Objetos e Classes em Python

Tempo estimado necessário: 10 minutos

Objetivos

Nesta leitura, você aprenderá sobre:

- Conceitos fundamentais de objetos e classes em Python.
- Estrutura de classes e código de objetos.
- Exemplos do mundo real relacionados a objetos e classes.

Introdução a classes e objetos

Python é uma linguagem de programação orientada a objetos (OOP) que utiliza um paradigma centrado em objetos e classes.

Vamos examinar esses conceitos fundamentais.

Classes

Uma classe é um modelo ou template para criar objetos. Ela define a estrutura e o comportamento que seus objetos terão.

Pense em uma classe como um cortador de biscoitos e objetos como os biscoitos cortados desse template.

Em Python, você pode criar classes usando a palavra-chave `class`.

Criando classes

Quando você cria uma classe, você especifica os `atributos` (dados) e `métodos` (funções) que os objetos dessa classe terão. `Atributos` são definidos como variáveis dentro da classe, e `métodos` são definidos como funções. Por exemplo, você pode projetar uma classe “Carro” com atributos como “cor” e “velocidade”, junto com métodos como “acelerar”.

Objetos

Um *objeto* é uma unidade fundamental em Python que representa uma entidade ou conceito do mundo real.
Os objetos podem ser tangíveis (como um carro) ou abstratos (como a nota de um estudante).

Todo objeto tem duas características principais:

Estado

Os *atributos ou dados* que descrevem o objeto. Para o seu objeto "Carro", isso pode incluir atributos como "cor", "velocidade" e "nível de combustível".

Comportamento

As *ações ou métodos* que o objeto pode executar. Em Python, métodos são funções que pertencem a objetos e podem alterar o estado do objeto ou realizar operações específicas.

Instanciando objetos

- Uma vez que você definiu uma classe, pode criar objetos individuais (instâncias) com base nessa classe.
- Cada objeto é independente e possui seu próprio conjunto de atributos e métodos.
- Para criar um objeto, você usa o nome da classe seguido de parênteses, assim: “meu_carro = Car()”

Interagindo com objetos

Você interage com objetos chamando seus métodos ou acessando seus atributos usando a notação de ponto.

Por exemplo, se você tiver um objeto Carro chamado **my_car**, você pode definir sua cor com **my_car.color = "azul"** e acelerá-lo com **my_car.accelerate()** se houver um método accelerate definido na classe.

Estrutura de classes e código de objeto

> Por favor, não copie e use este código diretamente, pois é um modelo para explicação e não para resultados específicos.

Declaração de classe (class ClassName)

- A palavra-chave `class` é usada para declarar uma classe em Python.
- `ClassName` é o nome da classe, tipicamente seguindo as convenções de nomenclatura CamelCase.

```
class ClassName:
```

Atributos de classe (class_attribute = valor)

- Atributos de classe são variáveis compartilhadas entre todas as instâncias da classe (objetos).
- Eles são definidos dentro da classe, mas fora de qualquer método.

```
class ClassName:
    # Class attributes (shared by all instances)
    class_attribute = value
```

Método construtor (def init(self, atributo1, atributo2, …):)

- O método `__init__` é um método especial conhecido como construtor.
- Ele inicializa os **atributos de instância** (também chamados de variáveis de instância) quando um objeto é criado.
- O parâmetro `self` é o primeiro parâmetro do construtor, referindo-se à instância que está sendo criada.
- **atributo1**, **atributo2**, e assim por diante são parâmetros passados para o construtor ao criar um objeto.
- Dentro do construtor, `self.atributo1`, `self.atributo2`, e assim por diante são usados para atribuir valores aos atributos de instância.

```
class ClassName:
    # Class attributes (shared by all instances)
    class_attribute = value
    # Constructor method (initialize instance attributes)
    def __init__(self, attribute1, attribute2, ...):
        pass
        # ...
```

Atributos de instância (self.attribute1 = attribute1)

