HINTS

PYTHON Y SUS CLASES

Python viene con multitud de tipos de datos integrados, como: diccionarios, listas, conjuntos. Podemos crear tipos de datos propios como una persona, un vehículo, universidades, entre otros, los cuales son útiles en la construcción de nuestros propios desarrollos y en Python se realiza por medio de las clases y los objetos.

¿Cómo creamos una clase en Python?

```
class NombreClase:
    declaraciones
```

Para crear una clase persona, que por el momento no contendrá nada, excepto la declaración de **pass**. La sentencia **pass** no hace nada. Puede ser utilizada cuando se requiere una sentencia sintácticamente, pero el programa no requiere acción alguna.

ATRIBUTOS

Son como propiedades que queremos añadir a la clase (tipo). Por ejemplo, para nuestra clase `Persona`, vamos a añadir dos atributos: nombre y escuela.

```
class Persona:
    nombre = ''
    escuela = ''

#Creando el objeto
jorge = Persona()
jorge.nombre = 'Jorge'
jorge.escuela = 'Universidad de la vida'
```

MÉTODOS

Son como funciones en Python, ya que se definen con la palabra clave **def**, y cuentan con el mismo formato que las funciones.

```
class Persona:
    nombre = ''
    escuela = ''

    # Método Get
    def imprimir_nombre(self):
        print (self.nombre)
```

```
 8
 9     def imprimir_escuela(self):
10         print (self.escuela)
11
12
13 jorge = Persona()
14 2orge.nombre = 'Jorge'
15 2orge.escuela = 'Universidad de la vida'
16 2orge.imprimir_nombre()
17 2orge.imprimir_escuela()
```

Si no colocamos **self** como argumento en **`imprimir_nombre()`**, Python arrojará un error como el siguiente:

```
1 Traceback (most recent call last):
2   File "classPersona_test.py", line 16, in <module>
3     2orge.imprimir_nombre()
4 TypeError: imprimir_nombre() takes no arguments (1 given)
```

INICIALIZACIÓN. METODO CONSTRUCTOR __init__

En la sección anterior, hemos inicializado nombre y escuela, dándoles un valor vacío " ". Pero hay una forma más elegante de inicializar variables con sus valores predeterminados. El inicializador es un método especial, con nombre: **`__init__`**

```
 1 class Persona:
 2     def __init__(self, nombre, escuela):
 3         self.nombre = nombre
 4         self.escuela = escuela
 5
 6     def imprimir_nombre(self):
 7         print (self.nombre)
 8
 9     def imprimir_escuela(self):
10         print (self.escuela)
11
12 jorge = Persona('Jorge', 'Universidad de la vida')
13 jorge.imprimir_nombre()
14 jorge.imprimir_escuela()
```

Salida

```
1 Jorge
2 Universidad de la vida
```

Observa que el inicializador necesita dos argumentos. Por ejemplo, si no incluimos el argumento “nombre” y “escuela” en el inicializador, obtendremos el siguiente error:

```
Traceback (most recent call last):
  File "classPersona_test.py", line 13, in <module>
    jorge = Persona()
TypeError: __init__() takes exactly 3 arguments (1 given)
```

PRIMER ARGUMENTO SELF

Los objetos tienen una característica muy importante: son conscientes de que existen. Cuando se ejecuta un método desde un objeto (no desde una clase), se envía un primer argumento implícito que hace referencia al propio objeto. Si lo definimos en nuestro método podremos capturarlo y ver qué es:

```
class Persona:
    nombre = " "
    alto = false

    def saludar(soy_el_objeto):
        print("Soy un objeto de la clase persona")
        print(soy_el_objeto)

persona = Persona()
persona.saludar()
print()
print(persona.alto)
```

Salida

```
Soy un objeto de la clase persona
<__main__.Persona instance at 0x7f98100a70a0>
<__main__.Persona instance at 0x7f98100a70a0>
False
```

Podemos acceder al propio objeto desde el interior de sus métodos. Lo único es que como este argumento hace referencia al objeto en sí mismo, por convención se le llama **self**.

Poder acceder al propio objeto desde un método es muy útil, ya que nos permite acceder a sus atributos.

```
class Persona:
    nombre = " "
    alto = false

```

```
    def saludar(self):
        alto = true

persona = Persona()
persona.saludar()
print(persona.alto)
```

Salida

```
False
```

En cambio, si hacemos ver que **self** es el propio objeto:

```
class Persona:
    nombre = ""
    alto = False

    def saludar(self):
        self.alto = True


persona = Persona()
persona.saludar()
print(persona.alto)
```

Salida

```
True
```

En resumen, las instancias tienen que saber quiénes son, o no podrán acceder sus atributos internos, y por eso tienen que enviarse a sí mismas a los métodos.

Por defecto, el valor de un atributo se busca en la clase, pero para modificarlo en la instancia es necesario hacer referencia al objeto.

Las clases crean propios tipos de datos, y con los objetos se crean instancias de estos tipos de datos. Las clases también se componen de atributos (propiedades) y métodos que son acciones que realizamos en dichos atributos.