# Polimorfismo

Programação Orientada a Objetos

Python

Prof. Diógenes Furlan

# Polimorfismo

- Indica a habilidade de uma única instrução chamar diferentes funções e portanto assumir formas diferentes.
- Nos nossos exemplos:
  - Um Funcionário também é uma Pessoa
  - Um Gerente também é um Funcionário
- Isto permite que eu possa acessar o objeto por uma “interface” diferente.

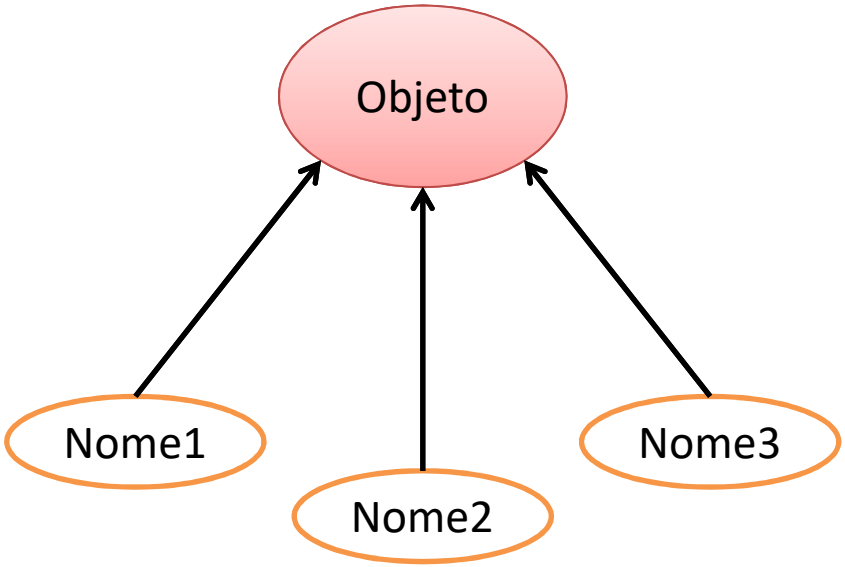
# Tipos de Polimorfismo

- Existem quatro tipos de polimorfismo que a linguagem pode ter
- Universal
  - Inclusão: um ponteiro da classe mãe pode apontar para uma instância de uma classe filha
  - Paramétrico: se restringe ao uso de templates (C++, por exemplo) e generics (C#/Java)
- Ad-Hoc
  - Sobrecarga: duas funções/métodos com o mesmo nome mas assinaturas diferentes
  - Coerção: conversão implícita de tipos sobre os parâmetros de uma função

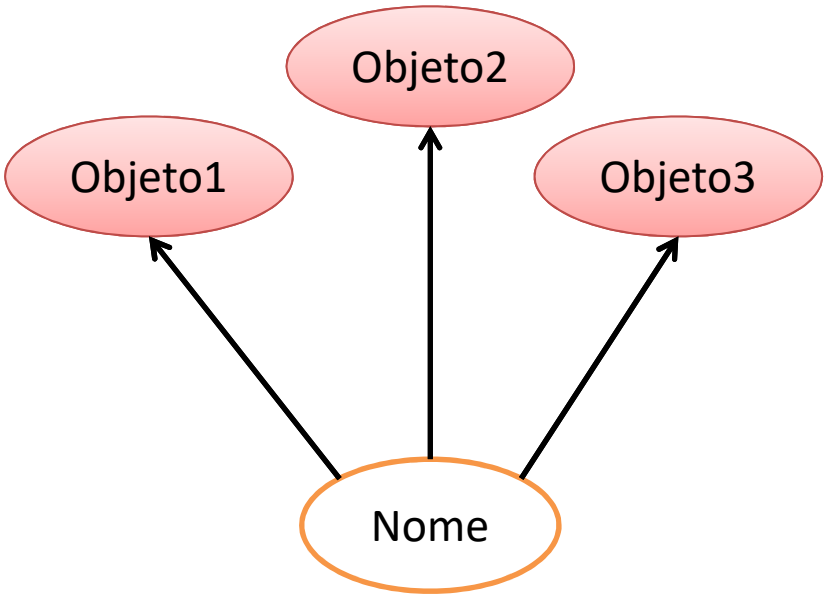
# Diferenças!!!



