Atividade II.2

Lucas Zacchi March 14, 2020

Programação Orientada a Objetos em Python

Python é uma linguagem de propósito geral multi-paradigma. Um dos paradigmas suportados é a Orientação a Objetos. O suporte a OOP existe desde o início do desenvolvimento da linguagem, e por conta disso, programar Orientação a Objetos em Python pode trazer mais simplicidade do que outras linguagens. O relatório a seguir apresenta um pequeno tutorial dos principais conceitos OOP em Python

Listing 1: Uma classe simples em Python

```
1 class Student():
2     def __init__(self, name, grade):
3         print("This is a constructor")
4         self.name = name
5         self.grade = grade
6 
7     def my_method(self):
8         print("Hi, my name is ", self.name)
```

O Exemplo 1 mostra uma classe simples em Python, com um construtor e um método. O construtor é o método `__init__()`. O propósito dele é inicialiar uma instância da classe Student e seus atributos. Um exemplo mostra a inicialização de um Ojbeto da classe `Student`:

```
1 >>> student = Student('Lucas', 10)
2 >>> student.name
3 'Lucas'
4 >>> lucas.grade
5 10
6 >>> student.my_method()
7 'Hi, my name is  Lucas'
```

No paradigma OOP, herança é o processo pelo qual uma classe recebe atributos de outra (por isso, "herda"). Python suporta herança "simples", quando uma classe herda de uma outra apenas, ou herança múltipla, quando uma classe herda de duas ou mais classes. Python suporta também polimorfismos:

Listing 2: Herança simples em Python.

```
class Father():
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Daughter(Father):
    def __init__(self, name, age):
        super().__init__(name, age)

>>> daughter = Daughter('Maria', 22)
>>> daughter.name
'Maria'
>>> daughter.age
22
```

No exemplo, a classe filha chama o método da superclasse através do método `super()`. Com heranças múltiplas, a construção é similar. A ordem de precedência das classes é feita da esquerda para a direita. Por exemplo:

```
class Mother(object):
    def greet_mom(self):
        print("The mother's name is Joana")

class Father(object):
    def greet_dad(self):
        print("The father's name is Pedro")

class Daughter(Mother, Father):
        pass

if __name__ == '__main__':
    Daughter().greet_mom()
    Daughter().greet_dad()
```

A composição de classes é uma ferramenta que podemos utilizar quando nossos objetos são compostos (ou seja, utilizam em sua construção instâncias de outros objetos). Em Python, composição pode ser feita instanciandp um objeto nos atributos de uma classe, como mostra o exemplo:

Listing 3: Composição de objetos em Python

```
class Car:
    def __init__(self, make, model, fuel_tank):
```

```
        self.make = make
        self.model = model
        self.fuel_tank = Fuel_Tank()

    def drive(self, distance):
        self.fuel_tank.spend(distance)
```

Python possui também métodos estáticos e abstratos. Métodos estáticos são independentes do objeto. Ou seja, não é necessário que o objeto seja instanciado de modo a utilizar um método estático. Por exemplo:

```
from math import sqrt

class Square():
    def area(a):
        return a*a

    @staticmethod
    def diagonal(a):
        return x*sqrt(2)

Square.area = staticmethod(Square.area)

>>> print('The area of a square with a=2 is: ', Square.area(2))
The area of a square with a=2 is:  4
>>> print('The area of a square with a=2 is: ', Square.diagonal(2))
The area of a square with a=2 is:  2.8284271247461903
```

Existem duas maneiras de implementar um método estático, como o exemplo acima mostra. Uma delas é com a função `staticmethod()` e a outra é através do decorator `@staticmethod()`

Por último, métodos abstratos são métodos declarados em uma superclasse mas que são implementados somente na subclasse que os herda. Isso pode ser feito da seguinte maneira:

```
class Art():
    def art(self):
        pass

class AbstractArt(Art):
    def art(self):
        print("I can't understand it")
```