PYTHON ACADEMY

CLASSES E OBJETOS NO PYTHON

Nesse ebook, você vai dominar a Programação Orientada a Objetos no Python. Aprenda a declarar Classes e instanciar Objetos no Python.

PYTHONACADEMY.COM.BR

Salve salve Pythonista!

Neste artigo, vamos explorar o conceito de **Classes** e **Objetos** da Programação Orientada a Objetos na linguagem de programação Python.

As classes são uma forma poderosa de organizar e estruturar nosso código, permitindo a criação de objetos personalizados com propriedades e comportamentos próprios.

É essencial entender o uso de Classes para aproveitar ao máximo o potencial da linguagem Python.

Então se prepare e vamos nessa!

```
<div class="suggestions-header">
    <h2>Leia também</h2>
</div>
<div class="suggestions-body">
    <ul>
        <li>
            <a href="https://pythonacademy.com.br/blog/introducao-a-
programacao-orientada-a-objetos-no-python">
                Introdução à Programação Orientada a Objetos no Python
            </a>
        </li>
    </ul>
</div>
```

Introdução

As classes podem ser definidas como **modelos** para a criação de objetos.

Elas contêm **atributos** (variáveis) e **métodos** (funções) que definem o comportamento do objeto criado a partir da classe.

Ao criar uma classe, estamos criando um tipo de dado **personalizado e reutilizável** em nosso código.

A utilização de classes é extremamente importante na programação orientada a objetos, pois permite a abstração de problemas complexos em entidades menores, além de facilitar a manutenção e reutilização do código.

Definindo uma Classe

No Python, podemos definir uma classe utilizando a palavra-chave `class`, seguida pelo nome da classe.

O nome da classe deve seguir algumas convenções, como começar com uma letra maiúscula e utilizar a notação CamelCase (iniciais das palavras compostas são maiúsculas e não há espaços ou underscores).

Aqui está um exemplo simples de uma classe chamada `Pessoa`:

```
class Pessoa:
    pass
```

Neste exemplo, utilizamos a palavra-chave `pass` para indicar que a definição da classe está vazia.

Isso permite que possamos criar a classe sem implementar nenhum atributo ou método por enquanto.

Criando um Objeto

Uma vez definida a classe, podemos criar objetos a partir dela.

Esses objetos são chamados de **instâncias da classe**.

Para criar uma instância, utilizamos o nome da classe seguido de parênteses:

```
pessoa1 = Pessoa()
```

No exemplo acima, criamos uma instância da classe `Pessoa` e a atribuímos à variável `pessoa1`.

Agora, podemos acessar essa instância e manipulá-la.

Atributos da Classe

A classes podem ter atributos, que são variáveis que pertencem à classe e são compartilhadas por todas as instâncias criadas a partir dessa classe.

Os atributos representam as características que o objeto da classe possui.

Podemos adicionar um atributo à classe `Pessoa` da seguinte forma:

```
class Pessoa:
    nome = "João"
    idade = 30
```

No exemplo acima, adicionamos os atributos `nome` e `idade` à classe `Pessoa`.

Agora, todas as instâncias dessa classe terão esses atributos definidos.

Podemos acessar e modificar os atributos de uma instância da seguinte forma:

```
pessoa1 = Pessoa()
print(pessoa1.nome)  # Saída: João

pessoa1.nome = "Maria"
print(pessoa1.nome)  # Saída: Maria
```

No exemplo acima, acessamos o atributo `nome` da instância `pessoa1` e o modificamos para “Maria”.

Podemos fazer o mesmo com o atributo `idade`.

Métodos da Classe

Além de atributos, as classes também podem ter métodos, que são funções que pertencem à classe e podem ser chamadas pelas instâncias dessa classe.

Os métodos representam os comportamentos que o objeto da classe pode realizar.

Podemos adicionar um método à classe `Pessoa` da seguinte forma:

```
class Pessoa:
    def falar(self, mensagem):
        print(f"{self.nome} diz: {mensagem}")
```

No exemplo acima, adicionamos o método `falar` à classe `Pessoa`.

O método recebe dois parâmetros: `self` e `mensagem`.

O parâmetro `self` é uma referência à própria instância da classe, permitindo o acesso aos atributos e métodos dessa instância.

Podemos chamar o método `falar` da seguinte forma:

```
pessoa1 = Pessoa()
pessoa1.nome = "João"
pessoa1.falar("Olá!")  # Saída: João diz: Olá!
```

E a saída será:

```
João diz: Olá!
```

No exemplo acima, chamamos o método `falar` da instância `pessoa1`, passando a mensagem “Olá!” como parâmetro.

> 💡 *Estou desenvolvendo o **DevBook**, uma plataforma que usa IA para gerar ebooks técnicos profissionais. Não deixe de conferir!*

Construtor da Classe

O construtor é um método especial que é executado automaticamente quando uma instância da classe é criada.

No Python, o construtor é chamado de `__init__`.

Podemos utilizar o construtor para definir valores iniciais para os atributos da classe.

Aqui está um exemplo de como definir um construtor na classe `Pessoa`:

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

No exemplo acima, definimos o construtor `__init__` com os parâmetros `nome` e `idade`.

