Introdução - Programação com Python

Informações do Autor

- **Nome:** ANIMA Lab
- **Data de atualização**: 11/09/2025

1) O que é programar

Programar é escrever **instruções compreensíveis pelo computador** para realizar uma tarefa. Neste início, vamos trabalhar com pequenos programas que exibem mensagens, recebem dados do usuário e guardam informações em variáveis.

2) “Hello World”: garantindo que o ambiente funciona

Em praticamente toda linguagem, o primeiro passo é imprimir uma mensagem na tela. Isso valida se interpretador/ambiente/editor estão corretos, sem nos distrair com complexidades de linguagem.

Exemplo mínimo:

```
print('Hello World')
```

Esse arquivo pode ser criado em qualquer editor/IDE (Emacs, Vim, Notepad++, Sublime, VS Code, PyCharm, Eclipse). Salvamos com extensão **`.py`**, pois um arquivo `.py` contém declarações (statements) Python que compõem um script/programa.

De onde vem `print` e para onde vai o texto?

`print()` é uma ação pronta do Python para mostrar coisas na tela. O texto aparece no terminal/console. Se esquecer uma das aspas, o interpretador reportará erro porque as aspas delimitam o início/fim do texto.

Como executar:

- Em IDE (ex.: PyCharm): clique em *Run* e depois no ícone verde para repetir.

- No terminal, navegue até a pasta do arquivo e rode:

  ```
  python hello.py
  ```

  (execute no diretório onde salvou o arquivo).

---

3) Tornando o programa interativo: `input()`

Agora vamos perguntar algo ao usuário e responder de forma personalizada.

```
print('Hello, world')
print("Hello! ", input("Digite seu nome "))
```

O que acontece, passo a passo:

1. O programa mostra uma mensagem: `Digite seu nome:`.
2. Ele espera você digitar e apertar Enter.
3. Em seguida, o programa mostra uma saudação usando o que foi guardado nessa caixinha.

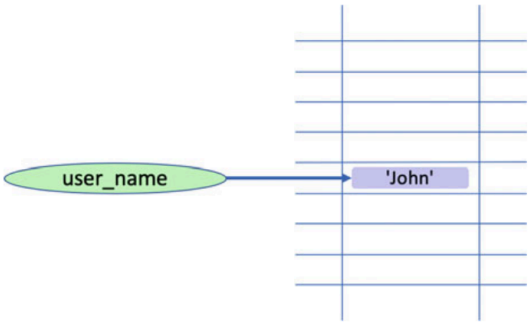
> Pense assim: `input('...')` é “perguntar” e guardar a resposta para usar depois.

---

4) Variáveis: “rótulos” para áreas de memória

Uma variável é um **nome** que referencia uma área de memória onde dados são guardados (texto, números, booleanos (verdadeiro ou falso)). Por exemplo, `nome_usuario` é um rótulo que aponta para onde ficou o texto digitado. A ideia é ilustrada como uma grade de endereços onde cada posição tem um endereço único; o *nome da variável* guarda o endereço do dado.

```
print('Hello, world')
nome_usuario = input('Digite seu nome: ')
print('Hello', nome_usuario)
```



Ao usar `print('Hello', nome_usuario)`, a ação `print` aceita vários itens separados por vírgula (um texto e o valor guardado por `nome_usuario`).

---

5) Reuso de variáveis

Já vimos que uma variável é como uma caixinha com nome, usada para guardar informações (textos, números, valores lógicos etc.).

Contudo, essa caixinha não fica presa a um único valor. Podemos substituir o que está dentro dela sempre que quisermos, ou seja, podemos reutilizar a mesma variável para guardar outro valor. Esse processo se chama reatribuição.

Ex.: primeiro guardamos o nome da pessoa; depois pedimos o melhor amigo e guardamos no mesmo nome de variável:

Exemplo 1 – Reaproveitando a mesma variável

```
print("Hello, world")

nome = input("Digite seu nome: ")
print("Olá,", nome)

# Reatribuindo a variável 'nome'
nome = input("Digite o nome do seu melhor amigo: ")
print("Olá, melhor amigo", nome)
```

O que acontece aqui:

1. O programa mostra `"Hello, world"`.
2. Pede o nome do usuário e guarda em `nome`.
    - Se o usuário digitar Ana, `nome` passa a conter `"Ana"`.
    - O programa imprime: `Olá, Ana`.

