

Programación Orientada a Objetos - Parte I

1. Introducción

Un Programa Orientado a Objetos (POO) se basa en el diseño de un dominio que debemos representar usando un grupo de objetos de distintas clases que interactúan entre sí y que, en conjunto, consiguen que un programa implemente un propósito determinado.

En este paradigma de programación se intenta emular el funcionamiento de los objetos que nos rodean en la vida real.

En Python cualquier elemento del lenguaje pertenece a una clase y todas las clases tienen el mismo rango y se utilizan del mismo modo.

A continuación, se declaran varios objetos Python y con la función `type()` se muestra a qué clase pertenecen cada uno:

Ejercicio de Inducción: Pruebe las siguientes líneas de código y verifique los resultados presentados:

```
lenguaje = "Python"
print(type(lenguaje)) # class str=""

def mitad(x):
    return x/2

print(type(mitad)) # class function=""

valor = mitad(25)
print(type(valor)) # class float=""

import os
print(type(os)) # class module=""

lista = [1, 2, 3, 4, 5]
print(type(lista)) # class list=""
```

2. Clases, atributos y métodos

Las clases en este contexto permiten definir los atributos y el comportamiento, mediante métodos, de los objetos de un programa. Una clase es un prototipo que se utiliza para crear instancias individuales del mismo tipo de objeto.

Los atributos definen las características propias del objeto y modifican su estado. Es información asociado a las clases y a los objetos creados a partir de ellas.

De ello, se deducen los dos tipos de atributos o de variables existentes: variables de clase y variables de instancia (objetos).

Los métodos son bloques de código (o funciones) de una clase que se utilizan para definir el comportamiento de los objetos.

Tanto para acceder a los atributos como para llamar a los métodos se utiliza el método denominado de notación de punto que se basa en escribir el nombre del objeto o de la clase seguido de un punto y el nombre del atributo o del método con los argumentos que procedan: `clase.atributo`, `objeto.atributo`, `objeto.método([argumentos])`.

Un ejemplo característico de objeto Python donde se identifican fácilmente los métodos son las listas. Una lista es un objeto que permite contener una colección o secuencia de datos. Los datos de una lista deben ir separados por comas (,) y todo el conjunto entre corchetes. Una lista es una estructura mutable porque no sólo se puede acceder a los datos, además, es posible agregar nuevos elementos o suprimir aquellos que no sean necesarios. La clase lista (**List**) incorpora varios métodos para facilitar este trabajo.

Ejercicio de Inducción: Pruebe las siguientes líneas de código y verifique los resultados presentados:2

```
lista = ['c', 'a', 'b'] # Declara lista con tres elementos
lista.append('d') # Agrega elemento al final de lista con append()
lista.pop() # Borra último elemento de lista con método pop()
lista.sort() # Ordena la lista con el método sort()
print(lista) # ['a', 'b', 'c']
```

3. Características de la Programación Orientada a Objetos

Abstracción: Se refiere a que un elemento pueda aislarse del resto de elementos y de su contexto para centrar el interés en lo qué hace y no en cómo lo hace (caja negra).

Modularidad: Es la capacidad de dividir una aplicación en partes más pequeñas independientes y reutilizables llamadas módulos.

Encapsulamiento: Consiste en reunir todos los elementos posibles de una entidad al mismo nivel de abstracción para aumentar la cohesión, contando con la posibilidad de ocultar los atributos de un objeto (en Python, sólo se ocultan en apariencia).

Herencia: Se refiere a que una clase pueda heredar las características de una clase superior para obtener objetos similares. Se heredan tanto los atributos como los métodos. Éstos últimos pueden sobrescribirse para adaptarlos a las necesidades de la nueva clase. A la posibilidad de heredar atributos y métodos de varias clases se denomina *Herencia Múltiple*.

Polimorfismo: Es la posibilidad de identificar de la misma forma comportamientos similares asociados a objetos distintos. La idea es que se sigan siempre las mismas pautas aunque los objetos y los resultados sean otros.

4. Variables de Clases y Variables de Instancias

En lenguajes que crean objetos a partir de clases, un objeto es una instancia de una clase. Y de una misma clase se pueden mantener activas en un programa más de una instancia al mismo tiempo.

Una variable de clase es compartida por todas las instancias de una clase. Se definen dentro de la clase (después del encabezado de la clase) pero nunca dentro de un método. Este tipo de variables no se utilizan con tanta frecuencia como las variables de instancia.

Una variable de instancia se define dentro de un método y pertenece a un objeto determinado de la clase instanciada.

5. Crear clases

Una clase consta de dos partes: un encabezado que comienza con el término `class` seguido del nombre de la clase (en singular) y dos puntos (`:`) y un cuerpo donde se declaran los atributos y los métodos:

```
class NombreClase:
    'Texto para documentar la clase (opcional)'
    varclase1 = "variable de clase1"

    def nombremetodo1(self, var1):
        self.var1 = var1
```

```
def nombremetodo2(self):  
    self.var1 += 1
```

La documentación de una clase debe situarse después del encabezado y justo antes del lugar donde se declaran las variables y los métodos de la clase.