- Atributos de instância são variáveis que armazenam dados específicos para cada instância da classe.
- Eles são inicializados dentro do método `__init__` usando a palavra-chave self seguida pelo nome do atributo.
- Esses atributos contêm dados exclusivos para cada objeto criado a partir da classe.

```
class ClassName:
    # Class attributes (shared by all instances)
    class_attribute = value
    # Constructor method (initialize instance attributes)
    def __init__(self, attribute1, attribute2, ...):
        self.attribute1 = attribute1
        self.attribute2 = attribute2
        # ...
```

Métodos de instância (def method1(self, parameter1, parameter2, …):)

- Métodos de instância são funções definidas dentro da classe.
- Eles operam nos dados da instância (atributos da instância) e podem realizar ações específicas para as instâncias.
- O parâmetro **self** é obrigatório nos métodos de instância, permitindo que eles acessem os atributos da instância e chamem outros métodos dentro da classe.

```
class ClassName:
    # Class attributes (shared by all instances)
    class_attribute = value
    # Constructor method (initialize instance attributes)
    def __init__(self, attribute1, attribute2, ...):
        self.attribute1 = attribute1
        self.attribute2 = attribute2
        # ...
    # Instance methods (functions)
    def method1(self, parameter1, parameter2, ...):
        # Method logic
        pass
```

Usando os mesmos passos, você pode definir múltiplos métodos de instância.

```
class ClassName:
    # Class attributes (shared by all instances)
    class_attribute = value
    # Constructor method (initialize instance attributes)
    def __init__(self, attribute1, attribute2, ...):
        self.attribute1 = attribute1
        self.attribute2 = attribute2
        # ...
    # Instance methods (functions)
    def method1(self, parameter1, parameter2, ...):
        # Method logic
        pass
    def method2(self, parameter1, parameter2, ...):
        # Method logic
        pass
```

Nota: Agora, você criou com sucesso uma classe de exemplo.

Criando objetos (Instâncias)

- Para criar objetos (instâncias) da classe, você chama a classe como uma função e fornece os argumentos que o construtor requer.
- Cada objeto é uma instância distinta da classe, com seus próprios atributos de instância e a capacidade de chamar métodos definidos na classe.

```
# Create objects (instances) of the class
object1 = ClassName(arg1, arg2, ...)
object2 = ClassName(arg1, arg2, ...)
```

Chamando métodos em objetos

- Nesta seção, você chamará métodos em objetos, especificamente `object1` e `object2`.
- Os métodos **method1** e **method2** estão definidos na Classe **class**, e você os está chamando em **object1** e **object2**, respectivamente.
- Você passa os valores **param1_value** e **param2_value** como argumentos para esses métodos. Esses argumentos são usados dentro da lógica do método.

Método 1: Usando notação de ponto

- Esta é a maneira mais direta de chamar um método de um objeto. Nela, usa-se a notação de ponto **(object.method())** para invocar o método no objeto diretamente.
- Por exemplo, `result1 = object1.method1(param1_value, param2_value, ...)` chama method1 em object1.

```
# Calling methods on objects
# Method 1: Using dot notation
result1 = object1.method1(param1_value, param2_value, ...)
result2 = object2.method2(param1_value, param2_value, ...)
```

Método 2: Atribuindo métodos de objeto a variáveis

- Aqui está uma maneira alternativa de chamar o método de um objeto atribuindo a referência do método a uma variável.
- `method_reference = object1.method1` atribui o método **method1** de **object1** à variável **method_reference**.
- Mais tarde, chame o método usando a variável assim: **result3 = method_reference(param1_value, param2_value, …)**.

```
# Method 2: Assigning object methods to variables
method_reference = object1.method1  # Assign the method to a variable
result3 = method_reference(param1_value, param2_value, ...)
```

Acessando atributos de objeto

- Aqui, você está acessando um atributo de um objeto usando notação de ponto.
- `attribute_value = object1.attribute1` recupera o valor do atributo **attribute1** de **object1** e o atribui à variável **attribute_value**.

```
# Accessing object attributes
attribute_value = object1.attribute1  # Access the attribute using dot notation
```

