

Programação Orientada a Objetos

Polimorfismo

Parte 6

Programação Orientada a Objetos

Polimorfismo

O polimorfismo é baseado nas palavras gregas Poli (muitas) e morphos (formas).

É a capacidade de um objeto adaptar o código ao tipo de dados que está processando.

Programação Orientada a Objetos

Polimorfismo

Objetos podem ser de muitos tipos ou formas. Por exemplo, botões possuem formas diferentes, existem botões redondos, quadrados, botões com imagem, mas todos compartilham a mesma lógica, executam uma ação chamada “clique”. A capacidade de acessarmos o método “clique” independente do tipo de botão é chamada de polimorfismo.



Programação Orientada a Objetos

Polimorfismo

O polimorfismo pode ser de dois tipos:

- Um objeto oferece diferentes implementações do método de acordo com os parâmetros de entrada;
- A mesma interface pode ser usada por objetos de tipos diferentes.

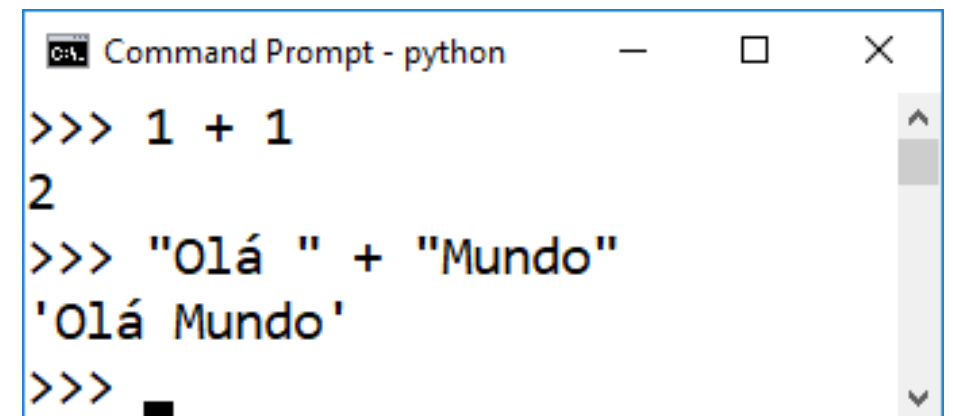
Programação Orientada a Objetos

Polimorfismo

Em Python, o polimorfismo é um recurso embutido na linguagem.

Por exemplo, o operador “+” pode ser usado para adicionar dois números mas também pode ser usado para concatenar duas strings.

Devido ao polimorfismo o “+” possui uma função adicional para concatenar texto, além da função de somar dois números.

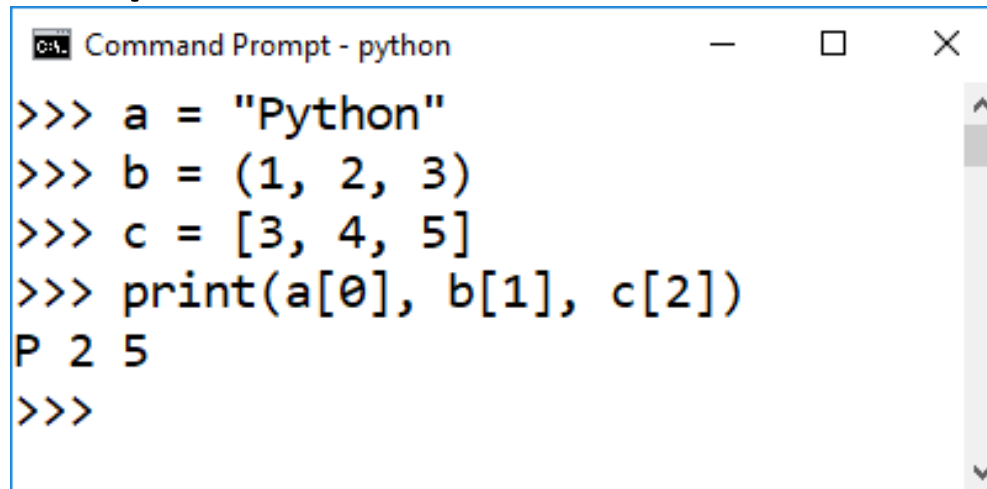


```
Command Prompt - python
>>> 1 + 1
2
>>> "Olá " + "Mundo"
'Olá Mundo'
>>>
```

Programação Orientada a Objetos

Polimorfismo

Strings, tuplas ou listas podem ser acessadas com um índice inteiro. Isto mostra como Python demonstra o polimorfismo em tipos embutidos.



```
Command Prompt - python
>>> a = "Python"
>>> b = (1, 2, 3)
>>> c = [3, 4, 5]
>>> print(a[0], b[1], c[2])
P 2 5
>>>
```

