5 - PSEUDOCÓDIGO

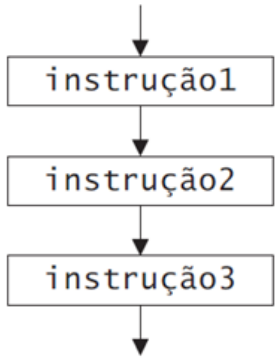
O pseudocódigo é uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples, sem ser necessário conhecer a sintaxe da linguagem de programação.

Na escrita de programas em pseudocódigo devem ser consideradas as seguintes opções:

- Os algoritmos são delimitados pelas etiquetas *início* e *fim;*
- As etiquetas *Entrada:* e *Saída:* são utilizadas para referir as entradas de dados (inputs) e saídas de dados (outputs) respetivamente;
- Os comentários são precedidos do símbolo '#' e servem apenas para documentar o algoritmo, não têm validade em termos de processamento;
- As ações são descritas através de verbos no infinitivo

5.1 - INSTRUÇÕES SEQUENCIAIS

As instruções do tipo sequencial são as mais simples de todas, apresentando uma estrutura atómica. São responsáveis pela entrada e saída de dados, execução de cálculos e atribuição de valores a variáveis. A noção de sequência é representada através de setas de fluxo (ver exemplo abaixo).




```
inicio
...
<instrução1>;
<instrução2>;
<instrução3>;
...
fim
```

5.2 - VARIÁVEIS, O QUE SÃO E PARA O QUE SERVEM

Considere uma variável como uma caixa que pode ser utilizada para armazenar dados ou informação. Ao longo dos nossos programas vamos utilizar variáveis para armazenar valores, que podem ser números, cadeias de texto, valores boleanos e até mesmo dados mais complexos como coleções e objetos. Estas variáveis podem ser manipuladas, testadas, comparadas e podem, ao longo do curso do programa verem o seu conteúdo alterado, ou em alguns casos, mantido inalterado ao longo do curso do programa. As variáveis podem conter determinados tipos de dados e geralmente operações só devem ser executados em variáveis do mesmo tipo de dado.

Estas variáveis são na prática áreas de armazenamento da memória RAM do computador onde podermos armazenar valores e mudar esse valor ao longo da execução do programa.

5.2 - VARIÁVEIS, O QUE SÃO E PARA O QUE SERVEM

As variáveis possuem três atributos:

- Nome

Todas as variáveis têm um nome, que serve para a distinguir das restantes. Cada linguagem de programação adota as suas regras para compor o nome de uma variável. Nos nossos algoritmos usaremos algumas regras:

  - O nome da variável deve começar por uma letra;
  - Não deve conter símbolos, exceto o underscore (_);
  - Não deve conter espaços em branco;
  - Não pode conter palavras reservadas a instruções do programa;

- Tipo de dado

Outro atributo importante das variáveis é o seu tipo de dado que ela pode armazenar, e define a natureza da informação nela contida.

- Conteúdo (informação)

O conteúdo não é mais do que a informação que a variável contém.

5.3 - TIPOS DE DADOS

As tarefas executadas por um computador baseiam-se na manipulação das informações contidas na sua memória. Estas informações podem ser classificadas em dois tipos principais:

- Instruções: São as instruções que comandam o funcionamento do computador e determinam a forma como os dados vão ser tratados.

- Dados: Os dados correspondem à porção das informações a serem processadas pelo computador.

5.3 - TIPOS DE DADOS (cont…)

A classificação abaixo não se aplica a nenhuma linguagem de programação específica, mas sintetiza as normas utilizadas na maioria das linguagens.

- Tipos Inteiros
  São considerados como dados do tipo inteiro todos os dados numéricos, positivos ou negativos, excluindo os números fracionários. Alguns exemplos deste tipo de dado: 1, 0, -35, 44, 2048 entre outros.

