**CLASES EN PYTHON**

Se manejan siguiendo el paradigma de POO, igual que java. Al igual que en este lenguaje, por convención, el nombre de las clases comienza con Mayus.

Las clases se declaran con la palabra reservada **class** seguida del nombre de la clase.

**PASS**

Pass es una palabra reservada que puede implementarse tanto en la definición de clases, como funciones, entre otras. Su funcion es evitar que se genere un error al crear una clase o función vacía. Funciona como placeholder para futuro código.

**METODO \_\_INIT\_\_**

El método que cumple la funcion de constructor es el método \_\_init\_\_. Este tipo de métodos (inician y terminan con 2 guiones bajos) se conocen como dunder y son característicos de las clases.

(dunder 🡪 dobule underscore)

Se dice que cumple la funcion de constructor, ya que en Python en realidad el constructor está “oculto”.

class Persona:

    def \_\_init\_\_(self):

        self.nombre = ''

        self.apellidos = ''

        self.edad = ''

chabon1 = Persona()

Cuando se crea la instancia **chabon1** se invoca al tipo Persona(). Se utilizan paréntesis porque en realidad se está “llamando” al método \_\_init\_\_ y no es necesario pasarle el argumento **self** ya que este se asigna automáticamente a este objeto en particular.

Luego, desde este self se crean los atributos **nombre, apellidos** y **edad.**

En este caso los atributos están vacíos, pero si se incluyese valor alguno en la declaración de la clase se tomarían como valor por default.

Para “acceder” a estos atributos se procedería, por ejemplo: **chabon1.nombre, chabon1.apellidos, chabon1.edad**

**CREANDO OBJETOS CON ARGUMENTOS**

Para crear objetos “personalizados” deberíamos modificar el método \_\_init\_\_

class Persona:

    def \_\_init\_\_(self, nombre, apellidos, edad):

        self.nombre = nombre

        self.apellidos = apellidos

        self.edad = edad

chabon1 = Persona('Agustin', 'Stringa', 18)

Ahora, con los parámetros especificiados en el \_\_init\_\_ ya podemos darle argumento a la invocación en **chabon1**

**CREANDO MÁS OBJETOS:**

Teniendo la clase anterior, podríamos crear otro objeto de esta forma:

guy2 = Persona('Dave', 'grohl', 52)

Si se hace un debug de toda la operación y se agregan breakpoints en la creación de los 2 objetos y en el método init, se puede ver como la palabra reservada **self** cambia la dirección de memoria al momento de “asociarse” con distintos objetos.

**ATRIBUTOS**

Hasta el momento, todos los atributos creados son llamados **atributos de objetos o atributos de instancia**. Esto significa que se asocian con la creación de una instancia de la clase y tienen distintos valores para cada objeto.

También existen los llamados atributos de clase, que responden a cada clase.

**METODOS DE INSTANCIA**

La diferencia entre método y funcion es que el primero se asocia directamente con una clase, mientras que una funcion se asocia al programa en sí, o están definidas por el lenguaje.

**UML -> UNIFIED MODELING LANGUAGE -> se utiliza para hacer diagramas y documentar en POO**

[www.umletino.com/umletino.html](http://www.umletino.com/umletino.html)

TODOS LOS METODOS DE INSTANCIA LLEVAN EL PARAMETRO **self**

Por ejemplo, si quisiésemos declarar un método de instancia, que se encargue de imprimir todos los atributos de instancia, podríamos resolver de la siguiente manera.

    def mostrar\_detalles(self):

        print(f'''

        Objeto: {self}

        Nombre: {self.nombre}

        Apellidos: {self.apellidos}

        Edad: {self.edad}

        ''')

Cabe resaltar que este trozo de código se debe agregar dentro de clase. Esto significa que debe respetar la tabulación necesaria para pertenecer a la misma.

**AGREGANDO ATRIBUTOS**

Podríamos agregar atributos a algun objeto (instancia) fuera de la declaración de la clase. Es importante entender que cuando se agregue un attr a un objeto en particular, este no se agrega a los demás objetos.

La sintaxis sería: **Objeto.nuevoAtributo = ‘valor’**

guy2.band = 'Foo Fighters'

**DOCUMENTANDO CLASES**