TIC

Lección 1:
Lo básico de POO

POO – Programación Orientada a Objetos

¿Qué se va a aprender?

- Lo básico de POO
- Clases, Objetos y Herencia

¿Qué se necesita para realizar este trabajo?

- Equipo de Cómputo
- Conexión a Internet
- Completar el módulo 2

Planteamiento de la sesión

Materiales

- Python Tutor
- 🟩 Programación Orientada a Objetos | El Libro De Python
- Guía para Principiantes de la Programación Orientada a Objetos (POO) en Python
- ¿Qué es la Programación Orientada a Objetos?
- Clases en Python 3: Fundamentos de OOP y Ejemplos ☑️
- Built-in Types – Python 3.12.2 documentation
- Tipos integrados – documentación de Python - 3.12.2

Introducción a la POO – Inicios y motivaciones

La Programación Orientada a Objetos POO o OOP en inglés, se trata de un paradigma de programación introducido en los años 1970s, pero que no se hizo popular hasta años más tarde.

Este modo o paradigma de programación nos permite organizar el código de una manera que se asemeja bastante a como pensamos en la vida real, utilizando las famosas **clases**. Estas nos permiten agrupar un conjunto de variables y funciones que veremos más adelante.

Cosas de lo más cotidianas como un perro o un coche pueden ser representadas con clases. Estas clases tienen diferentes características, que en el caso del perro podrían ser la edad, el nombre o la raza. Llamaremos a estas características, **atributos**.

Por otro lado, las clases tienen un conjunto de funcionalidades o cosas que pueden hacer. En el caso del perro podría ser andar o ladrar. Llamaremos a estas funcionalidades métodos.

Por último, pueden existir diferentes tipos de perro. Podemos tener uno que se llama Toby o el del vecino que se llama Laika. Llamaremos a estos diferentes tipos de perro objetos. Es decir, el concepto abstracto de perro es la clase, pero Toby o cualquier otro perro particular será el objeto.

En parte surgió debido a la creciente complejidad a la que los programadores se iban enfrentando conforme pasaban los años. En el mundo de la programación hay gran cantidad de aplicaciones que realizan tareas muy similares y es importante identificar los patrones que nos permiten no reinventar la rueda. La programación orientada a objetos intentaba resolver esto.

Uno de los primeros mecanismos que se crearon fueron las funciones, que permiten agrupar bloques de código que hacen una tarea específica bajo un nombre. Algo muy útil ya que permite también reusar esos módulos o funciones sin tener que copiar todo el código, tan solo la llamada.

Las funciones resultaron muy útiles, pero no eran capaces de satisfacer todas las necesidades de los programadores. Uno de los problemas de las funciones es que sólo realizan unas operaciones con unos datos de entrada para entregar una salida, pero no les importa demasiado conservar esos datos o mantener algún tipo de estado. Salvo que se devuelva un valor en la llamada a la función o se usen variables globales, todo lo que pasa dentro de la función queda olvidado, y esto en muchos casos es un problema.

Imaginemos que tenemos un juego con naves espaciales moviéndose por la pantalla. Cada nave tendrá una posición (x,y) y otros parámetros como el tipo de nave, su color o tamaño. Sin hacer uso de clases y POO, tendremos que tener una variable para cada dato que queremos almacenar: coordenadas, color, tamaño, tipo. El problema viene si tenemos 10 naves, ya que nos podríamos juntar con un número muy elevado de variables. Todo un desastre.

En el mundo de la programación existen tareas muy similares al ejemplo con las naves, y en respuesta a ello surgió la programación orientada a objetos. Una herramienta perfecta que permite resolver cierto tipo de problemas de una forma mucho más sencilla, con menos código y más organizado. Agrupa bajo una clase un conjunto de variables y funciones, que pueden ser reutilizadas con características particulares creando objetos.

Características

Casi todos los programas y técnicas que has utilizado hasta ahora pertenecen al estilo de programación procedimental. Es cierto que has utilizado algunos objetos incorporados, pero cuando nos referimos a ellos, se mencionan lo mínimo posible.

