**La POO puede variar según el programador.**

Este tipo de **programación** es **mucho más abierta**, aunque **favorece** una **estructuración ordenada**. Se **requiere** de una **cierta** **formación** previa, pero en la **práctica** hay varias **ventajas**. La organización del código se realiza en distintas clases que, posteriormente, podrán concretarse en objetos.

La **POO** **busca** que las **aplicaciones** que se **desarrollen** sean **cada** **vez** **más** **complejas** **sin** que eso suponga **desechar** el **código**. Esta filosofía permitirá reutilizarlo, de manera que progresar no supondrá renunciar.

***Herencia:***

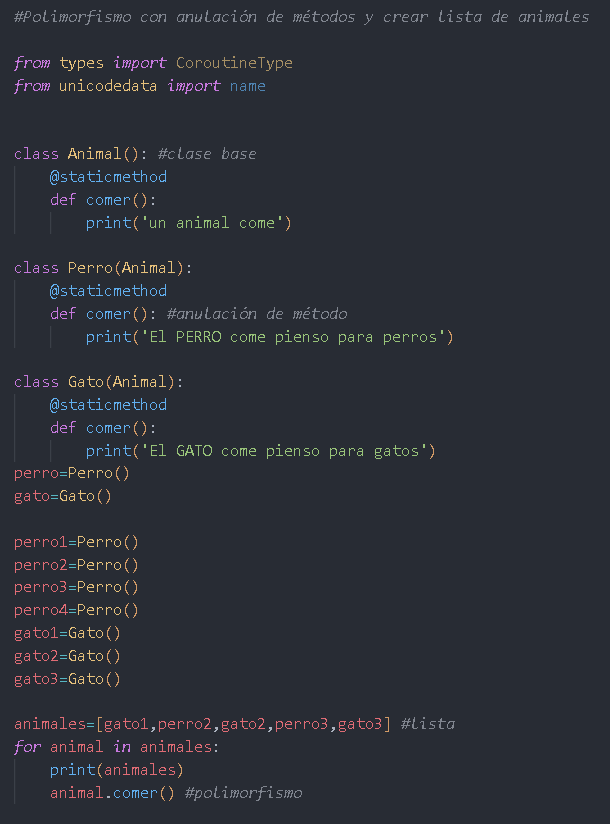
La **herencia** es un proceso mediante el cual se puede **crear** una **clase** **hija** que **hereda** **de** una **clase** **padre**, **compartiendo** sus **métodos** y **atributos**.



***Polimorfismo:***

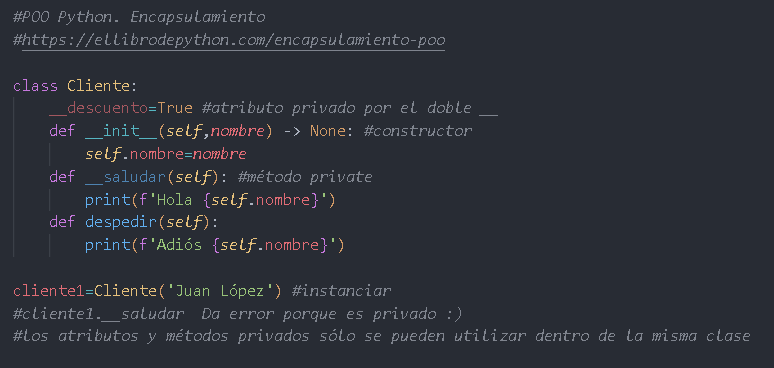
El término **polimorfismo** tiene origen en las palabras **poly (muchos)** y **morfo (formas)**, y aplicado a la programación hace referencia a que los **objetos** pueden **tomar** **diferentes** **formas**.

**Objetos de diferentes clases pueden ser accedidos utilizando el mismo interfaz**, mostrando un **comportamiento** **distinto** (tomando diferentes formas) según cómo sean accedidos.



***Encapsulamiento:***

El **encapsulamiento** o encapsulación en programación es un concepto relacionado con la programación orientada a objetos, y hace referencia al **ocultamiento** de los **estados** **internos** de una clase **al exterior**. Dicho de otra manera, **encapsular consiste en hacer que los atributos o métodos internos a una clase no se puedan acceder ni modificar desde fuera**, sino que tan solo el propio objeto pueda acceder a ellos.



***Sobreescritura:***

El término **sobreescritura** de funciones o métodos es comúnmente **usado en la herencia de la POO**. Consiste en **instanciar en una clase base** algún **método** los cuales **serán** **sobreescritos** **en la clase derivadas** con el **mismo nombre** del método **usado en la clase base**, de esta forma el **método** llamado **dependerá** del **tipo** del **objeto** que **llame** al **método** es importante resaltar que estos métodos sobreescritos pertenecen a la interfaz de ambas clases y **el método en la clase derivada sobreescribe al de la clase base**.

En la imagen se ve un ejemplo de sobreescritura en la línea 45 y 58. Donde la clase padre tiene definido el método consultar() y lo sobreescribimos en la clase hija con el mismo nombre consultar()



***Sobrecarga:***

La**sobrecarga de Métodos** **se apoya de métodos y constructores**.

La sobrecarga de métodos **hace** que **un mismo nombre** pueda **representar distintos métodos** con **distinto tipo** **y número** **de parámetros**, manejados dentro de la misma clase. La sobrecarga de métodos **se refiere a la posibilidad de tener dos o más métodos con el mismo nombre**, **pero distinta funcionalidad.** Es decir, dos o más métodos con el mismo nombre realizan acciones diferentes y el compilador usará una u otra dependiendo de los parámetros usados. Esto también se aplica a los constructores (de hecho, es la aplicación más habitual de la sobrecarga).



***Modificadores de acceso(public / private):***

La mayoría de los lenguajes de programación tienen **tres formas** de modificadores de acceso, que son **Público** , **Protegido** y **Privado** en una clase. En el caso de **Python solo** usamos **Público y Privado**.

**Python usa el símbolo '\_'** para determinar el control de acceso para un miembro de datos específico o una función miembro de una clase. Los especificadores de acceso en Python tienen un papel importante que desempeñar para proteger los datos del acceso no autorizado y evitar que sean explotados.

Se ve mejor con un ejemplo visual:

En este ejemplo de encapsulamiento se ve como utilizamos la ‘\_’ para marcar como privado el descuento del cliente en la línea 114 o en la 117 en saluda.

