PROGRAMACIÓN EN PYTHON

POO:

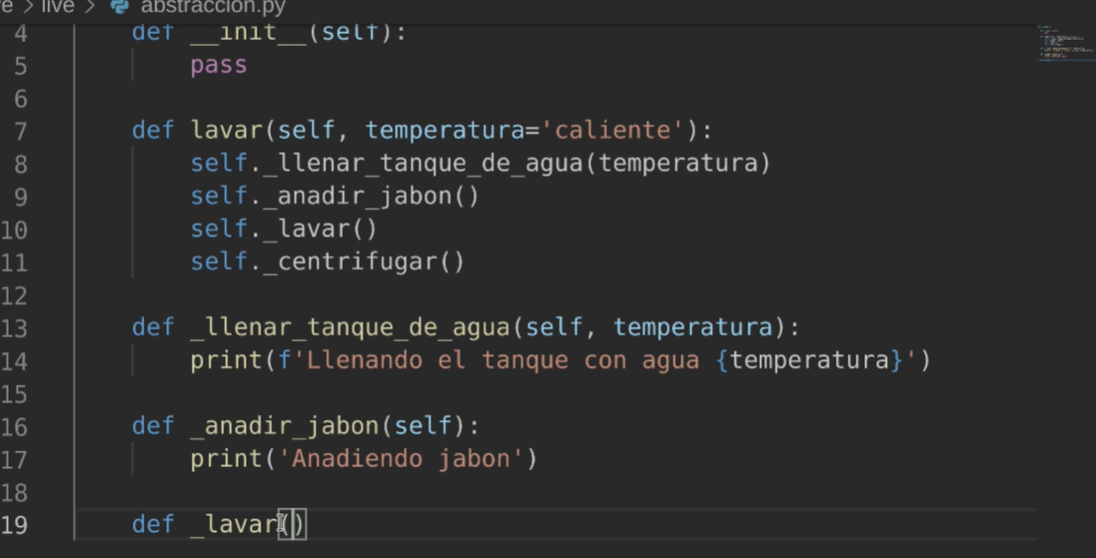
* El paradigma de Programación orientada a objetos permite la separación por módulos de un programa mediante la creación de clases. Mayor escalabilidad.
* Las clases son un conjunto de datos, las instancias son las clases convertidas en objetos cuando esos datos toman valores. Los métodos son las interacciones que se permite realizar sobre esos datos, ya sea mostrar, actualizar, eliminar.
* El garbage collector de Python se ocupa de eliminar de memoria las instancias que ya no están siendo utilizadas.
* La palabra reservada para crear clases es: **class <nombre de la clase> (<super clase>):**
* La instancia se crea: **<variable> = <nombre de la clase>(<argumentos>)**
* Los atributos se determinan dentro de la función constructor.
* Se utiliza el keyword self para hacer referencia a la clase misma al indicar la asignación de un parámetro:  **self.atributo = parámetro**
* El constructor de la clase es: **def \_\_init\_\_(self, parámetro1, parámetro2):**
* Hay parámetros por default **parámetro1 = `lento`**
* Por convención los atributos y métodos privados se especifican anteponiendo un guion bajo: **\_nombreAtributo** o  **def \_nombreMétodo():**
* El punto de entrada es la línea a partir de la cual el programa empieza a correr cuando se lo ejecuta a través de la terminal:  **if \_\_name\_\_ = `\_\_main\_\_`:**
* Para atributos son inicializar se usa la keyword **None**.
* Usar **pass** para definir que en una función no hay un bloque de código escrito.
* Añadir una variable a una cadena de texto: **(f`**cadena de texto**{variable}`)**.
* Uso **isinstance(valor, clase)** para validar que un objeto corresponda a una clase. Por ejemplo, para asegurar que el valor ingresado sea un valor entero o un float: **isinstance(number, float)**.
* **Convertir datos a minúscula: .lower()**
* Operador in: si un elemento esta dentro de una lista.
* Separar una cadena: .split(‘separador’)

