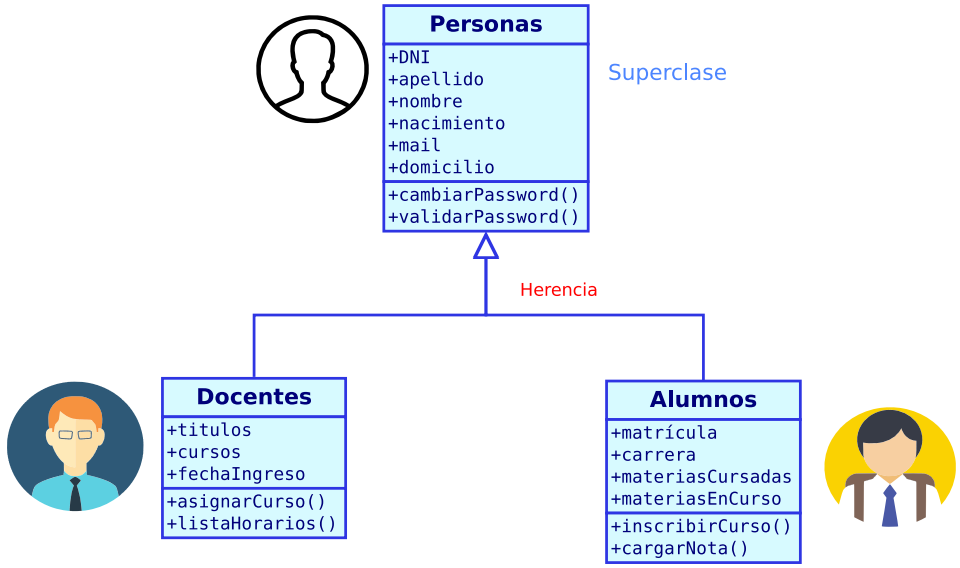
ESTUDIOS SUPERIORES DE BUENOS AIRES
ESBA

Programación Orientada a Objetos (OOP)

OOP

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que se basa en modelar los atributos y el comportamiento de clases de objetos desde un nivel muy abstracto hasta uno más concreto dependiendo de la aplicación mediante la herencia.

Python es un lenguaje orientado a objetos. Incluso cuando se pueden utilizar otros paradigmas en un programa Python se siguen utilizando objetos para hacer casi todo, eso es porque en Python todo es un objeto.



POO en Python

Los objetos definidos en Python tienen las siguientes características:

- **Identidad**. Cada objeto debe ser distinguido y ello debe poder demostrarse mediante pruebas. Las pruebas **is** e **is not** existen para este fin.
- **Estado** Cada objeto debe ser capaz de almacenar el estado. Para este fin, existen atributos, tales como variables de instancias y campos.
- **Comportamiento**. Cada objeto debe ser capaz de manipular su estado. Para este fin existen métodos.

Python incluye las características siguientes para dar soporte a la programación orientada a objetos:

- **Creación de objetos basada en clases**. Las clases son plantillas para la creación de objetos. Los objetos son estructuras de datos con el comportamiento asociado.
- **Herencia con polimorfismo**. Python da soporte a la herencia individual y múltiple. Todos los métodos de instancias de Python son polimórficos y se pueden alterar temporalmente mediante subclases.
- **Encapsulación con ocultación de datos**. Python permite ocultar los atributos. Cuando se ocultan los atributos, se puede acceder a los mismos desde fuera de la clase únicamente mediante los métodos de la clase. Las clases implementan métodos para modificar los datos.

Clases

Para definir una clase en python lo hacemos de la siguiente manera:

```
class Persona():
    pass
```

Por costumbre, el nombre de la clase que se haya creado suele nombrarse comenzando por mayúsculas y si tiene más de una palabra se acostumbra a usar la notación **camelCase** (la primera letra de cada palabra en mayúsculas y sin espacios intermedios).

Los paréntesis luego del nombre de la clase no son obligatorios, si se omiten o se dejan vacíos se interpreta que la clase persona es hija de una superclase genérica llamada **object** de la que derivan todas las clases.

Si la clase es hija de una o más clases (herencia múltiple) se especifican entre los paréntesis.

```
class Profesor( persona ):
    pass
```

La sentencia **pass** indica implícitamente que el bloque de instrucciones no se encuentra redactado, es para evitar que se produzca un error de sintaxis y también puede servir para definir lo que en java se conoce como plantilla de clase, o sea un clase que tenga enunciado los métodos que se definirán en las clases hijas pero que no haga nada.

Atributos y métodos

Luego de hacer la declaración de una clase debemos definir sus atributos y comportamientos, para hacer esto debemos dejar la sangría correspondiente para indicarle que el código siguiente pertenece a esa clase.