La programación procedimental fue el enfoque dominante para el desarrollo de software durante décadas de TI, y todavía se usa en la actualidad. Además, no va a desaparecer en el futuro, ya que funciona muy bien para proyectos específicos (en general, no muy complejos y no grandes, pero existen muchas excepciones a esa regla).

El enfoque orientado a objetos es bastante joven (mucho más joven que el enfoque procedimental) y es particularmente útil cuando se aplica a proyectos grandes y complejos llevados a cabo por grandes equipos formados por muchos desarrolladores.

Este tipo de programación en un proyecto facilita muchas tareas importantes, por ejemplo, dividir el proyecto en partes pequeñas e independientes y el desarrollo independiente de diferentes elementos del proyecto.

Python es una herramienta universal para la programación procedimental y orientada a objetos. Se puede utilizar con éxito en ambos enfoques.

Además, puedes crear muchas aplicaciones útiles, incluso si no se sabe nada sobre clases y objetos, pero debes tener en cuenta que algunos de los problemas (por ejemplo, el manejo de la interfaz gráfica de usuario) puede requerir un enfoque estricto de objetos.

Afortunadamente, la programación orientada a objetos es relativamente simple.

Fuente: Skills for All, Python Essentials 2 – POO

La programación orientada a objetos está basada en 4 principios o pilares fundamentales:

- Herencia
- Abstracción
- Polimorfismo
- Encapsulamiento

Conceptos que se revisan a lo largo de la explicación y ejercicios.

Enfoque procedimental versus el enfoque orientado a objetos

En el enfoque procedimental, es posible distinguir dos mundos diferentes y completamente separados: el mundo de los datos y el mundo del código. El mundo de los datos está poblado con variables de diferentes tipos, mientras que el mundo del código está habitado por códigos agrupados en módulos y funciones.

Las funciones pueden usar datos, pero no al revés. Además, las funciones pueden abusar de los datos, es decir, usar el valor de manera no autorizada (por ejemplo, cuando la función seno recibe el saldo de una cuenta bancaria como parámetro).

El **enfoque orientado a objetos** sugiere una forma de pensar completamente diferente. Los datos y el código están encapsulados juntos en el mismo mundo, divididos en clases.

Cada clase es como una receta que se puede usar cuando quieres crear un objeto útil. Puedes producir tantos objetos como necesites para resolver tu problema.

Cada objeto tiene un conjunto de rasgos (se denominan propiedades o atributos; usaremos ambas palabras como sinónimos) y es capaz de realizar un conjunto de actividades (que se denominan métodos).

Las recetas pueden modificarse si son inadecuadas para fines específicos y, en efecto, pueden crearse nuevas clases. Estas nuevas clases heredan propiedades y métodos de los originales, y generalmente agregan algunos nuevos, creando nuevas herramientas más específicas.
Ventajas de la POO de Python
Todos los lenguajes de programación modernos utilizan la POO

Este paradigma es independiente del lenguaje. Si aprendes POO en Python, podrás utilizarlo en lo siguiente:

- Java
- PHP
- Ruby
- Javascript
- C#
- Kotlin

Todos estos lenguajes están orientados a objetos de forma nativa o incluyen opciones para la funcionalidad orientada a objetos. Si quieres aprender cualquiera de ellos después de Python, será más fácil: encontrarás muchas similitudes entre los lenguajes que trabajan con objetos.

La POO te permite codificar más rápido

Codificar más rápido no significa escribir menos líneas de código. Significa que puedes implementar más funciones en menos tiempo sin comprometer la estabilidad de un proyecto.

La programación orientada a objetos te permite reutilizar el código mediante la implementación de la abstracción. Este principio hace que tu código sea más conciso y legible.

Como ya sabrás, los programadores pasan mucho más tiempo leyendo código que escribiéndole. Es la razón por la que la legibilidad es siempre más importante que sacar características lo más rápido posible.

La POO te ayuda a evitar el código espagueti

La programación orientada a objetos nos da la posibilidad de comprimir toda la lógica en objetos, evitando así largos trozos de, por ejemplo, if's anidados.

La POO mejora el análisis de cualquier situación

Una vez que tengas algo de experiencia con la POO, podrás pensar en los problemas como objetos pequeños y específicos.

Esta comprensión conduce a una rápida puesta en marcha del proyecto.

