CITCEA

Módulo 2

Big data y machine learning

Introducción a Python

CONTENIDOS

1. Introducción a Jupyter Notebooks

Jupyter notebooks es un entorno (IDE) que nos permite intercalar contenido de texto (se llamará cela 'Markdown') con código Python (tipo de cela 'Code') que se puede ejecutar cela por cela como se puede ver en los siguientes ejemplos.

Los comando básicos para poder seguir este curso y empezar a usar Jupyter Notebooks son:

- `Shift+Enter` : Ejecutar cela y seleccionar siguiente cela
- `Alt+Enter` : Ejecutar cela y insertar una cela nueva
- `ESC` : para entrar en modo 'command'. Una vez en este modo:
  - `H` lista de comandos
  - `A` insertar cela debajo
  - `B` insterar cela arriba
  - `D,D` D dos veces eleimina la cela
  - `Y` cambiar cela a tipo 'Code'
  - `M` cambiar cela a 'Markdown'
  - `Enter` pasar a modo edit

Ejecutar la cela de debajo usando alguno de los comandos aprendidos

In [1]:
```
3 + 4 + 9
```

Out[1]:
```
16
```

In [2]:
```
# Ejemplo de código en Python
# Cambiar los valores de a y b, y observar el resultado al ejecutar la cela

a=87
b=34
c=a+b

print('El resultado de sumar {} y {} es {}.'.format(a,b,c))
```

```
El resultado de sumar 87 y 34 es 121.
```

2. Variables y estructuras de datos (SARA)

Las variables en Python se crean cuando se definen por primera vez, es decir, cuando se les asigna un valor por primera vez (utilizando el operador `=`). Por esta razón, en Python no hace falta declarar las variables.

Los principales tipos de variables que se pueden encontrar son:

- **Numéricas**
  - **Integer (int):** Números naturales positivos y negativos.
  - **Float:** Números reales.
  - **Complex:** Números complejos. La parte imaginaria se multiplica por `j` para representar la raíz de `-1`. Por ejemplo: `a = 2+3j`
- **Boleanas** Son variables que pueden tomar como valor cierto o falso, representados en Python por `True` y `False` usando mayúsculas.
- **Estructuras de datos (secuencias de datos)** Lista ordenada de valores del mismo o distinto tipo. En Python existen:
  - **String:** cadenas de texto
  - **Lista:** conjuntos ordenados de elementos
  - **Tupla:** conjuntos ordenados e inmutables de elementos
  - **Diccionario:** conjuntos de elementos caracterizados por un identificador y un valor

Existen algunas reglas generales para los nombres de variables:

- Se usan solo letras, números y "guiones bajos" (_)
- NO se usan espacios en ningún lugar del nombre
- El nombre no puede empezar con un número
- Las letras mayúsculas se tratan como distintas de las minúsculas (es decir, Python distingue entre mayúsculas y minúsculas)

In [3]:
```
# El nombre no puede empezar como un número
3a = 10
```

```
  File "<ipython-input-3-cbdb5ae4000b>", line 2
    3a = 10
     ^
SyntaxError: invalid syntax
```

Cada vez que se crea o modifica una variable, Python interpreta qué tipo de variable es. Cuando se asigna un nuevo valor a un objeto existente (*escritura dinámica*), los valores anteriores de ese objeto se borran de la memoria de la computadora. El primer valor de **a** es el número real 13.0, pero después de una nueva asignación:

In [4]:
```
# Escritura dinámica
a = 13.0
a = 'b'
print(a)
```

```
b
```

La función `type()` muestra el TIPO del dato que le pasamos como argumento:

In [5]:
```
# Declarar una variable a y mirar qué tipo de variable es utilizando "type(nombre_va

a = 2
type(a)
```

Out[5]:
```
int
```

In [6]:
```
# Probar de declarar otro tipo de variable y mirar qué tipo es

a = 2.0
type(a)
```

Out[6]:
```
float
```

2.1 Tipos y operadores básicos

Los operadores básicos en Python son:

- Suma (también para strings, tuplas o listas): `a + b`
- Resta: `a - b`
- Multiplicación (también para strings, tuplas y listas): `a * b`
- División: `a / b`
- Division entera: `a // b`
- Resto de la división o residuo: `a % b`
- Exponencial: `a ** b`
- Asignamiento: `=`, `-=`, `+=`, `/=`, `=`, `%=`, `//=`, `*=`
- Comparaciones booleanas: `==`, `!=`, `<`, `>`, `<=`, `>=`
- Operaciones booleanas: `and`, `or`, `not`
- Operaciones para comprobar pertenecencia: `in`, `not in`
- Operaciones para identificar objetos: `is`, `is not`

In [7]:

```
# SARA
# Operadores de asignación

a=7; b=2

print("Operadores de asignación")
x=a; x+=b;  print("x+=", x)  # 9
x=a; x-=b;  print("x-=", x)  # 5
x=a; x*=b;  print("x*=", x)  # 14
x=a; x//=b; print("x//=", x) # 3
```

```
Operadores de asignación
x+= 9
x-= 5
x*= 14
x//= 3
```

In [8]:

```
# SARA
# Comparaciones booleanas (operaciones relacionales)


a>=b
```

Out[8]:

```
True
```

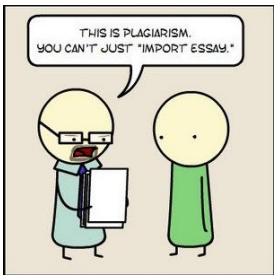
In [9]:

```
# SARA
# Operadores booleranos o lógicos

print(a>b and b<a)   # True
print(a==7 and b==5)   # True

print(a==7 or b==5)   # True
```

```
True
False
True
```

Módulos

Algunas funciones matemáticas más complejas no están disponibles en el módulo básico de Python y deben importarse desde un módulo específico. Para importar librerías, solo se debe escribir "import + el nombre de la librería".



En cada caso se deberá leer bien la documentación de las librerías usadas para concocer sus características y saber como implementarlas. Por ejemplo, para math:

https://docs.python.org/3/library/math.html (https://docs.python.org/3/library/math.html)

In [14]:

```
import math   # solo se ejecutará si la librería no ha estado importada previamente
print(math.pi + math.sin(100) + math.ceil(2.3))  # The math.ceil() method rounds a n
```

```
5.635227012480034
```

Operadores booleanos

In [15]:

```
a = 4
b = 40
(a>2) and (b>30)
```

Out[15]:

```
True
```

In [16]:

```
(a>2) or (b>100)
```

Out[16]:

```
True
```

In [10]:

```
# SARA
# Operadores para identificar pertenencia

mi_lista = [1, 3, 2, 7, 9, 8, 6]
print(4 in mi_lista)
```

```
False
```

In [11]:

```
# SARA
# Operadores para identificar objetos

a = 7
b = 2
c = 7
print(a is c) # muestra True
print (a is not b) # muestra True
print (a is not c) # muestra False
```

```
True
True
False
```

Python como calculadora

Python tiene una notación concisa para la aritmética que se parece mucho a la forma tradicional de escribir operaciones.

In [12]:

```
a = 3+2
b= 3.5 * -8
c = 10/6
print(a, b, c, 10./6.)
```

```
5 -28.0 1.6666666666666667 1.6666666666666667
```

In [13]:

```
# Ejemplo

a = 12
b = 13.3
c = a + b
print (c, type(a), type(b), type(c))
```

```
25.3 <class 'int'> <class 'float'> <class 'float'>
```

In [17]:

```
not(a>2)
```

Out[17]:

```
False
```

2.2 Estructuras de datos

Strings o cadenas de texto

El texto en Python se trata como una lista de carcateres. De esta manera, la metodología que aplica al tratamiento de listas, puede ser aplicada a las cadenas de texto.

