

# Programação em Python (Back-end)



Instrutor: Pablo Araujo  
(21) 97172-1697

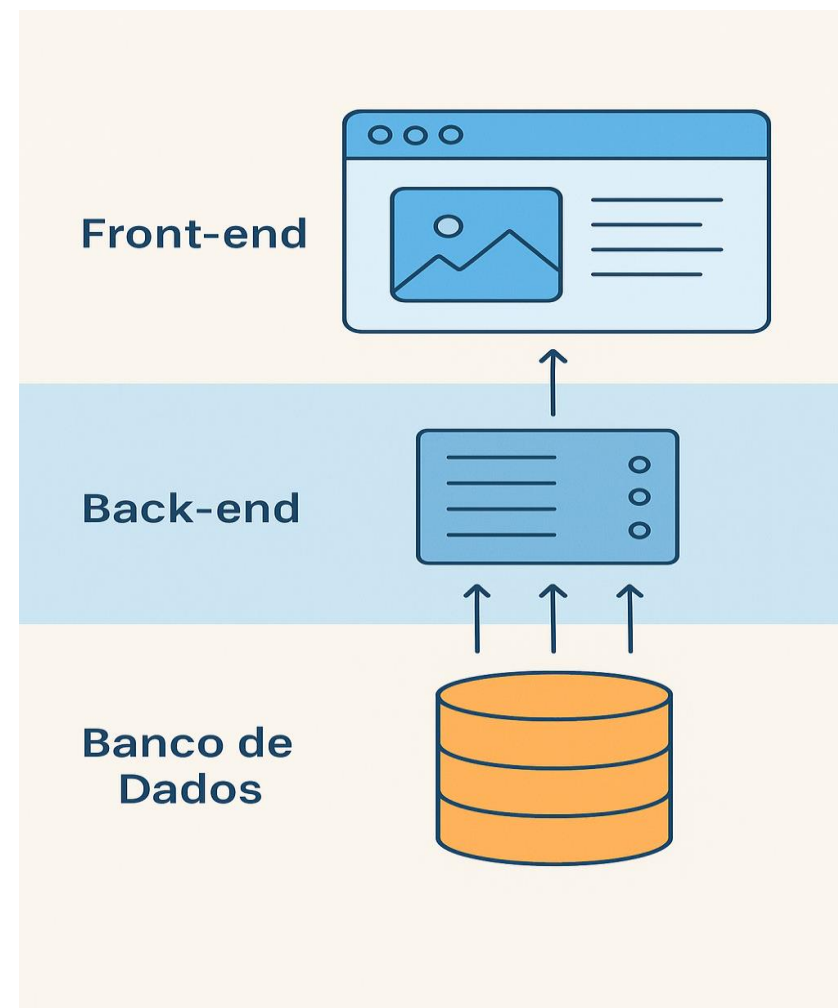


# Aula 1

# Entendendo

## O que é Back-end?

- Camada responsável pelas regras de negócio do sistema
- Faz o desacoplamento entre a interface e a lógica da aplicação
- Controla acesso a dados e garante que operações sigam as regras definidas
- Interage com bancos de dados, APIs e outros serviços
- Exemplo: validar login, verificar saldo, processar uma compra



# Entendendo

## Linguagens do Back-end

- **Python** → simples, moderno e versátil, muito usado em APIs e análise de dados
- **Java** → robusto, consolidado em sistemas corporativos de grande porte
- **C# (.NET)** → forte no ambiente Microsoft e aplicações empresariais
- **Node.js (JavaScript)** → indicado para aplicações em tempo real e alta performance
- **PHP** → ainda bastante utilizado em sites e aplicações web tradicionais



# Aprendendo

## Por que escolher Python no Back-end ?

- Sintaxe simples e curva de aprendizado rápida
- Grande comunidade e ampla documentação
- Diversos frameworks voltados para desenvolvimento web (Django, Flask, FastAPI)
- Integração facilitada com bancos de dados e APIs
- Forte suporte para testes e boas práticas de código