Desde cualquier lugar de un programa se puede acceder a la cadena de documentación de una clase accediendo al atributo especial: **NombreClase.__doc__**

Todo lo que se incluye en una clase es opcional. De hecho, la clase más elemental aunque no tenga mucha utilidad puede estar vacía:

```
class Clase:  
    pass
```

La declaración **pass** indica que no se ejecutará ningún código. Sin embargo, esta clase una vez definida permite que se instancien objetos de ella e incluso es posible realizar algunas operaciones elementales.

```
objeto1 = Clase() # Crea objeto1 de clase Clase  
objeto2 = Clase() # Crea objeto2 de clase Clase  
print(objeto1 == objeto2) # Retorna False...  
# Los objetos aunque sean de la misma clase son diferentes.
```

En el siguiente ejemplo se define una clase mucho más completa:

Ejercicio de Inducción: Pruebe las siguientes líneas de código y verifique los resultados presentados:2

```
class Alumno:  
    'Clase para alumnos'  
    numalumnos = 0  
    sumanotas = 0  
  
    def __init__(self, nombre, nota):  
        self.nombre = nombre  
        self.nota = nota  
        Alumno.numalumnos += 1  
        Alumno.sumanotas += nota  
  
    def mostrarNombreNota(self):  
        return(self.nombre, self.nota)  
  
    def mostrarNumAlumnos(self):  
        return(Alumno.numalumnos)  
  
    def mostrarSumaNotas(self):  
        return(Alumno.sumanotas)  
  
    def mostrarNotaMedia(self):  
        if Alumno.numalumnos > 0:  
            return(Alumno.sumanotas/Alumno.numalumnos)  
        else:  
            return("Sin alumnos")
```

La clase **Alumno** consta de dos variables de clase **Alumno.numalumnos** y **Alumno.sumanotas** que son accesibles desde los métodos de la clase. Además, sus valores son compartidos por todas las instancias que existan de esta clase.

A continuación, se declaran varios métodos (funciones) que incluyen como primer argumento a **self** que contiene la referencia del objeto específico que llama al método en un momento dado. Como su valor es implícito cuando se llama a un método no es necesario pasar este argumento.

El método `__init__()` es especial porque se ejecuta automáticamente cada vez que se crea una nuevo objeto. Este método, que es opcional, se llama **constructor** y se suele utilizar para inicializar las variables de las instancias (en este caso para inicializar las variables `self.nombre` y `self.nota`).

El resto de métodos se utilizan para acceder y mostrar el valor de las variables de clase y de instancia. Por último, el método `mostrarNotaMedia()` realiza un cálculo y después muestra su resultado.

6. Crear objetos (instancias) de una clase

Para crear instancias de una clase se llama a la clase por su propio nombre pasando los argumentos que requiera el método constructor `__init__` si existe.

Ejercicio de Inducción: Pruebe las siguientes líneas de código y observe los resultados:

```
alumno1 = Alumno("Maria", 8)
alumno2 = Alumno("Carlos", 6)
```

Todos los argumentos se pasan escribiéndolos entre paréntesis y separados por comas (","). El primer argumento `self` se omite porque su valor es una referencia al objeto y es implícito para todos los métodos.

7. Accediendo a los atributos y llamando a los métodos

Para acceder a la variable de un objeto se indica el nombre del objeto, seguido de un punto y el nombre de la variable:

Ejercicio de Inducción: Pruebe las siguientes líneas de código y observe los resultados:

```
print(alumno1.nombre) # María
print(alumno1.nota)   # 8
```

Para modificar la variable de un objeto se utiliza la misma notación para referirse al atributo y después del signo igual (=) se indica la nueva asignación:

Ejercicio de Inducción: Pruebe las siguientes líneas de código y observe los resultados:

```
alumno1.nombre = "Carmela"
```

Para acceder a las variables de la clase se sigue la misma notación pero en vez de indicar el nombre del objeto se indica el nombre de la clase instanciada.

Ejercicio de Inducción: Pruebe las siguientes líneas de código y observe los resultados:

```
print(Alumno.numalumnos) # 2
print(Alumno.sumanotas)  # 14
```

Para llamar a un método se indica el nombre de objeto, seguido de un punto y el nombre del método. Si se requieren varios argumentos se pasan escribiéndolos entre paréntesis, separados por comas (","). Si no es necesario pasar argumentos se añaden los paréntesis vacíos "()" al nombre del método.

Ejercicio de Inducción: Pruebe las siguientes líneas de código y observe los resultados:

```
print(alumno1.mostrarNombreNota()) # ('Carmen', 8)
print(alumno2.mostrarNombreNota()) # ('Carlos', 6)
```

Para suprimir un atributo: `del alumno1.nombre`

Si a continuación, se intenta acceder al valor del atributo borrado o se llama a algún método que lo utilice, se producirá la siguiente excepción:

```
print(alumno1.mostrarNombreNota())
```

```
AttributeError: 'Alumno' object has no attribute 'nombre'
```

Para crear nuevamente el atributo realizar una nueva asignación: `alumno1.nombre = "Carmen"`