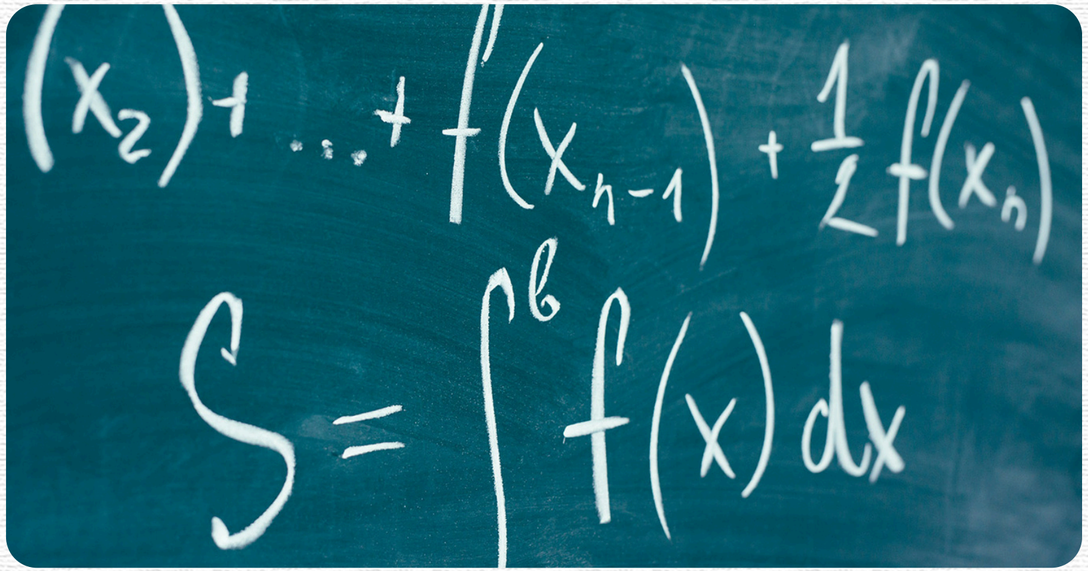
FÁBRICA DE SOFTWARE

Workshop Back-End

Quarta-feira

FUNÇÕES

O QUE SÃO FUNÇÕES?



Funções são blocos de código que executam tarefas específicas. Elas são como pequenos assistentes que você pode chamar sempre que precisar realizar uma ação particular.

DEFININDO FUNÇÕES

Para criar uma função, usamos a palavra reservada ***def***, seguida pelo nome da função e parênteses.

```
def saudacao(nome):
    print(f"Olá, {nome}!")
```

PARÂMETROS E ARGUMENTOS

Os parâmetros são os nomes que você coloca entre os parênteses na definição da função (como nome no exemplo anterior). Quando você chama a função, fornece argumentos (valores) para esses parâmetros.

```
saudacao("Alice")
# Saída: Olá, Alice!
```

RETORNO DE VALORES

Algumas funções retornam valores. Você pode usar a palavra-chave ***return*** para especificar o valor que a função deve devolver.

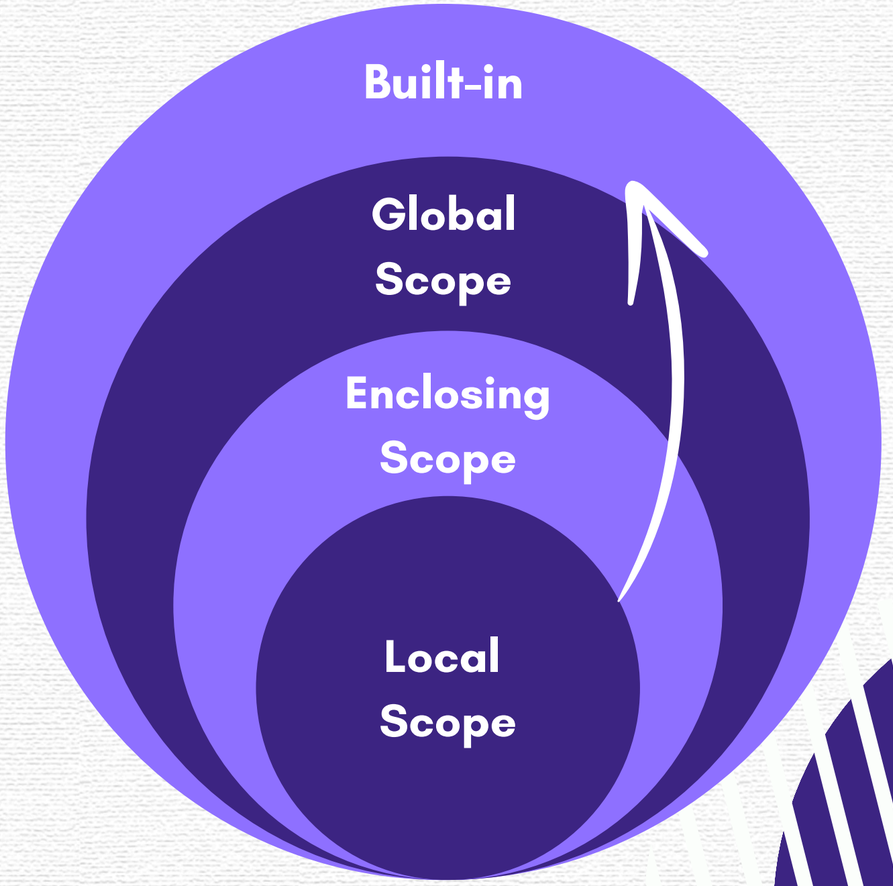
```
def soma(a, b):
    return a + b
resultado = soma(3, 5)
# resultado agora é 8
```

