Programacion orientada a objetos - Parte 1

La Programacion orirntada a objetos (Object Oriented Programming, OOP), es un "paradigma de programación": esto es, una forma de crear código, una metodología de programación que permite modelar un objeto del mundo real en base a código.

La ventaja de la OOP es la capacidad de encerrar en una estructura de código un conjunto de funciones que pueden utilizarse nuevamente sin tener en consideración como fueron implementadas. Adicionalmente, se puede utilizar esta estructura para crear nuevas estructuras, reduciendo la cantidad de código que un programa puede tener.

La mala noticia es que, por razones teóricas, la terminología que acompaña a la OOP es oscura y compleja. Las buenas noticias es que ya a estas alturas, todos saben manipular objetos...

...¡PORQUE EN PYTHON TODO ES UN OBJETO!

Es decir cuando se escribe la siguiente instrucción:

```
num = 4
```

lo que esta sucediendo es la creación de un *objeto* tipo *int* (esta es la *clase* del objeto). En este caso se ha creado el objeto num y ha sido *instanciado* con el valor de 4 (esto es, se ha inicializado con el valor de 4). El valor numérico es una propiedad del objeto num (esto se conoce como *campo* o *field*) y este objeto tiene ciertas capacidades o funciones (llamadas *metodos*), como suma (+), resta (-), etc.

Lea el parrafo anterior un par de veces mas para que quede claro antes que todo se ponga muy complicado...

Otro ejemplo de un objeto en Python es una lista:

```
lista = [1, 2, 3, 4, 5]
```

Lo que aqui sucede es la creación del objeto lista clase list y se ha instanciado con una secuencia de valores. Internamente, una lista se inicializa de la siguiente manera:

```
lista = 1 -> 2 -> 3 -> 4 -> 5
```

Donde 1 es el primer valor de la lista, -> es un "puntero", un tipo de dato especial que almacena una dirección de memoria, direccion donde se aloja el número 2 (de alli el nombre puntero, porque apunta a un dato) y luego otro puntero que apunta al tercer elemento de la lista, etc.

No es necesario conocer estos detalles internos del funcionamieto de una lista (per, ¿ahora entiende porque cuando de hace *lista2 = lista1* ambas no solo tienen los mismos valores, sino que además cuando se modifica un valor en una lista en la otra también se modifica?): estos detalles estan escondidos dentras de la definición del objeto. Como programador solo es necesario saber que hay que colocar numeros encerrados entre [] y separarlos por comas. Toda esta información esta *encapsulada* dentro de la definición de la clase *list*.

Una vez establecido esto, se sabe que hay ciertas acciones que se pueden realizar con una lista, como:

```
lista.append(6)
iista.pop()
lista.reverse()
```

Estas son acciones que se ejecutan sobre el objeto *lista*: son los *métodos* disponibles para este objeto. Estos métodos son los que se observan cuando se hace:

```
dir(lista)
```

Entonces, para llamar a los métodos de una lista se utiliza la nomenclatura *objeto.metodo(parametros)*.

Si se crean dos listas:

```
lista1 = [1, 2, 3, 4]
lista2 = [10, 20, 30, 40]
```

Lo que se tiene son dos *instancias* diferentes del mismo objeto clase *list*. Sin embargo, ambos comparten los mismos aributos y los mismos métodos.

Si revisa los métodos de las clases num y list:

```
dir(2)
dir([1, 2])
```

Observara que uno de los primeros métodos disponibles en ambos casos es "__add__". Estos caracteres "__" al principio especifican que es un *método privado*, es decir, una acción sobre la que no se tiene acceso directo sino a a través de algun mecanismo. Esta función es la encargada de resolver la suma de enteros y listas; y como se sabe, hacen cosas muy diferentes: en el caso de enteros se suman los valores y en el caso de listas se concatenan los elementos de ambas listas en una sola. Esto es un ejemplo de *sobrecarga*, es decir, dos métodos son el mismo nombre que realizan diferentes acciones porque estan asociados a diferentes clases.

Toda esta información será de mucha utilidad cuando se trate de comprender la OOP. Familiaricese con esta información antes de pasar a la siguiente parte. La OOP suele ser complicada para muchos nuevos programadores porque es una vuelta completa en la forma de programar: se suele atacar los problemas de programación yendo directamente al diseñar un algoritmo que modele un proceso con variables y acciones; en la OOP se requiere diseñar una especie de tablero de juego donde se van a colocar las piezas y se les da a cada pieza su propia naturaleza para que empiecen a interactuar entre si. Manejarse corretamente en la OOP requiere tiempo y esfuerzo.

Pero sobre todo pensar de una forma diferente...