Clases y Objetos en Python

JAIME ALBERTO GUZMAN LUNA
Universidad Nacional de Colombia
Medellín

Contenido

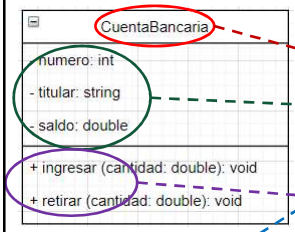
- Clases y Objetos:
  - Atributos
  - Métodos

Clases y Objetos

Visión general

Clases básica en Python

La declaración de la Clase, permite programar los atributos y métodos que tendrá, los cuales luego cada objeto de esta Clase utilizará en el programa.




```
class CuentaBancaria:

    pass
```

Atributos

Métodos

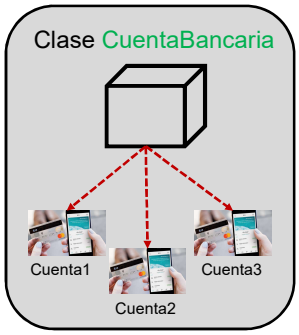
Se utiliza pass cuando la clase no tiene mas información, de lo contrario se omite

Python usa la función __init__ como "inicializador" de atributos

```
class CuentaBancaria:
    def __init__(self, numero, titular, saldo):
        self.numero = numero
        self.titular = titular
        self.saldo = saldo

    def ingresar(self, cantidad):
        self.saldo = self.saldo + cantidad

    def retirar(self, cantidad):
        if cantidad >= self.saldo :
            print ("saldo insuficiente")
        else:
            self.saldo = self.salto - cantidad
```

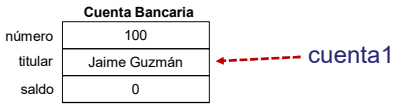
Python usa self para hacer referencia al objeto y a sus atributos. En los métodos siempre va al inicio

Los Objetos



Objeto: es una instancia de una clase.

- Declarar un objeto es declarar una referencia a un objeto. Los objetos en Python se definen así:
  - nombre_objeto = nombre_clase (parámetros)
- Crear un objeto igual que en java significa reservar espacio en memoria para sus atributos
  - cuenta1 = CuentaBancaria(100,“Jaime Guzman", 0)



- Los objetos en Python igual que Java siempre utilizan memoria dinámica

Los objetos

```
class CuentaBancaria:
    def __init__(self, numero, titular, saldo):
        self.numero = numero
        self.titular = titular
        self.saldo = saldo

    def ingresar(self, cantidad):
        self.saldo = self.saldo + cantidad

    def retirar(self, cantidad):
        if cantidad >= self.saldo :
            print ("saldo insuficiente")
        else:
            self.saldo = self.saldo - cantidad

if __name__ == "__main__":
    cuenta1 = CuentaBancaria(100,"jaime Guzman", 0)
    cuenta1.ingresar(5000)
    cuenta1.retirar(2000)
    print(cuenta1.saldo)
    print (cuenta1)
```

Ejecutando

Salida del programa

```
3000
<__main__.CuentaBancaria object at 0x0000027A4C17FFD0>
```

Cuando se imprimen, los objetos nos dicen de qué clase son y en qué dirección de memoria viven

- Nota: Las clases no pueden ejecutarse por si mismas.
  - Desarrollar sección de código donde crear y usar instancias de la clase (objetos)

Identidad y tipo

- Python es un lenguaje de programación orientado a objetos, y como tal, trata a todos los tipos de datos como objetos.
- Cada objeto viene identificado por su identidad y tipo:
  - Identidad- Id(): Nunca cambia e identifica de manera unívoca al objeto. El operador Id() nos permite saber si dos objetos son en realidad el mismo. Es decir, si dos variables hacen referencia al mismo.
  - Tipo- type(): Nos indica el tipo al que pertenece, como un float o una lista. La función type() nos indica el tipo de un determinado objeto. Es la clase a la que pertenece.

```
# un string
x = "Hello, world!"
print("Identidad:", id(x))
print("Tipo: ", type(x))
#Entero
y = 5
print("Identidad:", id(y))
print("Tipo: ", type(y))

#Definiendo la clase prueba
class Prueba():
    pass

#Objeto p de la clase prueba
p = Prueba()
print("Identidad:", id(p))
print("Tipo: ", type(p))
```

```
Identidad: 2815457253360
Tipo:  <class 'str'>
Identidad: 2815450704304
Tipo:  <class 'int'>
Identidad: 2815457271616
Tipo:  <class '__main__.Prueba'>
```

Mutabilidad

- Los diferentes tipos de Python u otros objetos en general, se clasifican en:
  - Mutables: Si permiten ser modificados una vez creados.
  - Inmutables: Si no permiten ser modificados una vez creados.

