TEMA 7

Clases

Curso de Python

Alfredo Sánchez Sánchez
@alfsan04
www.educoteca.com

ÍNDICE

Programación orientada a objetos

Tras todo lo aprendido en los seis capítulos anteriores podemos empezar a hablar de la **programación orientada a objetos**. Este tema va a ser tratado únicamente en este capítulo y tiene una gran relevancia, pero al mismo tiempo requiere ciertos conocimientos de programación y no se puede llegar al verdadero potencial, aunque sí se puede vislumbrar y trataré de que lo consigas ver. Intenta a lo largo del capítulo entender cómo la programación orientada a objetos puede ser de ayuda para usarla en el futuro.

Comencemos, imagina que quieres hacer un juego de batallas en una tierra fantástica. En dicho juego hay un montón de cosas que tienes que crear: elfos, enanos, humanos, orcos, caballos, espadas, hachas… Pues bien, todas esas cosas que deberías crear para el juego son llamadas, en programación, **objetos**. Por lo tanto, a partir de ese momento, al ir creando el juego empezarás a enfocar tu programación a los objetos, es decir, necesitarás programar qué hará cada uno de los humanos (ya que cada uno será un objeto), cada elfo, cómo interactuarán, qué aspecto tendrán, que tamaño…

Pues bien, todo ello, así, de forma muy resumida, es la programación orientada a objetos. Dicha programación se enfoca a algo más que variables, listas y diccionarios, se enfoca a objetos completos que tienen sus propias variables, sus propias listas…

Ahora que entendemos qué es un objeto, vamos a centrarnos en ellos. Un objeto se puede describir (¿cómo es?) y puede hacer cosas (¿qué habilidades tiene?). Pongamos, siguiendo con el ejemplo, que quiero crear un elfo. El elfo tendrá un tamaño, un peso, una fuerza… es decir, **se puede describir o tiene atributos**. El elfo también podrá correr, saltar… es decir, **podrá realizar acciones**.

- En programación, a los atributos de un objeto los denominaremos **propiedades.**
- Así mismo, a las acciones que puede realizar un objeto las denominaremos **métodos.**

Además de todo lo anterior, los objetos se pueden agrupar por tener propiedades y métodos similares. Sigamos con el ejemplo: un elfo tendrá un nombre, una fuerza, una altura… pero un humano también tendrá un nombre, una fuerza y una altura. Por ello, a los objetos que comparten propiedades y métodos se les denominan **clases.**

Clases

Todos los objetos que comparten propiedades y métodos se pueden construir desde un mismo **modelo.** Imagina que quieres hacer un ejército de elfos, enanos y humanos y necesitas describir a todos los soldados de una forma un poco genérica, podrías decir que…

- Todos los soldados tienen una raza
- Todos los soldados tienen un nombre
- Todos los soldados poseen una fuerza
- Todos los soldados son capaces de correr a una velocidad
- Todos los soldados tienen un arma

Pues bien, a un modelo que representa a un conjunto de objetos se le denomina **clase.**

Recopilemos: una **clase** es un modelo que define a una serie de **objetos** similares que, simplificando, se puede decir que son cosas. De esta forma, podemos tener la clase soldado, que será un modelo para definir a una serie de soldados que tendrán una raza, una fuerza, podrán correr... También podríamos tener la clase piedra que tiene un tamaño y puede romper el dedo meñique de los soldados cuando van descalzos.

Vamos a trabajar con el ejemplo de los soldados ya que es habitual en la programación de videojuegos. Haremos un script que permita crear diferentes soldados que podrían ser usados en un hipotético juego.

Creando una clase

En Python, para crear una clase no hay más que indicarlo de la siguiente forma:

```
class Soldado:
```

Si te fijas, al definir la clase, he usado una palabra con la primera letra en mayúscula. Si bien no es obligatorio, en programación se suelen llamar a las clases con la primera letra en mayúscula para diferenciarlas de variables, listas o diccionarios.

