GUÍA 1
PYTHON EN POO

DECLARACIÓN DE UNA CLASE Y CREACIÓN DE OBJETOS

La programación orientada a objetos se basa en la definición de clases; a diferencia de la programación estructurada, que está centrada en las funciones.

Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.

Un poco más abajo se define una clase Persona y luego se crean dos objetos de dicha clase.

Una clase es una plantilla (molde), que define atributos (lo que conocemos como variables) y métodos (lo que conocemos como funciones).

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Debemos declarar una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

Problema 1:

Implementaremos una clase llamada Persona que tendrá como atributo (variable) su nombre y dos métodos (funciones), uno de dichos métodos inicializará el atributo nombre y el siguiente método mostrará en la pantalla el contenido del mismo.

Definir dos objetos de la clase Persona.

```
class Persona:

   def inicializar(self,nom):
      self.nombre=nom

   def imprimir(self):
      print("Nombre",self.nombre)


# bloque principal

persona1=Persona()
persona1.inicializar("Pedro")
persona1.imprimir()

persona2=Persona()
persona2.inicializar("Carla")
persona2.imprimir()
```

Siempre conviene buscar un nombre de clase lo más próximo a lo que representa. La palabra clave para declarar la clase es class, seguidamente el nombre de la clase y luego dos puntos.

Los métodos de una clase se definen utilizando la misma sintaxis que para la definición de funciones.

Como veremos todo método tiene como primer parámetro el identificador self que tiene la referencia del objeto que llamó al método.

Luego dentro del método diferenciamos los atributos del objeto antecediendo el identificador self:

```
self.nombre=nom
```

Con la asignación previa almacenamos en el atributo nombre el parámetro nom, los atributos siguen existiendo cuando finaliza la ejecución del método. Por ello cuando se ejecuta el método imprimir podemos mostrar el nombre que cargamos en el primer método.

Decíamos que una clase es un molde que nos permite definir objetos. Ahora veamos cual es la sintaxis para la creación de objetos de la clase Persona:

```
# bloque principal

persona1=Persona()
persona1.inicializar("Pedro")
persona1.imprimir()

persona2=Persona()
persona2.inicializar("Carla")
persona2.imprimir()
```

Definimos un objeto llamado persona1 y lo creamos asignándole el nombre de la clase con paréntesis abierto y cerrado al final (como cuando llamamos a una función)

Luego para llamar a los métodos debemos disponer luego del nombre del objeto, el operador y por último el nombre del método (función)

En el caso que tenga parámetros se los enviamos (salvo el primer parámetro (self) que el mismo Python se encarga de enviar la referencia del objeto que se creó):

```
persona1.inicializar("Pedro")
```

También podemos definir tantos objetos de la clase Persona como sean necesarios para nuestro algoritmo:

```
persona2=Persona()
persona2.inicializar("Carla")
persona2.imprimir()
```

La declaración de clases es una de las ventajas fundamentales de la Programación Orientada a Objetos (POO), es decir reutilización de código (gracias a que está encapsulada en clases) es muy sencilla.

VARIABLES DE CLASE

Hemos visto como definimos atributos en una clase anteponiendo la palabra clave self:

```
class Persona:

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

Los atributos son independientes por cada objeto o instancia de la clase, es decir si definimos tres objetos de la clase Persona, todas las personas tienen un atributo nombre pero cada uno tiene un valor independiente:

```
class Persona:

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


# bloque principal

persona1=Persona("Juan")
persona2=Persona("Ana")
persona3=Persona("Luis")

print(persona1.nombre) # Juan
print(persona2.nombre) # Ana
print(persona3.nombre) # Luis
```

En algunas situaciones necesitamos almacenar datos que sean compartidos por todos los objetos de dicha clase, en esas situaciones debemos emplear variables de clase.

Para definir una variable de clase lo hacemos dentro de la clase pero fuera de sus métodos:

```
class Persona:

   variable=20

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


# bloque principal

persona1=Persona("Juan")
persona2=Persona("Ana")
persona3=Persona("Luis")
```

```
print(persona1.nombre) # Juan
print(persona2.nombre) # Ana
print(persona3.nombre) # Luis

print(persona1.variable) # 20
Persona.variable=5
print(persona2.variable) # 5
```

Se reserva solo un espacio para la variable "variable", independientemente que se definan muchos objetos de la clase Persona. La variable "variable" es compartida por todos los objetos persona1,persona2 y persona3.

Para modificar la variable de clase hacemos referencia al nombre de la clase y seguidamente el nombre de la variable:

```
Persona.variable=5
```

Problema 1:
Definir una clase Cliente que almacene un código de cliente y un nombre.
En la clase Cliente definir una variable de clase de tipo lista que almacene todos los clientes que tienen suspendidas sus cuentas corrientes.
Imprimir por pantalla todos los datos de clientes y el estado que se encuentra su cuenta corriente.

