**Semana 5 día 3**

Modificadores

Conceptos que van a permitir a una variable tipificarla como publica o privada.

Han de estar dentro de una clase

\_x # es una variable protegida. Se puede llamar a la función dentro del propio fichero/carpeta en el que se creó, aunque se solicite fuera de la clase. Si están fuera del fichero/carpeta no se puede acceder

\_\_X # es una variable privada. Privada encapsulada. Sólo se puede acceder a ella desde dentro de la propia clase en la que se ha creado. Funciones a las que tengan acceso otras personas o no. Se evita que algunos parámetros obligatorios para que funcione la clase se puedan cambiar por una persona externa. Puede aparecer dentro del fichero/carpeta donde se ha creado pero no permite el acceso. Desde otro fichero/carpeta ni siquiera aparecería esta función

If \_\_name\_\_ == ‘\_\_main\_\_’

se ejecuta lo de abajo

**class properties**

# se pueden crear atributos de una clase fuera de la clase al crear una instancia pero no es una buena práctica

Puedo acceder a una función privada de una clase creando otra función en es misma clase que me llame a la función privada

# @ es el decorador de la función

@property (es un decorador ya creado por defecto)

Para llamar a la función no hace falta que coloque los “()”, puedo acceder a una función como un atributo, aunque sea una función

def \_\_init\_\_(self, x):

self.x=\_\_x

@property

def x(self):

Return self.\_\_x

p.x # llamo a la función como un atributo(sin parentésis), en vez de como una función

@x.setter # se llama a la propiedad x para poder modificar la variable. Hay que crear la función con el mismo nombre de la propiedad (x)

def x(self,x):

if x< 0:

self.\_\_x = 0

elif x> 0:

self.\_\_x = 1000

else::

self.\_\_x = x

p.x # estoy llamando a la propiedad

p.x = -9999 # si asignamos un valor a x llamamos a setter

@classmethod # es una función que permite construir una nueva instancia utilizando unos atributos diferentes a los especificados en el constructor de la clase

Class person:

Def \_\_init\_\_ (self, name, age, is\_alive):

self,\_name = name

self.\_age = age

@classmethod

def from\_birthyear(cls, name, birthyear):

return cls(name=name, age=(date.today().year - birthyear), is\_alive = True)

Lucia = person,from\_birth\_year(cls, name, birthyear):

@staticmethod # va a utilizar algún concepto de la clase. Sin tener que crear una instancia de la clase pero son funciones que utilizan algún atributo de la clase

def is\_adult(age):

return age > 18

person.is\_adult()

# función método de la clase tablero

@staticmethod

def check\_boats(number\_of\_boats):

if number\_of\_boats ¡= 10:

print (“no tiene 10 barcos)

else:

print(‘tiene los barcos necesarios’)

# función de la instancia tablero

def check\_boats2(self, number\_of\_boats):

if number\_of\_boats ¡= 10:

print (“no tiene 10 barcos)

else:

print(‘tiene los barcos necesarios’)

tablero().check\_boats(number\_of\_boats=10)

poner al abrir archivos siempre ‘low\_memory=False’ para que lea todo correctamente. Para ficheros muy grandes

leer en Python con diccionarios, en caso de ficheros muy grandes. Panda tiende a ser eficiente e interpreta los datos de forma no muy rigurosa

users2=users.copy()

users2.drop(columns =[‘occupation’], inplace=True) # para que me haga el cambio directamente en la variable. Atribuir los cambios en la nueva base a la primera como aparece arriba, hay que ponerlo con users.copy para que no me cambie la base original

El None para numpy es NaN

Para el proyecto: 1000 filas y más o igual a 5 columnas

**13-04-2021**

Comprobar siempre que la versión de Python de jupyter y de Python es la misma. Aparecen, la de jupyter en la parte de arriba de visual studio y la de Python en la barra de abajo

**Correlaciones**

# pearson

Es un valor numérico entre -1 < r < 1

0 -> no correlación

1 -> las dos variables aumentan de igual forma. Correlación positiva alta entre 0.5 y 1

-1 ->

Plt.figure(figsize =(15,10))

C=df.corr()

Sns.heatmap(c