Modificando atributos de objeto

- Você modificará o atributo de um objeto usando notação de ponto.
- `object1.attribute2 = new_value` define o atributo **attribute2** de **object1** para o novo valor **new_value**.

```
# Modifying object attributes
object1.attribute2 = new_value  # Change the value of an attribute using dot notation
```

Acessando atributos da classe (compartilhados por todas as instâncias)

- Por fim, acesse um atributo de classe compartilhado por todas as instâncias da classe.
- `class_attr_value = ClassName.class_attribute` acessa o atributo de classe `class_attribute` da classe ClassName e atribui seu valor à variável. `class_attr_value`.

```
# Accessing class attributes (shared by all instances)
class_attr_value = ClassName.class_attribute
```

Exemplo do mundo real

Vamos escrever um programa em python que simula uma classe de carro simples, permitindo que você crie instâncias de carro, acelere-as e exiba suas velocidades atuais.

1. Vamos começar definindo uma classe `Car` que inclui os seguintes atributos e métodos:

- Atributo de classe `max_speed`, que é definido como **120 km/h**.
- Método construtor `__init__` que recebe parâmetros para a **marca, modelo, cor do carro e uma velocidade opcional (padrão 0)**. Este método inicializa os atributos de instância para marca, modelo, cor e velocidade.
- Método `accelerate(self, acceleration)` que permite que o carro acelere. Se a aceleração não exceder a `max_speed`, atualize o atributo de **velocidade do carro**. Caso contrário, defina a velocidade como **max_speed**.
- Método `get_speed(self)` que retorna a velocidade atual do carro.

```
class Car:
    # Class attribute (shared by all instances)
    max_speed = 120  # Maximum speed in km/h
    # Constructor method (initialize instance attributes)
    def __init__(self, make, model, color, speed=0):
        self.make = make
        self.model = model
        self.color = color
        self.speed = speed  # Initial speed is set to 0
    # Method for accelerating the car
    def accelerate(self, acceleration):
        if self.speed + acceleration <= Car.max_speed:
            self.speed += acceleration
        else:
            self.speed = Car.max_speed
    # Method to get the current speed of the car
    def get_speed(self):
        return self.speed
```

2. Agora, você irá instanciar dois objetos da classe `Car`, cada um com as seguintes características:

- car1: **Marca = “Toyota”, Modelo = “Camry”, Cor = “Azul”**
- car2: **Marca = “Honda”, Modelo = “Civic”, Cor = “Vermelho”**

```
# Create objects (instances) of the Car class
car1 = Car("Toyota", "Camry", "Blue")
car2 = Car("Honda", "Civic", "Red")
```

3. Usando o método `accelerate`, você aumentará a velocidade do carro1 em 30 km/h e do carro2 em 20 km/h.

```
# Accelerate the cars
car1.accelerate(30)
car2.accelerate(20)
```

4. Por fim, você exibirá a velocidade atual de cada carro utilizando o método `get_speed`.

```
# Print the current speeds of the cars
print(f"{car1.make} {car1.model} is currently at {car1.get_speed()} km/h.")
print(f"{car2.make} {car2.model} is currently at {car2.get_speed()} km/h.")
```

Próximos passos

Em conclusão, esta leitura fornece uma compreensão fundamental de objetos e classes em Python, conceitos essenciais na programação orientada a objetos. Classes servem como modelos para a criação de objetos, encapsulando atributos de dados e métodos. Objetos representam entidades do mundo real e possuem seu estado e comportamento únicos. O exemplo de código estruturado apresentado na leitura delineia os elementos-chave de uma classe, incluindo atributos de classe, o método construtor para inicializar atributos de instância e métodos de instância para definir a funcionalidade específica do objeto.

Na próxima sessão de laboratório, você poderá aplicar os conceitos de objetos e classes para ganhar experiência prática.

Autor

Akansha Yadav

Registro de Alterações

Data	Versão	Alterado por	Descrição da Mudança
2023-09-05	1.0	Akansha Yadav	Criado leitura
2024-02-20	2.0	Gagandeep Singh	Revisão de ID
2024-02-20	2.1	Mary Stenberg	Revisão de QA
2024-02-20	2.2	Sydney Marsing	Revisão de UXQA