Para definir un **atributo de clase** simplemente creamos la variable con total normalidad y le asignamos un valor por defecto:

```
class Persona:
    DNI= None
    nombre = ""
    apellido = ""
    nacimiento = []

    def __init__(self):
        self.nacimiento = (2000, 1, 1)
        self.password = "1234"
```

A continuación de los atributos tiene definido un método (comportamiento) de forma similar a una función con la palabra por defecto **def** y el nombre de dicho método, dentro de sus paréntesis el parámetro self hace referencia al objeto sobre el cual se ejecuta.

La variable self.password no existe en la clase, comenzará a existir en el objeto cuando se instancie, es un **atributo de objeto**.

Instanciación

Para definir un objeto (**instanciar**) simplemente lo asignamos como si se tratara de un variable (recuerden que todo en python es un objeto) indicándole como contenido la clase a la que va a pertenecer.

```
p = Persona()
print( Persona.nacimiento ) # []
print( p.nacimiento ) #(2000, 1, 1)
print( p.password ) #1234
```

Se puede acceder a los atributos de clase sin necesidad de instanciarla referenciando como clase.atributo como se ve en Persona.nacimiento.

En el código anterior se definió el método __init__ (dos underscores antes y después) que es un método que se ejecuta automáticamente al instanciar el objeto. En este método se modifica un atributo de clase (nacimiento) y se crea un atributo de objeto (self.password).

Se denomina atributo de objeto a un atributo añadido en tiempo de ejecución, que en la clase no está definido. De hecho si ejecutamos:

```
print( Persona.password )
```

obtenemos:

```
AttributeError: type object 'Persona' has no attribute 'password'
```

Métodos estándar

Los siguientes métodos y algunos otros se encuentran definidos en la clase object de forma estándar y tienen un comportamiento determinado, en cada clase se puede reemplazar el comportamiento para ajustarlo a la clase.

__init__: conocido como **constructor**, método que es ejecutado automáticamente al instanciar la clase, por lo tanto es el elegido para asignar valores a los atributos más relevantes, e incluso puede evaluar no instanciar el objeto si las circunstancias no lo permiten lógicamente o los requerimientos no se cumplen.

__del__: conocido como **destructor**, antes de desaparecer el objeto, por ejemplo al final del programa o después de ejecutar la instrucción del, se invoca este método para hacer algún proceso antes de eliminarlo (por ejemplo registrar un log).

__dir__: retorna una enumeración con los nombres de todos los métodos y atributos del objeto. Se accede directamente mediante la función dir(*objeto*).

__str__: este método aporta un método personalizado para devolver la descripción del objeto en un string. Se accede cuando se invoca a la función str(*objeto*).

Herencia

Implica que puede crearse un objeto a partir de otro objeto ya existente. El nuevo objeto hereda todas las cualidades del o los objetos del que deriva y además puede añadir nuevas funcionalidades o modificar las ya existentes.

```
class padre():
    atributo1 = 10
    def metodo1(self):
        return self.atributo1

class hija(padre):
    def __init__(self):
        self.atributo1 += 10

H = hija()
print( H.metodo1() ) # 20
```

Todas las clases son hijas en última instancia del objeto Object, que es un objeto genérico que incluye los atributos y métodos estándar.

Polimorfismo

El concepto de polimorfismo (del griego muchas formas) implica que si en una porción de código se invoca un determinado método de un objeto, podrán obtenerse distintos resultados según la clase del objeto.

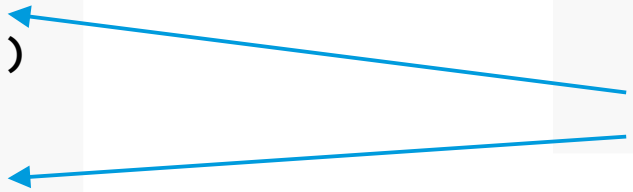
```python
class Animal:
    def sonido(self):
        pass

class Perro(Animal):
    def sonido(self):
        print("Guau!")

class Gato(Animal):
    def sonido(self):
        print("Miau!")
```

```python
def escuchar(animal):
    animal.sonido()

if __name__ == '__main__':
    g = Gato()
    p = Perro()
    escuchar(g)
    escuchar(p)
```



Encapsulación

La encapsulación es una forma de darle uso exclusivo a los comportamientos o atributos que posee una clase, es decir, protege esos atributos y comportamientos para que no sean usados de manera externa.

```
class Ejemplo():
    atributoPublico = 0
    __atributoPrivado = 10
    def publico(self):
        return "Soy un método público, a la vista de todo"
    def __privado(self):
        return "Soy un metodo privado, para ti no existo"

objeto = Ejemplo()
print(objeto.publico())
print(objeto.__privado())
```