Con la instrucción anterior estás creando un modelo o esquema de cómo va a ser cualquier objeto que sea de la clase *Soldado*. Luego podrás crear tantos objetos *Soldado* como quieras, pero al declarar una clase no creas ningún *Soldado*, sólo especificas cómo son esos objetos.

Para llamar a una clase sólo hay que almacenarla en una variable y añadir tras el nombre unos paréntesis:

```
#Creo mi primer soldado e indico que es de la clase Soldado
soldado1 = Soldado()
```

Es normal, al principio, tener dificultades para entender la diferencia entre una función y una clase. Una función es un trozo de código que se aloja bajo un nombre para poder ser usado en cualquier momento. Al llamar a una función, el programa reproduce su código y cuando termina deja de estar, aunque nos puede entregar valores que sí permanezcan. En cambio, una clase, al ser llamada, crea un objeto que puede ser almacenado y permanecerá listo para ser usado en cualquier momento.

Creando el constructor

Al crear nuestra clase *Soldado* la hemos dejado un poco huérfana de **propiedades y métodos**. Veamos cómo podemos definir ese esquema o modelo de *Soldado*. Al crear la clase lo primero que debemos crear es lo que se denomina **constructor**, que contendrá todo aquello que deba contener el objeto al ser creado. El constructor se ejecutará automáticamente al crear un objeto de la clase en la que se encuentra.

Imagina que te toca construir *Soldados* y tienes que crearlos en su forma más básica para ser entrenados y convertidos en verdaderas armas de guerra. En el constructor introduciremos todos esos atributos que deba tener cada soldado (raza, nombre, vida…).

El constructor hay que definirlo de la misma forma que definíamos las funciones hace unos capítulos. Para indicar que estamos definiendo el constructor debemos usar *__init__* (con doble barra baja antes y después de *init*):

```
class Soldado:
   #Definimos el constructor, es decir, la función __init__
   def __init__():
```

Si recuerdas, en las funciones podíamos pasar parámetros. En el caso de las **funciones que van dentro de una clase** (y el constructor es una de ellas) también podemos enviar parámetros, aunque no es obligatorio hacerlo. Eso sí, hay un parámetro que siempre debemos introducir: *self* o, en castellano, yo.

Como hemos dicho antes, el constructor se ejecutará nada más crear un objeto de la clase que lo contiene, no hace falta llamarlo para que se ejecute, como pasaba con las funciones.

Ahora que ya sabemos lo que es un constructor, vamos a ver cómo lo creamos de forma que contenga lo necesario para nuestros soldados. Al constructor que creará nuestros soldados le vamos a pasar dos parámetros: raza y nombre.

```
class Soldado:
   #Situamos los parámetros que queremos y el obligado self al principio
   def __init__(self, nombre, raza):
```

Si no lo tienes claro piensa que el objeto, al ser creado, necesita para darle sentido a su existencia ciertos parámetros básicos. Eso no quiere decir que no se pueda crear un objeto sin pasar ningún parámetro. Yo podría crear una clase *Soldado* en la que, al ser materializada como objeto (es decir, llamada) todo soldado tuviese la misma vida, nombre y raza, pero no tiene mucho sentido hacer un ejército de clones y ponerles a todos el mismo

nombre. El general iba a tener ciertos problemas en el campamento cuando llamase al centro de mando a Légolas y viniesen 50.000 Légolas.

Una vez tenemos los parámetros que vamos a pasar a la clase para crear un objeto toca programar qué hacemos con esos parámetros. En nuestro programa el parámetro *nombre* llegará desde una variable que contenga un nombre para el soldado. Lo mismo ocurrirá con el parámetro raza, aunque ambos podrían ponerse como strings directamente al crear el objeto *Soldado*.