3. Depois, o mesmo nome `nome` recebe outro valor.
    - Se o usuário digitar João, `nome` agora passa a conter `"João"`.
    - O programa imprime: `Olá, melhor amigo João`.

👉 **Repare:** a variável não guarda os dois valores ao mesmo tempo. O último valor substitui o anterior.

Exemplo 2 – Mudando o tipo da variável

No Python, uma variável pode mudar de tipo durante a execução.

```
x = 10          # aqui x é um número inteiro
print(x)

x = 3.14        # agora x é um número decimal
print(x)

x = "dez"       # agora x é um texto (string)
print(x)

x = True        # agora x é um valor lógico (booleano)
print(x)
```

Saída:

```
10
3.14
dez
True
```

Observação:

- Em Python isso é permitido porque o tipo da variável é definido automaticamente pelo valor que ela recebe**.
- Em linguagens como **C** ou **Java**, isso não seria possível. Se você declarasse `int x = 10`, não poderia depois transformar `x` em um texto sem erro.

Reflexão

- Essa flexibilidade do Python facilita a vida do iniciante, porque não é preciso se preocupar em declarar tipos de variáveis.

- Por outro lado, essa mesma liberdade pode gerar erros difíceis de encontrar se você não souber exatamente que tipo de dado está manipulando.
- Em Ciência de Dados, por exemplo, muitas vezes trabalhamos com números (para cálculos estatísticos), mas também com textos (nomes de pessoas, categorias, descrições).
    - Se uma variável que deveria guardar um número inteiro passa a guardar uma palavra, o programa pode falhar em uma conta.
    - Por isso, é importante ter atenção ao valor que cada variável está guardando no momento.

---

7) Convenções de nomes em Python

Quando escrevemos programas, precisamos dar nomes a variáveis, funções, classes e outros elementos. Esses nomes são chamados de identificadores. O Python permite quase qualquer nome válido, mas a comunidade segue um conjunto de convenções que tornam o código mais claro, organizado e fácil de entender.

Essas convenções não são regras obrigatórias da linguagem — ou seja, o interpretador não vai impedir que você use um nome diferente. Contudo, se você não seguir essas boas práticas, seu código pode ficar confuso para outras pessoas (e até para você mesmo no futuro!).

🔹 Boas práticas ao nomear variáveis

1. **Usar letras minúsculas**
    - Correto:

```
nome = "Maria"
idade = 20
```

    - Evite:

```
Nome = "Maria"   # Misturar maiúsculas pode confundir
IDADE = 20       # Letras maiúsculas geralmente são usad
```

2. **Escolher nomes descritivos**
    - Correto:

```
total_vendas = 1500
media_notas = 8.5
usuario_email = "teste@gmail.com"
```

- Evite:

```
x = 1500      # o que é x?
a = 8.5       # o que significa "a"?
```

3. **Separar palavras com underscore (_)**
   - Correto:

```
numero_alunos = 30
valor_medio_compra = 75.90
```

   - Errado:

```
numeroAlunos = 30   # estilo de Java, não é comum em Pyt
valorMedioCompra = 75.90
```

4. **Não usar espaços em nomes**
   - Errado:

```
user name = "Ana"  # Python entende como dois comandos a
```

   - Correto:

```
user_name = "Ana"
```

🔹 Convenções para outros elementos

- **Constantes:** usar **maiúsculas com underscores**

```
TAXA_JUROS = 0.15
PI = 3.14159
```

- **Classes:** usar o padrão **CamelCase** (cada palavra começa com letra maiúscula, sem underscore).

```
class ContaBancaria:
    pass
```

- **Funções:** nomes em minúsculas com underscores.

```
def calcular_media(valores):
    return sum(valores) / len(valores)
```

Por que seguir essas convenções?

- **Legibilidade:** facilita a vida de quem lê o código (incluindo você mesmo no futuro).
- **Padronização:** torna o projeto mais organizado e consistente.
- **Colaboração:** quando várias pessoas trabalham no mesmo código, todos entendem facilmente a função de cada variável.
- **Compatibilidade:** muitas bibliotecas e tutoriais seguem essas convenções, então você estará alinhado com a comunidade Python.