BUILT-IN

- Python já vem com várias funções embutidas, como ***print(), len(), max(), min()***, etc.
- Você pode usá-las diretamente sem precisar defini-las.

ESCOPO DE VARIÁVEIS

- Variáveis definidas dentro de uma função têm escopo local. Isso significa que elas só existem dentro da função.
- Variáveis definidas fora da função têm escopo global e podem ser acessadas em qualquer lugar do código.



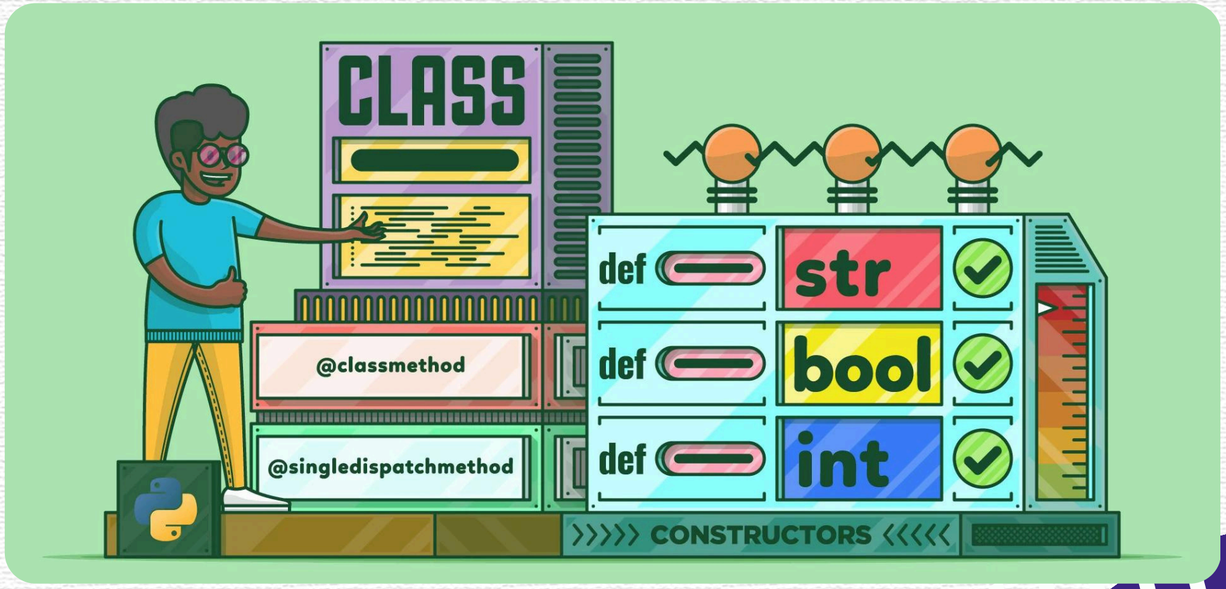
FUNÇÃO COMPLETA

Lembre-se de que as funções tornam seu código mais organizado, reutilizável e fácil de entender.

