

Guía Teórica N°02

Profundizando en Programación Orientada a Objetos (POO)

Ha llegado el momento de profundizar un poco en uno de los paradigmas de la programación más importantes en este mundo y el cual es implementado por muchos lenguajes dentro de su conjunto de instrucciones para poder llevar a cabo diferentes tareas que nosotros, los programadores, requerimos.

Antes de profundizar como tal en la Programación Orientada a Objetos y conocer nuevos términos y funcionalidades que nos pueden llegar a ser de mucha utilidad, hagamos un breve repaso de lo aprendido en el Curso Básico.

Breve Repaso:

Hay cuatro (4) términos fundamentales que debemos manejar al momento de sentarnos a trabajar con POO, estos términos son los siguientes:

- **Clase:** Conjunto de atributos y métodos que comparten diferentes objetos. En otras palabras, se define como una plantilla la cual, a partir de esta, se pueden crear objetos.
- **Objeto:** De manera resumida, un objeto es la instancia de una clase, es decir, cuando tomamos los atributos y métodos, y comenzamos a darle valores y un uso a cada uno.
- **Método:** Es la acción a definir para un objeto.
- **Atributo:** Característica que comparte un conjunto de objetos.

Veamos la implementación de estos términos a nivel de código en Python.

Para ello, observemos la siguiente imagen:

A screenshot of a code editor with a dark background. At the top left, there are three colored circles (red, yellow, green) representing window controls. The code is written in Python and defines a class named 'Persona'. It includes a constructor method '__init__' that takes a 'nombre' parameter and assigns it to 'self.nombre'. There is also a class method 'saludar' that prints a greeting using 'self.nombre'. Finally, an instance of the class is created named 'ejemplo' with the name 'Elizabeth', and the 'saludar' method is called on this instance.

```
# Definición de Clase Persona
class Persona:

    # Método Constructor
    def __init__(self, nombre):
        # Atributo Nombre
        self.nombre = nombre

    # Método de Clase Persona
    def saludar(self):
        print(f'Hola, mi nombre es {self.nombre}')

# Definición de objeto de tipo Persona
ejemplo = Persona('Elizabeth')

# Utilizamos el método 'saludar'
ejemplo.saludar()
```

Ya habiendo hecho un vistazo por encima de los términos fundamentales de la Programación Orientada a Objetos, es turno de conocer lo nuevo que aprenderemos en este curso. Así que, vayamos a ello.

Métodos Especiales:

Las clases en Python contienen un conjunto de métodos especiales que se utilizan para definir comportamientos propios de la clase, algunos son utilizados para la documentación de la clase y otros para la purificación del código para evitar un mal uso de la memoria al programar.

Cabe destacar que los métodos especiales (o integrados) están definidos por dos (2) pisos bajo (__) tanto como prefijo como sufijo.

- `__init__`: Método constructor cuya tarea es definir el valor de cada uno de los atributos al crear un nuevo objeto perteneciente a la clase.
- `__str__`: Su tarea principal es la devolver una cadena de caracteres con el valor actual de cada uno de los atributos del objeto definido.
- `__del__`: Método destructor cuya tarea es la de liberar la memoria cuando el objeto ya ha cumplido con su ciclo de ejecución dentro de nuestro código. No se recomienda modificar este método al menos que deseamos realizar una tarea de purificación en especial. Se invita a investigar sobre la función integrada de Python **del()**.
- `__len__`: Tiene por objeto conocer la longitud (cantidad de elementos) de un atributo en especial. Se invita a investigar sobre la función integrada de Python **len()**.

Sobrecarga de Métodos:

La sobrecarga de métodos se define como la posibilidad de redefinir métodos de las clases padres en cada una de las clases hijas, es decir, tomar un método padre y dentro de la clase hija, modificar el comportamiento de este método.