---

7) O Operador de Atribuição (=)

Quando programamos, precisamos guardar informações em algum lugar para usá-las depois. No mundo da programação, usamos variáveis como caixinhas com um nome, onde podemos colocar valores. Porém, como colocamos algo dentro dessa caixinha? É aí que entra o **operador de atribuição (=)**.

🔹 O que o = realmente faz?

No dia a dia, estamos acostumados a ver o sinal de igual (=) como uma igualdade matemática. Por exemplo:

```
2 + 2 = 4
```

Aqui significa que os dois lados são iguais.

No Python (e em outras linguagens de programação), o sinal **= NÃO significa "igual a"**, e sim **"atribuir um valor"**. Ou seja:

- **O lado direito** do = é avaliado (calculado).
- O resultado dessa avaliação é armazenado no nome que está à esquerda.

🔹 Exemplo prático

```
nome = "Ana"
idade = 21
print("Nome:", nome)
print("Idade:", idade)
```

Saída:

```
Nome: Ana
Idade: 21
```

- `nome = "Ana"` → cria uma variável chamada `nome` e guarda o texto `"Ana"`.
- `idade = 21` → cria uma variável chamada `idade` e guarda o número `21`.
- O `print()` mostra os valores guardados.

🔹 Atribuindo o resultado de uma expressão

Podemos usar = não apenas para guardar um valor pronto, mas também o resultado de um cálculo:

```
a = 10
b = 5
soma = a + b   # soma recebe o valor calculado de 10 + 5
print("Resultado da soma:", soma)
```

Saída:

```
Resultado da soma: 15
```

Aqui:

1. O Python calcula `10 + 5` (resultado: `15`).
2. Esse valor é colocado dentro da variável `soma`.
3. O `print()` usa o conteúdo da variável para mostrar na tela.

🔹 Reatribuindo valores

O valor dentro da variável pode mudar:

```
x = 100
print(x)   # 100

x = x + 50
print(x)   # 150
```

No segundo passo, o Python pega o valor atual de `x` (100), soma `50` e guarda o novo valor (150) de volta em `x`.

Assim:

- = não é “igualdade”, é atribuição.
- **Esquerda** = nome da variável.
- **Direita** = valor ou expressão que será avaliada.

---

8) Declarações (Statements) x Expressões

Quando escrevemos um programa em Python (ou em qualquer outra linguagem), precisamos dar instruções claras ao computador. Essas instruções podem assumir **duas formas principais**:

🔹 Declaração (Statement)

Uma declaração é uma ordem completa para o Python executar. É como uma frase inteira em português: ela transmite uma ideia completa.

Exemplo simples:

```
print("Olá, mundo")
```

- Aqui, o Python entende que deve executar a ação de mostrar o texto na tela.
- Essa linha já é uma declaração por si só.

Resumo: um *declaração (statement)* é um **comando completo**, algo que o Python pode executar do jeito que está.

🔹 Expressão

Uma expressão é uma combinação de valores, variáveis e operadores que resulta em um valor. É como uma conta de matemática ou uma frase incompleta que ainda precisa ser “resolvida”.

Exemplo:

```
4 + 5
```

- Isso é uma expressão, pois resulta em um valor (9).
- Mas, sozinha, ela não faz nada visível — apenas existe o valor calculado.

🔹 Relação entre os dois

Uma *declaração* pode **conter** uma expressão. Por exemplo:

```
resultado = 4 + 5
print(resultado)
```

Explicando:

1. `resultado = 4 + 5` → a declaração de atribuição pede para calcular a expressão `4 + 5` e guardar o valor (9) dentro da variável `resultado`.
2. `print(resultado)` → outra *declaração* que executa a impressão do valor armazenado.

🔹 Quebrando uma declaração em várias linhas

Em Python, uma **declaração pode ocupar mais de uma linha**. Isso ajuda a organizar o código e deixá-lo mais legível.

Exemplo (uma única declaração de impressão dividida em várias linhas):

```
print(
    "A população da cidade de",
    "Goiânia",
    "foi de",
    1_536_097,
    "habitantes em",
    2022
)
```

Saída:

```
A população da cidade de Goiânia foi de 1536097 habitantes em 2022
```

Mesmo escrito em várias linhas, o Python entende que tudo isso é uma única declaração.

