ESTRUTURAS DE CONTROLO (Continuação)

ESTRUTURAS DE REPETIÇÃO (ou CICLOS)

- Ciclo for (Para):
- Ciclo while (Enquanto):
- Ciclo repeat (Repetir)

Ciclo FOR (para)

- ✓ utilizado quando sabemos o número de vezes que pretendemos repetir uma ação.
- ✓ sintaxe:

```
para (posição_inicial; condição_de_paragem; incremento)
    instruções
fimpara
```

▶ **Exemplo:**

1) Algoritmo que lê 10 números e calcula a sua soma

```
Variáveis:
        inteiro num, i, soma=0
Início
        para(i=0; i<10 ; i++)
                Escrever "Introduza um número"
                ler num
                soma = soma + num (ou soma += num)
        fimpara
        Escrever "A soma dos números é de " + soma
fim
```

NOTA:

O operador += em Python é um operador de atribuição composta que adiciona o valor à direita do operador ao valor da variável à esquerda e atribui o resultado de volta à variável.

```python
num1 = 10
num1 += 3
print("num1 = ",num1)  # num1 agora é 13
```

```python
str1 = "Bom"
str1 += " Dia!"
print("str1 = ",str1)  # str1 agora é "Bom dia!"
```

▶ **Exercício:** Converte o algoritmo em linguagem Python

```python
soma = 0
for x in range(1,11):
    num = int(input(f"Introduza o número {x}: "))
    soma = soma + num   #ou soma += num
print(f"A soma dos 10 números é {soma}")
```

FUNÇÃO RANGE()

Argumentos da função range(start,stop,step):

1º **start (opcional)**:
- ✓ Indica onde a sequência começa.
- ✓ Valor padrão: 0.

2º **stop (obrigatório)**:
- ✓ Indica onde a sequência termina.
- ✓ **Atenção**: O número especificado em stop **não é incluído** na sequência.

3º **step (opcional)**:
- ✓ Define o tamanho do incremento ou decremento entre os números da sequência.
- ✓ Valor padrão: 1.

NOTA: A função range() gera números sequencialmente, mas o objeto retornado por range() não é uma lista.

Exemplos:

1) range(stop) → sequência de números inteiros que começa em **0 (zero)** e vai até ao número anterior a **stop**.

2) range(start, stop) → sequência de números inteiros que começa em **start** e vai até ao número anterior a **stop**.

3) range(start, stop, step) → sequência começa em **start**, vai até ao número anterior a stop, e incrementa (se o step é positivo) ou decrementa (se o step é negativo), de acordo com o valor do **step**.

▶ **Experimenta:**

```
#_____________________________________________________________
for i in range(5):
    print(i)

#_____________________________________________________________
for i in range(2, 6):
    print(i)

#_____________________________________________________________
for i in range(1, 10, 2):
    print(i)

#_____________________________________________________________
for i in range(10, 0, -2):
    print(i)
```

NOTA:

O uso do underscore (_) num ciclo **for** em Python tem um significado específico:

- Variável descartável: O underscore é usado como uma variável que recebe valores, mas que não será utilizada no corpo do ciclo
- Economia de memória: Ao usar _ , evita-se criar uma variável desnecessária recorrendo a menos memória

```
for _ in range(5):
    print("Olá")
```

Ciclo WHILE (enquanto)

- ✓ utilizado para repetir uma ação se verificar uma determinada condição
- ✓ poderá nunca ser executado pois a verificação ocorre no início!
- ✓ sintaxe

```
enquanto (condição)
    instruções
fimenquanto
```

▶ **Exemplo:**

1) Algoritmo que imprime os números de 1 a 4

```
Variáveis:
        inteiro num=1
Início
        enquanto (num<5)
                escrever num
                num = num + 1 (ou num+=1)
        fimenquanto
fim
```

▶ **Exercício**: Converte o algoritmo em linguagem Python

```python
num = 1
while num<5:
    print(f"Num: {num}")
    num += 1
```

Ciclo Repeat (enquanto)

- ✓ utilizado para repetir uma ação se verificar uma determinada condição.
- ✓ é executado pelo menos uma vez, visto que a condição é verificada no fim!
- ✓ em Python não existe o ciclo ***do … while***

NOTA:

A principal diferença entre "do...while" e "while" é a ordem em que a condição é verificada.
A escolha entre while e do-while depende do contexto específico do problema.

- No ciclo "**while**", a condição é verificada **antes** da execução do bloco de código. Se a condição for falsa inicialmente, o bloco pode nunca ser executado.
  - ⇨ O controlo é feito na entrada.
  - ⇨ Previne execuções desnecessárias.

- No ciclo "**do...while**", o bloco de código é executado pelo menos uma vez, e só então a condição é verificada. Isso garante que o bloco seja **executado no mínimo uma vez**, independentemente da condição inicial.
    - ⇨ O controlo é feito na saída.
    - ⇨ É útil para validações de entrada de dados.

▶ Em pseudocódigo, a estrutura do ciclo *repeat* é deste tipo:

```
repeat
   // Bloco de código a ser executado
until (condição)
```

Em python, podemos simular um ciclo repeat (do-while) da seguinte forma:

```python
a=4
while True:
    if a<10:
        a += 1
        print(a)
    else:
        break
print("Terminou")
```

O comando **while True** cria um **ciclo infinito**, ou seja, continuará a executar o código até que seja interrompido manualmente ou por uma instrução **break**.