Conceitos de Programação Orientada a Objetos (POO) em Python

Índice

Seção 1: Definição de Classe e Objeto

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

    def emitir_som(self):
        pass

gato = Animal("Gato")
print(gato.nome)  # Saída: Gato
```

Definição da Classe `Animal`:

- A classe `Animal` é criada. Dentro dela, há um método especial chamado `__init__`, que é o construtor da classe. Este método é chamado automaticamente quando um novo objeto é instanciado a partir da classe.
- `__init__(self, nome)` recebe dois parâmetros: `self` (que representa a instância atual do objeto) e `nome` (o nome do animal).
- `self.nome = nome` atribui o valor do parâmetro `nome` ao atributo `nome` da instância.

Definição do Método `emitir_som`:

- O método `emitir_som` é definido, mas ele não faz nada (a instrução `pass` indica que nenhuma ação é realizada). Esse método é um espaço reservado para ser potencialmente sobrescrito em subclasses.

Criação de um Objeto:

- `gato = Animal("Gato")` cria uma instância da classe `Animal` chamada `gato`, passando o argumento "Gato" para o construtor. Isso define o atributo `nome` do objeto `gato` como "Gato".

Impressão do Nome:

- `print(gato.nome)` imprime o valor do atributo `nome` do objeto `gato`, que é "Gato".

Seção 2: Herança

```
class Gato(Animal):
    def emitir_som(self):
        return "Miau"

gato = Gato("Bichano")
print(gato.nome)  # Saída: Bichano
print(gato.emitir_som())  # Saída: Miau
```

Definição da Subclasse `Gato`:

- A classe `Gato` é criada como uma subclasse da classe `Animal`. Isso é indicado pelo fato de `Gato` herdar de `Animal` (`class Gato(Animal)`).

Sobrescrita do Método `emitir_som`:

- A classe `Gato` sobrescreve o método `emitir_som` da classe `Animal`, de modo que agora ele retorna "Miau".

Criação de um Objeto `Gato`:

- `gato = Gato("Bichano")` cria uma instância da classe `Gato`, passando "Bichano" como o nome. A classe `Gato` herda o construtor da classe `Animal`, então `nome` é definido como "Bichano".

Impressão do Nome e Som:

- `print(gato.nome)` imprime "Bichano".
- `print(gato.emitir_som())` chama o método `emitir_som` da instância `gato`, que agora retorna "Miau".

Seção 3: Polimorfismo

```
class Animal:
    def emitir_som(self):
        pass
```

```
class Gato(Animal):
    def emitir_som(self):
        return "Miau"

class Cachorro(Animal):
    def emitir_som(self):
        return "Au Au"

animais = [Gato(), Cachorro()]

for animal in animais:
    print(animal.emitir_som())
```

Classes `Gato` e `Cachorro`:

- `Gato` e `Cachorro` são subclasses de `Animal`, e ambos sobrescrevem o método `emitir_som` para retornar sons diferentes: "Miau" e "Au Au", respectivamente.

Lista de Animais:

- `animais = [Gato(), Cachorro()]` cria uma lista contendo instâncias das classes `Gato` e `Cachorro`.

Iteração e Polimorfismo:

- O loop `for animal in animais` percorre cada objeto na lista `animais`. Mesmo que `animal` possa ser um `Gato` ou um `Cachorro`, Python chama o método `emitir_som` correto, mostrando o comportamento polimórfico.

A saída será:

- "Miau" para o objeto `Gato`.
- "Au Au" para o objeto `Cachorro`.

Seção 4: Abstração

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def emitir_som(self):
        pass

class Gato(Animal):
    def emitir_som(self):
        return "Miau"
```

```
gato = Gato()
print(gato.emitir_som())  # Saída: Miau
```

Uso da Classe Abstrata `Animal`:

- `Animal` é uma classe abstrata, definida usando `ABC` (Abstract Base Class). O método `emitir_som` é decorado com `@abstractmethod`, o que significa que qualquer subclasse de `Animal` deve implementar esse método.

Subclasse `Gato`:

- `Gato` herda de `Animal` e implementa o método `emitir_som`, retornando "Miau".

Criação de um Objeto e Chamada do Método:

- `gato = Gato()` cria uma instância de `Gato`, e `print(gato.emitir_som())` imprime "Miau".

Seção 5: Encapsulamento

```
class ContaBancaria:
    def __init__(self, saldo_inicial):
        self.__saldo = saldo_inicial

    def depositar(self, quantia):
        self.__saldo += quantia

    def sacar(self, quantia):
        if quantia <= self.__saldo:
            self.__saldo -= quantia
            return True
        return False

    def obter_saldo(self):
        return self.__saldo

conta = ContaBancaria(100)
conta.depositar(50)
print(conta.obter_saldo())  # Saída: 150
conta.sacar(75)
print(conta.obter_saldo())  # Saída: 75
```

Atributos Privados:

- `__saldo` é um atributo privado da classe `ContaBancaria`, indicado pelos dois underscores no início do nome. Isso significa que ele não pode ser acessado diretamente fora da classe.

Métodos Públicos:

- `depositar`, `sacar` e `obter_saldo` são métodos públicos que permitem manipular e acessar o saldo da conta de forma controlada.

Exemplo de Uso:

- Uma instância de `ContaBancaria` é criada com um saldo inicial de 100. Em seguida, 50 são depositados, e o saldo é verificado (150). Após um saque de 75, o saldo é verificado novamente (75).