- Tipos Reais
  São considerados como dados to tipo real os dados numéricos, positivos ou negativos, incluindo os números fracionários. Alguns exemplos deste tipo de dado: 44, 32.354, -62.524 entre outros.

5.3 - TIPOS DE DADOS (cont…)

- Tipos Caracter

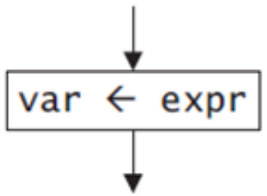
  São caracterizados como tipo carater as sequencias contendo letras, números e símbolos especiais. Uma sequência de carateres deve ser contida entre aspas (“”). Este tipo de dado também é chamado de alfanumérico, string ou cadeia de texto. Como exemplo deste tipo de dados temos: “Rua da Boavista”, “65”, “00351919813074”, “” entre outros.

- Tipos Lógicos

  São considerados dados do tipo lógico os dados com o valor verdadeiro (true) e falso (false), sendo que este tipo de dado apenas pode representar um dos dois valores. É também apelidado de tipo booleano devido à contribuição do filósofo e matemático George Boole na área da matemática.

5.4 - ATRIBUIÇÃO DE VALORES A VARIÁVEIS

A expressão de atribuição permite atribuir um valor ou resultado de uma expressão a uma variável. A variável é escrita do lado esquerdo e vai receber o valor que aparece do lado direito. Como valor podemos ter números texto, o resultado de uma operação ou o valor de uma outra variável. Vejamos a sintaxe para a atribuição:




```
inicio
...
<variável> ← <expressão>;
...
fim
```

5.5 - EXEMPLOS

```
inicio
# ler a variável x;
ler(x);
# ler as variáveis nome e idade
ler(nome, idade);
fim
```

5.5 – EXEMPLOS (cont…)

```
inicio
# atribuir o valor 5 à variável x;
x ⟵ 5;
# atribuir o resultado da operação 5 * 5 – 2 = 23 à variável resultado;
resultado ⟵ 5 * 5 * - 2;
# atribuir o resultado da variável n à variável máximo
maximo ⟵ n;
# atribuir o texto “Olá Mundo” à variável texto;
texto ⟵ “Olá Mundo”;
fim
```

5.6 - DECLARAÇÃO DE VARIÁVEIS EM ALGORITMOS

Nos algoritmos, todas as variáveis utilizadas são definidas no início do mesmo, através das seguintes formas:

```
VAR <nome da variável>: <tipo da variável>
VAR <lista de variáveis>: <tipo das variáveis>
```

Variáveis de tipos diferentes devem ser declaradas em linhas separadas.

```
VAR nome: carater;
VAR idade: inteiro;
VAR media: real;
VAR casado: lógico;
```

5.7 - CONSTANTES

Define-se como constante tudo o que é fixo ou estável durante a execução do nosso programa. As constantes podem ser usadas em programação sempre que um valor é imutável ao longo do curso do programa.

As constantes são normalmente definidas no início do programa e têm nomes descritivos que facilitam o seu entendimento pelos programadores.

Em algoritmia definimos as contantes da seguinte forma:

```
CONST <nome_da_constante> = <valor>
```

Exemplo declaração de constante:

```
CONST pi = 3.14;
CONST chave_api = ”sad33esd44e33dccxsec45450”;
```

5.7 – CONSTANTES (cont..)

Em Python, as constantes são inicializadas exatamente como uma variável, usando um nome para a constante, o operador de atribuição “=” e o respetivo valor.

A única alteração significativa na sintaxe de uma constante está relacionada com o nome, que por convenção deve ser sempre em maiúsculas e os termos separados por underscore “_”.

