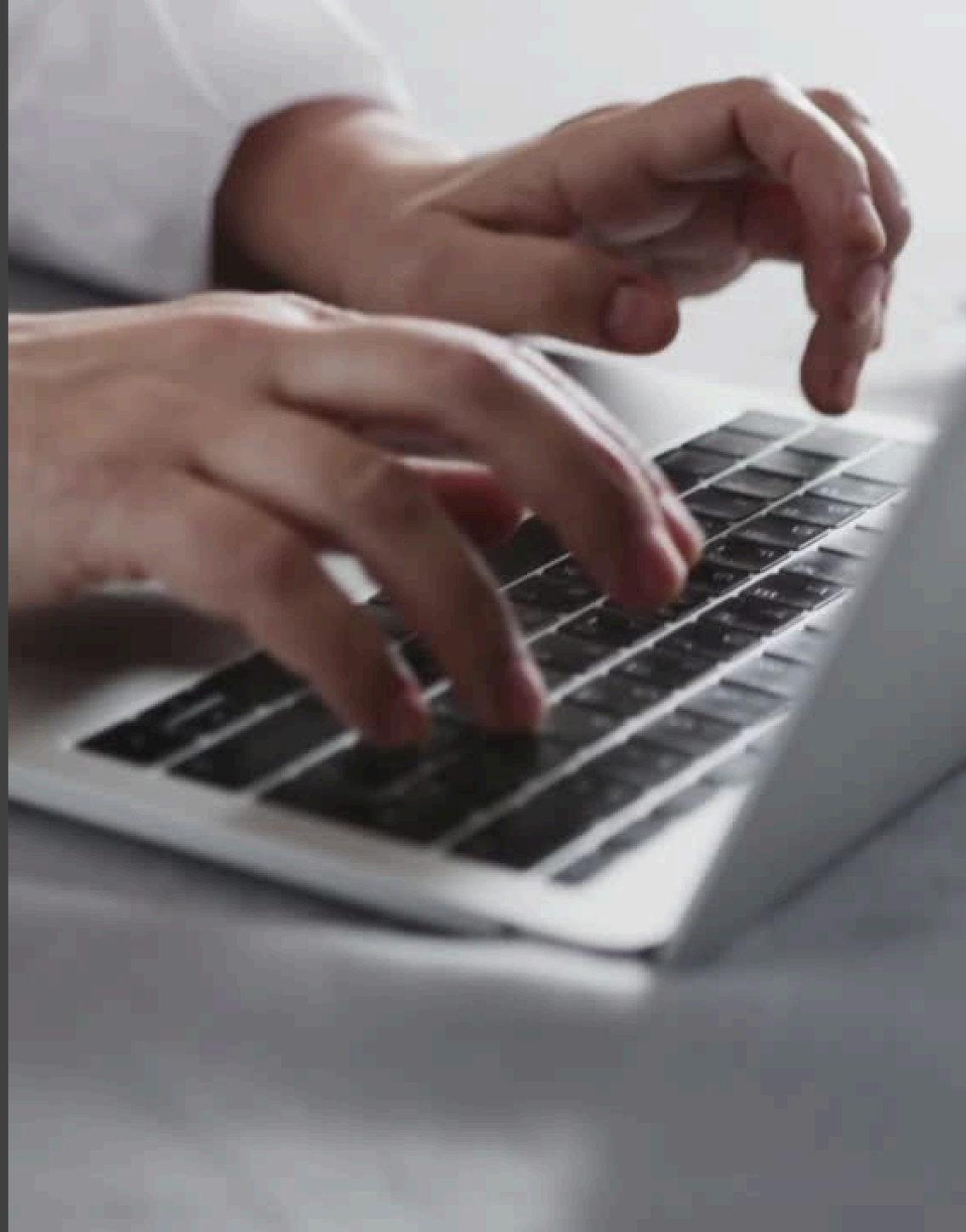
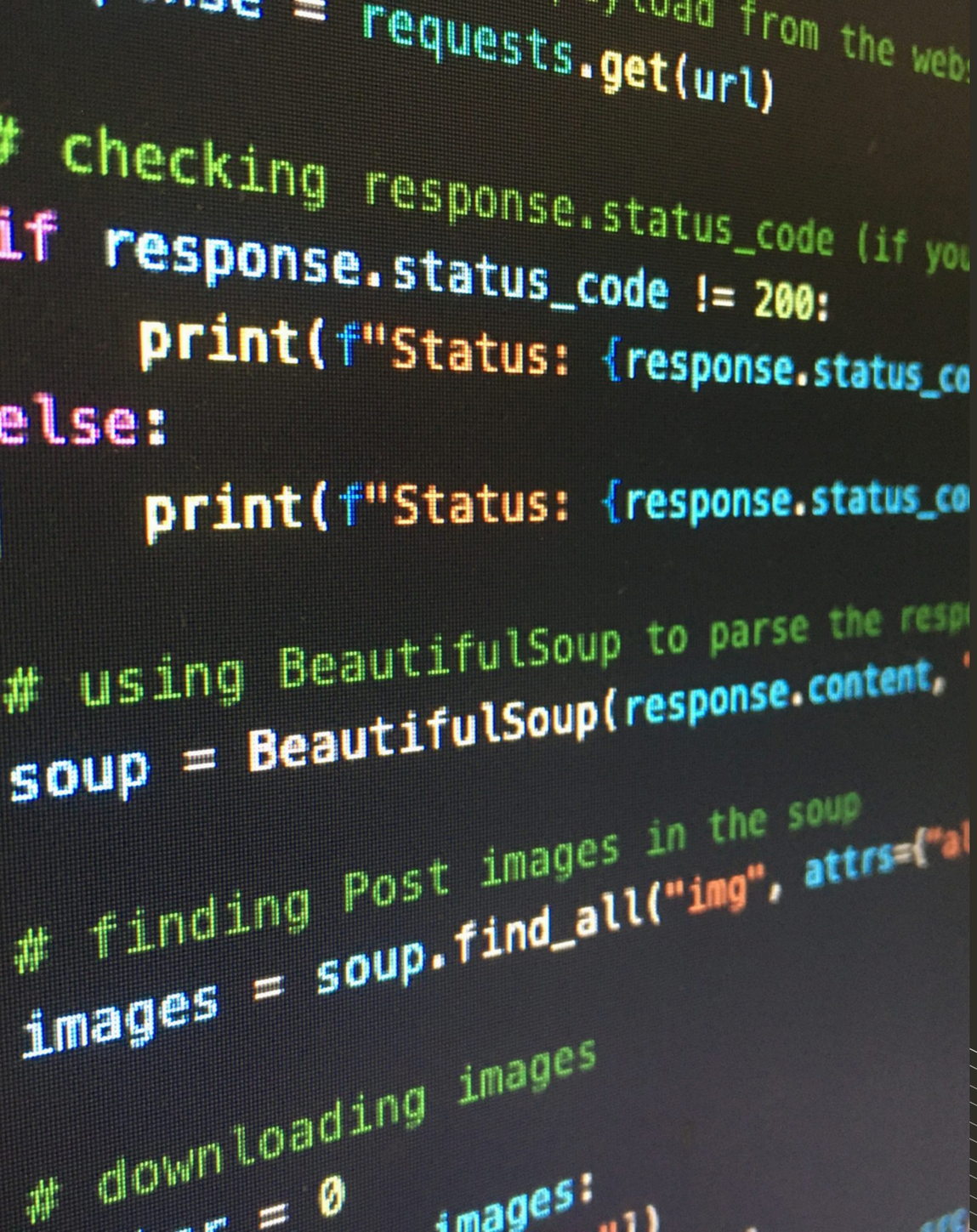
Python: Visão Geral e Orientação a Objetos