# Aprendendo – Versões

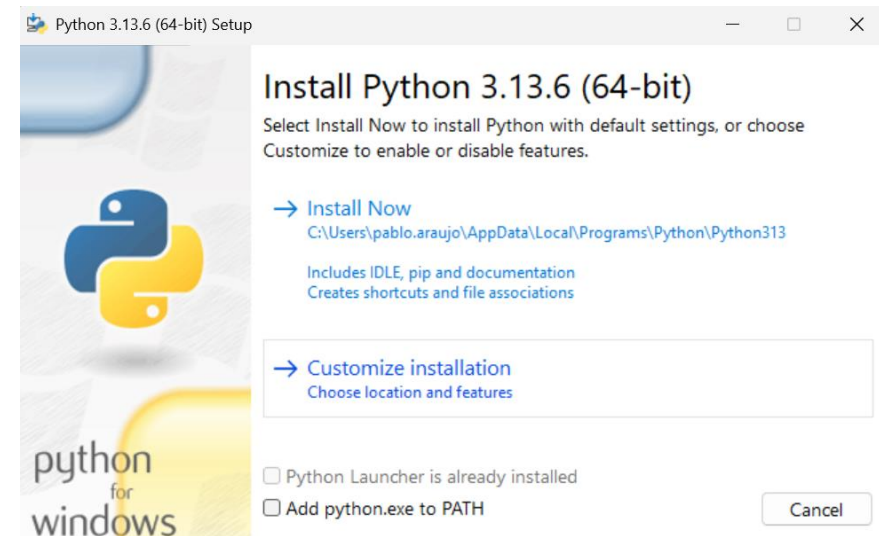
1º Acesse o site para instalação em ambiente Windows:

<https://www.python.org/downloads/windows/>

2º Selecione a versão desejada e baixe o arquivo.

3º Abra o instalador e lembre de selecionar a opção 'Add python.exe to PATH'

4º Após a instalação, abra o terminal e execute `python --version`.



# Aprendendo – Comandos python

## O que é o py?

- É um launcher oficial do Python no Windows.
- Permite escolher qual versão do Python executar, sem precisar mexer no PATH.
- Já vem junto com a instalação oficial do Python (a partir do 3.3).

```
pablo.araujo>py --version  
Python 3.13.7
```

```
pablo.araujo>py --list  
-V:3.13 *      Python 3.13 (64-bit)  
-V:3.12        Python 3.12 (64-bit)  
-V:3.10        Python 3.10 (64-bit)
```

```
pablo.araujo>py -3.12 --version  
Python 3.12.10
```

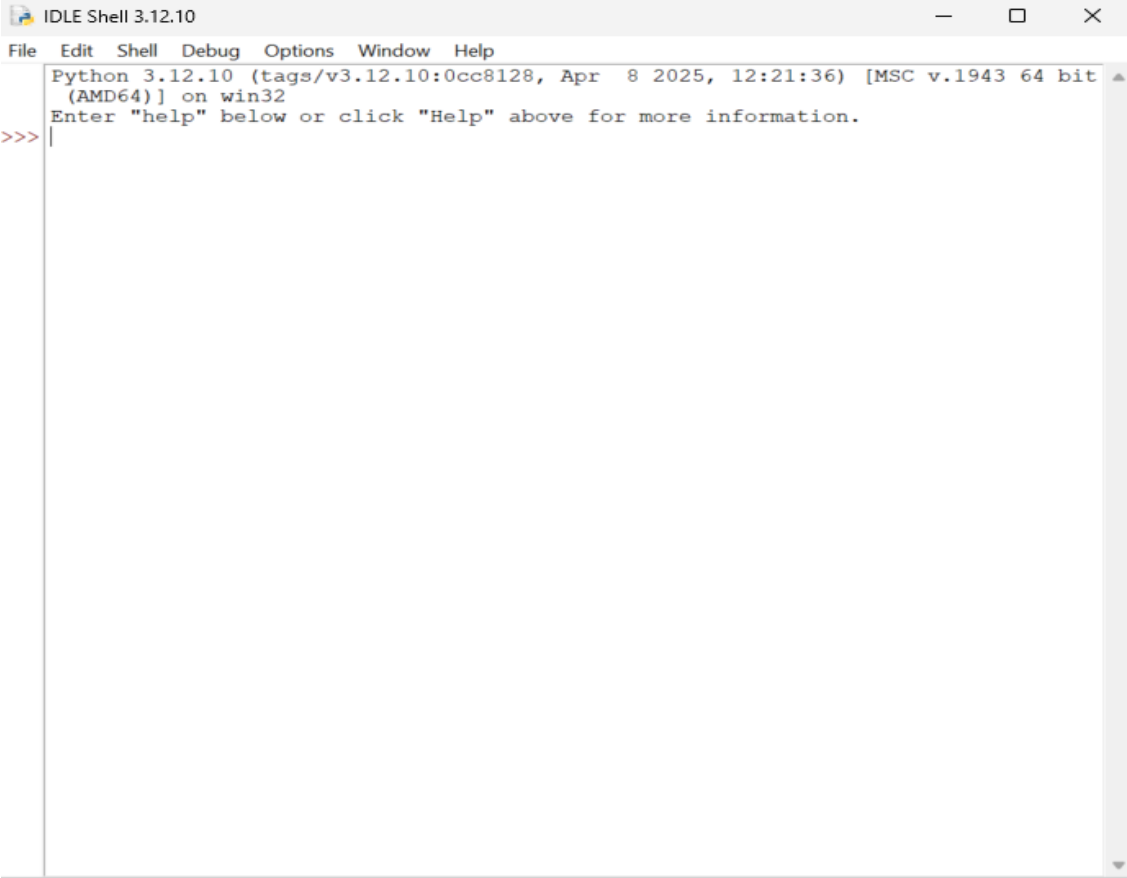
## Vantagem:

