



Python: Programación Orientada a Objetos

JESSE PADILLA AGUDELO

Ingeniero Electrónico


Licencia de la Presentación



Reconocimiento-No comercial-Compartir bajo la misma licencia 2.5 Colombia

Usted es libre de:





copiar, distribuir y comunicar públicamente la obra




hacer obras derivadas


Bajo las condiciones siguientes:



Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).



No comercial. No puede utilizar esta obra para fines comerciales.



Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Advertencia

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.
Esto es un resumen fácilmente legible del [texto legal](#) (la licencia completa).

Objetivos

- Dar un repaso general a la Programación Orientada a Objetos, comprendiendo de manera general conceptos tales como ***Objetos, Clases, Métodos, Herencia, polimorfismo, etc.***
- Conocer e implementar cada uno de los conceptos claves de la POO usando Python como lenguaje de programación.

Introducción

- Hoy en día la POO es un paradigma de programación fundamental para el desarrollo de cualquier tipo de aplicación, por eso hoy en día se la mayoría de los lenguajes de alto nivel (Como Java, C#, C++ entre otros) soportan este paradigma y buscan explotarlo al máximo, Python no es la excepción de hecho en Python trabajar con POO es muy fácil y agradable. Razón por la cual durante esta presentación analizaremos de forma detallada las virtudes de Python a la hora de desarrollar software bajo este paradigma.

Índice

1. Conceptos de Programación Orientada a Objetos
2. Python: Clases y Objetos
3. Python: Herencia
4. Python: Herencia Múltiple
5. Python: Polimorfismo
6. Python: Encapsulación
7. Python: Clases de Nuevo Estilo
8. Python: Métodos Especiales

POO – Programación Orientada a Objetos

- La programación orientada a objetos es un paradigma de programación que busca representar entidades u objetos agrupando datos y métodos que puedan describir sus características y comportamiento.

POO– Programación Orientada a Objetos

- La POO paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se modelan a través de clases y objetos, y en el que nuestro programa consiste en una serie de interacciones entre estos objetos.

Ventajas de la POO

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Facilita la creación de programas visuales.
- Construcción de prototipos
- Agiliza el desarrollo de software
- Facilita el trabajo en equipo
- Facilita el mantenimiento del software

Modelo Orientado a Objetos

- Para entender la POO vamos a revisar unos conceptos básicos:
 1. Objeto
 2. Clase
 3. Mensaje
 4. Método
 5. Interfaz
 6. Herencia

POO: El Objeto

- Un ***objeto*** es una unidad que engloba en sí mismo características y comportamiento necesarias para procesar información. Cada objeto contiene datos y funciones. Y un programa se construye como un conjunto de objetos, o como un único objeto.

POO: El Objeto

- Ejemplo
 - Carro BMW
 - Características
 - 4 Ruedas Micheline
 - Motor BMW
 - Caja de cambios de 7 Velocidades
 - Color Azul
 - 2 Espejos

POO: La Clase

- La ***clase*** es un modelo o prototipo que define las variables y métodos comunes a todos los objetos de cierta clase. También se puede decir que una clase es una plantilla genérica para un conjunto de objetos de similares características.

POO: La Clase

- Ejemplo:
 - Clase Vehículo
 - Numero de Ruedas
 - Tipo de Motor
 - Capacidad del Tanque de Gasolina
 - Numero de Velocidades de la Caja de Cambios
 - Color

POO: Mensaje

- El ***mensaje*** es el modo en que se comunican los objetos entre si.
- Ejemplo:
 - Cuando llamemos a una función de un objeto, diremos que estamos enviando un mensaje a ese objeto.

POO: Método

- Un ***Método*** es en POO se encarga de procesar los mensajes que lleguen a un objeto, un método no es otra cosa que una función o procedimiento perteneciente a un objeto.

POO: Interfaz

- Las clases y por lo tanto también los objetos, tienen partes públicas y partes privadas. Algunas veces llamaremos a la parte pública de un objeto su interfaz. Se trata de la única parte del objeto que es visible para el resto de los objetos, de modo que es lo único de lo que se dispone para comunicarse con ellos.

POO: Herencia

- La herencia es uno de los conceptos más cruciales en la POO. La herencia básicamente consiste en que una clase puede heredar sus variables y métodos a varias subclases. Esto significa que una subclase, aparte de los atributos y métodos propios, tiene incorporados los atributos y métodos heredados de la superclase.

Python: Clases y Objetos

- Python está completamente orientado a objetos: puede definir sus propias clases, heredar de las que usted defina o de las incorporadas en el lenguaje, e instanciar las clases que haya definido.
- En Python las clases se definen mediante la palabra reservada ***class*** seguida del nombre de la clase, dos puntos (:) y a continuación, indentado, el cuerpo de la clase.