Dentro do construtor, atribuímos os valores desses parâmetros aos atributos da classe `Pessoa`.

Podemos criar uma instância da classe `Pessoa` e passar os valores para o construtor da seguinte forma:

```
pessoa1 = Pessoa("João", 30)
print(pessoa1.nome)
print(pessoa1.idade)
```

Resultando em:

```
João
30
```

No exemplo acima, criamos uma instância da classe `Pessoa` com o nome "João" e idade 30.

Ao imprimir os valores dos atributos `nome` e `idade`, obtemos os valores passados no construtor.

Herança de Classes

Uma vantagem das classes é a possibilidade de criar novas classes a partir de classes existentes. Isso é chamado de **Herança**.

A classe derivada herda todos os atributos e métodos da classe base.

Aqui está um exemplo de como criar uma classe derivada (também chamada de subclasse) a partir da classe `Pessoa`:

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

No exemplo acima, criamos a classe `Estudante` que herda da classe `Pessoa`.

Utilizamos o método `super().__init__()` para chamar o construtor da classe `Pessoa` e passar os valores para os atributos `nome` e `idade`.

Podemos criar uma instância da classe `Estudante` e acessar os atributos e métodos da classe base da seguinte forma:

```
estudante1 = Estudante("Maria", 18, "12345")
print(estudante1.nome)
print(estudante1.idade)
print(estudante1.matricula)
estudante1.falar("Olá!")
```

O que resultará em:

```
Maria
18
12345
Maria diz: Olá!
```

No exemplo acima, criamos uma instância da classe `Estudante` com o nome “Maria”, idade 18 e matrícula “12345”.

Ao imprimir os valores dos atributos e ao chamar o método `falar`, temos acesso aos atributos e métodos da classe base `Pessoa`.

Conclusão

As classes são fundamentais na linguagem de programação Python e permitem a criação de código modular, organizado e reutilizável.

Elas fornecem uma forma concisa e eficiente de representar e manipular objetos em nossos programas.

Neste artigo, vimos como definir uma classe, criar objetos, adicionar atributos e métodos, utilizar um construtor e explorar a herança de classes.

Agora, você tem o conhecimento necessário para começar a utilizar as classes em seus projetos Python.

Espero que este artigo tenha sido útil e que você possa aproveitar ao máximo as classes na sua jornada de programação Python!

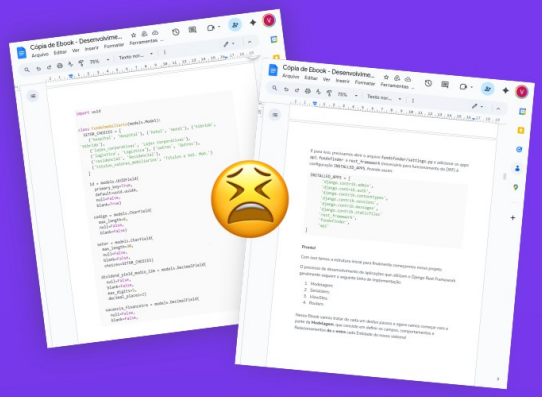
Nos vemos no próximo Post! 👋

Não se esqueça de conferir!

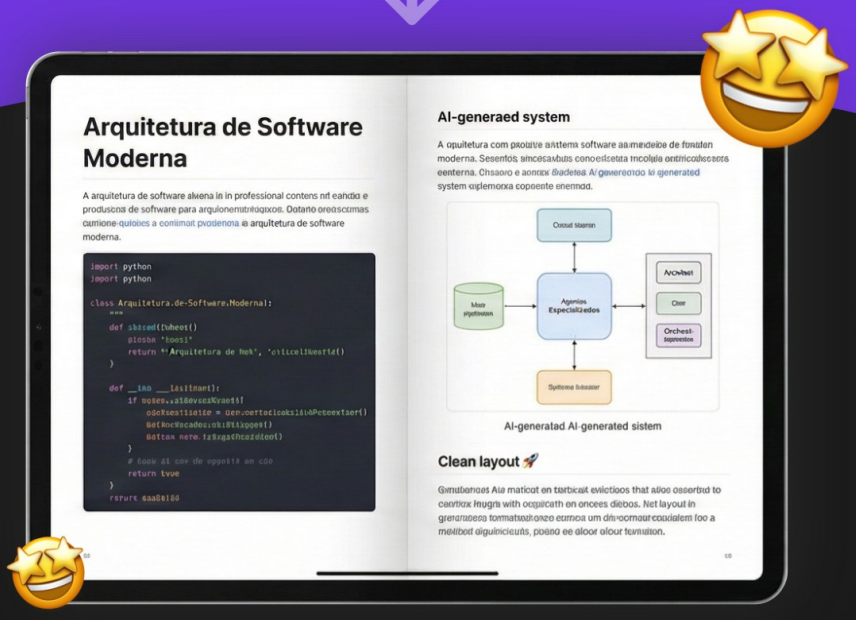
DevBook

Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código**!



Chega de formatar código no Google Docs



Syntax Highlight

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS