

Sesión 6

Programación Orientada a Objetos

CTIC - UNI

¿Qué es una clase?

 Si ha llevado algún curso en ciencias de la computación anteriormente, es probable que haya escuchado el término programación orientada a objetos (POO).

• ¿Qué es la POO y por qué debería importarme?



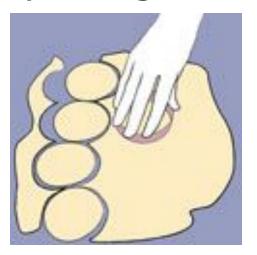
Respuesta corta

- La respuesta corta es que la programación orientada a objetos es una forma de pensar acerca de "Objetos" en un programa (como variables, funciones, etc.)
- Un programa se convierte menos en una lista de instrucciones y más en un conjunto de objetos y cómo interactúan.

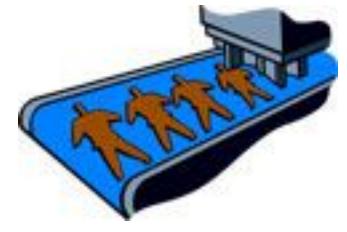


Clases versus instancia

- Una de las cosas más difíciles de conseguir entender es qué es una clase y qué es una instancia de una clase.
- La analogía del molde o cortador de galletas y una galleta.









Plantilla vs ejemplar

- El molde es una plantilla para estampar galletas, la galleta es lo que se hace cada vez que se usa el cortador.
- Una plantilla se puede utilizar para hacer un número infinito de galletas, cada una igual que la otra.



Lo mismo en POO

- Define una clase como una forma de generar nuevas instancias de esa clase.
- Tanto las instancias como las clases son en sí mismas objetos
- la estructura de una instancia comienza igual, según lo dictado por la clase.
- Las instancias responden a los mensajes definidos como parte de la clase.



¿Por qué una clase?

- Hacemos clases porque necesitamos tipos de datos más complejos y definidos por el usuario para construir instancias que podamos usar.
- Cada clase tiene potencialmente dos aspectos:
 - los datos (tipos, número, nombres) que puede contener cada instancia.
 - los mensajes a los que cada instancia puede responder.



Una Primera Clase



Nombres de clase estándar

La forma estándar de nombrar una clase en Python se llama Palabras Capitales:

- •Cada palabra de una clase comienza con una letra mayúscula.
- sin subrayados
- •a veces llamado Notación Camello
- •facilita el reconocimiento de una clase



POO, Definiendo una Clase

- Python fue diseñado como un lenguaje procedural
 - POO existe y trabaja bien.
 - Java probablemente hace las clases mejor que Python
- Declaración de una clase:

class nombre: instrucciones



Campos

nombre = valor

– Ejemplo:

```
class Punto:
    x = 0
    y = 0

# main
p1 = Punto()
p1.x = 2
p1.y = -5
```

point.py

```
1 class Point:
2 x = 0
y = 0
```

- Puede ser declarada directamente dentro de la clase (como se muestra aquí) o en constructores (más común)
- Python realmente no tiene campos de encapsulación o privados.



Probando una Clase

importando una clase

Los programas cliente deben importar las clases que usan.

```
#main
pl = Point()
pl.x = 7
pl.y = -3

#Los objetos Python son dinamicos
#Puede agregar campos en cualquier momento
pl.name="Victor Melchor"
```



Métodos de los Objetos

```
def name(self, parametro, ..., parametro):
   instrucciones
```

- self debe ser el primer parámetro para cualquier método del objeto
 - representa el "parametro implicito" (similar a this en Java)
- Los objetos deben acceder a los campos a través de la referencia self

```
class Point:
    def trasladar(self, dx, dy):
        self.x += dx
        self.y += dy
```



Parámetro "Implicito" (self)

• Java: this, implicito

• Python: self, explicito

```
def trasladar(self, dx, dy):
    self.x += dx
    self.y += dy
```

Ejercicio: Escriba los métodos distancia, set_ubicacion,
 y distancia del origen.



