PYTHON ACADEMY

CLASSES E OBJETOS NO PYTHON

Guia completo de Classes e Objetos: declaração, __init__, métodos de instância/classe/estáticos, __str__/__repr__, casos reais (produto, usuário), atributos classe vs instância.

PYTHONACADEMY.COM.BR

> ✅ ***Atualizado para Python 3.13*** *(Dezembro 2025)*
> *Conteúdo enriquecido com métodos especiais (**str**, **repr**), tipos de métodos e casos práticos.*

Salve salve Pythonista!

Classes são moldes para criar **objetos** com atributos (dados) e métodos (comportamentos). São a base da Programação Orientada a Objetos!

Neste guia, você vai aprender: - ✅ **Declarar classes** e instanciar objetos - ✅ **Métodos especiais** - **init**, **str**, **repr** - ✅ **Tipos de métodos** - instância, classe, estáticos - ✅ **Casos práticos** - Produto, Usuário, Contador

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.

Métodos Especiais (Magic Methods)

str vs repr

```python
class Produto:
    def __init__(self, nome, preco):
        self.nome = nome
        self.preco = preco

    def __str__(self):
        # Para usuários finais (legível)
        return f"{self.nome} - R$ {self.preco:.2f}"

    def __repr__(self):
        # Para desenvolvedores (debug)
        return f"Produto(nome='{self.nome}', preco={self.preco})"

p = Produto("Mouse", 45.90)
print(str(p))   # Mouse - R$ 45.90
print(repr(p))  # Produto(nome='Mouse', preco=45.9)
```

len e getitem

```python
class Playlist:
    def __init__(self, nome):
        self.nome = nome
        self.musicas = []

    def adicionar(self, musica):
        self.musicas.append(musica)

    def __len__(self):
        return len(self.musicas)

    def __getitem__(self, index):
        return self.musicas[index]

playlist = Playlist("Favoritas")
playlist.adicionar("Música 1")
playlist.adicionar("Música 2")

print(len(playlist))  # 2
print(playlist[0])    # Música 1
```

Métodos de Instância vs Classe vs Estáticos

```python
class Contador:
    # Atributo de classe (compartilhado)
    total_instancias = 0

    def __init__(self, nome):
        # Atributo de instância (único por objeto)
        self.nome = nome
        self.contagem = 0
        Contador.total_instancias += 1

    # Método de instância (acessa self)
    def incrementar(self):
        self.contagem += 1

    # Método de classe (acessa cls)
    @classmethod
    def total(cls):
        return cls.total_instancias

    # Método estático (não acessa nada)
    @staticmethod
    def validar_nome(nome):
        return len(nome) > 0

c1 = Contador("A")
c2 = Contador("B")

c1.incrementar()
print(c1.contagem)  # 1
print(c2.contagem)  # 0
print(Contador.total())  # 2
print(Contador.validar_nome("teste"))  # True
```

Casos de Uso Reais

1. Sistema de Usuários

```python
from datetime import datetime

class Usuario:
    def __init__(self, username, email):
        self.username = username
        self.email = email
        self.criado_em = datetime.now()
        self.ativo = True

    def desativar(self):
        self.ativo = False

    def dias_desde_criacao(self):
        return (datetime.now() - self.criado_em).days

    def __str__(self):
        status = "Ativo" if self.ativo else "Inativo"
        return f"{self.username} ({status})"

user = Usuario("alice", "alice@example.com")
print(user)  # alice (Ativo)
print(f"Dias: {user.dias_desde_criacao()}")
```

2. Catálogo de Produtos

```python
class Produto:
    desconto_global = 0.1  # 10% desconto

    def __init__(self, nome, preco, estoque):
        self.nome = nome
        self.preco = preco
        self.estoque = estoque

    def preco_com_desconto(self):
        return self.preco * (1 - Produto.desconto_global)

    def vender(self, quantidade):
        if quantidade <= self.estoque:
            self.estoque -= quantidade
            return True
        return False

    @classmethod
    def alterar_desconto(cls, novo_desconto):
        cls.desconto_global = novo_desconto

p1 = Produto("Teclado", 200, 10)
p2 = Produto("Mouse", 100, 20)

print(p1.preco_com_desconto())  # 180.0

Produto.alterar_desconto(0.2)  # 20% agora
print(p1.preco_com_desconto())  # 160.0
print(p2.preco_com_desconto())  # 80.0
```