CONVENCIONES AL ESCRIBIR CÓDIGO:

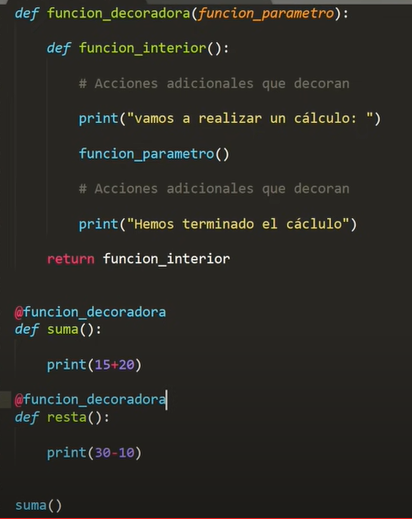
* Las funciones o nombres de variables compuestos se escriben comenzando en minúscula y separados por guión bajo.
* Las clases comienzan con Mayúscula.
* Atributos protegidos con un guión bajo, privados con dos.

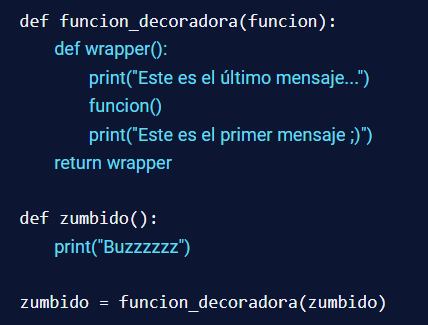
DECOMPOSICIÓN: Se trata de dividir un problema en problemas más pequeños mediante clases. Cada clase se encarga de una parte del problema. Permite un mejor mantenimiento del programa y abstraernos del funcionamiento interno de los componentes.

ABSTRACCIÓN: Saber separar lo que es importante de lo que es secundario, por ejemplo, la composición interna de un método. Lo importante es conocer que hace, que necesita y que retorna. No pensar en cómo están conformadas las cosas internamente. Se utiliza para esto la especificación de atributos y métodos privados que no le interesan al usuario. Por ejemplo, solo encender un lavarropas, pero no mostrar los pasos internos.



* Las funciones en Python se toman como objetos de primera clase, pueden ser utilizados como argumentos como cualquier otro objeto.
* Función decoradora: Se denomina a una función que recibe como parámetro una función para añadir nuevas funcionalidades y luego retorna una función. Dentro del bloque de código se establecen acciones propias del decorador, o se declara una función interna llamada **wrapper** que las llevará a cabo y ejecuta la función recibida. Se retorna la función interior. Se retorna sin los parentesís **return funciónInterna**. Se utiliza para añadir funcionalidades a un grupo de funciones sin necesidad de modificar los códigos de las mismas una por una, esto colocando la sintaxis **@funcionDecoradora** justo por encima de la declaración de la función que decora. Un uso práctico es la conexión y cierre con las bases de datos.





MANIPULAR ATRIBUTOS PRIVADOS: @property, @.setter

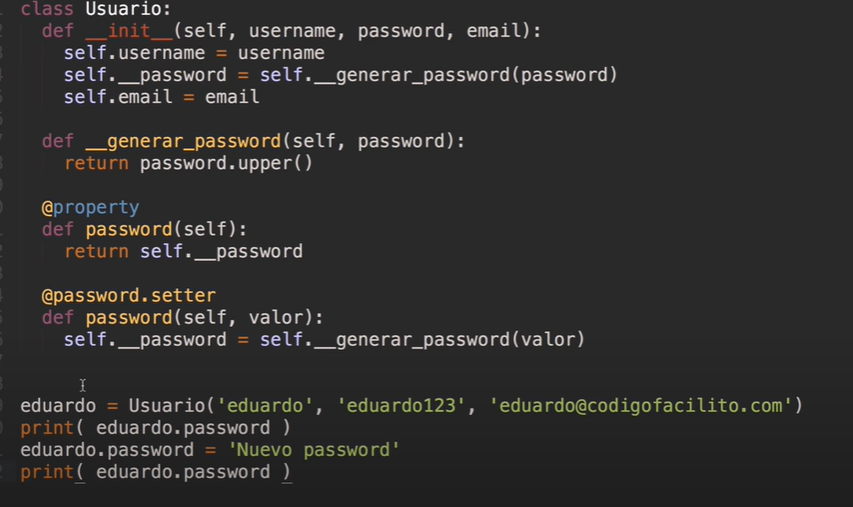
Por convención el atributo o método privado se identifica con la anteposición del guión bajo al nombre de la variable o función.

