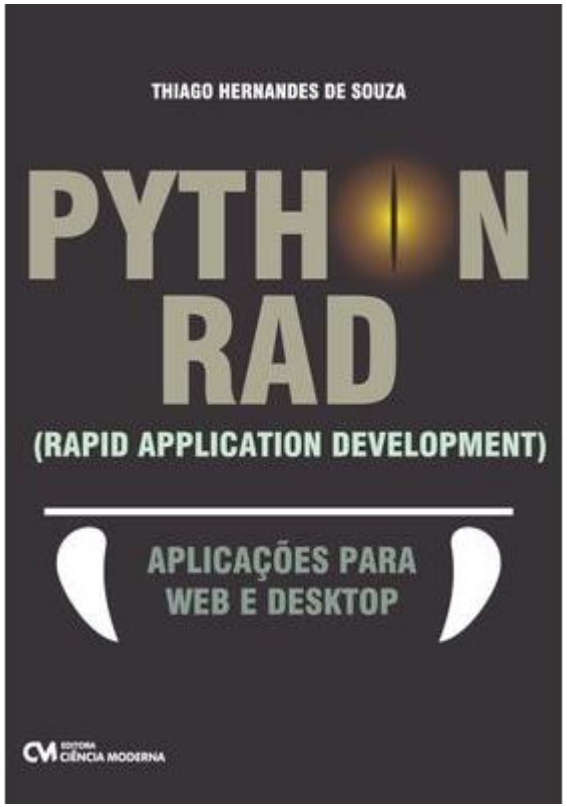
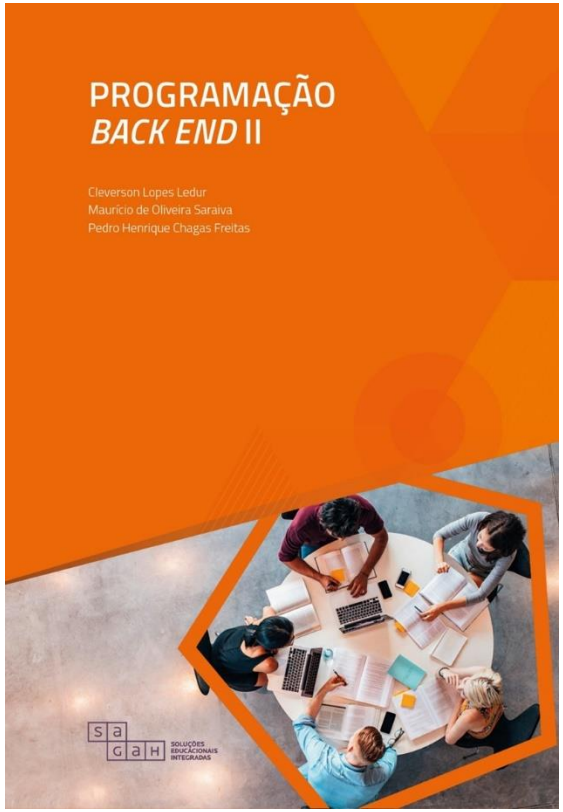
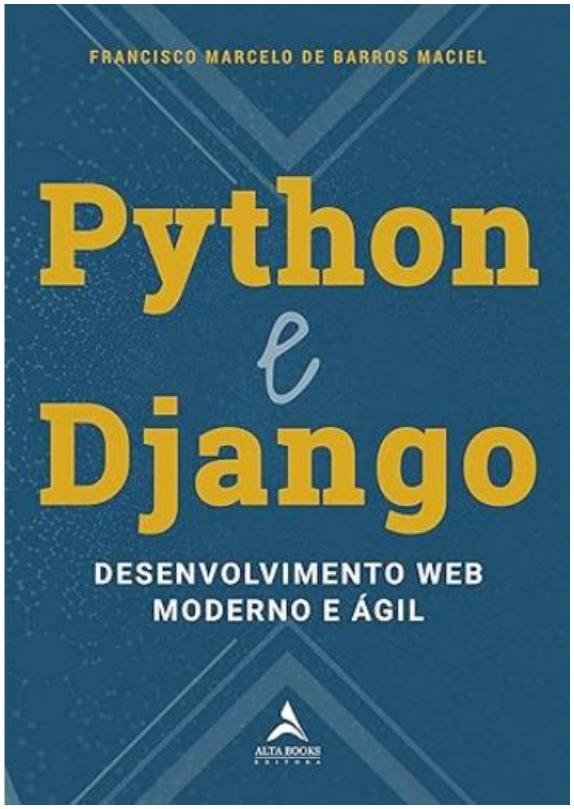
PROGRAMAÇÃO BACK-END







Programação Back-End

Python

Classes e Objetos

Classes/objetos Python

- Python é uma linguagem de programação orientada a objetos.
- Quase tudo em Python é um objeto, com suas propriedades e métodos.
- Uma classe é como um construtor de objetos ou um "modelo" para criar objetos.

