrito no código e recebendo as variáveis necessárias, executa o cálculo.



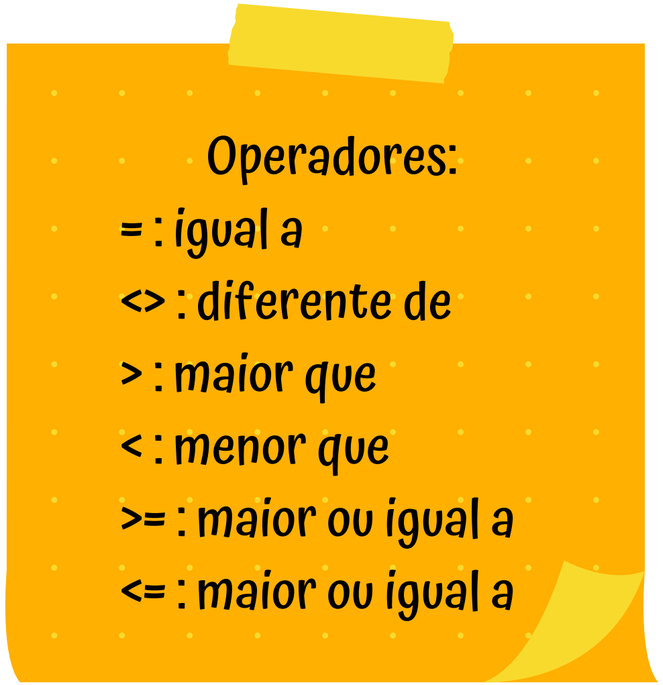

Essas funções já estão implementadas nas linguagens para que não seja necessário criar uma função apenas para calcular o seno ou o logaritmo de um número, por exemplo. Isso existe no sentido de facilitar para a pessoa programadora e para que o código fique mais organizado e "limpo".

```
programa {

    inclua biblioteca Matematica --> mat

    funcao inicio() {

        real numero, raiz

        numero = 4.0
        raiz = mat.raiz(numero,2.0)

        escreva("A raiz quadrada de ", numero, " é: ", raiz)
    }
}
```

```
A raiz quadrada de 4.0 é: 2.0
Programa finalizado.
```

Nesse exemplo, precisamos incluir uma *biblioteca* específica que contém as funções aritméticas que vamos utilizar (a definição de *biblioteca* em linguagem de programação está ao final deste material, não se preocupe).

**Operadores relacionais:** utilizamos para comparação de valores, e essa comparação sempre nos dará um resultado booleano (verdadeiro ou falso).




Utilizamos operadores relacionais para validações e comparações de constantes e/ou variáveis dentro do nosso código.

No exemplo, utilizamos os operadores de comparação *menor igual que*, *menor que* e *maior que*.

```
programa {
    funcao inicio() {
        real score

        escreva("SOLICITAÇÃO DE CARTÃO DE CRÉDITO {SCORE}\n\n")
        escreva("Qual é o score do seu cliente? ")
        leia(score)
        limpa()

        se(score <= 100){
            escreva("Indeferimento automático: cliente inadimplente.\n")
        }

        senao se(score<250){
            escreva("Seu cliente não foi aprovado. Solicite a regularização do seu CPF.\n")
        }

        senao se(score>500){
            escreva("Cartão de crédito premium aprovado!\n")
        }senao{
            escreva("Score provado: liberar crédito.\n")
        }
    }
}
```

## Algoritmos Sequenciais e Estruturas de Controle

**Algoritmos sequenciais:** comando sendo executados numa sequencia pré-estabelcida, um após o outro. Estruturas de decisão permitem escolher qual caminho seguir de acordo com as decisões tomadas pelo usuário.

Início > Entrada > Processamento > Saída > Fim

```
programa
{
    funcao inicio() // INÍCIO
    {
        const inteiro MAIORIDADE = 18

        inteiro idade, anos

        escreva("Digite sua idade: ") // ENTRADA
        leia(idade)

        anos = MAIORIDADE - idade // PROCESSAMENTO

        se (anos > 0)
        {
            escreva("Falta(m) ", anos, " ano(s) para você atingir a maioridade\n") // SAÍDA
        }
        senao
        {
            escreva("Você já atingiu a maioridade\n") // SAÍDA
        }
    } // FIM
}
```

Essa sequência é importante para a construção lógica do seu código, e para que ele solucione o problema ou execute sua função da melhor forma possível.

**Estrutura de Controle:** ordem em que instruções, chamadas de funções e expressões são executadas ou avaliadas em um programa.

