Algorítmica y Programación

Enero - Mayo 2020

Dr. Iván S. Razo Zapata
(ivan.razo@itam.mx)

ITAM

Programación Orientada a Objetos

Temas para esta sesión

- **Métodos especiales**
- **Ejercicios**

Métodos especiales

Métodos especiales

- **__str__(self)**
  - Método llamado para crear una cadena de texto que represente a nuestro objeto. Se utiliza cuando usamos **print** para mostrar nuestro objeto o cuando usamos la función **str**(obj) para crear una cadena a partir de nuestro objeto
- **__del__(self)**
  - Método llamado cuando el objeto va a ser borrado. También llamado destructor, se utiliza para realizar tareas de limpieza

Métodos especiales

```
class personaFisica(empleado,ciudadano):
    __RFC = ''
    def __init__(self,val1):
        self.__RFC = val1
        self.setNombre("Bob")

    def getRFC(self):
        return self.__RFC

    def setRFC(self,val1):
        self.__RFC = val1

    def __str__(self):
        return 'Persona : ' + self.getNombre() + ' RFC: ' + self.getRFC()

    def __del__(self):
        print("Eliminando ", self.getNombre())
```

Métodos especiales

- **__call__(self)**
    - Permite que un objeto sea llamado como una función (con o sin parámetros)

Métodos especiales

$$y = c_0 + c_1 x$$

Métodos especiales

```
import numpy as np
import matplotlib.pyplot as plt


class Linea():

    def __init__(self):
        self.__c0 = 1
        self.__c1 = 2

    def __call__(self, x):
        return self.__c0 + self.__c1 * x

    def tabla(self, inicio, final, n):
        """Regresa tabla con n puntos para L <= x <= R."""
        T = []
        # np.linspace
        # Returns evenly spaced numbers over a specified interval.
        for x in np.linspace(inicio, final, n):
            y = self(x)
            T.append([x,y])
        return np.asarray(T)
```

Métodos especiales, herencia y polimorfismo

$$y = c_0 + c_1 x + c_2 x^2$$

Métodos especiales, herencia y polimorfismo

```
l1 = Linea()
puntos = l1.tabla(1,10,5)
X = puntos[:,0]
Y = puntos[:,1]
plt.plot(X,Y) # graficando puntos
plt.show()

c1 = Curva()
puntos2 = c1.tabla(10,100,5)
X2 = puntos2[:,0]
Y2 = puntos2[:,1]
plt.plot(X2,Y2) # graficando puntos
plt.show()
```

Métodos especiales, herencia y polimorfismo

- Revisando herencia

```
In [69]: isinstance(l1, Linea)
Out[69]: True

In [70]: isinstance(c1, Curva)
Out[70]: True

In [71]: isinstance(l1, Curva)
Out[71]: False

In [72]: isinstance(c1, Linea)
Out[72]: True

In [73]: isinstance(c1, Curva)
Out[73]: True
```

ITAM

Métodos especiales, herencia y polimorfismo

- Accediendo a atributos especiales

```
In [78]: l1.__class__
Out[78]: __main__.Linea
```

```
In [79]: c1.__class__
Out[79]: __main__.Curva

In [80]: l1.__class__.__name__
Out[80]: 'Linea'

In [81]: c1.__class__.__name__
Out[81]: 'Curva'
```

ITAM

Métodos especiales, herencia y polimorfismo

- Accediendo a atributos especiales

```
In [82]: c1.__class__.__name__ == Curva
Out[82]: False

In [83]: c1.__class__ == Curva
Out[83]: True

In [84]: l1.__class__.__name__ == Linea
Out[84]: False

In [85]: l1.__class__ == Linea
Out[85]: True
```

ITAM

Métodos especiales, herencia y polimorfismo

- Accediendo a atributos especiales

```
In [86]: l1.__class__.__name__ == 'Linea'
Out[86]: True

In [87]: l1.__class__ == Linea
Out[87]: True

In [88]: c1.__class__.__name__ == 'Curva'
Out[88]: True

In [89]: c1.__class__ == Curva
Out[89]: True
```

Ejercicios

Idea general

Three essential components of computer simulation

Initialize. You will need to set up the initial values for all the state variables of the system.

Observe. You will need to define how you are going to monitor the state of the system. This could be done by just printing out some variables, collecting measurements in a list structure, or visualizing the state of the system.

Update. You will need to define how you are going to update the values of those state variables in every time step. This part will be defined as a function, and it will be executed repeatedly.

Objetivo principal

- Simular el sistema discreto para 40 pasos
- Valores dados por el usuario: **a** y x0

$$x_t = ax_{t-1}$$

```python
class sistemaD:
    __a = 0
    __x = 0

    def __init__(self,val1,val2):
        self.__a = val1
        self.__x = val2

    def observa(self):
        return self.__x

    def actualiza(self):
        self.__x = self.__a * self.__x

# Inicialización
sistema1 = sistemaD(1.01,1)

for x in range(1,41):
    # Observación
    print(sistema1.observa())
    # Actualización
    sistema1.actualiza()
```