Python: Clases y Objetos

- *Ejemplo:*

class Ejemplo:

#1

pass

#2

- En este ejemplo el nombre de la clase es Ejemplo y no hereda de otra clase. Por convención las clases empiezan en Mayúscula.
- Esta clase no define atributos pero no puede estar vacía para eso usamos la función ***pass***, equivalente en otros lenguajes a usar {}

POO en Python: el método `__init__`

```
1 import math
2
3 class complejo:
4     def __init__(self, real, imaginario):
5         self.real = real
6         self.img = imaginario
7     def abs(self):
8         print math.sqrt((self.real * self.real) + (self.img * self.img))
9
10
```

Las clases de Python no tienen constructores o destructores explícitos. Las clases de Python tienen algo similar a un constructor: el método `__init__`.

POO en Python: el método `__init__`

- `__init__` se llama inmediatamente tras crear una instancia de la clase.
- Sería tentador pero incorrecto denominar a esto el constructor de la clase. Es tentador porque parece igual a un constructor (por convención, `__init__` es el primer método definido para la clase), actúa como uno (es el primer pedazo de código que se ejecuta en una instancia de la clase recién creada), e incluso suena como uno.
- Incorrecto, porque el objeto ya ha sido construido para cuando se llama a `__init__`, y ya tiene una referencia válida a la nueva instancia de la clase. Pero `__init__` es lo más parecido a un constructor que va a encontrar en Python, y cumple el mismo papel.

POO en Python: el método `__init__`

- El primer atributo o variable de cada método de clase, incluido `__init__`, es siempre una referencia a la instancia actual de la clase.
- Por convención, este argumento siempre se denomina ***self***. En el método `__init__`, ***self*** se refiere al objeto recién creado; en otros métodos de la clase, se refiere a la instancia cuyo método ha sido llamado.
- Los métodos `__init__` pueden tomar cualquier cantidad de argumentos, e igual que las funciones, éstos pueden definirse con valores por defecto, haciéndoles opcionales para quien invoca.

POO en Python: el método `__init__`

- Por convención, el primer argumento de cualquier clase de Python (la referencia a la instancia) se denomina ***self***.
- Cumple el papel de la palabra reservada ***this*** en C++ o Java, pero ***self*** no es una palabra reservada en Python, sino una mera convención.
- Aunque necesita especificar ***self*** de forma explícita cuando define el método, *no se especifica al* invocar el método; Python lo añadirá de forma automática.

POO en Python: Instanciación de las Clases

- Crear un objeto o instanciar una clase en Python es muy sencillo. Para instanciar una clase, simplemente se invoca a la clase como si fuera una función, pasando los argumentos que defina el método `__init__`. El valor de retorno será el objeto recién creado.

POO en Python: Instanciación de las Clases

```
1 import math
2
3 class complejo:
4     def __init__(self, real, imaginario):
5         self.real = real
6         self.img = imaginario
7     def abs(self):
8         print math.sqrt((self.real * self.real) + (self.img * self.img))
9
10 def main():
11     numero = complejo(3,4) # Se crea al objeto y se inicializa haciendo
12                            # el llamado al metodo __init__ pasando los valores
13                            # real = 3 y imaginario = 4
14
15     numero.abs()           # Convocamos el metodo abs de la clase complejo
16
17     #raw_input()           # generamos una pausa en la aplicacion esperando
18                            # que se precione una tecla
19 if __name__ == '__main__':
20     main()
```

POO en Python: Borrar Objetos

- Crear instancias nuevas es sencillo, destruirlas lo es más. En general, no hay necesidad de liberar de forma explícita las instancias, porque se eliminan automáticamente cuando las variables a las que se asignan salen de ámbito. Son raras las pérdidas de memoria en Python.

POO con Python: Atributos de datos

- Python admite atributos de datos (llamados variables de instancia en Java, y variables miembro en C++).
- Para hacer referencia a este atributo desde código que esté fuera de la clase, debe calificarlo con el nombre de la instancia, **instancia.data**, de la misma manera que calificaría una función con el nombre de su módulo.
- Para hacer referencia a atributos de datos desde dentro de la clase, use **self** como calificador. Por convención, todos los atributos de datos se inicializan en el método **`__init__`**. Sin embargo, esto no es un requisito, ya que los atributos, al igual que las variables locales, comienzan a existir cuando se les asigna su primer valor.