```
class Cliente:
   suspendidos=[]

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

   def imprimir(self):
      print("Codigo:",self.codigo)
      print("Nombre:",self.nombre)
      self.esta_suspendido()

   def esta_suspendido(self):
      if self.codigo in Cliente.suspendidos:
         print("Esta suspendido")
      else:
         print("No esta suspendido")
      print("_____")

   def suspender(self):
      Cliente.suspendidos.append(self.codigo)


# bloque principal

cliente1=Cliente(1,"Juan")
```

```
cliente2=Cliente(2,"Ana")
cliente3=Cliente(3,"Diego")
cliente4=Cliente(4,"Pedro")

cliente3.suspender()
cliente4.suspender()

cliente1.imprimir()
cliente2.imprimir()
cliente3.imprimir()
cliente4.imprimir()

print(Cliente.suspendidos)
```

La clase Cliente define una variable de clase llamada suspendidos que es de tipo lista y por ser variable de clase es compartida por todos los objetos que definamos de dicha clase:

```
class Cliente:
  suspendidos=[]
```

En el método imprimir mostramos el código, nombre del cliente y si se encuentra suspendida su cuenta corriente:

```
  def imprimir(self):
    print("Codigo:",self.codigo)
    print("Nombre:",self.nombre)
    self.esta_suspendido()
```

El método suspender lo que hace es agregar el código de dicho cliente a la lista de clientes suspendidos:

```
  def suspender(self):
    Cliente.suspendidos.append(self.codigo)
```

El método que analiza si está suspendido el cliente verifica si su codigo se encuentra almacenado en la variable de clase suspendidos:

```
  def esta_suspendido(self):
    if self.codigo in Cliente.suspendidos:
      print("Esta suspendido")
    else:
      print("No esta suspendido")
    print("_____")
```

Para probar esta clase en el bloque principal creamos cuatro objetos de la clase Cliente:

```
# bloque principal

cliente1=Cliente(1,"Juan")
cliente2=Cliente(2,"Ana")
cliente3=Cliente(3,"Diego")
cliente4=Cliente(4,"Pedro")
Suspendemos dos clientes:
```

```
cliente3.suspender()
cliente4.suspender()
```

Y luego imprimimos los datos de cada cliente:

```
cliente1.imprimir()
cliente2.imprimir()
cliente3.imprimir()
cliente4.imprimir()
```

Podemos imprimir la variable de clase suspendidos de la clase Cliente:

```
print(Cliente.suspendidos)
```

Es importante remarcar que todos los objetos acceden a una única lista llamada suspendidos gracias a que se definió como variable de clase.

HERENCIA

Existe un tipo de relaciones entre clases que es la Herencia.

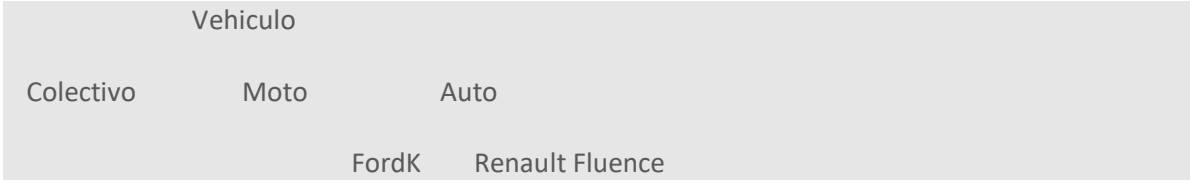
La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todas los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios.

clase padre
Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la clase padre.

Subclase
Clase descendiente de otra. Hereda automáticamente los atributos y métodos de su superclase. Es una especialización de otra clase.
Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

Veamos algunos ejemplos teóricos de herencia:

1) Imaginemos la clase Vehículo. ¿Qué clases podrían derivar de ella?



Siempre hacia abajo en la jerarquía hay una especialización (las subclases añaden nuevos atributos y métodos.

2) Imaginemos la clase Software.¿ Qué clases podrían derivar de ella?

El primer tipo de relación que habíamos visto entre dos clases, es la de colaboración. Recordemos que es cuando una clase contiene un objeto de otra clase como atributo.
Cuando la relación entre dos clases es del tipo "...tiene un..." o "...es parte de...", no debemos implementar herencia. Estamos frente a una relación de colaboración de clases no de herencia.

Si tenemos una ClaseA y otra ClaseB y notamos que entre ellas existe una relacion de tipo "... tiene un...", no debe implementarse herencia sino declarar en la clase ClaseA un atributo de la clase ClaseB.

Por ejemplo: tenemos una clase Auto, una clase Rueda y una clase Volante. Vemos que la relación entre ellas es: Auto "...tiene 4..." Rueda, Volante "...es parte de..." Auto; pero la clase Auto no debe derivar de Rueda ni Volante de Auto porque la relación no es de tipo-subtipo sino de colaboración. Debemos declarar en la clase Auto 4 atributos de tipo Rueda y 1 de tipo Volante.

Luego si vemos que dos clase responden a la pregunta ClaseA "..es un.." ClaseB es posible que haya una relación de herencia.

Por ejemplo:

```
Auto "es un" Vehiculo
Circulo "es una" Figura
Mouse "es un" DispositivoEntrada
Suma "es una" Operacion
```

Problema 1:
Plantear una clase Persona que contenga dos atributos: nombre y edad. Definir como responsabilidades la carga por teclado y su impresión.
En el bloque principal del programa definir un objeto de la clase persona y llamar a sus métodos.

Declarar una segunda clase llamada Empleado que herede de la clase Persona y agregue un atributo sueldo y muestre si debe pagar impuestos (sueldo superior a 3000)
También en el bloque principal del programa crear un objeto de la clase Empleado.

```
class Persona:

  def __init__(self):
    self.nombre=input("Ingrese el nombre:")
    self.edad=int(input("Ingrese la edad:"))

  def imprimir(self):
    print("Nombre:",self.nombre)
    print("Edad:",self.edad)


class Empleado(Persona):

  def __init__(self):
    super().__init__()
    self.sueldo=float(input("Ingrese el sueldo:"))

  def imprimir(self):
    super().imprimir()
    print("Sueldo:",self.sueldo)

  def paga_impuestos(self):
    if self.sueldo>3000:
```

```
        print("El empleado debe pagar impuestos")
    else:
        print("No paga impuestos")


# bloque principal

persona1=Persona()
persona1.imprimir()
print("_____")
empleado1=Empleado()
empleado1.imprimir()
empleado1.paga_impuestos()
```

La clase Persona no tiene ninguna sintaxis nueva, como vemos definimos sus métodos __init__ e imprimir, y sus dos atributos nombre y edad:

```
class Persona:

  def __init__(self):
    self.nombre=input("Ingrese el nombre:")
    self.edad=int(input("Ingrese la edad:"))

  def imprimir(self):
    print("Nombre:",self.nombre)
    print("Edad:",self.edad)
```

En el bloque principal creamos un objeto de la clase Persona:

```
# bloque principal

persona1=Persona()
persona1.imprimir()
```

La herencia se presenta en la clase Empleado, en la declaración de la clase indicamos entre paréntesis el nombre de la clase de la cual hereda:

```
class Empleado(Persona):
```

Con esta sintaxis indicamos que la clase Empleado hereda de la clase Persona. Luego de esto podemos decir que la clase Empleado ya tiene dos atributos heredados que son el nombre y la edad, también hereda las funcionalidades __init__ e imprimir.

La clase Empleado define un nuevo atributo que es el sueldo y tres funcionalidades __init__, imprimir y paga_impuestos.

En el método __init__ de la clase Empleado primero llamamos al método __init__ de la clase padre y luego cargamos el sueldo:

```
  def __init__(self):
    super().__init__()
    self.sueldo=float(input("Ingrese el sueldo:"))
```

Mediante la función super() podemos llamar al método __init__ de la clase padre.

Lo mismo sucede con el método imprimir donde primero llamamos al imprimir de la clase padre y luego mostramos el sueldo del empleado:

```
  def imprimir(self):
    super().imprimir()
    print("Sueldo:",self.sueldo)
```

La tercer funcionalidad es:

```
  def paga_impuestos(self):
    if self.sueldo>3000:
      print("El empleado debe pagar impuestos")
    else:
      print("No paga impuestos")
```

En el bloque principal de nuestro programa definimos un objeto de la clase Empleado y llamamos a sus funcionalidades:

```
empleado1=Empleado()
empleado1.imprimir()
empleado1.paga_impuestos()
```