Programação Orientada a Objetos

Polimorfismo

```
class Cachorro(object):  
    def som(self):  
        print("Au au")  
  
class Gato(object):  
    def som(self):  
        print("Miau")  
  
def emitir_som(animal):  
    animal.som()  
  
cachorrinho = Cachorro()  
gatinho = Gato()  
  
emitir_som(cachorrinho)  
emitir_som(gatinho)
```

Esta é uma forma de utilizarmos polimorfismo com função. Criamos duas classes: Cachorro e Gato, ambos podem fazer um som distinto. Em seguida, criamos duas instâncias e chamamos sua ação usando o mesmo método denominado “som”.

Programação Orientada a Objetos

Polimorfismo

```
class Cachorro(object):  
    def som(self):  
        print("Au au")  
  
class Gato(object):  
    def som(self):  
        print("Miau")  
  
def emitir_som(animal):  
    animal.som()  
  
cachorrinho = Cachorro()  
gatinho = Gato()  
  
emitir_som(cachorrinho)  
emitir_som(gatinho)
```

A classe Cachorro possui um método chamado "som" que imprime o texto "Au au"

A classe Gato possui um método chamado "som" que imprime o texto "Miau"

O método emitir_som recebe um objeto chamado "animal" e executa o método "som" desse objeto.

Foram instanciados dois objetos, um da classe Cachorro e outro da classe Gato.

O método emitir_som foi executado duas vezes, na primeira passando um objeto do tipo Cachorro e na segunda, um objeto do tipo Gato.

O resultado foi:
Au au
Miau

Programação Orientada a Objetos

Polimorfismo

```
class Porta(object):  
    def fechar(self):  
        print("A porta foi fechada!")  
  
    def abrir(self):  
        print("A porta foi aberta!")  
  
class Janela(object):  
    def fechar(self):  
        print("A janela foi fechada!")  
  
    def abrir(self):  
        print("A janela foi aberta!")
```

```
def realizar_abertura(o_que_abrir):  
    o_que_abrir.abrir()  
  
def realizar_fechamento(o_que_fechar):  
    o_que_fechar.fechar()  
  
porta = Porta()  
janela = Janela()  
  
realizar_abertura(porta)  
realizar_abertura(janela)  
realizar_fechamento(porta)  
realizar_fechamento(janela)
```