POO en Python: Atributos de datos

```
1 import math
2
3 class complejo:
4     def __init__(self, real, imaginario):
5         self.real = real
6         self.img = imaginario
7     def abs(self):
8         print math.sqrt((self.real * self.real) + (self.img * self.img))
9
10 def main():
11     numero = complejo(3,4) # Se crea al objeto y se inicializa haciendo
12                           # el llamado al metodo __init__ pasando los valores
13                           # real = 3 y imaginario = 4
14
15     print numero.real      # variable de instancia real
16     print numero.img       # variable de instancia imaginaria
17
18     numero.abs()           # Convocamos el metodo abs de la clase complejo
19
20     #raw_input()           # generamos una pausa en la aplicacion esperando
21                           # que se precione una tecla
22 if __name__ == '__main__':
23     main()
```

POO en Python: Sobre Carga de Métodos

- C++ y Java admiten la sobrecarga de funciones por lista de argumentos, es decir una clase puede tener varios métodos con el mismo nombre, pero con argumentos en distinta cantidad, o de distinto tipo. Python no admite sobrecarga de funciones. Los métodos se definen sólo por su nombre y hay un único método por clase con un nombre dado.
- De manera que si una clase sucesora tiene un método `__init__`, siempre sustituye al método `__init__` de su clase padre, incluso si éste lo define con una lista de argumentos diferentes. Y se aplica lo mismo a cualquier otro método

POO en Python: Sobre Carga de Operadores

- La sobrecarga de operadores permite redefinir ciertos operadores, como "+" y "-", para usarlos con las clases que hemos definido. Se llama sobrecarga de operadores porque estamos reutilizando el mismo operador con un número de usos diferentes, y el compilador decide cómo usar ese operador dependiendo sobre qué opera.

POO en Python: Sobre Carga de Operadores

- `__add__(self, other)` -> Oper. Suma
- `__sub__(self, other)` -> Oper. Resta
- `__mul__(self, other)` -> Oper. Multiplicacion
- `__rmul__(self, other)` -> Oper. Multi. Por Escalar
- `__floordiv__(self, other)` -> Oper. division Redondeo
- `__mod__(self, other)` -> Oper. modulo
- `__divmod__(self, other)` -> Oper. division
- `__pow__(self, other[, modulo])` -> Oper. Potencia
- `__and__(self, other)` -> Oper. and
- `__xor__(self, other)` -> Oper. xor
- `__or__(self, other)` -> Oper. or

POO en Python: Sobre Carga de Operadores

```
1 import math
2
3 class complejo:
4     def __init__(self, real, imaginario):
5         self.real = real
6         self.img = imaginario
7
8     def abs(self):
9         print math.sqrt((self.real * self.real) + (self.img * self.img))
10
11     def __add__(self, otro):
12         return complejo(self.real + otro.real, self.img + otro.img)
13
14     def __sub__(self, otro):
15         return complejo(self.real - otro.real, self.img - otro.img)
16
17     def mostrar(self):
18         print self.real
19         print self.img
20
21 def main():
22     complejo1 = complejo(3,4)
23     complejo2 = complejo(3,4)
24     complejo3 = complejo1 + complejo2
25     complejo4 = complejo1 - complejo2
26     complejo3.mostrar()
27     complejo4.mostrar()
28
29 if __name__ == '__main__':
30     main()
```

Python: Herencia

- Una de las principales propiedades de las clases es la ***herencia***. Esta propiedad nos permite crear nuevas clases a partir de clases existentes, conservando las propiedades de la clase original y añadiendo otras nuevas.
- La nueva clase obtenida se conoce como clase derivada, y las clases a partir de las cuales se deriva, clases base. Además, cada clase derivada puede usarse como clase base para obtener una nueva clase derivada.

Python: Herencia

- Definición de una clase heredada en Python.

class Instrumento:

pass

class Guitarra(Instrumento):

pass

class Bajo(Instrumento):

pass

Herencia: El Método `__init__`

- Cuando creamos una clase derivada a partir de una clase padre y tenemos que la clase derivada proporciona o requiere su propio método `__init__` , este método de la clase derivada debe llamar explícitamente el método `__init__` de la clase base.

Herencia: El Método `__init__`

```
1 class Animal:
2     def __init__(self):
3         print "Animal creado"
4
5     def quiensoy(self):
6         print "Animal"
7
8     def comer(self):
9         print "Estoy comiendo"
10
11 class Perro(Animal):
12     def __init__(self):
13         Animal.__init__(self)
14         print "Perro Creado"
15
16     def quiensoy(self):
17         print "Perro"
18
19     def ladrar(self):
20         print "Woof Woof Woof Woof!"
21
22 def main():
23     d = Perro()
24     d.quiensoy()
25     d.comer()
26     d.ladrar()
27
28 if __name__ == '__main__':
29     main()
```

Python: Herencia Múltiple

```
class acuatico:  
    pass
```

```
class terrestre:  
    pass
```

```
class anfibio(acuatico, terrestre):  
    pass
```