Veamos un pequeño ejemplo:

```
class Padre:
    # Método 'saludar'
    def saludar(self):
        print('Hola desde el padre')

class Hija(Padre):
    # Método 'saludar' (redifinición)
    def saludar(self):
        print('Hola desde el hijo')
```

Encapsulamiento:

Este término se define como la posibilidad de brindarle un permiso de visualización a los diferentes atributos y métodos que nosotros coloquemos dentro de una clase. Aunque Python no haga uso directo del encapsulamiento dado a la gran flexibilidad que nos ofrece al escribir código, esto no implica que otros lenguajes no hagan uso de este, lenguajes de programación como C/C++, Java y PHP tienen por obligación definir los permisos de cada atributo y método.

Existen tres tipos de permisos:

- Privado: Únicamente es visible para la clase en la que se está definiendo.
- Protegido: Las clases hijas tienen la posibilidad de acceder a los atributos y métodos que tengan este permiso. Utilizado principalmente para la herencia.
- Público: Cualquier clase es capaz de acceder a los atributos y métodos.

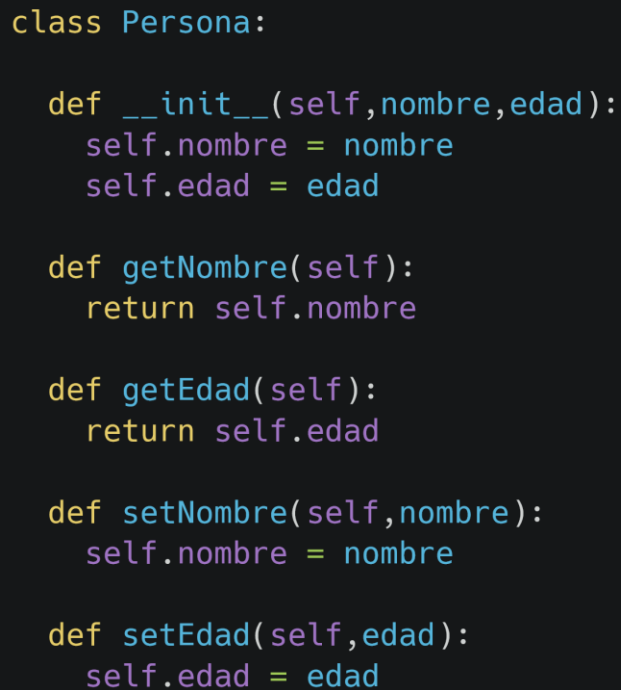
Por buenas prácticas de programación, los atributos suelen ser definidos como **privados** y los métodos como **públicos**. En caso de utilizar el mecanismo de herencia, los atributos y métodos que deseamos heredar, tendrán el permiso de **protegido**.

Ahora bien, Python no utiliza este mecanismo directamente y, por ende, podemos concluir que todos los métodos y atributos están definidos como públicos, aun así, hay dos practicas recomendadas a utilizar cuando queremos modificar y obtener los atributos de una clase, getters y setters.

Getters y Setters:

Estos son dos métodos definidos que nos permiten obtener los valores de los atributos y darles nuevos valores a los atributos. Para ello, definimos tantos getters como setters según la cantidad de atributos tengamos en nuestra clase. Por ejemplo, si tenemos dos (2) atributos, definimos un método getter y un setter por cada atributo.

Veamos un ejemplo para entender muy bien cómo se implementan estos métodos en nuestra clase:

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for a class named 'Persona'.

```
class Persona:

    def __init__(self,nombre,edad):
        self.nombre = nombre
        self.edad = edad

    def getNombre(self):
        return self.nombre

    def getEdad(self):
        return self.edad

    def setNombre(self,nombre):
        self.nombre = nombre

    def setEdad(self,edad):
        self.edad = edad
```

¡Nos vemos en la próxima clase!

De esta manera, finalizamos los bloques de repaso y profundización de temas muy importantes tratados en el Curso Básico, esto no significa que sea TODO lo relacionado a cada uno, simplemente son los puntos más relevantes a considerar cuando realicemos proyectos más grandes. Como siempre, se invita a investigar mucho más sobre cada detalle que se aprende en el curso y concretar muy bien las bases teóricas y prácticas para una mejor comprensión de los temas tratados.