Início > N1, N2 > Média = (N1+N2)/2 > Média > Fim

```
programa {
    funcao inicio() { //cálculo de média

    real n1, n2, media

    escreva("Digite a primeira nota: ")
    leia(n1)
    escreva("\nDigite a segunda nota: ")
    leia(n2)

    media = (n1 + n2)/2

    escreva("\nA média da sua nota é: ", media)

    }
}
```

A ordem de execução é lógica e a mais correta para a resolução do problema proposto. Precisamos que o usuário informe as duas médias antes de realizarmos o cálculo, e precisamos manter o padrão de estrutura da linguagem (função início, declaração do tipo e nome das variáveis, abrir e fechar todos os parênteses e chaves...).

```
Digite a primeira nota: 7

Digite a segunda nota: 8

A média da sua nota é: 7.5
Programa finalizado.
```

## Estruturas de Laços de Repetição

**Fundamento:** estrutura que permite que permite executar um comando (ou um conjunto de comandos) mais de uma vez, de acordo com uma condição ou um contador. *Exemplo: realizar uma multiplicação até o resultado ser um determinado número, distribuir itens entre pessoas até que cada uma possua 5 itens.*

**PARA** <variável contadora> **DE** <valor inicial> **ATE** <valor final> [**PASSO** <valor de incremento>] **FAÇA**
<instruções a serem executadas repetidamente até a <variável contadora> atingir o valor final> **FIM-PARA**

**Estrutura PARA:** executa o código em looping, várias vezes, de acordo com as condições informadas dentro do parênteses que acompanha a expressão *para*.

```
programa {
    funcao inicio() {

        inteiro numero, resultado, contador

        escreva("Insira o número para realizar a tabuada: ")
        leia(numero)
        limpa()

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

No exemplo, a função PARA é composta de três condições: *contador inicia no número 1* (contador=1), a função vai ser executada *até que o contador seja menor ou igual ao número 10* (contador<=10), e *cada vez que a função é executada é somado um número ao contador* (contador++).

**Estrutura ENQUANTO:** executa o código em looping, como a estrutura PARA, sendo mais indicado o uso para casos em que não sabemos ao certo quantas vezes o código deverá ser executado. Possui apenas uma condição para sua execução, e ela precisa ser verdadeira.

**ENQUANTO** <expressão booleana> **FAÇA** <instruções a serem executadas ENQUANTO a expressão booleana for VERDADEIRA> **FIM-ENQUANTO**

```
programa {
    funcao inicio() {

        escreva("Insira um número para o contador: ")

        inteiro numero
        leia(numero)

        escreva("Contando de ", numero,  " até 1: \n")

        enquanto(numero>0){
            escreva(numero+"\n")
            numero = numero - 1
        }
        escreva("Fim da contagem")
    }
}

```

Enquanto o número que o usuário informou -1 for maior que zero, o programa seguirá fazendo a contagem. Quando utilizar a estrutura ENQUANTO, tome cuidado para não colocar o programa em um looping infinito.

**Estrutura FAÇA ENQUANTO:** parecida com a estrutura enquanto, executa um conjunto de instruções enquanto a condição for verdadeira. A diferença é que o FAÇA ENQUANTO executa pelo menos uma vez o conjunto de instruções, depois verifica se a condição é verdadeira para que possa executar novamente o código.

```
programa {
    funcao inicio() {

        inteiro idade

        faca{
            escreva("Informe sua idade para criar uma conta no Facebook: ")
            leia(idade)
        }enquanto(idade<18 ou idade>90)

        escreva("\nCriação de conta liberada")
    }
}
```

No portugol, utilizamos a expressão *faca* para iniciar uma estrutura faça enquanto, seguida de *chaves {}* e, ao fechar chaves, a expressão *enquanto()* com a condição a ser verificada para executar novamente dentro do parênteses.

## Vetores e Matrizes

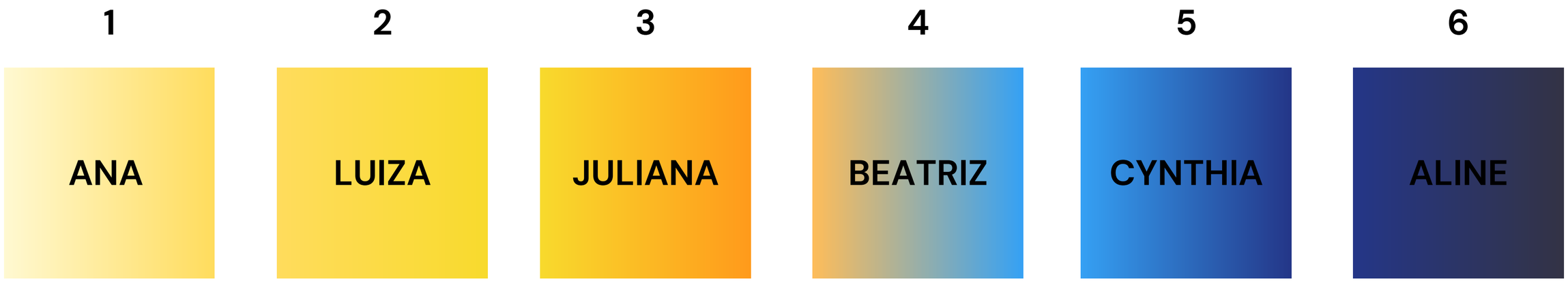
**Estrutura de dados:** ramo da computação que se dedica a estudar os mecanismos de organização de dados para atender aos diferentes requisitos de processamento. Elas definem a organização, o método de acesso e as opções de processamento para as informações manipuladas pelo programa.

**Dados homogêneos:** grupo de dados do mesmo tipo, armazenados em uma variável. Elas podem ser unidimensionais (vetores) ou multidimensionais (matrizes).

**Dados heterogêneos:** conjunto de tipos de dados distintos, em uma mesma estrutura.

**Estruturas de Dados mais utilizadas:** array, lista, fila, pilha e árvore.

**Arrays:** compreendem objetos com vários valores de um mesmo tipo, ou seja, dados homogêneos. Esses valores são identificados ou referenciados por um índice. Esse índice é um valor inteiro que é responsável por determinar a posição de cada elemento dentro de um array. Os arrays podem ser de dois tipos: vetor (unidimensionais) e matrizes (multidimensionais).



**Vetores:** é uma variável que armazena várias variáveis do mesmo tipo. Vetor é um tipo de array unidimensional.