- Algunos tipos que son mutables:
  - Listas
  - Diccionarios
  - Sets
  - Y clases definidas por el usuario

- Algunos tipos que son inmutables:
  - Booleanos
  - Complejos
  - Enteros
  - Float
  - Cadenas
  - Tuplas
  - Bytes

- Una lista es mutable

```
# La asignación se puede realizar
l = [1, 2, 3]
print(id(l))
#Se puede modificar luego de creada
l[0] = 0
print(id(l))
```

```
2165281150208
2165281150208
```

- Una tupla es inmutable

```
# La asignación no se puede realizar
t = (1, 2, 3)
t[0] = 0
```

```
TypeError: 'tuple' object does not support item assignment
```

ERROR

- Un entero es inmutable

```
# Un entero es inmutable
x = 6
print(id(x)) # Tiene un id
x = x + 1
print(id(x)) # Cambia el id
```

```
2275542264272
2275542264304
```

Python no puede cambiar el 6 al ser el entero un tipo inmutable

Paso por valor/referencia

- Paso por valor
  - Los tipos inmutables son pasados por valor, por lo tanto dentro de la función se accede a una copia y no al valor original

```
def doblar_valor(numero):
    numero *= 3

n = 7
doblar_valor(n)
print(n)
```

7

- Paso por referencia
  - Los tipos mutables son pasados por referencia, como es el caso de las listas y los objetos de clases definidas por el usuario.

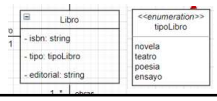
```
def doblar_valores(numeros):
    for i,n in enumerate(numeros):
        numeros[i] *= 2

ns = [5,10,50]
doblar_valores(ns)
print(ns)
```

[10, 20, 100]

Los atributos

Tipados de Atributos y su acceso

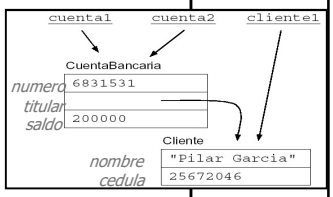
- **Tipado Dinámico**
  - En Python, al ser interpretado, es posible evitar declarar los tipos de las variables. Luego el ambiente hará los chequeos de tipos en tiempo de ejecución.
  - Por lo anterior si se le pasa en el parámetro del método `__init__` un `tipo primitivo` **`el atributo será`** un tipo inmutable (paso por valor), si se le pasa un objeto, **el atributo será** del tipo del objeto pasado (paso por referencia)
  - Enumerados: clases que representan un conjunto finito de valores. Asignar un enumerado a una variable, es por referencia.



La clase enumerado (se verá mas adelante en herencia)

```
class CuentaBancaria:
       def __init__(self, numero, titular, saldo):
        self.numero = numero
        self.titular = titular
        self.saldo = saldo

class Cliente:
    def __init__(self, nombre, cedula):
        self.nombre = nombre
        self.cedula = cedula


if __name__ == "__main__":
    cliente1 = Cliente ("jaime Guzman", 25672046)
    cuenta1 = CuentaBancaria(6831531,cliente1, 200000)
    cuenta2 = cuenta1
    print(cuenta1.saldo)
    print(cuenta2.saldo)
    cuenta2.titular.nombre = "Pilar Garcia"
    print (cuenta1.titular.nombre)
```

```
200000
200000
Pilar Garcia
```



Tipos de atributos

Atributos de clase: están vinculados a una clase; se comparten entre objetos de la misma clase.

Atributos de instancia: están vinculados a una instancia específica (objeto) de la clase; no se comparten entre objetos de la misma clase.

```
class CuentaCorriente:
    interes = 2.0 # Atributo de la clase
    def __init__(self, numero, interes):
        self.numero = numero # Atributo de instancia
        CuentaCorriente.interes = interes

if __name__ == "__main__":
    cuenta1 = CuentaCorriente(6831531, 0.7)
    cuenta2 = CuentaCorriente(1725462, 1.0)

    print (cuenta1.interes)
    print (CuentaCorriente.interes)
```

```
1.0
```

Acceso atributo de clase mediante objeto

Acceso atributo de clase mediante la clase

Sólo hay un ejemplar del atributo, de forma que todos los objetos acceden a la misma zona de almacenamiento

La memoria de una variable de clase se reserva al crear la clase en el intérprete luego de iniciar la ejecución

La memoria de una variable de instancia se reserva al crear el objeto en el intérprete

Que imprime?

```
class CuentaCorriente:
    interes = 2.0  #Atributo de la clase
    def __init__(self, numero, interes):
       self.numero = numero # Atributo de instancia
       CuentaCorriente.interes = interes

if __name__ == "__main__":
    cuenta1 = CuentaCorriente(6831531, 0.7)
    cuenta2 = CuentaCorriente(1725462, 1.0)
    print(cuenta1.interes)          (1)
    print(cuenta2.interes)          (2)
    print(CuentaCorriente.interes)  (3)
    cuenta1.interes = 0.5

    print(cuenta1.interes)          (4)
    print(cuenta2.interes)          (5)
    print(CuentaCorriente.interes)  (6)
```

Que imprime las líneas 1, 2 y 3

```
1.0
1.0
1.0
```

Que imprime las líneas 4, 5 y 6

```
0.5
1.0
1.0
```

Que paso?