Respuesta al Ejercicio

point.py

```
from math import *
   class Point:
        x = 0
        v = 0
7
8
        def set location(self, x, y):
            sel\overline{f}.x = x
            self.y = y
10
11
        def distance from origin(self):
12
            return sqrt(self.x * self.x + self.y * self.y)
13
14
        def distance(self, other):
15
            dx = self.x - other.x
16
            dy = self.y - other.y
17
            return sqrt(dx * dx + dy * dy)
```

Llamada a Métodos

- Un cliente puede llamar a los métodos de un objeto de dos maneras:
 - (el valor de self puede ser un parámetro implicito o explicito)
 - 1) objeto.metodo(parametros)

O

- 2) Clase.metodo (objeto, parametros)
- Ejemplo:

```
p = Punto(3, -4)
p.trasladar(1, 5)
Punto.trasladar(p, 1, 5)
```



método versus función

 Un método y una función están estrechamente relacionados. Ambos son "pequeños programas" que tienen parámetros, realizan algunas operaciones y (potencialmente) devuelven un valor.

• La principal diferencia es que los métodos son funciones vinculadas a un objeto en particular.



diferencia en llamadas

se llaman a las funciones, y también se llaman a los métodos en el contexto de un objeto:

•función:

```
hacer algo(param1)
```

•método:

```
un_objeto.hacer_algo(param1)
```

iEsto significa que el objeto al que llama el método siempre es implícitamente un parámetro!



diferencia en la definición

- los métodos se definen dentro del conjunto de una clase.
- los métodos siempre vinculan el primer parámetro de la definición al objeto que lo llamó.
- Este parámetro puede llamarse cualquier cosa, pero tradicionalmente se llama self

```
class MiClase(objeto):
    def mi_metodo(self,param1):
        instrucciones
```



más sobre self

- self es una variable importante. En cualquier método, está vinculado al objeto que llamó al método.
- mediante self podemos acceder a la instancia que llamó al método (y a todos sus atributos como resultado).



Constructor

- Cuando se define una clase, se realiza una función con el mismo nombre que la clase.
- Esta función se llama el constructor. Al llamarlo, puede crear una instancia de la clase
- Podemos afectar esta creación (más adelante), pero por defecto Python puede crear una instancia.



referencia punto

 podemos referirnos a los atributos de un objeto haciendo una referencia punto, de la forma:

objeto.atributo

- el atributo puede ser una variable o una función
- es parte del objeto, ya sea directamente o por ese objeto como parte de una clase.



ejemplos

```
print(mi_instancia.mi_val)
  imprime una variable asociada con el objeto
  mi_instancia
mi_instancia.mi_metodo()
  llama a un método asociado con el objeto
  mi instancia
```

se puede distinguir la variable o método, por los paréntesis al final de la referencia.



Cómo hacer un valor local de objeto

- Una vez que se crea un objeto, los datos se crean de la misma manera que en cualquier otra situación de Python, por asignación.
- Por lo tanto, cualquier objeto puede aumentarse agregando una variable

```
mi instancia.atributo = 'hola'
```



Constructores

- Cuando se define una clase, se realiza una función con el mismo nombre que la clase.
- Esta función se llama el constructor. Al llamarlo, puede crear una instancia de la clase
- Podemos afectar esta creación (más adelante), pero por defecto Python puede crear una instancia.



referencia punto

 podemos referirnos a los atributos de un objeto haciendo una referencia punto, de la forma:

- el atributo puede ser una variable o una función
- es parte del objeto, ya sea directamente o por ese objeto como parte de una clase.



Métodos estándar Python

Python proporciona una serie de métodos estándar que, si el diseñador de la clase proporciona, se pueden usar de forma normal "Python"

- •muchos de estos tienen el doble subrayado delante y detrás de su nombre
- •Al usar estos métodos, "encajamos" en el flujo normal de Python.



Método estándar: Constructor

 Se llama al constructor cuando se realiza una instancia, y proporciona al diseñador de la clase la oportunidad de configurar la instancia con variables, por asignación.