In [18]:

```
a = 'python'
type(a)
```

Out[18]:

```
str
```

In [19]:

```
print("Hello" )
```

```
Hello
```

In [20]:

```
print("Esto es 'un ejemplo' de como usar comillas dentro de un texto que se escribe
print('Aquí tenemos "otro ejemplo" pero ahora al revés')
```

```
Esto es 'un ejemplo' de como usar comillas dentro de un texto que se e
scribe entre comillas dobles
Aquí tenemos "otro ejemplo" pero ahora al revés
```

Para juntar cadenas de carácteres se puede usar el operador '+':

In [21]:

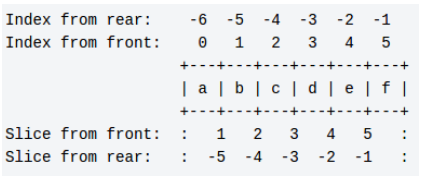
```
a = 'He'
b = 'llo'
c = a+b+'!'
print(c)
```

```
Hello!
```

Para acceder a solo una parte de una lista o, en este caso, de una *string*, se usa la técnica denominada **slicing**. El formato a seguir para coger solo una parte de la string es el siguiente:

nombre_variable[indice_primer_elemento:indice_fin

Donde el funcionamiento de los índices se pueden ver en la siguiente imagen



In [22]:

```
#Ejemplos
a = 'Python'
print(a[:], a[0], a[2:], a[:3], a[2:4], a[::2], a[1::2])
```

```
Python P thon Pyt th Pto yhn
```

Ejercicio

Con una línia de codigo, conseguir los siguientes sets de caracteres:

- Primera letra.
- De la letra 2 a la 4
- Úlimas 3 letras
- Solo las letras en posición par, des de la número 4 al final
- Todo el texto al revés

In [23]:

```
texto = "Texto de prueba para hacer el ejercicio"
primera_letra = print(texto[0])

#########################################################
print('De la letra 2 a la 4:',texto[2:4])
print('Últimas tres letras:', texto[-3:])
print('Solo las letras en posición par, des de la número 4 al final:', texto[4::2])
print('Todo el texto al revés:', texto[::-1])
```

```
T
De la letra 2 a la 4: xt
Últimas tres letras: cio
Solo las letras en posición par, des de la número 4 al final: od reapr
ae leecco
Todo el texto al revés: oicicreje le recah arap abeurp ed otxeT
```

En el caso de los *strings* hay métodos en Python que vienen por defecto que nos pueden ser útiles. Por ejemplo las funciones siguientes:

- len: devuelve la longitud el texto.
- str: convierte eun número en string.
- reverse: devuelve el texto al revés.

Para usar una fucnión se pondrá nombre_de_la_función(nombre_variable)

También podemos usar métodos propios para strings:

- lower: convierte todo el texto en mínusculas.
- upper: convierte todo el texto en mayúsculas.
- capitalize: pone en mayúsculas la primera letra del string.
- split(delimitador): devuelve una lista de substrings separados por el delimitador indicado (ver ejemplos más adelante)

Para usar un método se pondrá nombre_variable.nombre_del_métdodo

In [24]:

```
texto='vamos a hacer pruebas con las funciones y métodos de Strings!'
print(len(texto))
print(texto.lower())
print(texto.split(' '))

#########################################################################3
# En una línia, devuelve la 4a palabra del texto todo en mayúsculas!
print(texto[14:21].upper())
print(texto.capitalize())
```

```
61
vamos a hacer pruebas con las funciones y métodos de strings!
['vamos', 'a', 'hacer', 'pruebas', 'con', 'las', 'funciones', 'y', 'mé
todos', 'de', 'Strings!']
PRUEBAS
Vamos a hacer pruebas con las funciones y métodos de strings!
```

Lists

Las listas son un tipo de Python que sirven para agrupar objetos de otros tipos (incluso listas!). Por ejemplo una lista de números, de floats, de strings, de otras listas, etc.