Você pode ter várias versões do Python instaladas no mesmo computador e escolher facilmente qual usar.



# Aprendendo – O que é IDLE

- O IDLE é o ambiente padrão que já vem instalado junto com o Python.
- Ele é um editor simples, próprio para testar códigos e aprender a linguagem.
- Funciona como um bloco de notas inteligente, com destaque de sintaxe e execução imediata.
- Bom para quem está começando e quer escrever, salvar e rodar scripts de forma prática.



```
IDLE Shell 3.12.10
File Edit Shell Debug Options Window Help
Python 3.12.10 (tags/v3.12.10:0cc8128, Apr 8 2025, 12:21:36) [MSC v.1943 64 bit
(AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>> |
```

Ln: 3 Col: 0

# Revisando – Criando variáveis e Tipagem

## Objetivos

- Revisar criação de variáveis de vários tipos, porém utilizando o IDLE

## Como executar no IDLE (rápido)

1. Abra IDLE → File → New File
2. Salve o arquivo.(ex.: aula\_revisao.py).

## Crie várias variáveis:

```
char = "a"  
p = "Pablo"  
age = 34  
altura = 1.71  
instrutor = True  
cores = ["vermelho", "azul"]  
config = {"modo": "aula", "nivel": "inicial"}  
par = (1, 2)  
conjunto = {1, 2, 3}  
nada = None  
  
def saudacao():  
    print("oi")
```

## Verificando os tipos das variáveis

```
char <class 'str'>  
p <class 'str'>  
age <class 'int'>  
altura <class 'float'>  
instrutor <class 'bool'>  
cores <class 'list'>  
config <class 'dict'>  
par <class 'tuple'>  
conjunto <class 'set'>  
nada <class 'NoneType'>  
saudacao <class 'function'>
```

# Revisando – Trabalhando com Strings

File Edit Format Run Options Window Help

```
nome = "Pablo"
```

```
sobrenome = "Araujo"
```

```
# Concatenação
```

```
nome_completo = nome + " " + sobrenome
```

```
nome_completo2 = f"{nome} {sobrenome}"
```

```
print("Concatenação:", nome_completo)
```

```
print("Concatenação2:", nome_completo2)
```

```
# Fatiamento
```

```
print("Primeira letra:", nome[0])
```

```
print("Últimas 3 letras:", nome[1:])
```

```
print("Últimas 3 letras:", nome[-3:])
```

```
# Métodos principais
```

```
print("Maiúsculas:", nome.upper())
```

```
print("Minúsculas:", nome.lower())
```

