Object Oriented Programming with Python

Gramsci Hermozo

Session 09

Content

- Lets complete our game
- Inheritance
- Polymorphism

Complete the game

What is pending?

- Analyze functions that could be moved into Physics Lib
- Analyze to use constants instead of use “magic” numbers
- Add “shoot” logic to make the ball moved
- Calculate ball path

Game

If you want to made some changes

- All our classes should inherit from “object”
- Change the ball for some image
- Change the background for an image
- Try to change the line for an arrow
- Add a console to show the calculations in real time

Inheritance

Basic inheritance

In python is based on similar ideas used in other object oriented languages like Java, C#, etc. A new class can be derived from a existing class as follows.

```
class BaseClass(object):
  pass

class DerivedClass(BaseClass):
  pass
```

Inheritance

Add constructor function

```
# __init__() function
class Person(object):
  def __init__(self, name, lastname):
    self.name = name
    self.lastname = lastname

class Student(Person):
  def __init__(self, name, lastname, grade, id):
    Person.__init__(name, lastname)
    self.grade = grade
    self.id = id
```

Inheritance

Use the super() function

super() function could be used to call fathers funcitons

```
class Person(object):
  def __init__(self, name, lastname):
      self.name = name
      self.lastname = lastname

class Student(Person):
  def __init__(self, name, lastname, grade, id):
      super().__init__(name, lastname)
      self.grade = grade
      self.id = id
```

Inheritance

Abstract Classes

By default Python does not provide abstract classes. Python had/comes with a module that provides the base for defining Abstract Base classes.

```
# Firts thing that need to import to use abstract classes
from abc import ABC

class MyABC(ABC):
  pass
```

Inheritance

Multiple Inheritance

A class can be derived from more than one base class in python, similar to c++.

```
class Base1:
  pass

class Base2:
  pass

class MultiDerived(Base1, Base2):
  pass
```

Polymorphism

What it's?

Is the ability to perform an action on an object regardless of its type. This is generally implemented by creating base class and having two or more subclasses that all implement methods with the same signature.

```
class Shape(object):
  def calculate_area(self):
    raise NotImplemented

class Square(Shape):
  def calculate_area(self):
    pass

class Triangle(Shape):
  def calculate_area(self):
    pass
```

Polymorphism

Overloading

```
# functions that could be overload
__add__(self, other) # a1 + a2
__sub__(self, other) # a1 - a2
__mul__(self, other) # a1 * a2
# more operators
__int__(self) # int(a1)
```