Observa el siguiente código y trata de entenderlo con el párrafo que lo explica tras el mismo:

```
class Soldado:
   #Situamos los parámetros que queremos y el obligado self al principio
   def __init__(self, nombre, raza):
      self.nombre = nombre
```

Empecemos con el parámetro *nombre*. Para usar un parámetro y aplicárselo al objeto debemos usar, de nuevo, *self*. Vamos a indicar al objeto que, al ser creado, el parámetro que le llega llamado *nombre* lo tiene que almacenar en una variable propia llamada *nombre*.

A mí las primeras veces que me enfrenté a este tipo de estructura casi me explota la cabeza. Con el tiempo la normalizas y te parece de lo más razonable, pero es cierto que conviene pararse a explicar qué está pasando. Vamos a ver un esquema:

```
#Creamos un Elfo llamado Eldelbar
nombre = "Eldelbar"
raza = "Elfo"
primer_soldado = Soldado(nombre, raza)
```

Al llamar a *Soldado* primero se ejecuta la función *__init__( )*:

```
def __init__(self, nombre,raza)
   self.nombre = nombre
```

La cual está recibiendo los siguientes parámetros del programa:

```
def __init__(self, "Eldelbar","Elfo")
   self.nombre = "Eldelbar"
```

En la construcción de la clase decimos que al crear un objeto van a llegar dos parámetros, uno que llamamos *nombre* y otro que llamamos *raza*. Al crear un objeto, *nombre* tendrá un

valor (en nuestro ejemplo *"Eldelbar"*) que vamos a guardar en la variable *nombre* **propia del objeto** (que no tiene nada que ver con la variable *nombre* en la que almacenamos al principio el nombre *Eldelbar*.

Bueno, sigamos con nuestro constructor. Una vez tenemos el nombre para nuestro recién creado *primer_soldado* vamos a incorporar la raza también como variable interna del objeto:

```
class Soldado:
   def __init__(self, nombre, raza):
      self.nombre = nombre
      self.raza = raza
```

Como verás, el mecanismo es el mismo exactamente que con el nombre. Ya tenemos dos **variables propias del objeto**. Ahora vamos a ver cómo podemos acceder a ellas. Para acceder a cualquiera de las propiedades de una clase, tenemos que hacerlo con el nombre con el que se creó el objeto con el siguiente comando:

```
nombre_objeto.nombre
```

Vamos a ver cómo podríamos crear un soldado llamado Eldelbar e imprimir su nombre y su raza tras crearlo. En esta ocasión no vamos a pasar los parámetros como variables sino directamente escribiremos los string que correspondan al nombre y raza al usar la clase *Soldado*. Guarda el siguiente código en un script y ejecútalo desde Terminal o IDLE:

```
class Soldado:
   def __init__(self, nombre, raza):
      self.nombre = nombre
      self.raza = raza
#Creamos el primer objeto soldado
primer_soldado = Soldado("Eldelbar","Elfo")
#Imprimimos el nombre y raza de nuestro soldado
print(primer_soldado.nombre)
print(primer_soldado.raza)
```

Si todo ha ido correcto deberías haber recibido lo solicitado:

```
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 elfo.py
Eldelbar
Elfo
```

Ya tenemos unas nociones básicas de clases, ahora bien, el constructor sigue un poco huérfano. Vamos a incorporar, además de un nombre y una raza, una vida y una velocidad de movimientos, así como un coste de entrenamiento para incorporarlo a nuestro ejército.

El problema es que un elfo, un humano o un enano no se mueven igual ni tienen la misma vida (entiéndase vida como capacidad para sobrevivir en una refriega). Intenta entender el siguiente código antes de continuar:

```
class Soldado:
    def __init__(self,nombre,raza):
        self.nombre = nombre
        self.raza = raza
        if raza == "Elfo":
            self.vida = 80
            self.velocidad = 10
            self.coste = 200
        elif raza == "Humano":
            self.vida = 100
            self.velocidad = 7
            self.coste = 100
        elif raza == "Enano":
            self.vida = 120
            self.velocidad = 3
            self.coste = 180
```

Como verás, al crear el constructor hemos pasado dos parámetros que harán al objeto único, o al menos tendrá el nombre y raza que queramos, podrían ser iguales varios objetos si como usuarios le damos la misma raza y nombre. Así mismo, tendrá una serie de propiedades que serán iguales para todos los objetos que tengan una misma raza. Los elfos tendrán una vida, una velocidad y un coste diferente a los humanos, y los enanos diferente a elfos y humanos.

Esto pinta bastante bien. Seguro que has jugado a algún juego de estrategia de los antiguos, en los cuales tenías tres o cuatro tipos de tropas y venga a crear orcos, elfos, o cualquier fantástica raza. Fíjate que con esta pequeña introducción ya tendríamos para crear soldados e ir utílizándolos con sus valores internos.

Utilicemos el código anterior para crear un objeto *Soldado* con raza *Elfo* e imprimamos, además de su nombre y raza, su vida, velocidad y coste. Guarda el siguiente código como script y ejecútalo antes de seguir:

```
class Soldado:
    def __init__(self,nombre,raza):
```

```
        self.nombre = nombre
        self.raza = raza
        if raza == "Elfo":
            self.vida = 80
            self.velocidad = 10
            self.coste = 200
        elif raza == "Humano":
            self.vida = 100
            self.velocidad = 7
            self.coste = 100
        elif raza == "Enano":
            self.vida = 120
            self.velocidad = 3
            self.coste = 180
#Creamos el primer objeto soldado
primer_soldado = Soldado("Eldelbar","Elfo")
#Imprimimos el nombre y raza de nuestro soldado
print("Nombre:")
print(primer_soldado.nombre)
print("Raza:")
print(primer_soldado.raza)
print("Vida:")
print(primer_soldado.vida)
print("Velocidad:")
print(primer_soldado.velocidad)
print("Coste de entrenamiento:")
print(primer_soldado.coste)
#Veamos de qué tipo es la variable primer_soldado
print(type(primer_soldado))
```

Deberías haber obtenido lo siguiente:

```
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 elfo.py
Nombre:
Eldelbar
Raza:
Elfo
Vida:
80
Velocidad:
10
Coste de entrenamiento:
200
<class '__main__.Soldado'>
```

Si te fijas, al final del todo he impreso el tipo de la variable *mi_primer_soldado*, y Python nos indica que es una variable objeto de la clase *Soldado*. Es decir, para guardar los objetos que creamos también usamos variables (al fin y al cabo en algún sitio se tendrá que almacenar el objeto creado).

Creando métodos

Ya tenemos claro cómo se puede usar el constructor, ahora vamos a ver cómo podemos crear **métodos** para la clase. Por ejemplo, el soldado puede tener un arma, así que vamos a crear otra función con la misma estructura inicial que *__init__*, pero ahora ya podemos darle el nombre que nos parezca correcto:

```
class Soldado:
   def __init__(self,nombre,raza):
      #Todo el contenido ya creado para el constructor
   def arma(self, arma):
      self.arma = arma
```

Es importante incluir, en el constructor, la variable *arma* como variable propia del objeto, aunque inicialmente no le demos valor:

```
class Soldado:
    def __init__(self,nombre,raza):
        self.nombre = nombre
        self.raza = raza
        """Incluyo el arma como variable propia del objeto y valor
inicial vacío"""
        self.arma = ""
        if raza == "Elfo":
            self.vida = 80
            self.velocidad = 10
            self.coste = 200
        elif raza == "Humano":
            self.vida = 100
            self.velocidad = 7
            self.coste = 100
        elif raza == "Enano":
            self.vida = 120
            self.velocidad = 3
            self.coste = 180
```

Ahora, al llamar a nuestro objeto del tipo *Soldado* podríamos, posteriormente, darle un arma. Para ello sólo tendríamos que llamar a la función *arma* indicando que pertenece al objeto *Soldado*. Observa el siguiente código:

```
mi_primer_soldado = Soldado("Eldelbar","Elfo")
```

```
mi_primer_soldado.arma("Arco")
```

He creado un objeto *Soldado* llamado *mi_primer_soldado* y posteriormente he llamado a la función *arma* propia de la clase *Soldado* indicando por parámetros que el arma será un *"Arco"*.