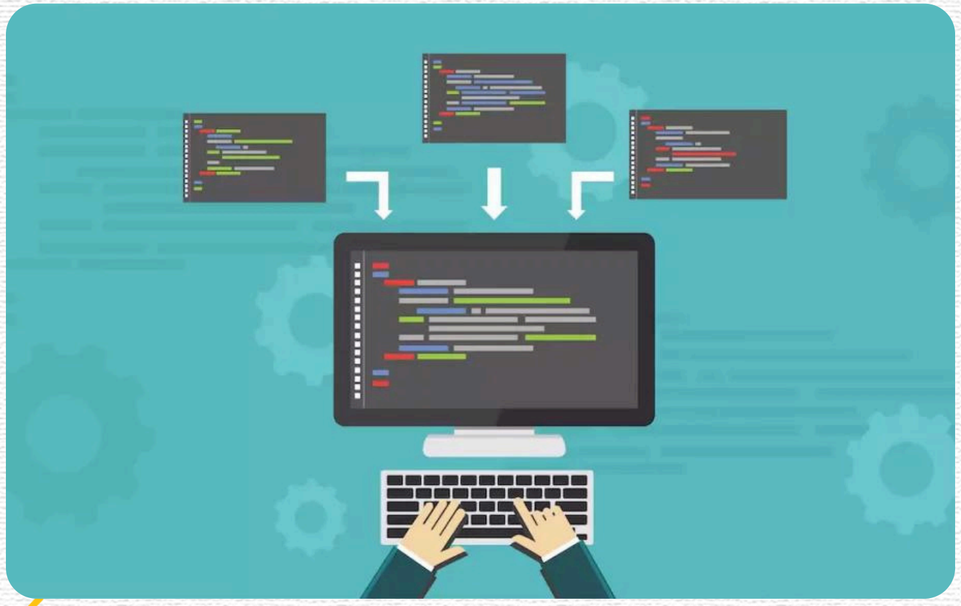
```python
def calcular_pagamento(qtd_horas, valor_hora):
    horas = float(qtd_horas)
    taxa = float(valor_hora)
    if horas <= 40:
        salario = horas * taxa
    else:
        h_excedentes = horas - 40
        salario = 40 * taxa + (h_excedentes * 1.5 * taxa)
    return salario

salario_final = calcular_pagamento(45, 20)
print(f"Salário final: R${salario_final:.2f}")
```

CLASSES


CLASS
@classmethod
@singledispatchmethod
def
str
def
bool
def
int
CONSTRUCTORS

O QUE SÃO CLASSES?



As classes são uma parte fundamental da programação orientada a objetos (POO) em Python. Elas nos permitem criar objetos complexos com atributos e métodos, tornando nosso código mais organizado e reutilizável.

O QUE SÃO CLASSES?

As classes são como moldes ou plantas para criar objetos. Cada classe define um novo “tipo” de objeto.

Imagine que queremos representar carros em nosso programa. Criaríamos uma classe chamada “Carro” com atributos como cor, marca e modelo, e métodos como “acelerar” e “frear”

DEFININDO CLASSE

Para criar uma classe, usamos a palavra-chave class, seguida pelo nome da classe.

```python
class Carro:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def acelerar(self):
        print(f"{self.marca} {self.modelo} acelerando!")

    def frear(self):
        print(f"{self.marca} {self.modelo} freando!")
```

ATRIBUTOS E MÉTODOS

Preste atenção no código abaixo.

```python
class Carro:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def acelerar(self):
        print(f"{self.marca} {self.modelo} acelerando!")

    def frear(self):
        print(f"{self.marca} {self.modelo} freando!")
```

ATRIBUTOS E MÉTODOS

A classe “Carro” tem dois atributos (marca e modelo)
e dois métodos (acelerar e frear).
Os métodos são funções definidas dentro da classe e
operam nos atributos do objeto.

CRIANDO OBJETOS

Agora podemos criar objetos (instâncias)
dessa classe

```
meu_carro = Carro(marca="Ford", modelo="Mustang")
```

ACESSANDO ATRIBUTOS

Podemos acessar os atributos do objeto

```
print(meu_carro.marca)  # Saída: Ford
```