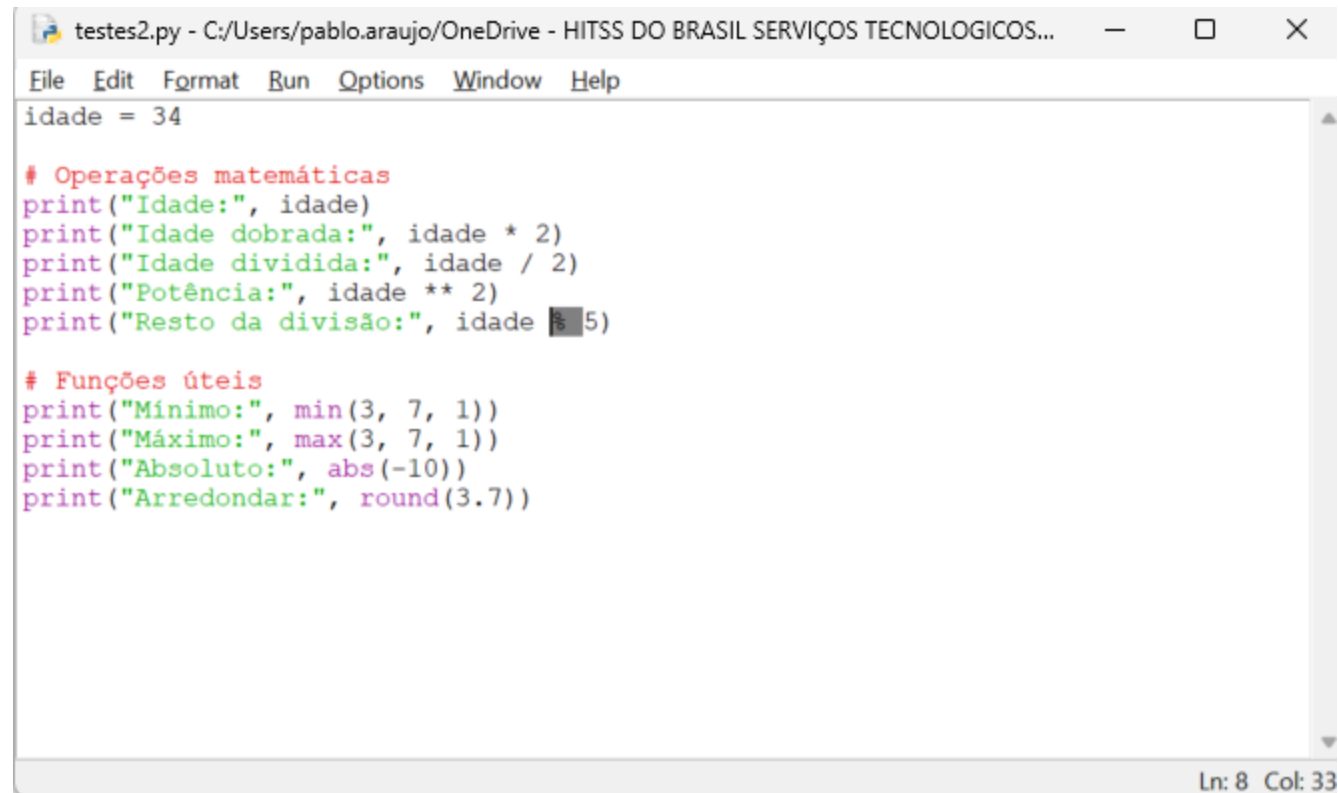
```
print("Título:", nome_completo.title())
```

```
print("Substituir:", nome_completo.replace("Araujo", "Oliveira"))
```

```
print("Está em maiúsculo?", nome.isupper())
```

Ln: 14 Col: 0

# Revisando – Trabalhando com Int



```
testes2.py - C:/Users/pablo.araujo/OneDrive - HITSS DO BRASIL SERVIÇOS TECNOLOGICOS...  
File Edit Format Run Options Window Help  
idade = 34  
  
# Operações matemáticas  
print("Idade:", idade)  
print("Idade dobrada:", idade * 2)  
print("Idade dividida:", idade / 2)  
print("Potência:", idade ** 2)  
print("Resto da divisão:", idade % 5)  
  
# Funções úteis  
print("Mínimo:", min(3, 7, 1))  
print("Máximo:", max(3, 7, 1))  
print("Absoluto:", abs(-10))  
print("Arredondar:", round(3.7))  
  
Ln: 8 Col: 33
```

# Revisando – Trabalhando com float

```
File Edit Format Run Options Window Help
altura = 1.71

# Operações com float
print("Altura:", altura)
print("Altura x 2:", altura * 2)
print("Divisão:", altura / 3)

# Arredondamento
print("Arredondado:", round(altura, 1))

# Biblioteca math
import math
print("Raiz quadrada:", math.sqrt(16))
print("Pi:", math.pi)
```

# Revisando – Trabalhando com bool

File Edit Format Run Options Window Help

```
instrutor = True
```

```
ativo = False
```

```
# Comparações retornam bool
```

```
print("10 > 5?", 10 > 5)
```

```
print("10 == 5?", 10 == 5)
```

```
print("10 != 5?", 10 != 5)
```

```
# Uso em if
```

```
if instrutor:
```

```
    print("É instrutor")
```

```
else:
```

```
    print("Não é instrutor")
```

# Revisando – Trabalhando com lista

```
File Edit Format Run Options Window Help
cores = ["vermelho", "azul", "verde"]

# Acesso e fatiamento
print("Primeira cor:", cores[0])
print("Últimas duas:", cores[-2:])

# Métodos principais
# adiciona
cores.append("amarelo")
print("Após append:", cores)
# remove
cores.remove("azul")
print("Após remove:", cores)

print("Quantidade de itens:", len(cores))
print("Existe 'verde'?", "verde" in cores)

# Iteração
for cor in cores:
    print("Cor:", cor)
```

- ◆ Listas são mutáveis: podemos alterar, adicionar e remover elementos.
- ◆ Muito utilizadas para coleções de dados dinâmicas.

# Revisando – Trabalhando com tupla

```
File Edit Format Run Options Window Help
par = (1, 2, 3)

print("Primeiro:", par[0])
print("Último:", par[-1])

# Tupla é imutável
# par[0] = 99 # ERRO

print("Comprimento:", len(par))
print("Existe 2?", 2 in par)
```

- ◆ Tuplas são imutáveis: depois de criadas, não podem ser alteradas.
- ◆ Usadas quando os dados não devem mudar.



# Revisando – Trabalhando com dicionário

```
File Edit Format Run Options Window Help
config = {"modo": "aula", "nivel": "inicial"}

# Acesso
print("Modo:", config["modo"])
print("Nivel:", config.get("nivel"))

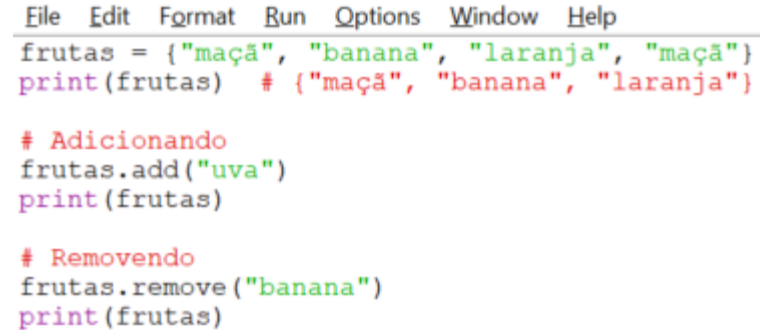
# Adição/alteração
config["turma"] = "Python 1"
print("Novo dicionário:", config)

# Métodos principais
print("Chaves:", config.keys())
print("Valores:", config.values())
print("Itens:", config.items())

# Iteração
for chave, valor in config.items():
    print(chave, ":", valor)
```

- ◆ Dicionários são mutáveis: armazenam pares chave:valor.
- ◆ Muito usados para representar objetos ou registros.

# Revisando – Trabalhando com set



```
File Edit Format Run Options Window Help
frutas = {"maçã", "banana", "laranja", "maçã"}
print(frutas)  # {"maçã", "banana", "laranja"}

# Adicionando
frutas.add("uva")
print(frutas)

# Removendo
frutas.remove("banana")
print(frutas)
```

- ◆ Sets são mutáveis, mas não permitem elementos duplicados.
- ◆ Muito úteis para remover duplicatas.

# Revisando – Atividade

Utilizando o IDLE crie funções que:

- Primeira função, receberá um texto como parâmetro e colocará todo em maiúscula.
- Segunda função receberá dois parâmetros o primeiro será a lista e o segunda será algum valor, ele deverá adicionar o valor na lista.
- Terceira função receberá dois parâmetros o primeiro será um set e o segunda será algum valor, ele deverá adicionar o valor no set.
- Quarta função deverá receber um dicionário, você deverá procurar a chave idade e somar +1. (Desafio)

Obs.: Crie variáveis que possam ser passadas para as funções.

Nas funções print o valor final e também exiba o tipo de cada parâmetro recebido.

# Revisando – Classes

- ◆ Uma classe é como um molde para criar objetos.
- ◆ Um objeto é uma instância dessa classe.
- ◆ Dentro da classe, usamos métodos (funções internas).
- ◆ O método `__init__` representar o inicializador da classe.
- ◆ Para representar a própria instancia utilizamos o `self`

```
File Edit Format Run Options Window Help
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def apresentar(self):
        return f"Olá, meu nome é {self.nome} e tenho {self.idade} anos."
```

# Revisando – Atividade Classes

- Crie uma classe chamada Carro.
- A classe deve ter:
  - Um construtor (`__init__`) com os atributos `marca` e `ano`.
  - Um método `ligar()` que imprime "O carro {marca} está ligado."
  - Um método `acelerar()` que imprime "O carro {marca} está acelerando."
  - Crie pelo menos dois objetos da classe Carro e teste os métodos.

## Desafio:

- Modifique a classe Carro para ter um atributo extra chamado `ligado` (valor inicial `False`).
- O método `ligar()` deve:
  - Alterar `ligado` para `True`.
  - Exibir "O carro {marca} está ligado."
- Crie o método `desligar()`
  - Alterar `ligado` para `False`.
  - Exibir "O carro {marca} está desligado."
- O método `acelerar()` só deve funcionar se `ligado == True`.
- Caso contrário, exiba "O carro {marca} precisa estar ligado para acelerar."
- Crie 2 objetos e teste o comportamento.

Dúvidas ?

Obrigado!

Instrutor: Pablo Araujo

