VARIÁVEIS

Em algoritmos e em programas de computadores, os **dados** podem surgir sob a forma de **variáveis:**

- Uma **variável** é um dado que pode sofrer alterações de valor ao longo do algoritmo;
- Um **identificador** é um nome que é associado a uma variável ou a outro tipo de elemento, em programação.

► As variáveis surgem sempre designadas através de identificadores, ou seja, nomes que as identificam

Exemplo: soma=0

- Chamam-se **instruções de atribuição** às operações internas de um programa ou algoritmo que atribuem valores às variáveis.

Exemplos:
nome = "Joana Silva" -> Indica que a variável **nome** recebe a *string* ou cadeia de caracteres "Joana Silva"
custo = 15 -> Indica que a variável **custo** recebe o valor 15.

➔ O operador de atribuição é o sinal de igual (=).

➔ **Nomes de variáveis:**
- Devem começar com uma letra ou underscore
- Podem conter letras, números e underscores
- São sensíveis a maiúsculas e minúsculas (case-sensitive)

  - Para dar maior legibilidade deve usar-se o _

Por exemplo: Total_Mes;
total = (custo_fixo + custo_variavel) * 1.23

➔ **Reatribuição:** O valor de uma variável pode ser alterado ao longo do programa.

➔ **Tipagem dinâmica:** Python permite que uma variável mude de tipo durante a execução do programa.

EXPRESSÕES ARITMÉTICAS E OPERADORES

As variáveis para além de receberem e guardarem temporariamente valores, também são utilizadas para fornecerem esses valores para a realização de outras operações:

```
Lucro = Custo * 1.25
Escrever (Custo + (1*Taxa))
Escrever ("Valor =", Custo*1.25)
```

✓ As expressões aritméticas também podem ser incluídas diretamente em instruções de escrita.

Operadores Aritméticos:

1. Adição (+): Soma dois valores

   Exemplo: 5 + 3 resulta em 8

2. Subtração (-): Subtrai o segundo valor ao primeiro

   Exemplo: 10 - 4 resulta em 6

3. Multiplicação (*): Multiplica dois valores

   Exemplo: 3 * 4 resulta em 12

4. Divisão (/): Divide o primeiro valor pelo segundo

   Exemplo: 20 / 5 resulta em 4.0 (devolve sempre um float)

5. Divisão inteira (//): Realiza a divisão e arredonda para baixo

   Exemplo: 7 // 2 resulta em 3

6. Módulo (%): Devolve o resto da divisão

   Exemplo: 17 % 5 resulta em 2

7. Exponenciação (** ou pow): Eleva um número a um expoente (potência)

   Exemplo: 2 ** 3 resulta em 8

8. Valor absoluto (abs): Devolve o valor absoluto ou módulo de um número

   Exemplo: 2 ** 3 resulta em 8

Operadores Lógicos:

1. AND (and): Devolve True se ambas as condições forem verdadeiras

   Exemplo: True and False resulta em False

2. OR (or): Devolve True se pelo menos uma condição for verdadeira

   Exemplo: True or False resulta em True

3. NOT (not): Inverte o valor booleano

   Exemplo: not True resulta em False

4. Igual a (==): Verifica se dois valores são iguais

   Exemplo: 5 == 5 resulta em True

5. Diferente de (!=): Verifica se dois valores são diferentes

   Exemplo: 5 != 3 resulta em True

6. Maior que (>), Menor que (<), Maior ou igual a (>=), Menor ou igual a (<=):

   Comparam valores

   Exemplo: 10 > 5 resulta em True

Tipos de Variáveis em Python

1. **Números**
   - **Inteiros (int)**: Representam números inteiros sem parte decimal. Exemplo: x = 5
   - **Ponto Flutuante (float)**: Representam números com parte decimal. Exemplo: y = 3.14
   - **Complexos (complex)**: Representam números complexos. Exemplo: z = 3 + 4j
2. **Sequências**
   - **Strings (str)**: Representam texto. Exemplo: nome = "Maria"
   - **Listas (list)**: Coleções ordenadas e mutáveis de itens. Exemplo: frutas = ["maçã", "banana", "laranja"]
   - **Tuplas (tuple)**: Coleções ordenadas e imutáveis de itens. Exemplo: coordenadas = (10, 20)
3. **Mapeamento**
   - **Dicionários (dict)**: Coleções de pares chave-valor. Exemplo: pessoa = {"nome": "João", "idade": 30}
4. **Conjuntos**
   - **Sets (set)**: Coleções não ordenadas de itens únicos. Exemplo: cores = {"vermelho", "verde", "azul"}
5. **Booleanos**
   - **Bool (bool)**: Representam valores lógicos True ou False. Exemplo: is_ativo = True
6. **Binários**
   - **Bytes (bytes)**: Sequências imutáveis de bytes. Exemplo: dados = b"hello"
   - **Bytearray (bytearray)**: Sequências mutáveis de bytes. Exemplo: arr = bytearray(b"hello")
7. **Nenhum**
   - **NoneType (None)**: Representa a ausência de valor. Exemplo: resultado = None

► f-string em Python

As **f-strings (formatted strings)** são uma maneira fácil e poderosa de incorporar expressões dentro de strings. Introduzidas no Python 3.6, permitem uma formatação de *strings* mais clara e concisa.

✓ Para criar uma f-string:
- **precede-se a string com a letra f** ou F.
- **dentro das chavetas {}, é possível inserir variáveis** que serão avaliadas e formatadas em tempo de execução.

Exemplo:

```
nome = "Maria"
idade = 25
mensagem = f"Meu nome é {nome} e eu tenho {idade} anos."
print(mensagem)
```

OBSERVAÇÕES:

1) **Arredondamentos**

```
troco = 3.14159
print(f"{troco:.2f}")  # Exibe 3.14, com duas casas decimais
```

f"{troco:.2f}": Esta é uma *f-string*. As *f-strings* permitem incluir expressões dentro de *strings* usando chaves {}. Aqui, **troco** é uma variável que tem atribuído um número decimal.

:.2f → Esta parte é responsável pela formatação do número:

: indica o início de uma especificação de formato.
.2 indica que queremos exibir o número com 2 casas decimais.
f significa que estamos a formatar um número decimal (float).

2) **Inserção de espaços**

- **print()**: Qualquer espaço adicional entre as aspas será refletido no output.
- **input()**: Captura exatamente o que foi digitado pelo utilizador, incluindo espaços.

  Pode aplicar-se um método para remover espaços (como strip()).
- A **escrita de espaços entre as vírgulas dentro de uma função em Python** é uma questão de estilo e legibilidade, mas não afeta a execução do código. Python ignora os espaços extras dentro das listas de argumentos de uma função