Cuidados con los atributos de clase

- Básicamente al hacer cuenta1.interes = 0.5, le estamos creando a cuenta1 una variable de instancia que se llama interés.
  - Si un ***atributo de clase*** se le pretende asignar un nuevo valor accediendo a una instancia (como el ejemplo anterior), anulará el valor solo para esa instancia. Esto esencialmente anula el atributo de la clase y la convierte en una variable de instancia disponible, **intuitivamente, solo para esa instancia**. Es decir, prioriza el atributo de la instancia.
- **Sugerencia:** nunca modifique una variable de clase desde una instancia. En la mayoría de los casos el error está al modificar una variable de clase por fuera de la definición de la clase.

Constantes

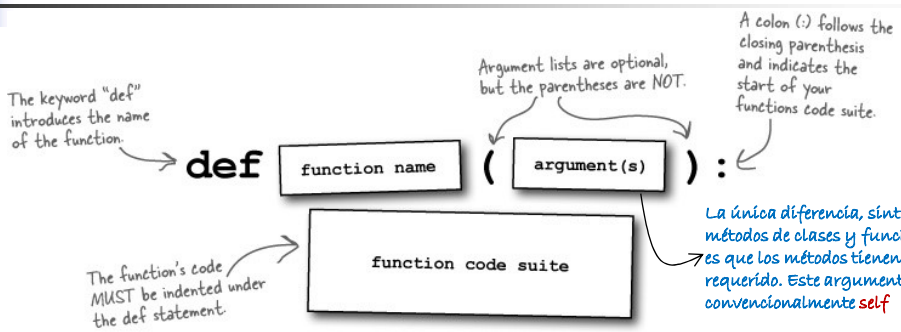
- Las constantes NO existen en python.
- La sugerencia es crear siempre el nombre de una variable o atributo que se desee ver como constante escribirlo en MAYÚSCULA (esto es mas indicativo para el desarrollador)

```
class Circulo:
   PI=3.1415
```

- Se recomienda igual para Java siempre colocar las constantes en MAYÚSCULA.

Los métodos

Estructura de un método en Python



La única diferencia, sintácticamente, entre métodos de clases y funciones fuera de clases es que los métodos tienen un argumento requerido. Este argumento se denomina convencionalmente self

- Instrucciones que pueden ir en un método:
  - Asignación
  - Estructuras condicionales y de iteración
  - Invocación a otros métodos
  - Creación de objetos

Métodos: Acceso a atributos y métodos

Los métodos referencian a los atributos de instancia de la clase haciendo uso del **self**

Los métodos referencian a los atributos de clase haciendo uso del **nombredelaclase**

Los métodos invocan otros métodos del mismo objeto con el **self**

Los métodos pueden invocar métodos de otro objeto con su referencia

```
cuenta1.transferencia(cuenta2, 1000)
    cuenta1.retirar (cantidad)
    cuenta2.ingresar (cantidad)
```

```
class CuentaCorriente:
    interes = 1.0
    def __init__(self, numero, titular, saldo, interes):
        self.numero = numero
        self.titular = titular
        self.saldo = saldo
        CuentaCorriente.interes = interes
    def ingresar(self, cantidad):
        self.saldo = self.saldo + cantidad

    def retirar(self, cantidad):
        if cantidad >= self.saldo :
            print ("saldo insuficiente")
        else:
            self.saldo = self.saldo - cantidad
    def transferencia (self, destino, cantidad):
        if cantidad <= self.saldo:
            self.retirar(cantidad)
            destino.ingresar(cantidad)
```

El argumento **self** es una referencia al objeto sobre el cual el método esta siendo ejecutado.

