

Investigación sobre la Programación Orientada a Objetos en Python

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un 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.

Un objeto es una entidad que agrupa un estado y una funcionalidad relacionadas. El estado del objeto se define a través de variables llamadas atributos, mientras que la funcionalidad se modela a través de funciones a las que se les conoce con el nombre de métodos del objeto.

Python no ofrece soporte nativo para registros, sino para clases, un concepto más general y potente. Usaremos registros a través de un módulo especial que ofrece una clase cuyo comportamiento es el que cabe esperar de los registros

En Python las clases se definen mediante la palabra clave `class` seguida del nombre de la clase, dos puntos (`:`) y a continuación, indentado, el cuerpo de la clase. Como en el caso de las funciones, si la primera línea del cuerpo se trata de una cadena de texto, esta será la cadena de documentación de la clase o `docstring`.

El método `__init__`, con una doble barra baja al principio y final del nombre, se ejecuta justo después de crear un nuevo objeto a partir de la clase, proceso que se conoce con el nombre de instanciación. El método `__init__` sirve, como sugiere su nombre, para realizar cualquier proceso de inicialización que sea necesario.

Para crear un objeto se escribiría el nombre de la clase seguido de cualquier parámetro que sea necesario entre paréntesis. Estos parámetros son los que se pasarán al método `__init__`, que como decíamos es el método que se llama al instanciar la clase.

Hay tres conceptos que son básicos para cualquier lenguaje de programación orientado a objetos: el encapsulamiento, la herencia y el polimorfismo.

- **Herencia:**
En un lenguaje orientado a objetos cuando hacemos que una clase (subclase) herede de otra clase (superclase) estamos haciendo que la subclase contenga todos los atributos y métodos que tenía la superclase. No obstante al acto de heredar de una clase también se le llama a menudo "extender una clase".

Para indicar que una clase hereda de otra se coloca el nombre de la clase de la que se hereda entre paréntesis después del nombre de la clase.

- **Polimorfismo:**
La palabra polimorfismo, del griego *poly* *morphos* (varias formas), se refiere a la habilidad de objetos de distintas clases de responder al mismo mensaje. Esto se puede conseguir a través de la herencia: un objeto de una clase derivada es al mismo tiempo un objeto de la clase padre, de forma que allí donde se requiere un objeto de la clase padre también se puede utilizar uno de la clase hija.

Python, al ser de tipado dinámico, no impone restricciones a los tipos que se le pueden pasar a una función, por ejemplo, más allá de que el objeto se comporte como se espera: si se va a llamar a un método `f()` del objeto pasado como parámetro, por ejemplo, evidentemente el objeto tendrá que contar con ese

método. Por ese motivo, a diferencia de lenguajes de tipado estático como Java o C++, el polimorfismo en Python no es de gran importancia.

- Encapsulación:

La encapsulación se refiere a impedir el acceso a determinados métodos y atributos de los objetos estableciendo así qué puede utilizarse desde fuera de la clase.

En Python no existen los modificadores de acceso, y lo que se suele hacer es que el acceso a una variable o función 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 variable o función privada, en caso contrario es pública. Los métodos cuyo nombre comienza y termina con dos guiones bajos son métodos especiales que Python llama automáticamente bajo ciertas circunstancias.

Existen otros métodos con significados especiales, cuyos nombres siempre comienzan y terminan con dos guiones bajos. A continuación se listan algunos especialmente útiles.

`__init__(self, args)` Método llamado después de crear el objeto para realizar tareas de inicialización.

`__new__(cls, args)` Método exclusivo de las clases de nuevo estilo que se ejecuta antes que `__init__` y que se encarga de construir y devolver el objeto en sí. Es equivalente a los constructores de C++ o Java. Se trata de un método estático, es decir, que existe con independencia de las instancias de la clase: es un método de clase, no de objeto, y por lo tanto el primer parámetro no es `self`, sino la propia clase: `cls`.

`__del__(self)` Método llamado cuando el objeto va a ser borrado. También llamado destructor, se utiliza para realizar tareas de limpieza.

`__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.

`__cmp__(self, otro)` Método llamado cuando se utilizan los operadores de comparación para comprobar si nuestro objeto es menor, mayor o igual al objeto pasado como parámetro. Debe devolver un número negativo si nuestro objeto es menor, cero si son iguales, y un número positivo si nuestro objeto es mayor. Si este método no está definido y se intenta comparar el objeto mediante los operadores `<=`, `>` o `>=` se lanzará una excepción. Si se utilizan los operadores `==` o `!=` para comprobar si dos objetos son iguales, se comprueba si son el mismo objeto (si tienen el mismo id).

`__len__(self)` Método llamado para comprobar la longitud del objeto.

Bibliografía

González Duque, R. G. *Python para todos*. España.

Marzal, A., & Gracia, I. *Introducción a la programación con Python*. Universitat Jaume I.

Foundation., P. S. (s.f.). *Tutorial de Python*. Obtenido de
<http://docs.python.org.ar/tutorial/3/classes.html>