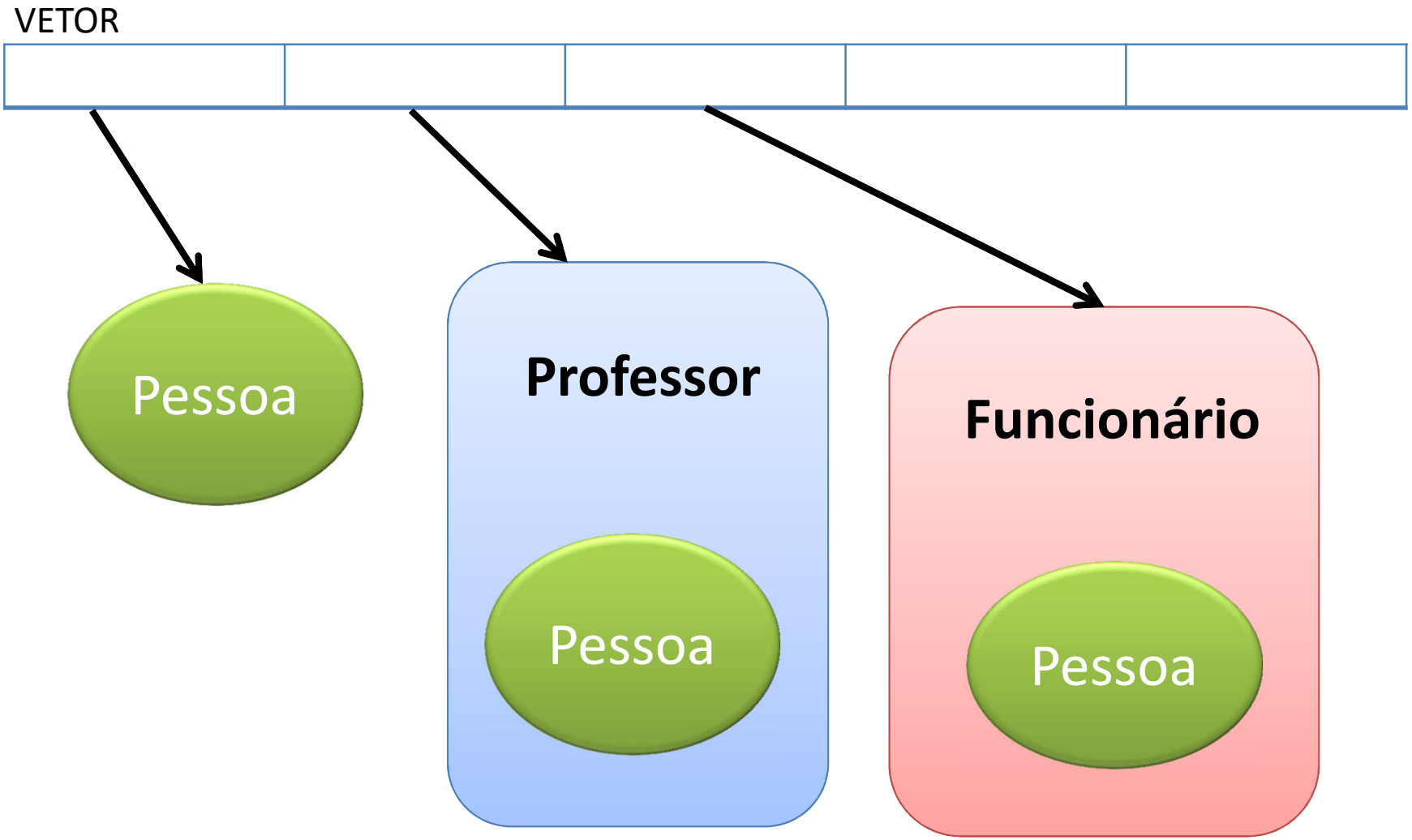
**Inclusão**




**Sobrecarga**



# Polimorfismo



# Polimorfismo

- O **polimorfismo** permite que diferentes classes tenham métodos com o mesmo nome, mas comportamentos diferentes.
- Em Python, isso pode ser facilmente implementado usando herança e sobrescrita de métodos.