Python é uma linguagem de programação de alto nível, interpretada e de propósito geral, que se destaca por sua **simplicidade**. Ela suporta múltiplos paradigmas de programação, incluindo programação orientada a objetos (POO). Neste 15 minutos vamos explorar os principais conceitos da POO em Python, incluindo classe, construtor, objeto, métodos especiais e herança.

Programação Orientada a Objetos

A POO é um paradigma que modela um sistema como uma coleção de objetos que interagem entre si. Vamos explorar seus principais componentes.

Imperativo

Declarativo

Procedural

Orientado Objeto

Lógico

Funcional

```python
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

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

# Criando uma instância da classe Pessoa
p1 = Pessoa("Marcus", 45)

print(p1.saudacao())  # Saída: Olá, meu
nome é Marcus e tenho 45 anos.
```

Classe

Uma classe é um modelo ou blueprint a partir do qual objetos são criados. Ela define as **propriedades** e comportamentos que os **objetos** dessa terão. Em Python, as classes são definidas usando a palavra-chave "class". Ela é formada por:

- Inicializador ou construtor (__init__)
- Atributos ou propriedades (dados)
- Métodos (funções).

Classe (exemplo para cópiar)

```
class Pessoa:
 def __init__(self, nome, idade):
 self.nome = nome
 self.idade = idade

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

# Criando uma instância da classe Pessoa
p1 = Pessoa("Marcus", 45)

print(p1.saudacao()) # Saída: Olá, meu nome é Marcus e tenho 45 anos.
```

Herança e polimorfismo

```python
class Funcionario(Pessoa):
    def __init__(self, nome, idade, cargo):
        super().__init__(nome, idade)
        self.cargo = cargo

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

f1 = Funcionario("Marcus", 45, "Professor")
print(f1.saudacao())  # Saída: Olá, meu
nome é Marcus, tenho 45 anos e sou
Professor.
```

Polimorfismo é uma característica de paradigma de programação onde um objeto pode se transformar em outro objeto.
Dentro deste paradigma podemos encontrar a herança.
Herança é o mecanismo pelo qual uma classe pode herdar atributos e métodos de outra classe, promovendo reutilização de código.

```
class Funcionario(Pessoa):
 def __init__(self, nome, idade, cargo):
 super().__init__(nome, idade)
 self.cargo = cargo

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

f1 = Funcionario("Marcus", 45, "Professor")
print(f1.saudacao()) # Saída: Olá, meu nome
é Marcus, tenho 45 anos e sou Professor.
```

```python
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def __str__(self):
        return f"Pessoa(nome={self.nome}, idade=
        {self.idade})"

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

p1 = Pessoa("Marcus", 45)
print(p1)  # Saída: Pessoa(nome=Marcus, idade=45)
```

Métodos Especiais

Métodos especiais são funções com nomes começando e terminando com __, como __init__, que possuem significados especiais.

Exemplo:

```
class Pessoa:
 def __init__(self, nome, idade):
 self.nome = nome
 self.idade = idade

 def __str__(self):
 return f"Pessoa(nome={self.nome}, idade={self.idade})"

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

p1 = Pessoa("Marcus", 45)
print(p1) # Saída: Pessoa(nome=Marcus, idade=45)
```

Meus contatos

- mvdiogoce@gmail.com
- mvdiogo.sociedadecosmica.com.br