Tipos de métodos (1)

- **Método de instancia**
  - Es el que se invoca siempre sobre una instancia (objeto) de una clase (métodos comunes).
    - Reciben como primer parámetro de entrada a **self**, que hace referencia a la instancia que llama al método
- **Método de clase**
  - Son métodos que se ejecutan desde la propia clase.
  - Para indicarle a python que se trata de un método de clase debemos utilizar el decorador **@classmethod**
  - Reciben como argumento inicial **cls** (por convensión), que hace referencia a la clase. Por lo tanto, pueden acceder a la clase pero no a la instancia.
    - Los métodos de clase utilizan el prefijo **cls** para referirse a los atributos de la clase.

```
class CuentaBancaria:
    ncuentas = 0
    def __init__(self):
        self.numero = self.generarNumero()
    @classmethod
    def generarNumero(cls):
        cls.ncuentas += 1
        return cls.ncuentas
```

- Pueden ser invocados desde la clase o desde los objetos

Desde el objeto | Desde la clase

```
cuenta1 = CuentaBancaria()
cuenta1.numero = cuenta1.generarNumero()
cuenta1.numero = CuentaBancaria.generarNumero()
```

- Restricciones:
  - Pueden acceder a atributos y métodos de la misma clase
  - **No** pueden acceder a atributos ni métodos de instancia de la clase
  - Los métodos de instancia sí pueden acceder a variables y métodos de la clase

Tipos de métodos (2)

- **Método Estático**
  - Los métodos estáticos se podrían ver como funciones normales, con la salvedad de que van ligadas a una clase concreta.
  - Se pueden definir con el decorador **@staticmethod** y no aceptan como parámetro ni de la instancia (self) ni de la clase (cls)
    - Por lo anterior, no pueden modificar el estado ni de la clase ni de la instancia. Pero por supuesto pueden aceptar parámetros de entrada.
    - No se pueden llamar métodos estàticos ni métodos de clase directamente desde el objeto
  - Su objetivo principal es contener lógica perteneciente a la clase, pero esa lógica no debería tener ninguna necesidad de datos de instancia de clase específicos

```
class Ave (object):
    'Clase para las aves' #Descripción
    def __init__(self):
        pass

    def hablar (self, color):
        print ("Soy una ave de color %s" %color) #Usa format

    @staticmethod        #decorador staticmethod
    def funcion_volar_ave(tiene_alas, kms): #Define el metodo
        if tiene_alas == True:
            print ("El ave se fue volando %i kilometro" %kms)
        else:
            print ("Esta ave no puede volar")

Ave.funcion_volar_ave(True, 1) #Llama el método (función)
```

Método Estático

Se llama desde la clase

El uso de los **métodos estáticos** pueden resultar útil para indicar que un método no modificará el estado de la instancia ni de la clase.

Pregunta de repaso

- Dado el siguiente código: Identifique que imprime

```
class Alcance:
    x=3
    def __init__(self):
        pass
    def metodo1(self, x):
        x = 2
        return x*x*x;
    def metodo2(self):
        self.x = self.x - 5
    @classmethod
    def metodo3(cls):
        x = 5
        cls.x = x * cls.x
```

```
    def metodo4(self):
        x = 6
        Alcance.metodo3()
        print("x1 = ", x )
        print("x2 = ", self.x )
        self.metodo2()
        x = self.metodo1(self.x)
        print("x3 = ", x )
        print("x4 = ", self.x )
        self.x = self.metodo1(x)
        print("x5 = ", x )
        print("x6 = ", self.x );
        print("X7 = ", Alcance.x)

if __name__ == "__main__":
    alc1 = Alcance ()
    alc1.metodo4()
```

Solución

```
x1 =  6
x2 =  15
x3 =  8
x4 =  10
x5 =  8
x6 =  8
X7 =  15
```

Que paso?

Se creo una variable de instancia x en el método 2 que opaca a la variable x de clase: esta última sigue intacta con valor de 15 luego del método 3.

Encontrar los errores

```
class Arbol:
    pi=3.14 # constante
    altura_media=90
    def __init__(self,altura):
        self._altura=altura

    @staticmethod
    def PromedioAltura():
        altura_media_actual=AlturaAcumulada()
        altura=Altura()
        return altura_media_actual*altura/2

    def Altura():
        return self.altura

    @classmethod
    def AlturaAcumulada(cls):
        return cls.altura_media
```

Al ser una constante debe ir en mayúscula: PI=3,14

Falta Arbol.Altura…

No se puede acceder a un método de instancia desde uno estático

Falta el parámetro self, que va en todos los métodos de instancia

Actividad – 6 de Septiembre

- Equipos conformados
- Un solo integrante del equipo deberá adjuntar una presentación en powerpoint en Google Classroom con el siguiente contenido:

  - **Diapositiva 1:** Título proyecto y nombres integrantes
  - **Diapositiva 2:** Breve resumen sobre el proyecto
  - **Diapositiva 3:** Diagrama de clases del proyecto, mínimo debe tener 6 clases, máximo 8
  - **Diapositiva 4:** Lista de funcionalidades a implementar (5)

- Video de 2 minutos explicando su aplicación