# Exemplo 1

```
from abc import ABC, abstractmethod

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

class Cachorro(Animal):
    def fazer_som(self):
        return "Latido"

class Gato(Animal):
    def fazer_som(self):
        return "Miau"

class Vaca(Animal):
    def fazer_som(self):
        return "Muuu"
```

# Exemplo 1

```
# Criando uma lista de animais
animais = [Cachorro(), Gato(), Vaca(), Cachorro()]

# Demonstrando polimorfismo
for animal in animais:
    print(animal.fazer_som())
```

# CLASSE OBJECT

# Classe Object

- ***Objetos*** são abstrações do Python para dados.
  - Todos os dados em um programa Python são representados por objetos
  - ou por relações entre objetos.
- Todo objeto tem:
  - uma identidade
  - um tipo
  - um valor

# Classe Object

- A identidade de um objeto nunca muda depois de criado (imutável)
  - similar ao endereço de objetos em memória.
  - O operador '**is**' compara as identidades de dois objetos
  - A função **id()** retorna um inteiro representando sua identidade.

- A função **type()** retorna o tipo de um objeto (que é também um objeto).
  - Como sua identidade, o tipo do objeto também é imutável.

# Identidade e Tipo

- Exemplos:

```
a = 120
print( id(a) )
print( type(a) )

b = 3.8
print( id(b) )
print( type(b) )

c = 'aula'
print( id(c) )
print( type(c) )

d = True
print( id(d) )
print( type(d) )
```

```
2732893474768
<class 'int'>
2732933405744
<class 'float'>
2732934812656
<class 'str'>
140708887186280
<class 'bool'>
```

# Verificando o Tipo de um Objeto

- isinstance(): verifica se o objeto é uma instância de <tipo> ou de qualquer subclasse de <tipo>.

```
if isinstance(obj, str):
    print('obj é uma string ou subclasse de string')
```

# Verificando o Tipo Exato

- type(): verificar se o tipo de um objeto é exatamente <tipo>, excluindo quaisquer subclasses.

```
if type(objeto) is str:
    print('obj é exatamente uma string')
```

# COMPARAÇÕES DE OBJETOS

# Qual deve ser a diferença??

- is
- ==

- Exemplo:
  - a = 2, b = 2
  - a is b ?
  - a == b ?

# Comparação

- is

  verifica a identidade dos objetos

- ==

  verifica o valor dos objetos

# IS versus ==

- Exemplo

```
x = 700
y = 600
print(x == y)
print(x is y)
```

False
False

```
x = 700
y = 700
print(x == y)
print(x is y)
```

True
True

```
x = 700
y = x
print(x == y)
print(x is y)
```

True
True

```
a = [2]
b = [2]
print(a == b)
print(a is b)
```

True
False

# Identidade

- Exemplo:

```
class Point:
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def __str__(self):
        return '(%g, %g)' % (self.x, self.y)


p1 = Point(3,4)
print(p1)
p2 = p1
print(p2)
p1.x = 78
print(p1)
print(p2)
```

# Identidade

- A "referencia" ao objeto faz com que p2 seja um "novo nome" para p1
- Similar ao uso de ponteiro em C

```
>>> p1 = Point(3,4)
>>> print(p1)
(3, 4)
>>> p2 = p1
>>> print(p2)
(3, 4)
>>> p1.x = 78
>>> print(p1)
(78, 4)
>>> print(p2)
(78, 4)
>>> p1 is p2
True
```

Clonagem

# CÓPIA DE OBJETOS

# Cópia

- "Alias" ou "Referencia" pode tornar um programa difícil de ler porque as alterações em um lugar podem ter efeitos inesperados em outro lugar.
- Existem situações onde queremos "copiar" um objeto
  - uso do módulo **copy**