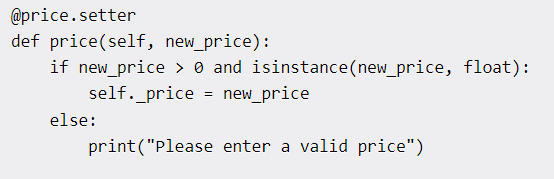
Cualquier desarrollador debe entender que esto significa que no debe manipularse directamente estos valores, si existiera las funciones getter y setter para acceder a ellos. Se identifican con los decoradores **@property**.

Property funciona como el getter. Convierte una función en un atributo, para ello el nombre de la función debe ser del nombre del atributo privado. Recibe como parámetros el objeto en si mimo self. Dentro de su desarrollo se accede al atributo privado con la doc notation.

Para settear un atributo privado se vuelve a designar una función con el nombre del atributo. Se antepone el decorador **@<nombre del atributo>.setter**. La función recibe como parámetro la referencia al objeto, y el valor nuevo. Dentro se accede a los atributos o métodos privados. En el programa principal se llama a la instancia y a la función decorada con doc notation y se le asigna con = el valor nuevo, **Objeto.atributo=valor** . Sin este decorador no es posible asignar un nuevo valor sin acceder directamente al atributo privado.

Nota:La función decorada con el setter puede tener cualquier nombre, pero para asignar un nuevo valor se debe acceder con el nombre de la función convertida a atributo con property.





class Personaje:

    def \_\_init\_\_(self, name):

        self.name = name

        self.\_\_element = self.\_\_assignElement()

    def \_\_assignElement(self):

        return 'water'

    @property

    def element(self):

        return self.\_\_element

    @element.setter

    def setElement(self, newElement):

        if isinstance(newElement, str):

            self.\_\_element = newElement

        else:

            print('You must assign a String')

    @element.deleter

    def deleteElement(self):

        del self.\_\_element

        print('The element has been deleted')

if \_\_name\_\_ == '\_\_main\_\_':

    camus = Personaje('camus')

    print(camus.name)

    print(camus.element)

    camus.setElement = 'fire'

    print(camus.element)

    del camus.deleteElement

    print(camus.element) # It throws an error when not finding attribute.

ENCAPSULAMIENTO:

* No hay restricción de datos como en JAVA. Se maneja por convención y se entiende que atributos o métodos con un guión bajo son protegidos, con doble guión bajo son privados.
* La restricción de métodos se relaciona con el encapsulamiento, agrupar un conjunto de datos, restringir los accesos de información a los usuarios de las clases y las sub clases.

HERENCIA:

* Sirve para crear gerarquías de clases y compartir atributos y comportamientos.
* La sintaxis de herencia: **class <Nombre\_De\_La\_Clase>(Superclase\_que\_extiende):**
* La subclase hereda todos los atributos y métodos de la superclase.
* No tiene acceso a las propiedas o métodos privados de esta, solo si existen getters y setters que lo permitan.
* Es mala práctica acceder directamente a atributos protegidos.
* El constructor de una subclase llama al constructor de la super clase: **super().\_\_init\_\_(parametro1,…,…)**

POLIMORFISMO:

* Poder modificar un método de una superclase y editarlo de forma que se comporte distinto en una subclase.
* Existen métodos del mismo nombre en clases diferentes, pero el desarrollo es distinto y al llamar al método de cada clase instanciada se ejecuta el que corresponda.

