DOCUMENTANDO LAS CLASES.

DOCUMENTANDO CLASES.

Para las clases necesitamos incluir en la documentación:

- Lo que la clase representa.
- Métodos públicos. Tenemos que mencionar cualquier método público que la clase tiene que podemos llamar desde fuera de la clase.
- Atributos de instancia y atributos de clase que son públicos.
- Efectos de herencia.

- La cadena de documentación de una clase debe resumir su comportamiento y enumerar los métodos públicos y las variables de instancia, si la clase está destinada a ser subclasificada y tiene una interfaz adicional para las subclases, entonces esta interfaz debe figurar por separado en el documento, el constructor de la clase debe ser documentado en la cadena doc para su método init, por lo que el método init debe ser documentado con una cadena doc, los métodos individuales deben documentarse con su propia cadena doc.,
- Si una subclase de una clase es otra clase y su comportamiento se hereda en su mayor parte de esa clase, su docstring debe mencionarlo y resumir las diferencias. Debe utilizar el término **override** para indicar que un método de la subclase sustituye a un método de la superclase y no llama al método de la superclase y utilizar el verbo **extend** para indicar que un método de la subclase llama al método de la superclase además de a su propio comportamiento, así que tenemos dos términos para la cadena doc **override** y **extend**.

- Así que la clave es que el constructor de la clase debe estar documentado en el docstring para su método __init__.
- Los métodos individuales deben tener su propio docstring (especialmente los métodos públicos).
- Si una clase subclase de otra clase y su comportamiento se hereda en su mayor parte de esa clase, su docstring debe mencionarlo y resumir las diferencias.

La siguiente clase tiene tres métodos y también tiene el método init. Podemos ver que inicialmente una instancia de mochila tiene una lista vacía de elementos. Podemos añadir un elemento, eliminar un elemento y comprobar si la mochila contiene un elemento o si tiene un elemento.
Así que empecemos a documentar esta clase para ayudar al usuario de la clase a saber más sobre el tipo de objetos

```
class Backpack:

    def _init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append (item)

    def remove_item(self, item):
        if item in self.items:
            self.items.remove(item)
        else:
            print("This item is not in the backpack")

def has_item(self, item):
    return item in self.items
```

DOCUMENTANDO LA CLASE.

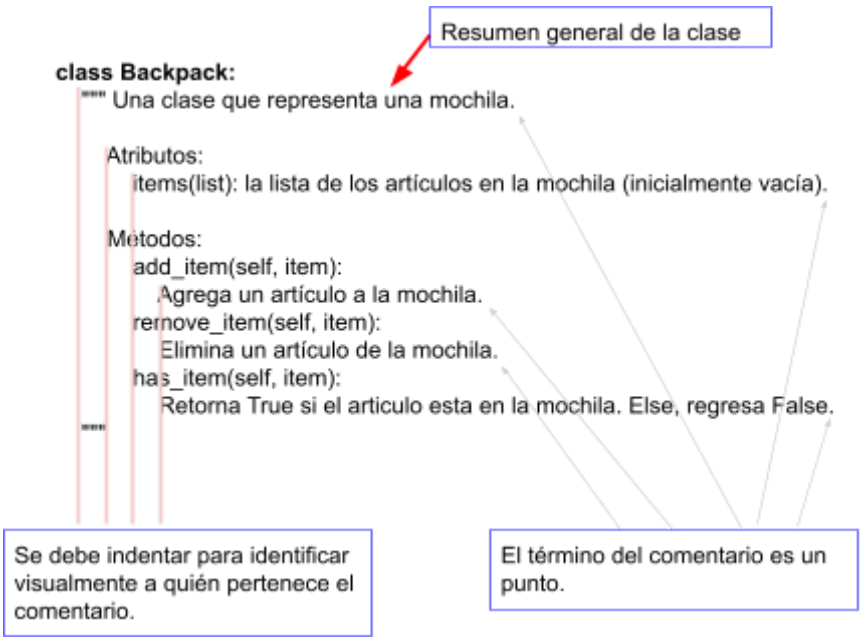
Documentado queda de la siguiente manera:

```
class Backpack:
    """ Una clase que representa una mochila.

        Atributos:
            items(list): la lista de los artículos en la mochila (inicialmente vacia).

        Métodos:
            add_item(self, item):
                Agrega un artículo a la mochila.
            remove_item(self, item):
                Elimina un artículo de la mochila.
            has_item(self, item):
                Retorna True si el articulo esta en la mochila. Else, regresa False.
    """

    def _init__(self):
        self.items = []
```



Se respetan las indentaciones para identificar la pertenencia de comentarios a sub comentarios.