PROGRAMAÇÃO BACK-END

Para criar uma classe, use a palavra-chave class.

A classe myClasse possui duas propriedades.

teste01.py > ...

```
1   class myClasse:
2       Nome = "Programação"
3       valor = 45
```

Criar objeto

Pode-se usar a classe chamada MyClasse para criar objetos

teste02.py > ...

```
1   class myClasse:
2       Nome = "Programação"
3       valor = 45
4
5   p1 = myClasse
6   print(p1.Nome)
7   print(p1.valor)
```

A função __init__()

Os exemplos anteriores são classes e objetos em sua forma mais simples e não são realmente úteis em aplicações da vida real.

Para entender o significado das classes, temos que entender a função integrada __init__().

Todas as classes possuem uma função chamada __init__(), que é sempre executada quando a classe está sendo iniciada.

A função __init__()

Use a função __init__() para atribuir valores às propriedades do objeto ou outras operações que sejam necessárias quando o objeto estiver sendo criado.

teste03.py > ...

```
1   class Pessoa:
2       def __init__(self, nome, idade):
3           self.nome = nome
4           self.idade = idade
5
6   p1 = Pessoa("João", 27)
7
8   print(p1.nome)
9   print(p1.idade)
```

Nota: A função __init__() é chamada automaticamente sempre que a classe está sendo usada para criar um novo objeto.

A função __str__()

A função __str__() controla o que deve ser retornado quando o objeto da classe é representado como uma string.

teste04.py > ...

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

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

p1 = Pessoa("João", 27)
print(p1)
```

Observação:

- No Python, o termo self é usado em métodos de classe para referenciar o objeto atual.
- Ele é semelhante ao this em outras linguagens de programação orientadas a objetos, como Java ou C++.
- O self ajuda a diferenciar entre atributos de instância e variáveis locais, e é utilizado para acessar variáveis e métodos associados ao objeto.

Exemplo:

classe.py

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

    def exibir_detalhes(self):
        print(f'Marca do carro: {self.marca}, Modelo: {self.modelo}')

meu_carro = Carro('Toyota', 'Corolla')
meu_carro.exibir_detalhes()
```

No exemplo:

- self.marca e self.modelo são atributos do objeto meu_carro.
- self é usado dentro do método exibir_detalhes para acessar esses atributos do objeto específico meu_carro.

Basicamente, self é uma referência ao próprio objeto e é usado para acessar outros atributos e métodos do objeto dentro da classe.

Métodos de objeto

Objetos também podem conter métodos. Métodos em objetos são funções que pertencem ao objeto.

teste05.py > ...

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

  def funcao(self):
    print("O nome é " + self.nome + " com idade " + str(self.idade))

p1 = Pessoa("João", 27)
p1.funcao()
```

Modificar propriedades do objeto

Pode-se modificar propriedades em objetos.

p1.idade = 37

teste06.py > ...

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

  def funcao(self):
    print("O nome é " + self.nome + " com idade " + str(self.idade))

p1 = Pessoa("João", 27)
p1.idade = 37
p1.funcao()
```

PROGRAMAÇÃO BACK-END

O código a seguir cria uma classe **Pessoa** com o método **alteranome** de atribuição de valor a propriedade **nome** e o método **mostranome** que apresenta o atributo **nome**.

classe.py

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

    def alteranome(self, nome):
        self.nome = nome

    def mostranome(self):
        print("O nome é "+self.nome)

p1 = Pessoa("João", 27)
p1.alteranome("Carlos")
p1.mostranome()
```

Excluir propriedades do objeto

Pode-se excluir propriedades de objetos usando a palavra-chave del:

```
del p1.idade
```

Excluir objetos

Pode-se excluir objetos usando a palavra-chave del:

```
del p1
```