# Cópia

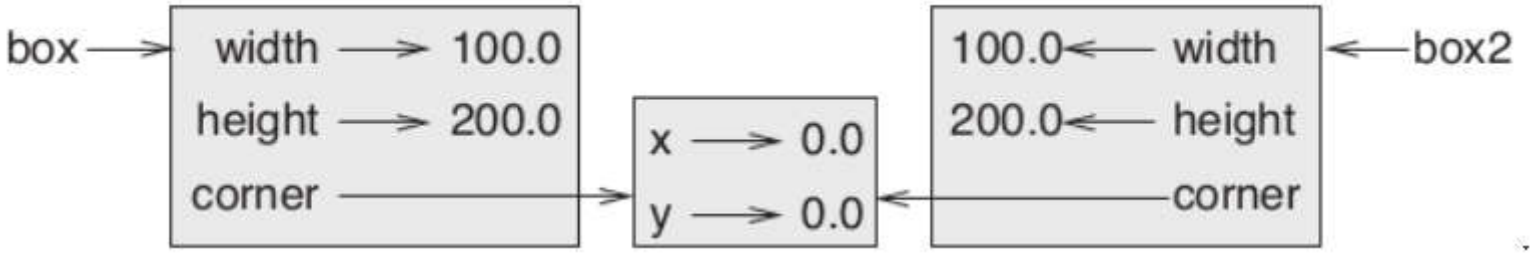
```
>>> import copy
>>> p1 = Point(3,4)
>>> print(p1)
(3, 4)
>>> p2 = copy.copy(p1)
>>> print(p2)
(3, 4)
>>> p1.x = 78
>>> print(p1)
(78, 4)
>>> print(p2)
(3, 4)
>>> p1 is p2
False
```

# Cópia Profunda

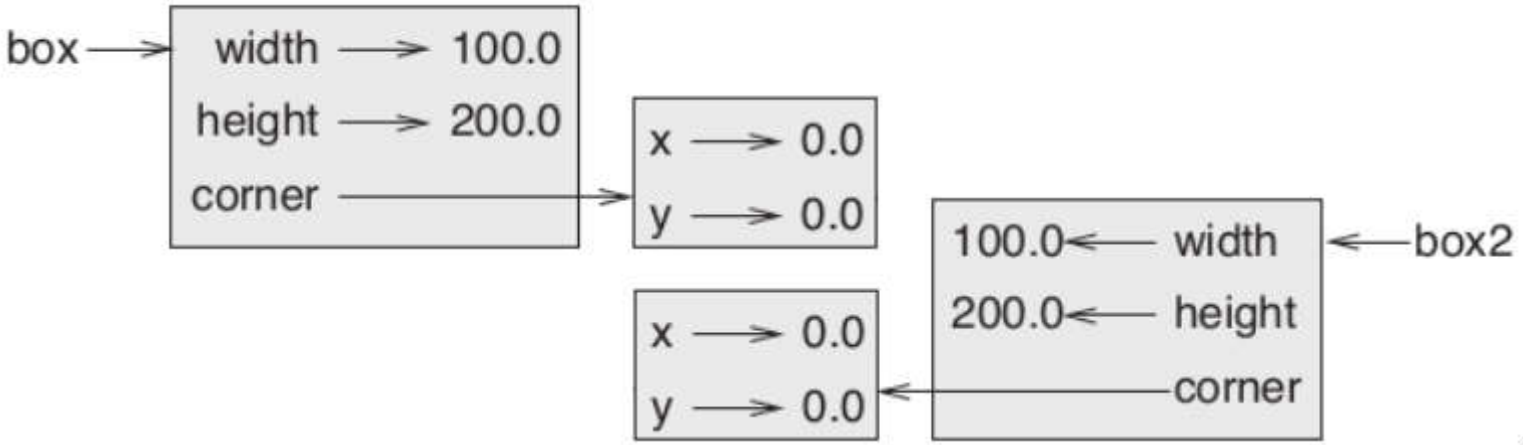
- O módulo **copy** oferece um método chamado **deepcopy** que copia não só o objeto, mas também
  - os objetos aos quais ele se refere
  - e os objetos aos quais estes se referem
  - e assim por diante.

# Copy x Deepcopy

- Copy




- Deepcopy



# EXERCÍCIOS

# Exercício 1

- Faça uma lista para a classe Conta e insira algumas:
  - contas correntes
  - contas poupança
  - contas especiais
- Teste o método de saque para
  - valores negativos
  - valores altos

# Exercício 2

- Faça uma lista para a hierarquia da classe Polígonos e insira objetos:
  - triângulo
  - retângulo
  - círculo
- Teste os métodos
  - área
  - perímetro

# Exercício 3

- Faça uma cópia correta do objeto:
  - Polígono Retângulo
  - Veículo Anfíbio
  - Playlist