@properties

Otro aspecto importante de la documentación es documentar las propiedades de una clase. En este caso, sólo debes documentar los **getters** de la propiedad e incluir información relevante de los **setters** si procede, entonces si una propiedad tiene un **getter** y un **setter**, sólo escribirás un **docstring** en el **getter** y ese único **docstring** debería incluir toda la información relevante para ambos métodos.

EJEMPLO 02 DE DOCUMENTACIÓN DE UNA CLASE.

```
import math

class Circulo:

    def __init__(self, radio, color):
        self._radio = radio
        self._color = color

    @property
    def radio(self):
        return self._radio

    @property
    def color(self):
        return self.color
```

```
    @color.setter
    def color(self, new_color):
        self._color = new_color

    @property
    def diametro(self):
        return 2 * self._radio

    def encuentra_area(self):
        return math.pi*(self._radio ** 2)

    def encuentra_perimetro(self):
        return 2 * math.pi * self._radio
```

DOCUMENTACIÓN DE LA CLASE.

```
import math

class Circulo:
    """Una clase que representa un círculo.

    Atributos:
        radio (float): Es la distancia del centro del círculo a su
        circunferencia.
    color (string): Es el color del círculo.
    diámetro (float): Es la distancia a través del centro del círculo
        de un lado a otro del círculo.

    Métodos:
        find_area(self):            <-- sefl es opcional depende del equipo
            Retorna el área de un círculo.
        find_perimeter(self):
            Retorna el perímetro del círculo.
    """

    def __init__(self, radio, color):
        """Inicializa una instancia de la clase circulo.

            Argumentos:
                radio (float): Es la distancia del centro del círculo a su
                    circunferencia.
                color (string): El color del círculo
        """

        self._radio = radio
        self._color = color
```

```
    @property
    def radio(self):
        """Retorna el radio del círculo.

            Este es un valor flotante que representa la distancia desde el
            centro del círculo de su circunferencia."""
        return self._radio

    @property
    def color(self):    # Aqui se documento el getter y el setter
        """Retorna el color de un círculo.

            El color es descrito por una cadena que debe capitalizarse.
            Por ejemplo: "Rojo", "Azul", "Verde", "Amarillo".
        """

        return self.color

    @color.setter       # El setter quedó documentado en el getter
    def color(self, new_color):
        self._color = new_color

    @property
    def diametro(self):
        """ Retorna el diámetro de un círculo.

        Este es un flotante que representa la distancia a través del
        centro del círculo desde un lado a otro del círculo.
        """

        return 2 * self._radio

    def encuentra_area(self):
        """Calcula y retorna el área de un círculo.

        El área es calculada con el radio del círculo usando la fórmula
        Pi * (radio ** 2 )
        """
        return math.pi*(self._radio ** 2)

    def encuentra_perimetro(self):
        """ Calcula y encuentra el perímetro de un círculo.

        El perímetro es calculado usando el radio del círculo usando
        la fórmula (2 * pi * radio).
        """
        return 2 * math.pi * self._radio
```

USO DE LA FUNCIÓN help ( ).

Para acceder a la documentación hacemos uso de la función help()
Al final del programa escribimos:

```
    def encuentra_perimetro(self):
        """ Calcula y encuentra el perimetro de un circulo.

        El perimetro es calculado usando el radio del circulo usando
        la formula (2 * pi * radio).
        """
        return 2 * math.pi * self._radio
```

```
help(Circulo)
```

Con lo anterior al momento de ejecutar obtenemos la documentación de la clase Circle.

```
help(Circulo.__doc__)
```