Desventajas

- Cambio en la forma de pensar de la programación tradicional a la orientada a objetos.
- La ejecución de programas orientados a objetos es más lenta.
- La necesidad de utilizar bibliotecas de clases obliga a su aprendizaje y entrenamiento.

Programación estructurada frente a POO – Código

El siguiente ejemplo es para una cafetería, el programa hace las veces del vendedor, donde dependiendo del dinero que tengamos no indicará qué tamaño de café no podemos comprar y si tenemos cambio.

Así por ejemplo, si decimos que tenemos 2, el vendedor nos dirá que podemos comprar uno pequeño y que no hay cambio.

```
small = 2
regular = 5
big = 6

user_budget = input('What is your budget? ')

try:
    user_budget = int(user_budget)
except:
    print('Please enter a number')
    exit()

if user_budget > 0:
  if user_budget >= big:
     print('You can afford the big coffee')
     if user_budget == big:
        print('It\'s complete')
     else:

print('Your change is', user_budget - big)
    elif user_budget == regular:
        print('You can afford the regular coffee')
        print('It\'s complete')
    elif user_budget >= small:
        print('You can buy the small coffee')
        if user_budget == small:
           print('It\'s complete')
        else:
           print('Your change is', user_budget - small)
```

Este código funciona perfectamente, pero tenemos tres problemas:

- Tiene mucha lógica repetida.
- Utiliza muchos condicionales if anidados.
- Será difícil de leer y modificar.

En resumen:

OOP	Programación Estructurada
Más fácil de mantener	Difícil de mantener
No te repitas (DRY)	Código repetido en muchos lugares
Pequeños trozos de código reutilizados en muchos lugares	Una gran cantidad de código en pocos lugares
Enfoque por objetos	Enfoque de código de bloques
Más fácil de depurar	Más difícil de depurar
Gran curva de aprendizaje	Una curva de aprendizaje más sencilla
Utilizado en grandes proyectos	Optimizado para programas sencillos

Ahora veamos como sería el ejemplo completo si se desarrolla utilizando POO, por ahora es posible no entender muchas de las sentencias que aparecen en el código pero se revisará más adelante

```
class Coffee:
    # Constructor
    def __init__(self, name, price):
        self.name = name
        self.price = float(price)
    def check_budget(self, budget):
        # Check if the budget is valid
        if not isinstance(budget, (int, float)):
            print('Enter float or int')
            exit()
        if budget < 0:
          print('Sorry you don\'t have money')
          exit()
    def get_change(self, budget):
        return budget - self.price

    def sell(self, budget):
        self.check_budget(budget)
        if budget >= self.price:
            print(f'You can buy the {self.name} coffee')
            if budget == self.price:
                print('It\'s complete')
            else:
                print(f'Here is your change {self.get_change
                (budget)}$')

            exit('Thanks for your transaction')
```

Hasta el momento solo hemos creado la receta de nuestro sistema, el código anterior representa una **clase** llamada «Coffee». Tiene dos atributos – «Name» y «Price» – y ambos se utilizan en los métodos. El método principal es «Sell», que procesa toda la lógica necesaria para completar el proceso de venta.

Si se ejecuta no ocurre nada visible, como cuando se crea una función pero no la invocamos, para invocar la clase es necesario crear una instancia de la clase, es decir, aplicar la receta a una preparación:

```
small = Coffee('Small', 2)
regular = Coffee('Regular', 5)
big = Coffee('Big', 6)

try:
  user_budget = float(input('What is your budget? '))
except ValueError:
  exit('Please enter a number')

for coffee in [big, regular, small]:
  coffee.sell(user_budget)
```

Aquí estamos haciendo **instancias**, u objetos de café, de la clase «Coffee», y luego llamando al método «sell» de cada café hasta que el usuario pueda pagar cualquier opción.

Obtendremos el mismo resultado con ambos enfoques, pero podemos ampliar la funcionalidad del programa mucho mejor con la POO.

Todo es un objeto en Python

Has estado usando POO todo el tiempo sin darte cuenta.

Incluso cuando se utilizan otros paradigmas en Python, se siguen utilizando objetos para hacer casi todo.

