Programando con Python

POO

POO introducción

- ¿Qué aprenderemos? (a la manera Python) :
  - Encapsulación
  - Abstracción de datos
  - Polimorfismo
  - Herencia

POO introducción

- **Clase**: El esquema o plano de construcción de donde se originan los objetos.
- En Python, todo es un objeto
- Se usa la función type() para determinar que clase origina un objeto
- Clase mínina:

```
class Robot:
    pass
```

POO introducción

- **Atributos**: habilidad específica o características que algo o alguien posee.
- Suele llamársele como propiedades
- En Python propiedades y atributos son algo diferentes
- Aunque los atributos se definen internamente en las clases. en Python, los atributos se puede definir dinámicamente
- Internamente en la clase se pueden verificar con __dict__

POO introducción

- **Métodos**: en Python son esencialmente funciones.
- Definen el comportamiento de los objetos
- Aunque pueden ser definidos fuera de la clase, no es una buena práctica
- El primer parámetro es usado como referencia de la instancia llamada. Es nombrado *self*
- *self* ¡no es una palabra reservada de Python!, usuarios de C++ o Java lo pueden llamar *this*, pero puede ser contraproducente.

POO introducción

- **El método __init__**: es llamado inmediata y automáticamente después que una instancia ha sido creada.
- Ejm:

```
Class A:
    def __init(self)__:
        print("¡¡__init__ ha sido ejecutado!!")
```

POO introducción

```
def __init__(self, radius):
        self.radius = radius
```

- El método **__init__** es conocido como el constructor
- Es invocado cada vez que un objeto es creado en memoria.
- Aparte del parámetro self, se pueden agregar tantos parámetros como se necesite para inicializar un objeto.
- Definir el constructor no es requerido y si no se hace, Python proporciona uno vacío: __init__()

POO introducción

- Definir una clase (ejemplo):

```
import math

class Circle:

    def __init__(self, radius):
        self.radius = radius

    def get_area(self):
        return math.pi * self.radius ** 2

    def get_perimeter(self):
        return 2 * math.pi * self.radius
```

- En Python el parámetro **self** es requerido en cada método y refiere al objeto que invocó tal método
- Al llamar el método no se necesita pasar ningún valor a self.
- Dentro de la clase se usa self, para tener acceso a los atributos y métodos propios del objeto

POO introducción

- **Conceptos:**
- **Encapsulación**: Ocultar o proteger los datos, de manera que solo puedan se accedidos usando funciones especiales, es decir los métodos
- **Ocultamiento de información**: principio de OOP por medio del cual los datos no puede ser cambiada accidentalmente.
- **Abstracción de datos:** Los datos están ocultos y se usó la encapsulación, es decir:

  Abstracción de datos = Encapsulación + Ocultamiento

POO introducción

- **Abstracción de datos:**
- **Métodos getter**: permiten obtener o tener acceso a los valores de los atributos, no cambian el valor del atributo.
- **Métodos setter**: son usados para cambiar los valores de los atributos.

POO introducción

- **Ejemplo:**

```
class Robot:
    def __init(self, name= None):
        self.name = name
    def decir_hola(self):
        if self.name:
            print("Hola, yo soy " + self.name)
        else:
            print("Hola, soy un robot sin nombre")
    def get_name(self):
        return self.name
    def set_name(self, name):
        self.name = name
```

POO introducción

- **El Zen de Python:**
  - se invoca usando import this
  - indica que: “Debe haber una, y preferiblemente solo una, forma obvia de hacer algo”
  - Ya lo trataremos más adelante

POO introducción

- **Los métodos __str__ y __repr__**
- Si son definidos en la clase son usados para obtener una representación personalizada del objeto.
- Si solo se define __str__, __repr__ retornará el valor predefinido
- Si solo se define __repr__, funcionará para llamadas de __repr__ y de __str__
- Considere siempre usar __str__, ya que __repr__ puede que no funcione para objetos diferentes de string, se debe cumplir:

```
obj == eval(repr(obj))
```

- __repr__ se usa para representación interna, __str__ para el usuario final

POO introducción

- **Ejemplo:**

```
class Robot:
    def __init(self, name, build_year):
        self.name = name
        self.build_year = build_year
    def __rpre__(self):
        return "Robot(\"" + self.name + "\", "
               + str(self.build_year) + ")"
    def __str__(self):
        return "Name: " + self.name + ", Build Year: "
               + str(self.build_year)
```

POO introducción

- **Atributos Public, Protected y Private**
- In POO significan:
  - Private, sólo puede ser usado por el propietario
  - Protected, úselo a su propio riesgo, solo por alguien que escriba una subclase
  - Public, puede ser usado sin restricción
- En Python, se pueden definir así:
  - name , Public
  - _name, Protected,
  - __name, Private

  (ver el ejemplo reescrito)

POO introducción

- **Atributos de clase y de instancia**
- In POO significan:
  - de instancia: son particulares a cada instancia
  - de clase: pertenecen a la clase
- En Python, atributo de clase:

```
Class A:
    a = "atributo de clase"

x = A()
y = A()
x.a
```

- OJO: Si se desean cambiar se debe hacer con el NombreClase.atributo, si se hace con un objeto, se crea un atributo de instancia.
(ver ejemplo)

POO introducción

- **Métodos estáticos**
- In POO significan:
  - se acceden desde la propia clase, no de una instancia
- En Python:

```
Class Robot:
    __counter = 0
    def __init(self):
        type(self).__counter += 1

    @staticmethod       # <- ojo con el decorador
    def RobotInstances():
        return Robot.__counter
```

- Simplemente no llevan *self*

POO introducción

- **Métodos de clase**
- Son como métodos estáticos, pero están atados a la clase, la que se pasa como referencia
- En Python:

```
Class Robot:
    __counter = 0
    def __init(self):
        type(self).__counter += 1

    @classmethod
    def RobotInstances(cls):
        return cls, Robot.__counter
```

POO introducción

- **Métodos de clase, cuando usarlos:**
- En la definición de métodos llamados factory, no se tratará el tema de patrones de diseño
- En conjunto con métodos estáticos, que deben llamar a otros métodos estáticos.

  (ver ejemplo)
- Se volverán a tratar mas adelante en herencia.

POO introducción

- **Uso de propiedades:**
- Cuando se desee usar la forma de acceso objeto.atributo, pero respetando el principio de encapsulamiento.
- Se eliminan los prefijos get_ y set_ y se sustituyen por @property (para el get_) y @atributo.setter por el setter, cada método simplemente lleva el nombre del atributo.

POO introducción

- **Uso de propiedades (ejemplo)**

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

    @property
    def x(self):
        return self.__x

    @x.setter
    def x(self, x):
        if x < 0:
            self.__x = 0
        elif x > 1000:
            self.__x = 1000
        else:
            self.__x = x
```

Fin parte 1