Polimorfismo

- En programación orientada a objetos se denomina ***polimorfismo*** a la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa. (wikipedia)

Python: Polimorfismo

- El polimorfismo es el proceso de la utilización de un operador o función de diferentes formas para diferentes datos de entrada. En términos prácticos, el polimorfismo significa que si la clase B hereda de la clase A, no tiene que heredar todo acerca de la clase A, que puede hacer algunas de las cosas que hace una clase diferente

Python: Polimorfismo

- A diferencia de Java y C++ el ***Polimorfismo*** en Python no es de gran importancia, dada su naturaleza de lenguaje dinámico.
- En Python no existe la sobrecarga de métodos, el ultimo método que se declare reemplazara a los anteriores, aunque se puede conseguir este comportamiento usando métodos de # argumentos variable (*otros, **otros)

Encapsulación

- Para proteger a las variables de modificaciones no deseadas se introduce el concepto de ***encapsulación***. Los miembros de una clase se pueden dividir en públicos y privados. Los miembros públicos son aquellos a los que se puede acceder libremente desde fuera de la clase. Los miembros privados, por el contrario, solamente pueden ser accedidos por los métodos de la propia clase.

Encapsulación

- La ***Encapsulación*** se consigue en otros lenguajes de programación como Java y C++ utilizando modificadores de acceso que definen si cualquiera puede acceder a ese método o atributo.
- En estos lenguajes tenemos las modificaciones:
- ***public*** -> hace visible los métodos y atributos fuera de la clase.
- ***private*** -> hace que los métodos y atributos solo sean accesibles por métodos dentro de la clase.

Python: Encapsulación

- En Python no existen los modificadores de acceso.
- El acceso a una atributo o a los métodos viene determinado por su nombre: si el nombre comienza con dos guiones bajos (y no termina también con dos guiones bajos) se trata de una atributo o método privada, si no es así estos son públicos.

Python: Encapsulación

```
1 class figura:
2     def __init__(self, lados = 0, longitud_lado = 0.0, apotema = 0.0):
3         self.lado = lados
4         self.long = longitud_lado
5         self.__apotema = apotema
6         self.__perimetro = self.lado * self.long
7     def __area(self):
8         return ((self.__apotema * self.__perimetro) / 2)
9     def imprimir(self):
10        a = self.__area()
11        print a
12
13 def main():
14     triangulo = figura(2,3,1.5)
15     print triangulo.lado
16     print triangulo.long
17
18     # Estas dos lineas me lanzara una excepcion, diciendo que los atributos no
19     # existen dado que son privados y solo se pueden acceder dentro de la clase
20     #print triangulo.__apotema
21     #print triangulo.__perimetro
22
23     # Al igual que los atributos se me presentara una excepcion dado que el
24     # metodo solo existe dentro de la clase
25     #triangulo.__area()
26
27     triangulo.imprimir()
28
29 if __name__ == '__main__':
30     main()
```

Python: Clases de Nuevo Estilo

- La ramas 2.x de Python tiene dos tipos de clases, las de estilo viejo y las de estilo nuevo conviviendo. Las de estilo viejo siguen exactamente el mismo modelo que cualquier programador de cualquier versión anterior debería conocer. Todas las características que se explicarán a continuación se aplican sólo a las clases de estilo nuevo. Eventualmente se dejarán de soportar las clases de estilo viejo, en Python 3.0

Python: Clases de Nuevo Estilo

- *¿Cómo se define una clase de estilo nuevo?* Se hace heredando de una clase existente. La mayoría de los tipos internos de Python, como enteros, listas, diccionarios, e incluso archivos son ahora clases de estilo nuevo. Hay además una clase de estilo nuevo llamada '**object**' que se convierte en la clase base para todos los tipos internos, de modo que si no queremos heredar de un nuevo tipo interno se puede heredar de este:

```
class MiClase(object):  
    def __init__(self):  
        pass
```

Preguntas



Referencias

- ***Python para todos***, Raúl González Duque
- ***Inmersión en Python***, MARK PILGRIM – FRANCISCO CALLEGO – RICARDO CÁRDENAS.
- ***Aprenda a Pensar Como un Programador con Python***, ALLEN DOWNEY - JEREY ELKNER – CHRIS MEYER

Enlaces

- www.python.org
- [http://es.wikipedia.org/wiki/Programaci%C3%B3n orientada a objetos](http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos)
- <http://www.python.org/doc/2.5.2/ref/numeric-types.html>
- <http://blog.rvburke.com/2006/11/22/programacion-orientada-a-objetos-en-python/>
- <http://www.freenetpages.co.uk/hp/alan.gauld/spanish/tutclass.htm>
- <http://juanjoalvarez.net/?q=cambiospython>

¡¡GRACIAS!!