Eso es porque, en Python, todo es un objeto.

Recuerda la definición de objeto: Un objeto en Python es una única colección de datos (atributos) y comportamiento (métodos).

Esto coincide con cualquier tipo de datos en Python.

Una cadena es una colección de datos (caracteres) y comportamientos (upper(), lower(), etc.). Lo mismo ocurre con los enteros, los flotantes, los booleanos, las listas y los diccionarios.

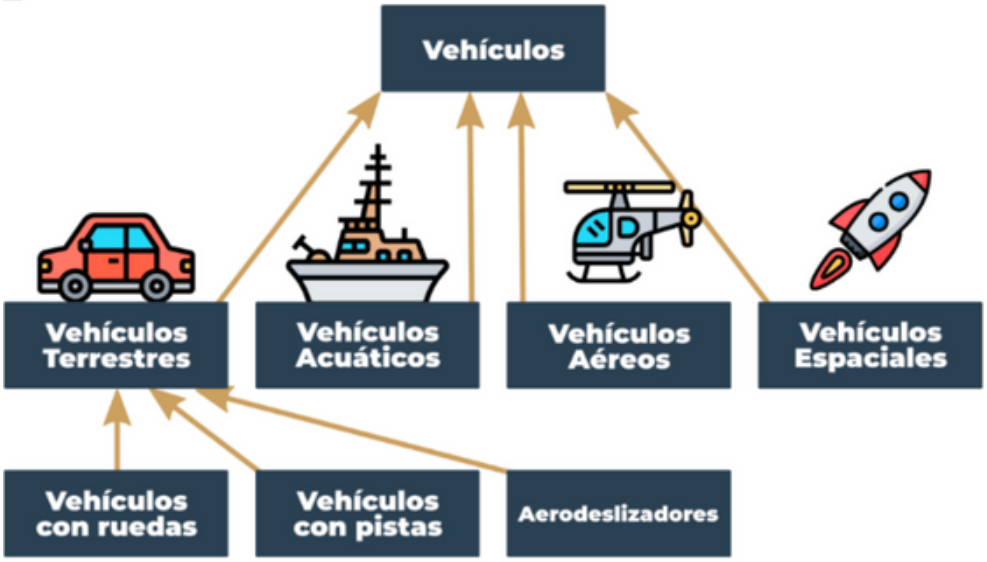
¿Qué es un objeto?

Una clase (entre otras definiciones) es un conjunto de objetos. Un objeto es un ser perteneciente a una clase.

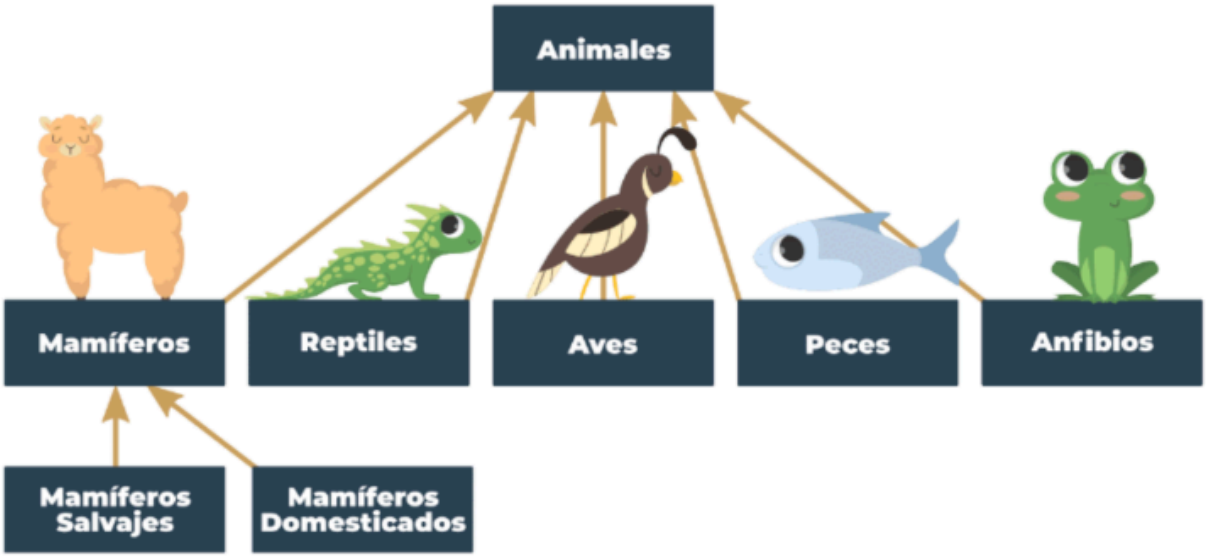
Un objeto es una encarnación de los requisitos, rasgos y cualidades asignados a una clase específica. Esto puede sonar simple, pero ten en cuenta las siguientes circunstancias importantes. Las clases forman una jerarquía.

