IFRN
PROGRAMAÇÃO ORIENTADA A OBJETOS

POO em Python – Herança

Prof. Gilbert Azevedo

Objetivos

- Compreender os conceitos de herança e polimorfismo da POO em Python

Herança

- Herança é o conceito da POO que define a relação de generalização – especialização
- Sintaxe de herança no Python

```
class ClasseBase:
    pass

class ClasseDerivada(ClasseBase):
    pass
```

Funções type e isinstance

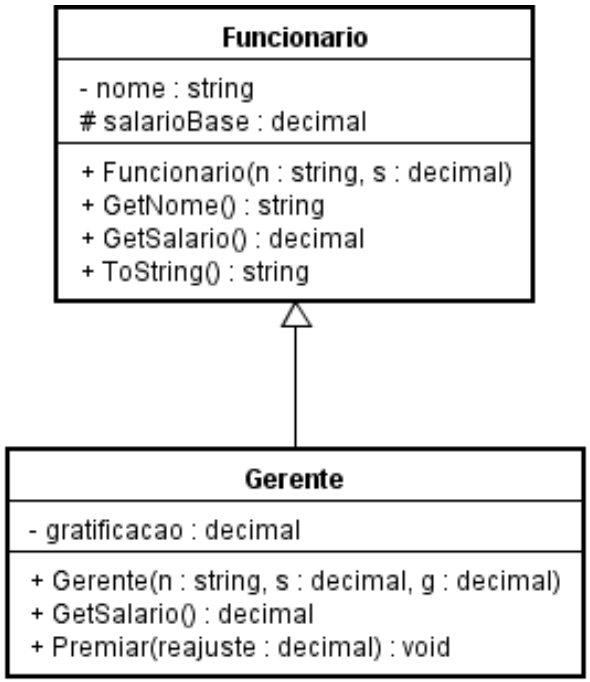
- Função *type* retorna a classe de um objeto
- Função *isinstance* verifica se o objeto é instancia de uma determinada classe (ou de uma classe ancestral dela)

```
class Mamifero:
  pass

class Gato(Mamifero):
  pass

x = Gato()
print(type(x))
print(type(x) == Mamifero)
print(type(x) == Gato)
print(isinstance(x, Mamifero))
print(isinstance(x, Gato))
```

```
<class '__main__.Gato'>
False
True
True
True
```

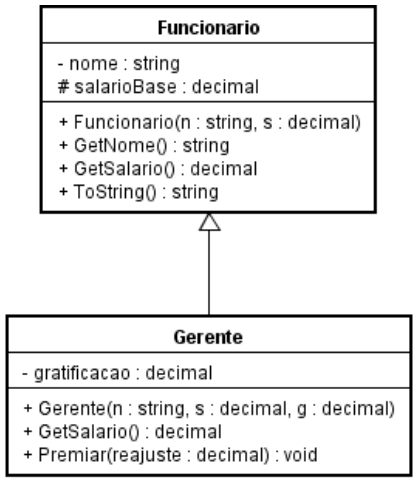
Polimorfismo

- Polimorfismo é o conceito da POO que permite a classes distintas em uma hierarquia realizar a mesma operação de forma diferente
  - Gerente é um Funcionário
  - Cálculo do Salário (GetSalario) possui operações diferentes



Função super

- Função *super* permite chamar métodos da classe base




```
class Funcionario:
    def __init__(self, nome, salarioBase):
        self.__nome = nome
        self._salarioBase = salarioBase
    def getNome(self):
        return self.__nome
    def getSalario(self):
        return self._salarioBase
    def __str__(self):
        return "Nome: " + str(self.__nome) + " - Salário = "
        + str(self.getSalario())

class Gerente (Funcionario):
    def __init__(self, nome, salarioBase, gratificacao):
        super().__init__(nome, salarioBase)
        self.__gratificacao = gratificacao
    def getSalario(self):
        return super().getSalario() + self.__gratificacao
```

```
x = Funcionario("Jose Maria", 5000)
y = Gerente("Maria Jose", 5000, 3000)
print(x)
print(y)
```

```
Nome: Jose Maria - Salário = 5000
Nome: Maria Jose - Salário = 8000
```

