IFRN
PROGRAMAÇÃO ORIENTADA A OBJETOS

POO em Python – Classes

Prof. Gilbert Azevedo

Objetivos

- Compreender os conceitos básicos de POO em Python
  - Definição de Classes
  - Atributos
  - Métodos Especiais
  - Encapsulamento

Classes

- Classes definem tipos de objetos e modelam o mundo real
- No Python, “tudo” é classe: números, textos, funções, métodos e módulos são instâncias de classes

```
import math
def dobro(x):
  return 2 * x;
x = 16
y = math.sqrt(x)
s = "Raiz Quadrada"
print(type(x))
print(type(y))
print(type(s))
print(type(dobro))
print(type(math.sqrt))
print(type(math))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'function'>
<class 'builtin_function_or_method'>
<class 'module'>
```

Definição de Classes

- Classes são definidas com a palavra reservada *class*
- Classe Triângulo em Python
  - *pass* é usado para definir uma classe “vazia”

```
class Triangulo:
  pass
```

- Instancias da classe Triângulo

```
x = Triangulo()
y = Triangulo()
print(x)
print(y)
print(x == y)
print(type(x))
```

```
<__main__.Triangulo object at 0x000002771DD29F70>
<__main__.Triangulo object at 0x000002771DDDC6A0>
False
<class '__main__.Triangulo'>
```

Atributos Dinâmicos

- Python permite que atributos sejam definidos dinamicamente em um objeto
  - “x” é uma instância de classe Triangulo e possui os atributos *base* e *altura*
  - “y” é uma instância de classe Triangulo mas não possui os atributos *base* e *altura*

```
class Triangulo:
  pass

x = Triangulo()
x.base = 10
x.altura = 20
print(x.base * x.altura / 2)

y = Triangulo()
```

Método __init__ e atributos

- O método __init__ é o método usado para iniciar as instâncias e comumente usado para definir os atributos da classe
  - Todos os métodos em Python devem receber um parâmetro *self*

```
class Triangulo:
  def __init__ (self, base, altura):
    self.base = base
    self.altura = altura

  def area(self):
    return self.base * self.altura / 2

x = Triangulo(10, 20)
print(x.base)
print(x.altura)
print(x.area())
```

```
10
20
100.0
```

Método __str__

- O método __str__ é o método usado para retornar um texto com informações sobre o objeto

```
class Triangulo:
  def __init__ (self, base, altura):
    self.base = base
    self.altura = altura

  def __str__ (self):
    return "Triângulo com base = " + str(self.base) +
    " e altura = " + str(self.altura)

  def area(self):
    return self.base * self.altura / 2

x = Triangulo(10, 20)
print(x)
```

```
Triângulo com base = 10 e altura = 20
```

Encapsulamento

- O encapsulamento, no Python, é realizado com o uso do caractere *underscore* "_" no início do nome do atributo
  - Públicos: não usa *underscore*
  - Protegidos: um *underscore*
  - Privados: dois *underscores*

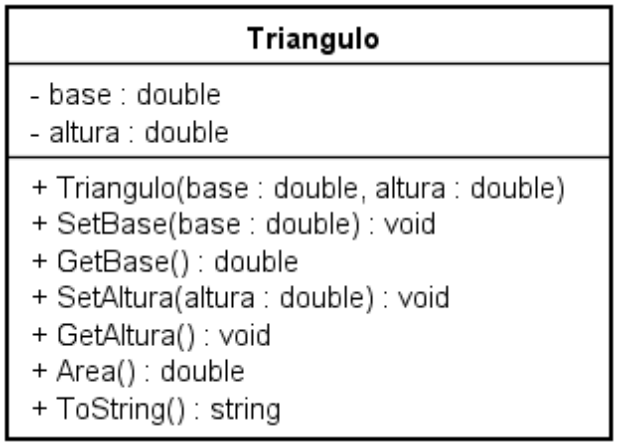
```
class Exemplo:
    def __init__ (self):
        self.atributo01 = "público"
        self._atributo02 = "protegido"
        self.__atributo03 = "privado"
```

```
x = Exemplo()
print(x.atributo01)
print(x._atributo02)
#print(x.__atributo03)
```

```
público
protegido
    print(x.__atributo03)
AttributeError: 'Exemplo' object has no attribute
'__atributo03'
```

Triângulo em Python

```
class Triangulo:
  def __init__ (self, base = 0, altura = 0):
    self.__base = base if base > 0 else 0
    self.__altura = altura if altura > 0 else 0

  def __str__ (self):
    return "Triângulo com base = " + str(self.__base) + " e altura = " +
    str(self.__altura)

  def area(self): return self.__base * self.__altura / 2

  def set_base(self, base):
    if (base > 0): self.__base = base

  def get_base(self): return self.__base

  def set_altura(self, altura):
    if (altura > 0): self.__altura = altura

  def get_altura(self): return self.__altura
```



Referências

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