Exemplo em Python:

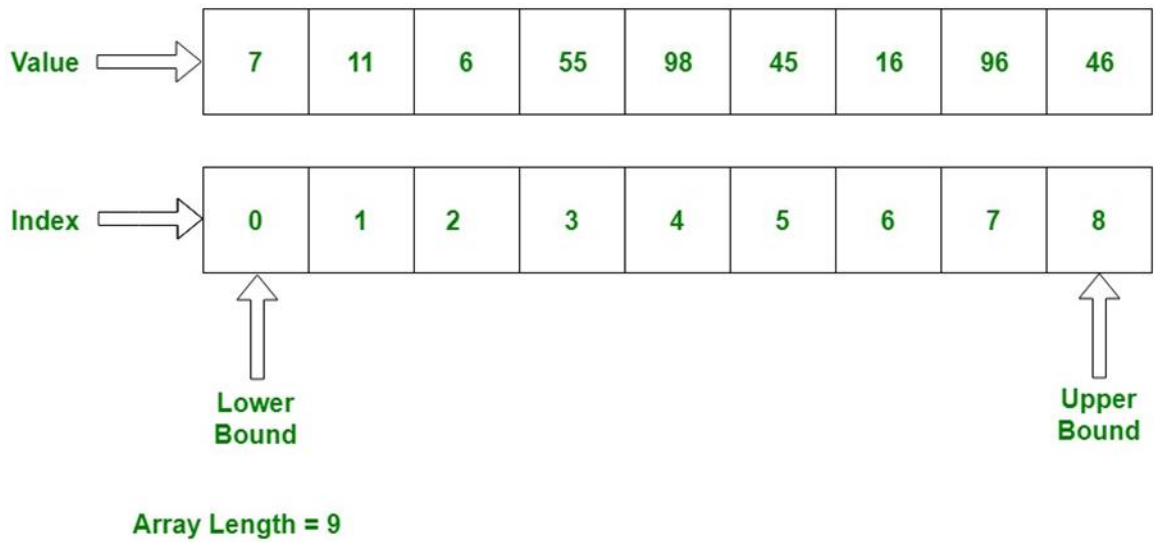
```
PI = 3.14
```

5.8 - VETORES

Um vetor é um tipo especial de variável unidimensional que pode armazenar várias variáveis ao mesmo tempo. A atribuição de valores a um vetor não pode ser feita diretamente de uma instrução “Ler”, sendo que cada valor deve ser lido e atribuído de forma independente à sua posição (índice). Para implementar este tipo de leitura utilizam-se geralmente ciclos.

Este tipo de variável é bastante útil quando queremos armazena um conjunto de dados do mesmo tipo. Imaginemos como exemplo um programa que necessita de armazenar o nome de 20 alunos de uma turma. Se utilizarmos variáveis normais, necessitaríamos de 20 variáveis com nomes distintos para armazenar toda a informação (exº: nome1, nome2, nome3…). Com os vetores, podemos criar uma variável que vai conter todos os nomes, cada um numa posição específica (índice). Cada valor poderá ser acedido pela sua respetiva posição.

5.8 – VETORES (cont…)



5.8 – VETORES (cont…)

Exemplo:

```
nome[0] = “Pedro Gouveia”;
nome[1] = “Luísa Rocha”;
nome[2] = “Jorge Mendes”;
…
nome[19] = “Sandra Coimbra”;
```

Nota: como deve ter reparado, o índice do vetor neste exemplo inicia em 0 (zero), o que é comum em várias linguagens de programação, sendo que um vetor de 20 posições tem como limites [0..19].

Como declarar vetores:

```
var nome_do_vetor[1..limite]:inteiro;
```

5.9 - VETORES EM PSEUDOCÓDIGO (cont…)

Usando o exemplo acima, como faríamos para implementar a leitura das notas de 20 alunos para um vetor?