Reflexão

- Pense na **declaração** como uma frase completa que o computador entende e executa.
- Pense na **expressão** como uma parte de frase que precisa ser resolvida para virar um valor.
- Uma declaração pode conter várias expressões.
- Usar múltiplas linhas em uma *declaração* longa ajuda na clareza, algo essencial em Ciência de Dados, onde as expressões matemáticas podem ser complexas.

---

9) Comentários de código (#)

Comentários não são executados — servem para explicar intenção/decisões. Em Python, tudo após # até o fim da linha é ignorado. É válido ter comentário ao fim de uma linha de código.

```
# Comentário de linha inteira
name = input('Enter your name: ')
print(name)  # comentário no final da linha
```

---

10) Formas de Executar Python: Script, Programa e Notebooks

O Python pode ser executado de várias formas, dependendo do contexto e da ferramenta que estamos usando.

---

1. Usando o Interpretador Interativo (REPL)

- REPL = **Read–Eval–Print–Loop** (Ler–Avaliar–Imprimir–Repetir).
- É o modo interativo do Python: abrimos o terminal e digitamos `python` ou `python3`.
- Cada linha digitada é executada imediatamente.
- Ideal para testar pequenos trechos de código rapidamente.

Exemplo:

```
$ python
>>> print("Olá, mundo")
Olá, mundo
>>> 2 + 3
5
```

---

2. Executando um Arquivo `.py`

- O modo mais comum: escrevemos o código em um arquivo com extensão `.py`.
- Esse é o formato usado em projetos maiores e na maioria dos tutoriais.
- Executamos no terminal:

```
python meu_programa.py
```

---

3. Associando `.py` ao Interpretador no Sistema Operacional

- O sistema pode ser configurado para abrir arquivos `.py` diretamente com o Python.
- Exemplo: no Windows, um duplo clique em `meu_programa.py` já executa o programa.
- Útil para usuários finais, mas para quem está aprendendo programação é melhor usar o terminal.

---

4. Usando o *Shebang* (`#!`)

- Em sistemas Linux/macOS, podemos indicar no início do arquivo qual interpretador deve rodar o código.
- Muito usado em scripts de automação no Linux.
- Isso é feito com o **shebang**:

```
#!/usr/bin/env python3
print("Olá, mundo")
```

- Depois damos permissão de execução:

```
chmod +x meu_programa.py
```

- E podemos rodar como se fosse um comando:

```
./meu_programa.py
```

---

5. Usando Notebooks (Google Colab / Jupyter)

Além das formas tradicionais, existe uma maneira muito popular em Ciência de Dados: os **Notebooks**, que permitem misturar código, texto explicativo, gráficos e fórmulas matemáticas no mesmo arquivo.

- **Google Colab:** roda diretamente no navegador, sem precisar instalar nada. Basta ter uma conta Google.
- **Jupyter Notebook:** roda localmente ou em servidores, com interface semelhante ao Colab.

Exemplo no Colab:

```
print("Olá, mundo no Colab!")
```

A saída aparece logo abaixo da célula.

Vantagens do Colab/Jupyter:

- Fácil de usar (não precisa instalar nada no caso do Colab).
- Perfeito para análises interativas, exploração de dados e criação de relatórios.
- Já vem com bibliotecas de Ciência de Dados (NumPy, Pandas, Matplotlib etc.).
- Permite compartilhar o notebook com colegas ou professores, como se fosse um Google Docs de programação.

Script x Programa: Qual a diferença?

- **Script** → arquivo `.py` simples, muitas vezes com *shebang*, feito para rodar diretamente.
- **Programa** → qualquer conjunto de código Python com um propósito maior (pode ser um arquivo único ou um projeto completo).

Na prática:

- Usamos “script” para coisas pequenas (ex.: automatizar uma tarefa simples).
- Usamos “programa” para projetos maiores e organizados.
- E em Ciência de Dados, usamos muito os notebooks, que funcionam como um meio-termo: não são “scripts” nem “programas”, mas um ambiente

interativo de estudo e experimentação.