```
programa {
    funcao inicio() {

        cadeia nome[] = {"Ada", "Grace", "Mary", "Carol", "Frances"}
        inteiro nascimento[] = {1815, 1906, 1913, 1955, 1932}

        escreva("Mulheres que marcaram a computação:\n")

        para(inteiro posicao=0; posicao<5; posicao++){
            escreva(nome[posicao], "\t\t", nascimento[posicao], "\n")
        }
    }
}
```

Nesse exemplo, a função *para* verifica a posição do item dentro do array, e o exibe na tela, até que ele seja menor que 5. Todo vetor começa em zero, então, nesse caso, o item *Ada* no array tipo cadeia *nome*, possui vetor 0

**Matrizes:** coleção de variáveis de mesmo tipo, acessíveis com um único nome e armazenados contiguamente na memória. A individualização de cada variável de um vetor é feita através do uso de índices; no caso da matriz, um índice vertical e um horizontal. Os índices iniciam em zero. Para declarar uma variável que seja uma matriz, colocamos o *tipo de dado*, *nome* da variável, seguido da *quantidade de linhas e colunas dentro de colchetes []*, separadamente. Os itens devem ser declarados dentro de *chaves {}*, separados por *vírgula*, e cada linha é declarada separadamente. Para acessar apenas um item de uma matriz, informamos o nome da variável acompanhada de parênteses, com o índice vertical e horizontal do item.

```
programa {
    funcao inicio() {

        inteiro notas[4][4] = {{1,2,3,4},
        {1,2,3,4},
        {1,2,3,4},
        {1,2,3,4}}

        para(inteiro l=0;l<4;l++){
            para(inteiro c=0;c<4;c++){
                escreva(notas[l][c],", ")
            }
            escreva("\n")
        }
    }
}
```

No exemplo, declaramos uma matriz de 4 linhas e 4 colunas, com notas atribuídas em cada posição. A função *para* imprime cada item até que se completem as 4 linhas e colunas.

```
1, 2, 3, 4,
1, 2, 3, 4,
1, 2, 3, 4,
1, 2, 3, 4,

Programa finalizado.
```

**Registro:** estrutura que fornece um formato especializado para armazenar informações em memória. Estão entre as estruturas de dados heterogêneas mais simples. O recurso do registro permite que se armazene mais de um tipo de dado. Um registro é um valor que contém outros valores, tipicamente em número fixo e sequência e normalmente indexados por nomes. Para acessar um registro específico, utilizamos o *nome da variável*, seguida de *ponto*, e o *nome do registro*.

**Portugol:** é uma pseudolinguagem que permite desenvolver algoritmos estruturados em português de forma relativamente mais simples e intuitiva

**Interface:** é o veículo que o usuário irá interagir com determinado sistema tanto fisicamente, perceptivamente assim como conceitualmente.

**Ada Lovelace:** foi a primeira pessoa programadora da história. Em 1815, ela escreveu o primeiro algoritmo.

**IDE:** sigla em inglês para *integrated development environment* que significa *ambiente de desenvolvimento integrado*. É um editor de texto que auxilia na criação de código de software por meio de funcionalidades como destaque da sintaxe com indicadores visuais, recurso de preenchimento automático específico da linguagem e verificação de bugs durante a criação. O Portugol Webstudio, que utilizamos nas aulas, é uma IDE.

**Biblioteca:** é uma coleção de subprogramas utilizados no desenvolvimento de software. Bibliotecas contém código e dados auxiliares, que provém serviços a programas independentes, o que permite o compartilhamento e a alteração de código e dados de forma modular.

**Open Source:** termo em inglês que significa *código aberto*. Isso diz respeito ao código-fonte de um software, que pode ser adaptado para diferentes fins.