Llamando a un constructor

Como se mencionó, se llama a un constructor usando el nombre de la clase como una llamada de función (agregando () después del nombre de la clase)

•crea una nueva instancia utilizando el constructor de la clase Estudiante.



el constructor ___init___

- uno de los nombres de métodos especiales en una clase es el nombre del constructor, ___init___
- asignando valores en el constructor, cada instancia comenzará con las mismas variables.
- También puede pasar argumentos a un constructor a través de su método init



Constructores

```
def __init__ (self, parametro, ..., parametro):
    instrucciones
```

- un constructor es un método especial con el nombre init
- Ejemplo:

```
class Point:
    def __init___(self, x, y):
        self.x = x
        self.y = y
```

• Cómo podríamos construir un Punto() sin parámetros para obtener (0, 0)?



constructor Estudiante

- self está vinculado a la instancia predeterminada a medida que se realiza.
- Si queremos agregar un atributo a esa instancia, modificamos el atributo asociado con self.



ejemplo

```
s1 = Estudiante()
print(s1.apellidos_str)

s2 = Estudiante(ape='Python', nom='Monty')
print(s2.apellidos_str)

Python
```



toString y str

```
def __str__(self):
    return string
```

- equivalente a toString de Java (convierte objeto a string)
- invocado automáticamente cuando se llama a str o print

Ejercicio: Escriba un método __str__ para objetos Punto que retorne cadenas como "(3, -14)"

```
def __str__(self):
    return "(" + str(self.x) + ", " + str(self.y) + ")"
```



Clase Point Completa

point.py

```
from math import *
   class Point:
       def init (self, x, y):
            self.x = x
            self.v = v
       def distance from origin(self):
            return sqrt(self.x * self.x + self.y * self.y)
10
11
       def distance(self, other):
12
            dx = self.x - other.x
13
            dy = self.y - other.y
14
            return sqrt(dx * dx + dy * dy)
15
16
       def translate(self, dx, dy):
17
            self.x += dx
18
            self.y += dy
19
20
       def str (self):
21
            return "(" + str(self.x) + ", " + str(self.y) + ")"
```

Sobrecarga de Operadores

 sobrecarga de operadores: Puede definir funciones de modo que los operadores internos de Python puedan ser usados con su clase.

Ver también: http://docs.python.org/ref/customization.html

Operador	Método de Clase
I	neg(self, other)
+	pos(self, other)
*	mul(self, other)
/	truediv(self, other)

_	neg(self)
+	pos(self)

Operador	Método de Clase
==	eq(self, other)
!=	ne(self, other)
<	lt(self, other)
>	gt(self, other)
<=	le(self, other)
>=	ge(self, other)

Operadores Unitarios



Ejercicios

- Ejercicio: **Escribir una clase Fraccion para** representar números racionales como 1/2 y -3/8.
- Las fracciones deberían siempre ser almacenadas en forma reducida; por ejemplo, almacenar 4/12 como 1/3 y 6/-9 como -2/3.
 - Sugerencia: Una función MCD (máximo comun divisor) puede ayudar.
- Defina los métodos sumar y multiplicar que acepten otra Fraccion como un parametro y modifique la Fraccion existente por la suma/multiplicacion por este parametro.



Generando Excepciones

```
raise ExceptionType("mensaje")
```

- útil cuando el cliente usa su objeto incorrectamente
- tipos: ArithmeticError, AssertionError, IndexError, NameError, SyntaxError, TypeError, ValueError
- Ejemplo:

```
class CuentaBancaria:
    ...
    def deposit(self, monto):
        if monto < 0:
            raise ValueError("monto negativo")
        ...</pre>
```



Herencia

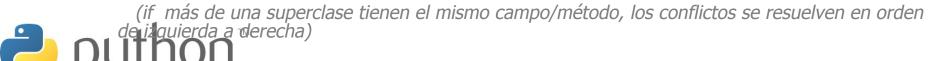
```
class name (superclase): instrucciones
```

– Ejemplo:

```
class Punto3D(Punto): # Punto3D extiende
Punto
    z = 0
```

• Python también soporta herencia múltiple

```
class nombre (superclase, ..., superclase): instrucciones
```



Llamada a Métodos de una Superclase

- metodos: clase.metodo (objeto, parametros)
- constructores: clase. init (parametros)

```
class Punto3D(Punto):
    z = 0
    def __init__(self, x, y, z):
        Punto.__init__(self, x, y)
        self.z = z

def trasladar(self, dx, dy, dz):
        Point.trasladar(self, dx, dy)
        self.z += dz
```