CHAMANDO MÉTODOS

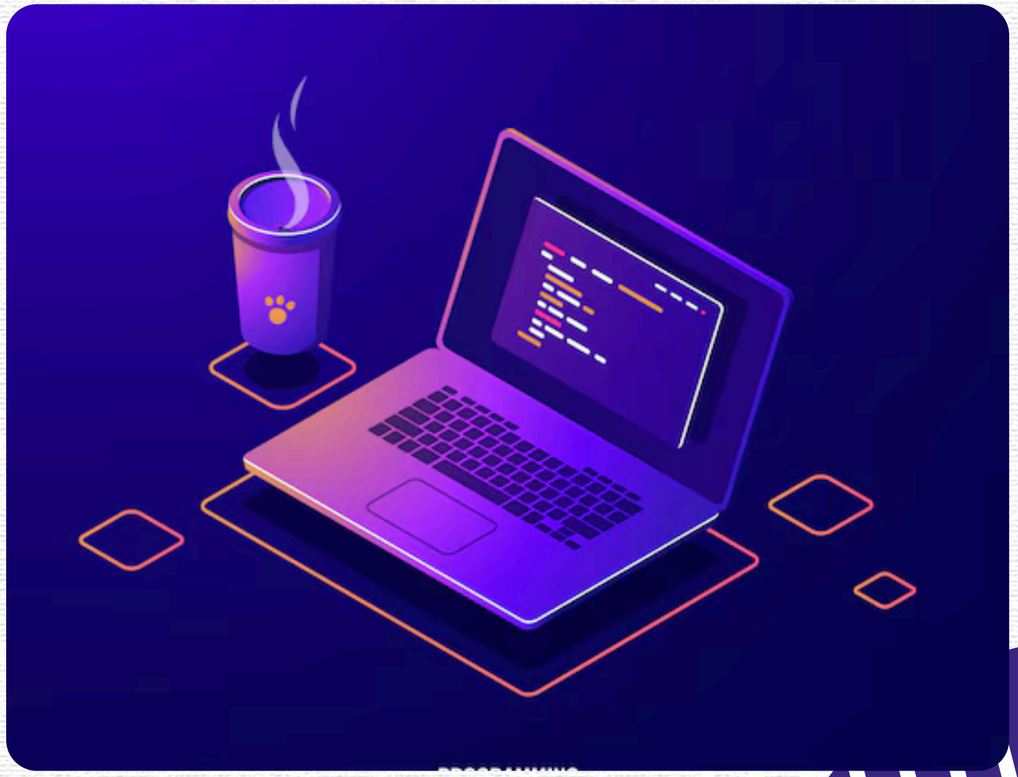
E podemos chamar os métodos

```
meu_carro.acelerar()  # Saída: Ford Mustang acelerando!
```

CONCLUSÃO

Classes são blocos de construção essenciais em Python. Elas nos ajudam a modelar o mundo real e a criar código mais estruturado.

HERANÇA
POLIMORFISMO
HEREDITARIEDADE



O QUE É HERANÇA?

A herança é um princípio fundamental na POO que permite que uma classe (chamada subclasse) herde propriedades e métodos de outra classe (chamada superclasse).

HERANÇA

Com a herança, podemos criar uma hierarquia lógica entre objetos. Por exemplo, imagine uma classe “Veiculo” com atributos como “marca” e “modelo”

```python
class Veiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo
```

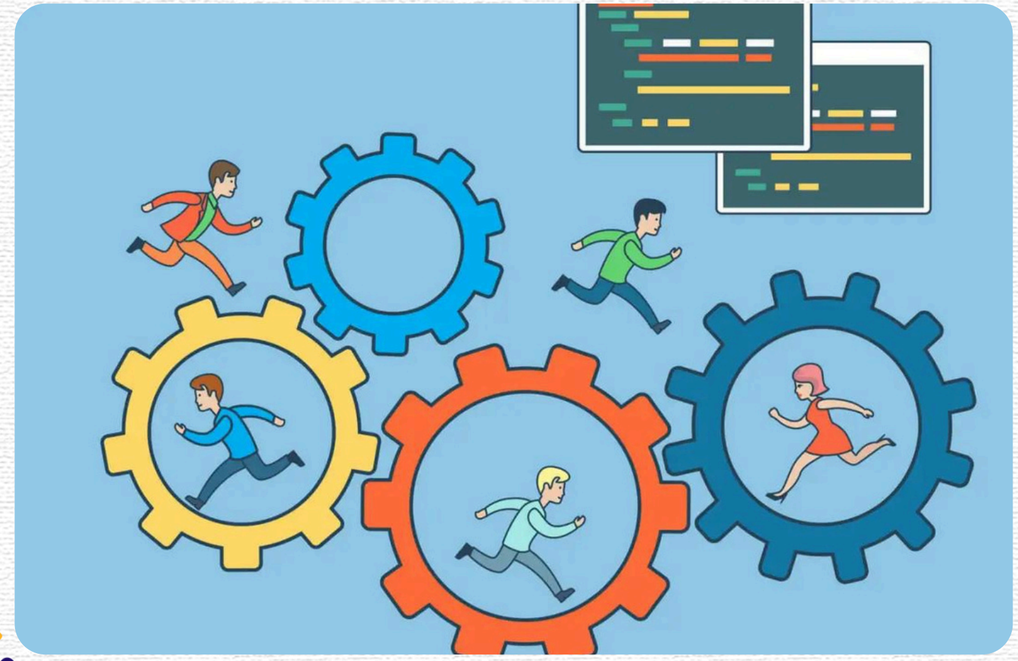
HERANÇA

Agora, podemos criar uma subclasse chamada “Carro” que herda da classe “Veiculo”

```python
class Carro(Veiculo):
    def __init__(self, marca, modelo, portas):
        super().__init__(marca, modelo)
        self.portas = portas
```

A classe “Carro” tem acesso aos atributos “marca” e “modelo” da superclasse “Veiculo”. Isso promove reutilização de código e facilita a manutenção.