Esto puede significar que un objeto que pertenece a una clase específica pertenece a todas las superclases al mismo tiempo. También puede significar que cualquier objeto perteneciente a una superclase puede no pertenecer a ninguna de sus subclases.

Por ejemplo: cualquier automóvil personal es un objeto que pertenece a la clase Vehículos Terrestres. También significa que el mismo automóvil pertenece a todas las superclases de su clase local; por lo tanto, también es miembro de la clase Vehículos.



Tu perro (o tu gato) es un objeto incluido en la clase Mamíferos Domesticados, lo que significa explícitamente que también está incluido en la clase Animales.



Cada subclase es más especializada (o más específica) que su superclase. Por el contrario, cada superclase es más general (más abstracta) que cualquiera de sus subclases.

{Jerarquía de Clases

Veamos por un momento los vehículos. Todos los vehículos existentes (y los que aún no existen) están **relacionados por una sola característica importante:** la capacidad de moverse. Puedes argumentar que un perro también se mueve; ¿Es un perro un vehículo? No lo es. Tenemos que mejorar la definición, es decir, enriquecerla con otros criterios, distinguir los vehículos de otros seres y crear una conexión más fuerte. Consideremos las siguientes circunstancias: los vehículos son entidades creadas artificialmente que se utilizan para el transporte, movidos por fuerzas de la naturaleza y dirigidos (conducidos) por humanos.

Según esta definición, un perro no es un vehículo.

La clase Vehículos es muy amplia. Tenemos que definir **clases especializadas.** Las clases especializadas son las **subclases**. La clase Vehículos será una **superclase** para todas ellas.

Nota: **la jerarquía crece de arriba hacia abajo, como raíces de árboles, no ramas.** La clase más general y más amplia siempre está en la parte superior (la superclase) mientras que sus descendientes se encuentran abajo (las subclases).

A estas alturas, probablemente puedas señalar algunas subclases potenciales para la superclase Vehículos. Hay muchas clasificaciones posibles. Elegimos subclases basadas en el medio ambiente y decimos que hay (al menos) cuatro subclases:

- Vehículos Terrestres.
- Vehículos Acuáticos.
- Vehículos Aéreos.
- Vehículos Espaciales.

En este ejemplo, discutiremos solo la primera subclase: Vehículos Terrestres. Si lo deseas, puedes continuar con las clases restantes.

Los vehículos terrestres pueden dividirse aún más, según el método con el que impactan el suelo. Entonces, podemos enumerar:

- Vehículos con ruedas.
- Vehículos con pistas.
- Aerodeslizadores.

La figura ilustra la jerarquía que hemos creado.

Ten en cuenta la dirección de las flechas: siempre apuntan a la superclase. La clase de nivel superior es una excepción: no tiene su propia superclase. Otro ejemplo es la jerarquía del reino taxonómico de los animales.

Podemos decir que todos los animales (nuestra clase de nivel superior) se puede dividir en cinco subclases:

- Mamíferos.
- Reptiles.
- Aves.
- Peces.
- Anfibios.

Tomaremos el primero para un análisis más detallado.
Hemos identificado las siguientes subclases:

- Mamíferos Salvajes.
- Mamíferos Domesticados.

La abstracción

La abstracción es cuando **el usuario interactúa solo con los atributos y métodos seleccionados de un objeto**, utilizando herramientas simplificadas de alto nivel para acceder a un objeto complejo.