Programação Orientada a Objetos

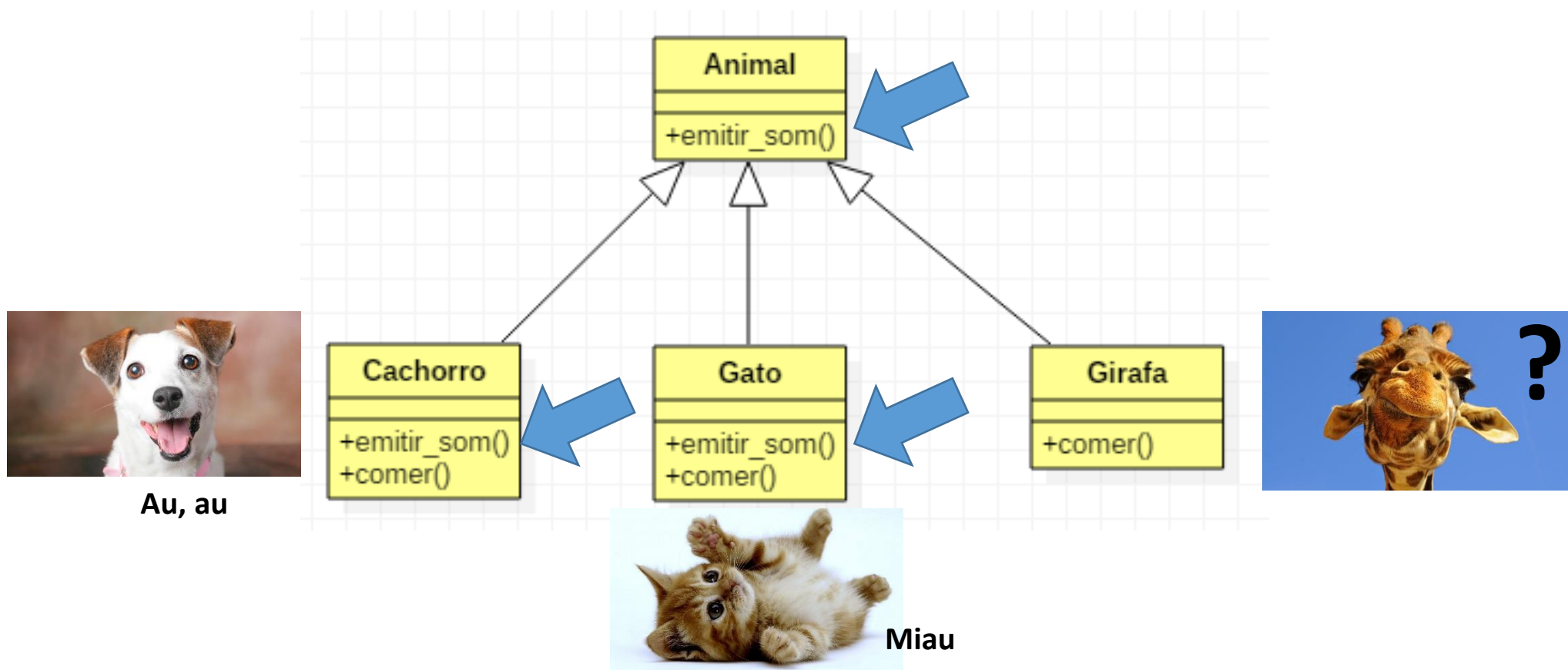
Polimorfismo

O mais comum é utilizarmos polimorfismo com classes abstratas.

Uma classe abstrata é uma classe que não possui implementação, possuindo apenas uma estrutura com os métodos que devem ser implementados nas subclasses. Desta forma, os métodos devem ser obrigatoriamente implementados na subclasse.

Programação Orientada a Objetos

Polimorfismo



Programação Orientada a Objetos

Polimorfismo

```
class Animal(object):
    def emitir_som(self):
        raise NotImplementedError("Não foi implementado o método emitir_som.")

class Cachorro(Animal):
    def emitir_som(self):
        print("Au au")

class Gato(Animal):
    def emitir_som(self):
        print("Miau")

class Girafa(Animal):
    def comer(self):
        print("A girafa está comendo.")
```

```
def barulho(animal):
    animal.emitir_som()

cachorrinho = Cachorro()
gatinho = Gato()
girafinha = Girafa()

barulho(cachorrinho)
barulho(gatinho)
barulho(girafinha)
```

O **raise** é uma forma de forçar um tipo de exceção. Neste caso se o método “emitir_som” não for implementado na classe derivada, irá emitir a mensagem “Não foi implementado o método emitir_som.” O método emitir_som não foi implementado na classe Animal, apenas criamos o “esqueleto” e terá que ser implementado nas subclasses.

```
Au au
Miau
NotImplementedError: Não foi implementado
o método emitir_som.
```

Programação Orientada a Objetos

Polimorfismo

Quando eu implemento um método na classe base (não será mais uma classe abstrata) e o mesmo não é implementado na subclasse, será executado o método da classe base. Veja:

```
class Animal(object):  
    def emitir_som(self):  
        print("Um som qualquer")  
  
class Girafa(Animal):  
    def comer(self):  
        print("A girafa está comendo.")
```

```
def barulho(animal):  
    animal.emitir_som()  
  
girafinha = Girafa()  
barulho(girafinha)
```

Será impresso: "Um som qualquer".

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

Se eu fizer assim, não será obrigatório implementar na subclasse, da mesma forma. Porque foi implementado e não dará erro, porém, não irá fazer nada.

Programação Orientada a Objetos

Polimorfismo

Quando o método é implementado na classe base e também implementado na subclasse, temos algo similar a sobrescrita de método que ocorre no Java (ou outras linguagens). O método executado é o da subclasse.

```
class Animal(object):  
    def emitir_som(self):  
        print("Um som qualquer")  
  
class Cachorro(Animal):  
    def emitir_som(self):  
        print("Au au")
```

```
def barulho(animal):  
    animal.emitir_som()  
  
cao = Cachorro()  
  
barulho(cao)
```

Será impresso: "Au au".

Programação Orientada a Objetos

Polimorfismo

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

    def abrir(self):
        raise NotImplementedError("O método abrir deve ser implementado na subclasse.")

class Pdf(Documento):
    def abrir(self):
        return 'Foi aberto um arquivo PDF!'

class Doc(Documento):
    def abrir(self):
        return 'Foi aberto um arquivo DOC!'

class Xls(Documento):
    pass
```

```
documentos = [Pdf('Arquivo.pdf'),
               Pdf('Lista.pdf'),
               Doc('Documento.doc'),
               Xls('Planilha.xls')]

for documento in documentos:
    print(documento.abrir(), "de nome", documento.nome)
```

CONTINUA...