Ahora podría crear todos los soldados que me diese la gana. Incorporaré la función *arma* a la clase *Soldado* y crearé unos cuantos soldados:

```
class Soldado:
    def __init__(self,nombre,raza):
        self.nombre = nombre
        self.raza = raza
        if raza == "Elfo":
            self.vida = 80
            self.velocidad = 10
            self.coste = 200
        elif raza == "Humano":
            self.vida = 100
            self.velocidad = 7
            self.coste = 100
        elif raza == "Enano":
            self.vida = 120
            self.velocidad = 3
            self.coste = 180
   def arma(self, arma):
      self.arma = arma
primer_soldado = Soldado("Eldelbar","Elfo")
mi_primer_soldado.arma("Arco")
#Creamos más soldados
segundo_soldado = Soldado("Bran","Humano")
segundo_soldado.arma("Espada")
tercer_soldado = Soldado("Glóin","Enano")
segundo_soldado.arma("Hacha")
```

Al ver el ejemplo anterior observarás que crear soldados no tiene misterio, no es más que ir creando variables y asignarles como valor la clase *Soldado* pasando por parámetros lo que necesita esa clase para materializarse. También podemos (o no) asignarle a cada soldado su arma con la función creada para la clase *Soldado* llamada *arma*.

El mundo de la programación orientada a objetos en Python es muy extenso. Podríamos ver cómo podemos utilizar una clase para construir otro tipo de clase que tiene las mismas propiedades de la primera y algunas más (a esto se le denomina **heredar de una clase**).

Así mismo, podríamos ver más métodos y cómo una clase puede tener dentro otra clase (por ejemplo la clase *Soldado* podría incluir dentro la clase *Armadura*). Pero realmente sería desviar la atención de lo básico y es preferible asentar un poco más cómo usar una clase.

Utilizando clases

Ya hemos visto cómo crear soldados, podríamos tener un enorme ejército, el problema es que deberíamos programar algo del estilo:

```
mi_primer_soldado = Soldado(...)
mi_segundo_soldado = Soldado(...)
mi_tercer_soldado = Soldado(...)
mi_cuarto_soldado = Soldado(...)
.
.
.
mi_quincuagesimotercer_soldado = Soldado(...)
.
.
.
mi_yoquesecuantosmillones_soldado = Soldado(...)
```

El código es árduo cuando queremos crear muchos objetos, por lo que tendríamos que buscar una forma de automatizar la creación de soldados. Podríamos usar un bucle, pero tenemos un problema: el objeto se crea almacenándolo en una variable y si hacemos algo así...

```
i = 1
while i<=10000:
    soldado = Soldado(...)
    i = i+1
```

... crearíamos 1 solo soldado, y no 10.000, ya que al crear un nuevo objeto y almacenarlo en la variable *soldado* pisaríamos (es decir, reemplazaríamos) el anterior objeto *Soldado*.

Es normal que surja este problema a la hora de programar si utilizamos clases, siempre hay un momento en que quieres crear muchos objetos de un mismo tipo y nos encontramos con que no es fácil automatizarlo, por ello hay que acudir a un ya conocido tipo de variable: **las listas**.

Las listas pueden contener cualquier tipo de variables, incluidas otras listas y, por suerte, **objetos.** De esta forma, yo puedo ir creando objetos y almacenándolos en listas. Además, en una lista no hace falta definir cuantos objetos introduciremos, ya que las listas son **mutables** y podemos ir cambiando su contenido.