O QUE É POLIMORFISMO?



O polimorfismo permite que objetos de diferentes classes sejam tratados como se fossem da mesma classe. Permite tratar objetos de diferentes classes de maneira unificada

POLIMORFISMO

Podemos criar outra subclasse chamada “Moto”

```python
class Moto(Veiculo):
    def __init__(self, marca, modelo, cilindradas):
        super().__init__(marca, modelo)
        self.cilindradas = cilindradas
```

POLIMORFISMO

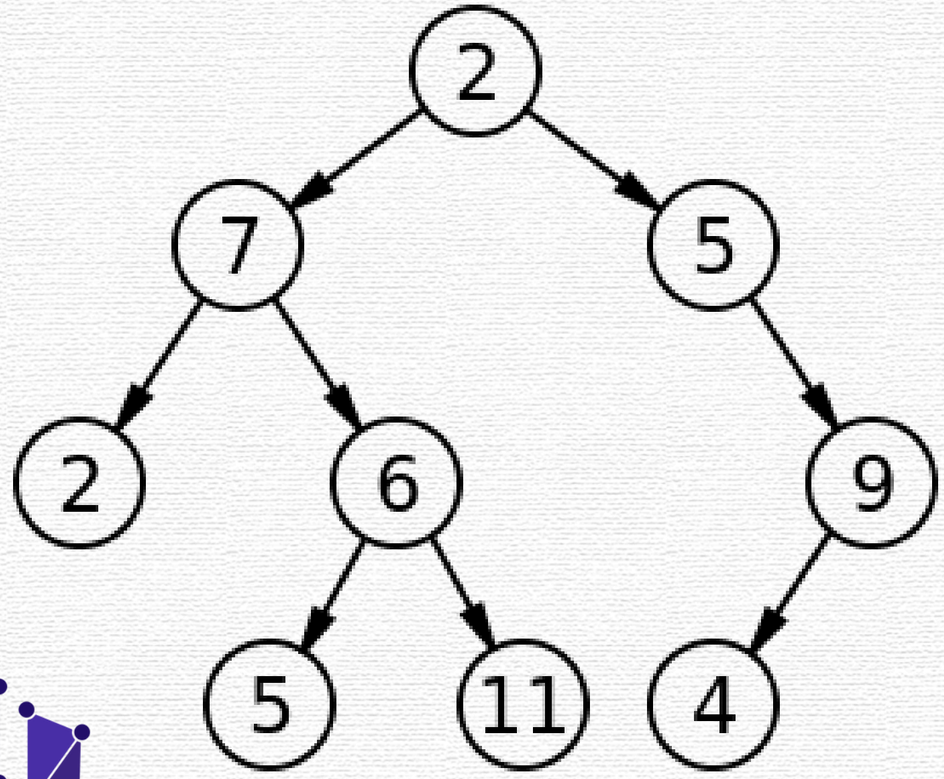
Agora, podemos criar uma função que interage
com objetos de diferentes classes

```
def mostrar_informacoes_veiculo(veiculo):
    veiculo.mostrar_detalhes()  # Método comum a todas as subclasses

carro = Carro("Ford", "Fusion", 4)
moto = Moto("Honda", "CBR", 650)

mostrar_informacoes_veiculo(carro)  # Exibe detalhes do carro
mostrar_informacoes_veiculo(moto)   # Exibe detalhes da moto
```

O QUE É HEREDITARIEDADE?



A hereditariedade é o processo pelo qual as características de uma classe (superclasse) são transmitidas às suas subclasses.

HEREDITARIEDADE

No exemplo acima, a classe “Carro” herda características da classe “Veiculo”.

```python
class Carro(Veiculo):
    def __init__(self, marca, modelo, portas):
        super().__init__(marca, modelo)
        self.portas = portas
```

CONCLUSÃO

Herança e Polimorfismo são fundamentais na POO em Python (e em qualquer linguagem que use esse paradigma). A herança permite a extensão e a especialização de classes, enquanto o polimorfismo facilita a interação com objetos de diferentes classes de maneira unificada.

DÚVIDAS??


não entendi
ahh entendi
mentira, não entendi
eu definitivamente entendi nada
PáginaChloeBr
PáginaChloe

AGORA VAMOS PARA PRÁTICA