Python Programming

Módulo 3

Clases y Objetos

Paradigma de Objetos

La **Programación Orientada a Objetos *POO*** (**OOP** en inglés) se trata de un paradigma de programación introducido en la década del 70’ del siglo pasado.

Este paradigma de programación nos permite organizar el código de una manera que se asemeja bastante a como pensamos y utilizamos **objetos** en la vida cotidiana. Pero nosotros vamos a utilizar o crear **clases,** como si fueran una fábrica de objetos.

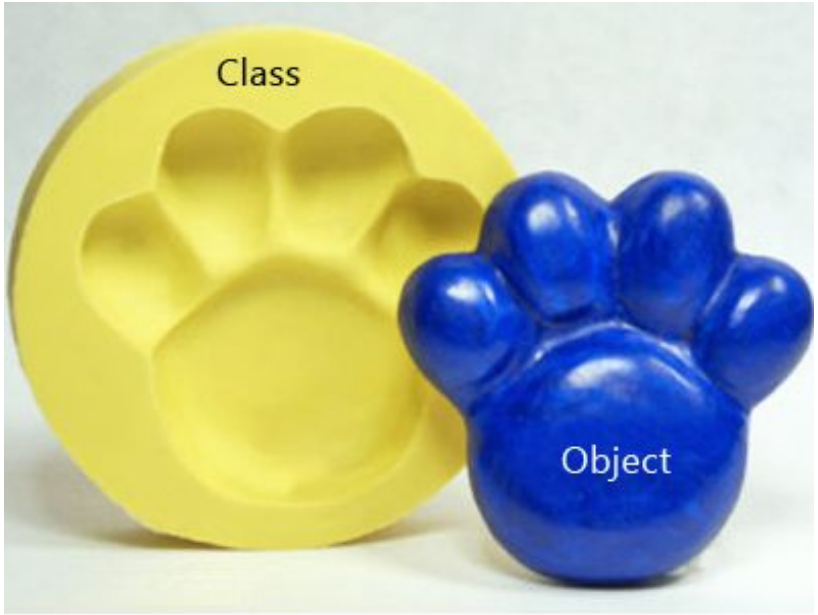
Las **clases** nos permiten agrupar un conjunto de variables y funciones, y que con las cuales podemos crear entidades que vivirán en nuestros programas.

En la programación orientada a objetos se definen estructuras de datos u **objetos**, cada uno con sus propios **atributos**. Cada objeto también puede contener sus propios **métodos**(funciones).

Paradigma de Objetos

Existen diversos lenguajes que trabajan con POO, pero los más populares son los lenguajes basados en clases en los que los *objetos son instancias de clases*.

Los principios fundamentales de la programación orientada a objetos son la encapsulación, la abstracción, la herencia y el polimorfismo.



Aspectos básicos de una clase

Las clases se definen vía la palabra reservada `class` seguida de su nombre y, entre paréntesis, el nombre de las clases de las cuales debe heredar, en caso de haber.

```
class MiClase:
      pass
```

La convención para nombrar una clase difiere de las funciones y otras sentencias; en las clases cada término se escribe con su primera letra en mayúscula y no se emplean guiones bajos.

Recordá que pass no ejecuta ninguna operación, sino que simplemente rellena un lugar que es requerido sintácticamente.

Ahora bien, todas las clases necesitan que se ejecute un codigo cada vez que se haga uso de ella y en donde el programador tendrá la posibilidad de inicializar lo que sea necesario.

```
class MiClase:
    def __init__(self):
        pass
```

A las funciones definidas dentro de una clase se las conoce con el nombre de *método*. Todas las funciones que empiezan y terminan con doble guión bajo son métodos especiales que utilizará Python y que no están diseñados para ser llamados manualmente (con algunas excepciones).

La función `__init__()` es un método que será invocado automáticamente por Python cada vez que se cree un objeto a partir de una clase. También denominado constructor en muchas bibliografías.

```
mi_objeto = MiClase()
```

Se dice que `mi_objeto` es una *instancia* de la clase `MiClase`. Para ilustrarlo mejor, prueba el siguiente código:

```
class MiClase:
    def __init__(self):
        print("Has creado una instancia de
MiClase.")

mi_objeto = MiClase()
```

Habrás observado que si bien el método `__init__()` requiere de un argumento, no hemos especificado ninguno al crear una instancia.

Esto se debe a que Python coloca el primer argumento de todos los métodos de una clase automáticamente (por convención se lo llama `self`). `self` es una referencia a la instancia actual de la clase.

Por ejemplo, consideremos el siguiente código, el cual crea dos instancias de `MiClase`.

```
mi_objeto = MiClase()
otro_objeto = MiClase()
```

En ambos casos, para crear la instancia Python automáticamente llama al método `__init__()`.

Sin embargo, en la primera línea `self` será una referencia a `mi_objeto`, mientras que en la segunda, a `otro_objeto`.

Por lo general estarás utilizando el método `__init__()` para inicializar objetos dentro de tu clase. Por ejemplo:

```
class MiClase:
    def __init__(self):
        self.a = 1

mi_objeto = MiClase()
print(mi_objeto.a)
```

Se dice que `a` es un atributo de `mi_objeto` o, en general, de `MiClase`. También podemos cambiar el valor de un atributo desde fuera de la clase:

```
mi_objeto = MiClase()
mi_objeto.a += 1
print(mi_objeto.a)  # Imprime 2
```

El método `__init__()` puede tener argumentos posicionales y por nombre como cualquier otra función convencional, que serán especificados al momento de crear una instancia de la clase.

```
class MiClase:
    def __init__(self, a):
        self.a = a

mi_objeto = MiClase(5)
print(mi_objeto.a)  # Imprime 5
```

Una clase también puede incluir funciones, aunque recuerda, siempre el primer argumento debe ser `self`.

```
class MiClase:
    def __init__(self, a):
        self.a = a

    def imprimir_a(self):
        print(self.a)

mi_objeto = MiClase(5)
mi_objeto.imprimir_a()
```

Otros ejemplos:

```
class MiClase:
    def __init__(self, a):
        self.a = a

    def imprimir_a(self):
        print(self.a)

    def sumar_e_imprimir(self, n):
        self.a += n
        print(self.a)

mi_objeto = MiClase(5)
mi_objeto.sumar_e_imprimir(7)  # Imprime 12
```

¡Muchas gracias!

¡Sigamos trabajando!