En la programación orientada a objetos, los programas suelen ser muy grandes y los objetos se comunican mucho entre sí. El concepto de abstracción **facilita el mantenimiento de un código de gran tamaño**, donde a lo largo del tiempo pueden surgir diferentes cambios.

Así, la abstracción se basa en usar **cosas simples para representar la complejidad**. Los objetos y las clases representan código subyacente, ocultando los detalles complejos al usuario. Por consiguiente, supone una extensión de la encapsulación. Siguiendo con el ejemplo del coche, no es necesario que conozcas todos los detalles de cómo funciona el motor para poder conducirlo.

Atributos y métodos

Los atributos son variables internas dentro de los objetos, mientras que los métodos son funciones que producen algún comportamiento.

Antes de crear nuestras clases y objetos, vamos revisar algunas de las que ya hemos utilizado, por ejemplo las de cadenas, definamos entonces primero una cadena:

cadena = "Hola, soy una cadena para revisar métodos y atributos de cadenas en Python."

print(type(cadena))

#<class 'str'>

Y ahora ejecutemos algunos métodos asociado a la clase cadena, recordemos el método dir() que devuelve todos los atributos y métodos de un objeto, en este caso de nuestra cadena, que es un objeto de la clase str, temas que cubriremos más adelante.

```
print(dir(cadena))

#['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join',
'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Por ejemplo utilizar __len__() para conocer la longitud de la cadena, que también funciona con len(cadena)

```
print(cadena.__len__())

#75
```

Esto mismo lo podemos repetir con un método como por ejemplo capitalize() o upper():

```
print(cadena.capitalize())
#Hola, soy una cadena para revisar métodos y atributos de cadenas en python.

print(cadena.upper())
#HOLA, SOY UNA CADENA PARA REVISAR MÉTODOS Y ATRIBUTOS DE CADENAS EN PYTHON.
```

En este punto, solo nos queda explorar la documentación Tipos integrados – documentación de Python – 3.12.2 o ejecutar diferentes métodos para conocer sus resultados.

Vamos identificando como todo este tiempo hemos interactuado con los métodos y objetos propios de otras clases, en el momento que comencemos a crear nuestras propias clases, vamos a revisar como creamos dichos elementos y la similitud que tienen con variables y funciones, solo que aplicadas a la definición de la clase.

Para finalizar, vamos a ver un par de métodos y formas especiales de trabajar con estos dentro de la definición de una clase:

Los objetos con los mismos atributos y métodos se agrupan **clases**. Las clases definen los atributos y los métodos, y por tanto, la semántica o comportamiento que tienen los objetos que pertenecen a esa clase. Se puede pensar en una clase como en un molde a partir del cuál se pueden crear objetos.

Los atributos se definen igual que las variables mientras que los métodos se definen igual que las funciones. Tanto unos como otros tienen que estar indentados en el cuerpo de la clase.

Los métodos de una clase son las funciones que definen el comportamiento de los objetos de esa clase.

Se definen como las funciones con la palabra reservada def. La única diferencia es que su primer parámetro es especial y se denomina self. Este parámetro hace siempre referencia al objeto desde donde se llama el método, de manera que para acceder a los atributos o métodos de una clase en su propia definición se puede utilizar la sintaxis self.atributo o self.método.

La razón por la que existe el parámetro self es porque Python traduce la llamada a un método de un objeto objeto.método(parámetros) en la llamada clase.método(objeto, parámetros), es decir, se llama al método definido en la clase del objeto, pasando como primer argumento el propio objeto, que se asocia al parámetro self.

Algo super sencillo sería una clase con un atributo y un método, que se vería algo así:

```
class Saludo:
#Definición de un atributo
  mensaje = "Bienvenido "
#Definición de un método
  def saludar(self, nombre):
      print(self.mensaje + nombre)
      return
```

Si creamos una instancia de clase e invocamos a su método tendríamos algo como:

```
s = Saludo()
s.saludar('Alf')

#Bienvenido Alf
```

Para avanzar con el tema de la POO, vamos a revisar la definición de clase objeto y herencia, para conocer un poco más de los pilares de la POO, de esta forma, vamos a poder crear nuestras propias clases, objetos e interactuar con ellos en prácticas y ejercitaciones.