Uma possível solução (ineficiente e desaconselhada) seria:

```
inicio
var notas[1..20]: inteiro;
escrever("introduza a nota do aluno 1");
ler(nota);
notas[1] = nota;
escrever("introduza a nota do aluno 2");
ler(nota);
notas[2] = nota;
escrever("introduza a nota do aluno 3");
ler(nota);
notas[3] = nota;
...
...
fim
```

5.9 - VETORES EM PSEUDOCÓDIGO (cont…)

A solução mais correta seria então:

```
inicio
para num ← 1; num < 20; num + 1 fazer
    escrever("Introduzir a nota do aluno ", num);
    ler(nota[num]);
fim-para
fim
```

5.9 - MATRIZES

As matrizes são variáveis semelhantes aos vetores, mas diferem num ponto, são multidimensionais em vez de unidimensionais.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

5.9 – MATRIZES (cont …)

Exemplo em pseudocódigo:

```
#Ler matriz 3x4
inicio
        var tabela:matriz[0..2, 0..3];
        var i, j :inteiro;
        para i de 0 até 2 passo 1 fazer
                para j de 0 até 3 passo 1 fazer
                        ler(tabela[i,j]);
                fim-para
        fim-para
fim
```

Desafio: Criar um algoritmo em pseudocódigo para escrever os valores da matriz anterior.

5.10 - OPERADORES

Na escrita de algoritmos são utilizados operadores relacionais, lógicos e aritméticos que são apresentados nas tabelas seguintes.

5.10.1 - RELACIONAIS

$<$	menor que
$>$	maior que
$\geq$	maior ou igual que
$\leq$	menor ou igual que
$=$	igual
$\neq$	diferente

5.10.2 - LÓGICOS

e , $\wedge$	conjunção
ou , $\vee$	disjunção
não, $\neg$	negação

5.10. 3 - ARITMÉTICOS

+	soma
-	subtracção
*	multiplicação
/	divisão
div	divisão inteira
%	resto da divisão inteira

5.11 - ESTRUTURAS LÓGICAS E DE CONTROLO

No que diz respeito à programação, a lógica tem a ver com a avaliação de uma instrução do ponto de vista da mesma ser verdadeira ou falsa. Com base nesta avaliação, decisões são efetuadas no fluxo de um programa, que levam ações e/ou resultados distintos.

Um programador, ao desenvolver um programa, tem de conseguir instruir o computador não só no que deve executar, mas também quando executar.

Na programação utilizamos estruturas de controlo para avaliar e controlar o fluxo de execução dos nossos programas, rotinas ou sub-rotinas. As estruturas de controlo podem ser de três tipos – estruturas de sequência, decisão/seleção ou de repetição/ciclos.

5.12 - ESTRUTURAS DE SEQUÊNCIA

Refere-se à execução linha a linha, na qual as instruções são executadas sequencialmente, na mesma ordem em que aparecem no programa. Eles podem, por exemplo, realizar uma série de operações de leitura ou gravação, operações aritméticas ou atribuições de valores a variáveis.

5.13 - ESTRUTURAS DE DECISÃO/SELEÇÃO

As estruturas de decisão/seleção permitem selecionar uma opção entre as disponíveis, executando sequências ou blocos de código alternativos mediante a verificação de uma determinada condição. As estruturas de decisão/seleção são compostas por:

- Decisão/seleção simples
- Decisão/seleção composta
- Decisão/seleção encadeada
- Decisão/seleção múltipla

5.13.1 – DECISÃO / SELEÇÃO SIMPLES

Este tipo de estrutura de controlo permite a execução de uma determinada ação mediante a satisfação de uma condição verdadeira. Caso a condição seja falsa, o programa continua a sua execução fora da condição ou termina.

Exemplo geral:

```
se <condição> então
        <instruções>
fim-se
```

Em programação esta estrutura habitualmente é referida como if-then.

5.13.1 – DECISÃO / SELEÇÃO SIMPLES

Abaixo um exemplo da utilização desta estrutura:

```
inicio
        ler (idade);
        se idade < 65 então
            dif ← 65 – idade;
            escrever("Faltam ", dif, " para atingir a reforma");
        fim-se;
fim
```

5.13.2 – DECISÃO / SELEÇÃO COMPOSTA

Através deste tipo de estrutura de decisão/seleção, podem existir duas opções possíveis para a execução de um conjunto de instruções que são selecionadas mediante um teste lógico (verdadeiro ou falso) de uma determinada condição ou expressão.