Atributos de Classe vs Instância

```python
class Carro:
    # Atributo de classe (compartilhado por todos)
    rodas = 4

    def __init__(self, marca, modelo):
        # Atributos de instância (únicos por objeto)
        self.marca = marca
        self.modelo = modelo

c1 = Carro("Toyota", "Corolla")
c2 = Carro("Honda", "Civic")

print(c1.rodas)  # 4 (acessa classe)
print(c2.rodas)  # 4 (acessa classe)

# Modificar instância (apenas c1)
c1.rodas = 6  # Cria atributo de instância
print(c1.rodas)  # 6
print(c2.rodas)  # 4 (ainda usa classe)

# Modificar classe (afeta todos)
Carro.rodas = 8
print(c2.rodas)  # 8 (atualizado)
print(c1.rodas)  # 6 (instância sobrescreve)
```

Conclusão

Neste guia de **Classes e Objetos**, você aprendeu:

✅ **Declaração** - Criar classes e instanciar objetos
✅ **Métodos especiais** - **init**, **str**, **repr**, **len**
✅ **Tipos de métodos** - Instância, classe (@classmethod), estáticos (@staticme-

thod)
✅ **Casos práticos** - Usuários, produtos, contadores
✅ **Atributos** - Classe (compartilhado) vs Instância (único)

Principais lições: - **self** refere-se à instância atual - **init** é o construtor (inicializa objeto) - **str** para exibição, **repr** para debug - **@classmethod** acessa atributos de classe - **@staticmethod** não acessa nem self nem cls

Próximos passos: - Aprenda @property para getters/setters - Explore Dataclasses (menos boilerplate) - Estude herança e polimorfismo - Pratique Design Patterns

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

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

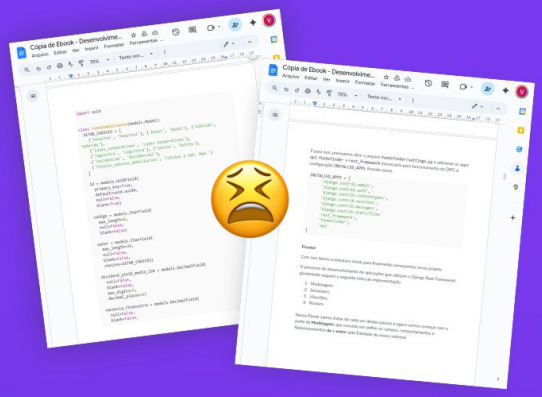
Nos vemos no próximo Post! 👋

Não se esqueça de conferir!

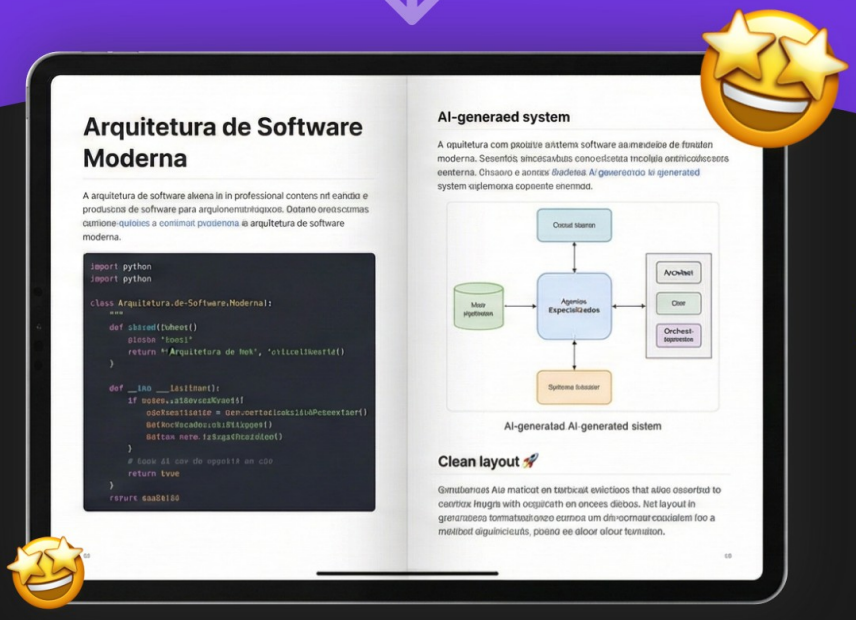
DevBook

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