CONHECENDO O PYTHON

1 – Primeiro vamos falar sobre programação, aqui vamos usar a linguagem de programação Python.

Toda linguagem de programação é dividida em

- ✓ **Comando de Entrada/Saída**
- ✓ **Variáveis**
- ✓ **Operadores**
- ✓ **Estruturas de Condição**
- ✓ **Estruturas de Repetição**

Comandos Entrada/Saída

Comandos de I/O (Input / Output) → Entrada e Saída

entrada_saida.py > ...

```
1   # Quando quisermos fazer um comentario
2   # 1. COMANDO DE SAÍDA: print() --> Serve para o programa "falar"
3   # ou mostrar algo na tela.
4   print("Olá! Eu sou um programa em Python.")
5
6   # 2. COMANDO DE ENTRADA: input() -->#  Serve para o programa "ouvir" o usuário.
7   # Ele pausa e espera você digitar algo.
8
9   nome = input("Qual é o seu nome? ")
10
11  # Mostrando o que foi digitado
12
13  print("Muito prazer em te conhecer,", nome, "!")
```

Resultado do programa

```
PS C:\Aula_Projeto> & "C:\Users\Evandro Vieira\AppData\
Olá! Eu sou um programa em Python.
Qual é o seu nome? Evandro
Muito prazer em te conhecer, Evandro !
PS C:\Aula_Projeto>
```

Variáveis

Variáveis: São espaços reservados na memória que o computador (CPU) usa para guardar dados e depois trabalhar com o seu processamento, pode ou não devolver um resultado. Pense como caixinhas com nomes e nela guardamos os dados que precisamos.

Quais tipo de variáveis temos em Python?

No Python, você não precisa dizer qual é o tipo da variável quando a cria; o próprio Python descobre isso sozinho (chamamos isso de **tipagem dinâmica**).

Os quatros tipos que vamos usar.

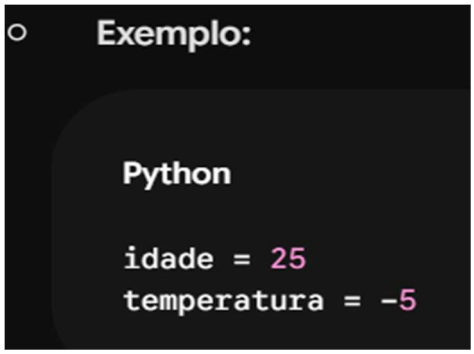
1. str → (String / Texto)

Serve para guardar qualquer tipo de texto, desde uma única letra até parágrafos inteiros. No Python, os textos sempre ficam entre aspas simples (') ou duplas (").

---

2. int (Integer / Inteiro)

Serve para números inteiros, ou seja, números sem casas decimais (sem vírgula/ponto). Podem ser positivos ou negativos.



---

3. float (Floating Point / Ponto Flutuante)

Serve para números com casas decimais (números quebrados). Atenção: o Python usa o padrão americano, então usamos ponto (.) e não vírgula para separar os centavos.

Exemplo:

```python
preco = 49.90
altura = 1.75
```

---

4. bool (Boolean / Booleano)

É o tipo mais simples de todos. Ele só aceita dois valores possíveis: Verdadeiro (True) ou Falso (False). É muito usado para tomadas de decisão no código (sim ou não). Atenção: a primeira letra deve ser sempre maiúscula.

Exemplo:

```python
usuario_logado = True
tem_desconto = False
```

Operadores

Os operadores são os símbolos que usamos para fazer cálculos, comparações e lógica no Python. Pense neles como os verbos do código: eles pegam as suas variáveis e fazem algo acontecer com elas.

Podemos dividi-los em 4 grupos principais que cobrem quase tudo o que vamos fazer:

1. Operadores Aritméticos (Matemática)

São os operadores para fazer contas matemáticas. A maioria você já conhece da escola, mas tem dois que são os "truques" do Python.

Operador	Operação	Exemplo	Resultado
+	Adição	5 + 3	8
-	Subtração	10 - 4	6
*	Multiplicação	4 * 3	12
/	Divisão (Sempre retorna quebrado)	10 / 4	2.5
//	Divisão Inteira (Descarta o quebrado)	10 // 4	2
**	Exponenciação (Potência)	2 ** 3	8 (2^3)
%	Resto da Divisão (Módulo)	10 % 3	1

2. Operadores de Comparação (Relacionais)

Servem para testar condições. Eles olham para dois lados e sempre respondem com um Booleano: True (Verdadeiro) ou False (Falso).

Operador	Significado	Exemplo	Resultado
==	Igual a	5 == 5	True
!=	Diferente de	5 != 3	True
>	Maior que	10 > 20	False
<	Menor que	10 < 20	True
>=	Maior ou igual a	5 >= 5	True
<=	Menor ou igual a	4 <= 3	False

3. Operadores Lógicos

Os operadores lógicos são as ferramentas que usamos para criar regras de negócio e tomar decisões complexas no código. Enquanto os operadores matemáticos lidam com números, os operadores lógicos lidam exclusivamente com valores **booleanos (True ou False).**

No Python, eles são escritos por extenso em inglês: **and, or e not**

1. O Operador `and` (E)

O **and** é extremamente exigente. Ele só vai resultar em **True** se todas as condições ao redor dele forem verdadeiras. Se apenas uma for falsa, tudo azeda e vira **False**.

Regra de ouro: **True and True é True**. Qualquer outra combinação é **False**.

Exemplo prático (Sistema de Caixa Eletrônico): Para sacar dinheiro, você precisa ter saldo suficiente E digitar a senha correta.

```python
saldo_suficiente = True
senha_correta = False  # Errou a senha

# O operador 'and' exige que ambos sejam True
autorizar_saque = saldo_suficiente and senha_correta

print(autorizar_saque)  # Resultado: False
```

2. O Operador `or` (OU)

O **or** é o operador "mãe", super compreensivo. Para ele resultar em **True**, basta que pelo menos uma das condições seja verdadeira. Ele só vai **dar False se absolutamente tudo for falso**.

Regra de ouro: **False or False é False**. Qualquer outra combinação é **True**.

Exemplo prático (Sistema de Desconto de Cinema): Você ganha desconto se for estudante OU se tiver mais de 60 anos.

```python
eh_estudante = False
eh_idoso = True  # Uma das condições é verdadeira

# O operador 'or' aceita se pelo menos um for True
ganha_desconto = eh_estudante or eh_idoso

print(ganha_desconto)  # Resultado: True
```

3. O Operador `not` (NÃO)

O **not** é o operador do contra (inversor). Ele não junta duas condições, ele simplesmente pega uma única condição e inverte o valor dela. **Se for True, vira False. Se for False, vira True.**

Exemplo prático (Bloqueio de Usuário):

```python
usuario_banido = False

# 'not False' vira 'True'
pode_acessar_o_site = not usuario_banido

print(pode_acessar_o_site)  # Resultado: True
```

Estruturas de Condição

As **Estruturas de Condição** (ou estruturas de decisão) são o mecanismo que permite ao seu programa deixar de ser um roteiro fixo e passar a "pensar" e tomar decisões. Elas avaliam se uma situação é **Verdadeira (True)** ou **Falsa (False)** e, com base nisso, mudam o caminho que o código vai seguir.

No Python, o controle de fluxo é feito por três palavras-chave simples: **if, elif** e **else.**

A Regra Mais Importante: Indentação e os Dois Pontos (:)

Diferente de outras linguagens que usam chaves { } para isolar os blocos de código, o Python usa o recuo do texto (chamado de indentação, que são aqueles 4 espaços ou um Tab).

Ao criar uma condição, você obrigatoriamente deve:

Terminar a linha da condição com dois pontos (:).

Deixar as linhas de código que dependem dessa condição com um recuo para a direita.

1. O if (Se...)

É o ponto de partida de qualquer decisão. O Python analisa a condição proposta. Se ela for real/verdadeira, o bloco de código logo abaixo (e recuado) é executado. Se for falsa, o Python ignora esse bloco e segue para a próxima linha alinhada à esquerda.

```python
temperatura = 32

# Se a temperatura for maior que 30, o código de dentro roda
if temperatura > 30:
    print("Está muito quente! Ligue o ar-condicionado.")
```

2. O else (Senão...)

O else é o "plano B". Ele nunca recebe uma condição direta (você não escreve nada na linha dele além de else:). Ele serve para ditar o que o programa deve fazer se a condição do if lá de cima der errada/falsa.

```python
saldo = 50
preco_produto = 80

if saldo >= preco_produto:
    print("Compra realizada com sucesso!")
else:
    print("Saldo insuficiente. Compra negada.")
```

3. O elif (Senão se...)

Quando você precisa avaliar mais do que apenas duas opções (Sim ou Não), entra o elif (uma junção de else com if). Você pode colocar quantos elif quiser entre o if inicial e o else final.

O Python vai testando a lista de cima para baixo. Assim que ele encontra a primeira condição verdadeira, ele executa o bloco dela e ignora todo o resto.

```python
# Verificando a fase da vida pela idade
idade = 22

if idade < 12:
    print("Você é uma criança.")
elif idade < 18:
    print("Você é um adolescente.")
elif idade < 60:
    print("Você é um adulto.")
else:
    print("Você é um idoso.")
```

Estruturas de Repetição

As Estruturas de Repetição (também conhecidas como Loops ou Laços) são os comandos que usamos para fazer o Python executar o mesmo bloco de código várias vezes, de forma automática.

Imagine o trabalho que daria escrever print("Olá") cem vezes seguidas. Com uma estrutura de repetição, você escreve apenas duas linhas de código e o Python faz o trabalho duro.

No Python, temos duas ferramentas principais para isso: o **for** e o **while**. Cada um tem uma "personalidade" diferente.

1. O Loop for (Repetição com limite definido)

O **for** é usado quando você já sabe exatamente quantas vezes quer que o código se repita, ou quando quer percorrer uma lista de coisas do início ao fim.

Para controlar o número de repetições, costumamos usar a **função range().**

Python

```
# Vai repetir o print 5 vezes (de 0 até 4)
for contador in range(5):
    print(f"Essa é a repetição número {contador}")
```

2. O Loop while (Repetição baseada em uma condição)

O **while** (que significa "enquanto") é usado quando você não sabe quantas vezes o código vai rodar. Ele funciona como um if, mas em vez de testar a condição uma vez só, ele continua repetindo o código enquanto a condição for Verdadeira (True).

Muito cuidado: Se a condição nunca virar False, o programa entra em um "loop infinito" e trava.

Python

```
energia = 3

# Enquanto a energia for maior que 0, o boneco continua correndo
while energia > 0:
    print(f"O personagem está correndo! Energia atual: {energia}")
    energia -= 1  # Diminui 1 da energia a cada volta (para o loop não ser infinito)

print("O personagem cansou e parou.")
```