Exemplo geral:

```
se <condição> então
        <instruções>
se-não
        <instruções>
fim-se
```

Em programação esta estrutura habitualmente é referida como *if-then-else.*

5.13.2 – DECISÃO / SELEÇÃO COMPOSTA (cont…)

Exemplo:

```
inicio
        ler (idade);
        se idade < 65 então
            dif ⟵ 65 – idade;
            escrever(“Faltam ”, dif, “para atingir a reforma”);
        se-não
            escrever(“Já atingiu a idade da reforma”);
        fim-se;
fim
```

5.13.3 – DECISÃO / SELEÇÃ0 ENCADEADA

Neste tipo de estrutura podem existir várias opções encadeadas para a execução das instruções do programa. A execução de cada opção depende to valor lógico da condição ou expressão.

```
se <condição> então
      <instruções>
se-não se
      <instruções>
se-não
      <instruções>
fim-se
```

5.13.3 – DECISÃO / SELEÇÃ0 ENCADEADA (cont…)

Exemplo:

```
inicio
        var idade, dif: inteiro;
        ler (idade);
        se idade < 65 então
                dif ⟵ 65 – idade;
                escrever (“Faltam”, dif, “para atingir a reforma”);
        se-não se idade > 65
                dif ⟵ idade - 65;
                escrever(“Já ultrapassou a idade da reforma em ”,  dif, “ anos”);
        se-não
                escrever(”Atingiu a idade da reforma”);
        fim-se;
fim
```

5.13.4 – DECISÃO / SELEÇÃO MÚLTIPLA

Em algumas linguagens de programação existe uma quarta estrutura chamada de decisão múltipla. Esta estrutura é também conhecida por “case” ou “switch” e permite a seleção de uma opção de entre um conjunto de opções, mediante a satisfação de uma condição baseada numa variável.

```
inicio
        escolha (var)
        início
            caso (valor1)
            instruções;
            caso (valor2)
            instruções;
            caso (valor3)
            instruções;
            caso (valorN)
            instruções;
        fim
fim
```

5.14 - ESTRUTURAS DE REPETIÇÃO/CICLOS

As estruturas de repetição ou ciclos permitem repetir um conjunto de instruções de um programa, sendo controladas pelo resultado lógico (verdadeiro ou falso) da avaliação de uma condição ou expressão.

Existem três tipos de ciclos:

- Ciclo FOR (para fazer)
- Ciclo WHILE (enquanto fazer)
- Ciclo REPEAT (repetir até)

A diferença entre o bloco enquanto e o bloco repetir está no momento em que o teste lógico é efetuado. No caso do repetir, o código dentro do bloco é pelo menos executado uma vez porque o teste lógico á apenas executado no final.

5.14.1- EXEMPLO PARA FAZER

```
#somar os número de 1 até 100
inicio
soma ← 0;
para i ← 1; i < 100; i ← i + 1 fazer
        soma ← soma + i;
fim-para
escrever("O valor da soma é: ", soma);
fim
```

Outro formato:

```
#somar os número de 1 até 100
inicio
soma ← 0;
para i  de 1 até 100 passo 1 fazer
        soma ← soma + i;
fim-para
escrever("O valor da soma é: ", soma);
fim
```

5.14.2 - EXEMPLO ENQUANTO FAZER

```
#ler o número e apresentar a tabuada
início
escrever("Introduzir o número");
ler(numero);
i ← 1;
enquanto i <= 10 fazer
        resultado ← numero * i;
        escrever(número, " * ", i, " = ", resultado);
        #incrementar a variável i
        i ← i + 1;
fim-enquanto
fim
```

5.14.3- EXEMPLO REPETIR ATÉ

Este exemplo garante que a nota introduzida está compreendida entre 0 e 20.

```
inicio
repetir
    escrever("introduzir a nota entre 0-20:");
    ler(nota);
até nota >=  0  e nota <= 20
fim
```