Vamos a utilizar nuestro script generador de soldados ampliándolo para que el usuario pueda introducir tantos soldados como quiera. Recuerda que al crear un soldado hay que darle un nombre y una raza. Veamos la estructura (o algoritmo) inicial que necesita nuestro programa para generar las clases:

- Creamos una lista llamada *ejercito*
- Preguntamos al usuario cuántos soldados quiere crear
- Utilizamos un bucle que se repita tantas veces como soldados haya que crear
- En cada repetición del bucle preguntamos al usuario como quiere llamar al nuevo soldado y de qué raza será
- Creamos un nuevo objeto con el nombre y raza indicados y lo introducimos al final de nuestra lista
- Para demostrar que funciona imprimimos el nombre de cada uno de los objetos de la lista

Intenta, antes de seguir, generar ese script que cree tantos soldados como quiera el usuario e imprima los nombres dados a los soldados. Utiliza la clase creada anteriormente para este programa. Una vez lo consigas (o no) podrás ver cómo lo he hecho yo, pero no sigas leyendo sin intentarlo.

Así lo he hecho yo:

```
class Soldado:
    def __init__(self,nombre,raza):
        self.nombre = nombre
        self.raza = raza
        self.arma = ""
        if raza == "Elfo":
            self.vida = 80
            self.velocidad = 10
            self.coste = 200
        elif raza == "Humano":
            self.vida = 100
            self.velocidad = 7
            self.coste = 100
        elif raza == "Enano":
            self.vida = 120
            self.velocidad = 3
            self.coste = 180
    def arma(self, arma):
```

```
        self.arma = arma
#Creamos lista ejercito para almacenar los objetos soldado
ejercito = []
cantidad_soldados = int(input("¿Cuántos soldados quieres crear?: "))
#Recorremos la cantidad de soldados declarada
for i in range(cantidad_soldados):
    #Preguntamos nombre y raza
    nombre = input("Nombre del nuevo soldado: ")
    raza = input("Raza del nuevo soldado (Elfo/Humano/Enano): ")
    #Creamos el nuevo soldado
    soldado = Soldado(nombre,raza)
    #Guardamos el objeto soldado en la lista ejército
    ejercito.append(soldado)
for i in range(len(ejercito)):
    #Llamamos a cada objeto por su posición en la lista ejército
    print(ejercito[i].nombre)
```

Si te fijas, he llamado al *nombre* de cada objeto imprimiéndolo desde su ubicación en la lista. La sentencia *ejercito[i].nombre* sirve para ubicar cada objeto *Soldado* dentro de la lista *ejercito*.

De la misma forma que recorremos la lista para imprimir el nombre de cada objeto, podríamos recorrerla para utilizarlos con otros fines, veamos algunos ejemplos:

- Utilizar cada objeto de la lista en un ataque a un objeto enemigo.
- Utilizar un objeto aleatorio (random) de la lista para que se enfrente a un enemigo.
- Enfrentar a dos objetos aleatorios de nuestra lista.
- Ver el coste de entrenamiento de todos los objetos introducidos en la lista.
- Ver la vida total de nuestro ejército.
- Encontrar el objeto con mayor o menor vida de la lista si esta va cambiando durante un juego.

Como veréis, podríamos hacer múltiples cosas en un hipotético juego que introdujese nuestro script como parte de su funcionamiento.

Pero, para nuestros conocimientos actuales, es suficiente con este documento para entender de qué va esto de programar pensando en objetos y crear clases para poder generar objetos basados en una estructura común.

Instrucciones de código usadas

```
#Crear una clase
   class Clase:
#Utilizar una clase
   objeto = Clase()
#Definir el constructor de una clase
   def __init__(self,parametros):
#Usar una variable propia de un objeto
   objeto.variable
#Definir un método de una clase
   def nombre(parametros):
```

Reto

Utilizando el programa ya creado que genera soldados a petición del usuario, crea una variable que almacene un valor de **monedas** o **dinero** disponible y ve restando de dicha cantidad el coste de entrenamiento de cada soldado. El script debe informar si nos hemos quedado sin monedas para entrenar más soldados.