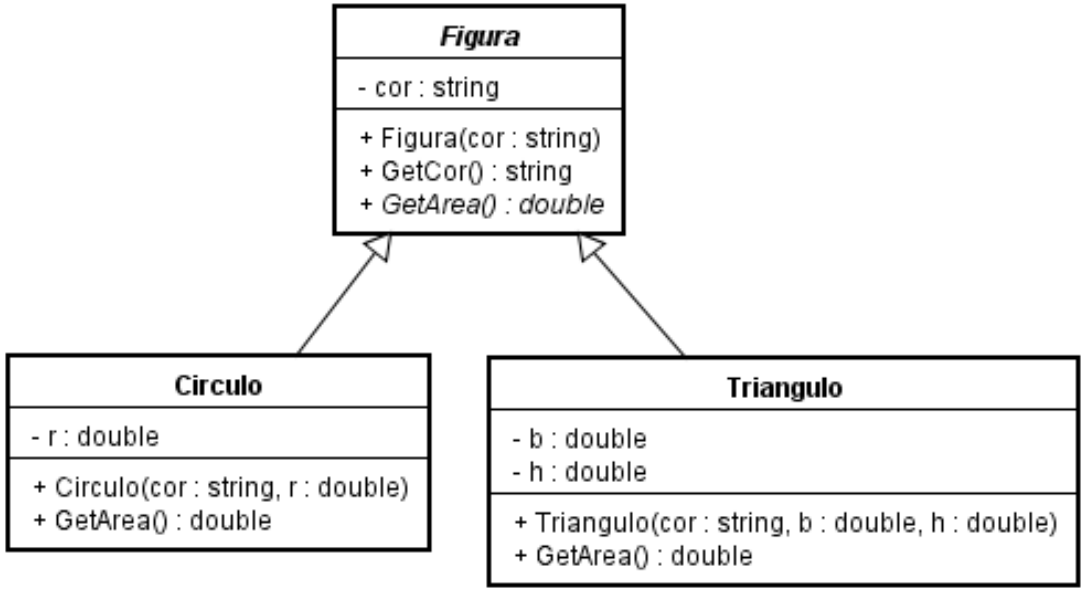
Classes Abstratas

- Classes abstratas no Python devem herdar da classe *ABC* (*Abstract Base Class*) que provê a funcionalidade necessária à implementação do conceito
- O módulo *abc* define a classe *ABC* e o decorador *abstractmethod*

```
from abc import ABC, abstractmethod

class ClasseAbstrata(ABC):
  def __init__(self, valor):
    super().__init()
    self.valor = valor

  @abstractmethod
  def metodoabstrato(self):
    pass
```

Exemplo

- Classe *Figura* é abstrata



Exemplo

- Classe *Figura*

```
from abc import ABC, abstractmethod
import math

class Figura(ABC):
  def __init__(self, cor):
    super().__init__()
    self.__cor = cor

  def getCor(self):
    return self.__cor

  @abstractmethod
  def getArea(self):
    pass
```

Classes *Círculo* e *Triângulo*

```
class Circulo(Figura):
  def __init__(self, cor, raio):
    super().__init__(cor)
    self.__r = raio
  def getArea(self):
    return math.pi * self.__r ** 2

class Triangulo(Figura):
  def __init__(self, cor, base, altura):
    super().__init__(cor)
    self.__b = base
    self.__h = altura
  def getArea(self):
    return self.__b * self.__h / 2
```

```
x = Circulo("Azul", 2)
y = Triangulo("Cinza", 10, 20)
print(x.getCor(), x.getArea())
print(y.getCor(), y.getArea())
```

```
Azul 12.566370614359172
Cinza 100.0
```

Referências

- OOP Python Tutorial
  - https://www.python-course.eu/python3_object_oriented_programming.php