2. Noções básicas de programação em Python

2.5 - Object-Oriented Programming

Cristiano Bacelar de Oliveira

Julho, 2020

Universidade Federal do Ceará

Summary

- Overall Concept
- Classes and Objects
- Examples

Overall Concept

Object-Oriented Programming

Object-oriented programming (OOP) is a programming paradigm that treats computer programs as a collection of basic units, named 'objects'.

- Each object contains data (attributes) and functions (methods) for manipulating data.
- An object can send/receive data to/from other objects.
- Objects are defined according to a data structure template named 'class'

Object-Oriented Design

Process-oriented programming treats a computer program as a series of commands.

This type of design results in the sequential execution of a set of functions.

The idea of Object-oriented Design (OOD) changes this paradigm since it derives from the natural concept of objects and classes.

- OOD does not specifically require OOP languages.
- An OOP language does not necessarily force you to write OO-related programs.
- In Python, the use of classes and OOP is not mandatory.

Classes and Objects

Classes

A class is a code template that defines a common set of attributes (data) and methods (functions) for objects.

'Class' is an abstract concept.

An object-oriented program can include multiple classes.

Objects

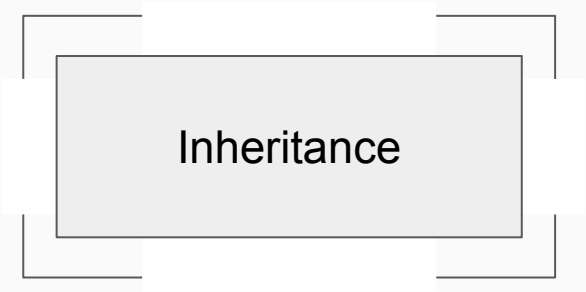
An object is an instance of a class. It holds the actual status (values) of attributes and methods.

Objects can represent many real-world entities, such as personal info, databases, multimedia, communication channels, etc.

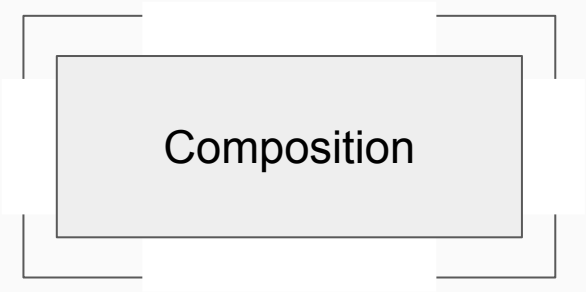
Multiple objects can be instantiated from the same class.

Classes Relationship

A class can be created by reusing one or more existing classes.



The new class (subclass) “**is a**” specialized version of another class (superclass).



The new class “**has**” objects from other classes.

Examples

Example 1

```
class MyClass(object):
        myData = 10
        def myFunction(self):
                print("MyData value is " + str(self.myData))

MyObject1 = MyClass()
MyObject1.myData = 50
MyObject2 = MyClass()

MyObject1.myFunction()
MyObject2.myFunction()
```

Example 2

```
class MyClass(object):
        myData = 10
        def __init__(self, myData):
                self.myData = myData
        def myFunction(self):
                print("MyData value is " + str(self.myData))

MyObject1 = MyClass(50)
MyObject2 = MyClass(30)

MyObject1.myFunction()
MyObject2.myFunction()
```

Example - Inheritance

```
class Pessoa:
  def __init__(self, nome, dt_nascimento):
        self.nome = nome
        self.dt_nascimento = dt_nascimento

  def calcula_idade(self):
        #Cálculo da idade
        #...
        pass

joe = Pessoa("Joe", "01-01-1900")
```

Example - Inheritance (cont.)

```
class Contribuinte(Pessoa):
  def __init__(self, nome, dt_nascimento, cpf):
        super().__init__(nome, dt_nascimento)
        self.cpf = cpf

joe = Contribuinte("Joe", "01-01-1900","111.111.111-11")
```

Example - Composition

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

class Poligono:
        vertice1 = Ponto(1,2)
        vertice2 = Ponto(-1,-5)
        vertice3 = Ponto(8,21)
        vertice4 = Ponto(13,17)
```

Thank You!

Next: 2.6 - Regular Expressions and File Manipulation