Las listas, como los strings, pueden usar la metodología de *slicing*

In [25]:

```
l=[]
type(l)
```

Out[25]:

```
list
```

In [26]:

```
# Crear lista
mi_lista = [0,5,'6',34,'Hola']
mi_lista
```

Out[26]:

```
[0, 5, '6', 34, 'Hola']
```

In [27]:

```
# Acceder a un elemento de la lista
a = mi_lista[2]
a
```

Out[27]:

```
'6'
```

In [28]:

```
# Canviar un elemento de una lista
mi_lista[-1]='Adiós'
mi_lista
```

Out[28]:

```
[0, 5, '6', 34, 'Adiós']
```

In [29]:

```
# Podemos tener lista de listas!
mi_lista=[45,[3,5,6],'palabra',34.6,['a','b','c']]
print(mi_lista)
print(mi_lista[1])
print(mi_lista[1][2])

#########################################################
# acceder a la letra c!
mi_lista[-1][2]
mi_lista[4][2]
```

```
[45, [3, 5, 6], 'palabra', 34.6, ['a', 'b', 'c']]
[3, 5, 6]
6
```

Out[29]:

```
'c'
```

Para las listas también hay funciones y métodos en Python que vienen por defecto que nos pueden ser útiles. Por ejemplo las funciones siguientes:

- len: devuelve la longitud de la lista.
- list: crear lista

Para usar una fucnión se pondrá nombre_de_la_función(nombre_variable)

También podemos usar métodos propios para listas:

- append: añadir elemento al final de la lista.
- insert: añdir un elemento en una posición particular.
- pop: quitar el último elemento de la lista.
- remove: quitar un elemento en particular.
- sort: ordenar la lista

Para usar un método se pondrá nombre_variable.nombre_del_métdodo

In [30]:

```
mi_lista=['primero','hola','prueba','adiós','zoológico','máster']
print(mi_lista)
mi_lista.append('último')
print(mi_lista)
mi_lista.insert(1,'segundo')
print(mi_lista)
mi_lista.pop()
print(mi_lista)
mi_lista.remove('prueba')
print(mi_lista)
mi_lista.sort()
print(mi_lista)
```

```
['primero', 'hola', 'prueba', 'adiós', 'zoológico', 'máster']
['primero', 'hola', 'prueba', 'adiós', 'zoológico', 'máster', 'últim
o']
['primero', 'segundo', 'hola', 'prueba', 'adiós', 'zoológico', 'máste
r', 'último']
['primero', 'segundo', 'hola', 'prueba', 'adiós', 'zoológico', 'máste
r']
['primero', 'segundo', 'hola', 'adiós', 'zoológico', 'máster']
['adiós', 'hola', 'máster', 'primero', 'segundo', 'zoológico']
```

Dictionaries

Un diccionario es otra manera de agrupar datos donde se puede acceder a un *element* a través de su *key*. Por ejemplo:

In [31]:

```
esta_instalacion = {
    'Potencia': 10,
    'Tipo': 'Solar',
    'Año_construccion':2012
}
```

In [32]:

```
esta_instalacion['Tipo']
```

Out[32]:

```
'Solar'
```

Otro ejemplo podría ser:

In [33]:

```
dict = {"Instalacion_1": "Solar", "Instalacion_2": "Eólica", "Instalacion_3": "Eólic
print(dict)
```

```
{'Instalacion_1': 'Solar', 'Instalacion_2': 'Eólica', 'Instalacion_3':
'Eólica', 'Instalacion_4': 'Hidráulica', 'Instalacion_5': 'Solar'}
```

Para acceder a algun elemento, modificar o añadir, se podrá hacer de forma parecida a las listas:

In [34]:

```
# Acceder
dict["Instalacion_1"]
```

Out[34]:

```
'Solar'
```

In [35]:

```
# modificar
print(dict)
dict["Instalacion_1"]="Eólica"
print(dict)
```

```
{'Instalacion_1': 'Solar', 'Instalacion_2': 'Eólica', 'Instalacion_3':
'Eólica', 'Instalacion_4': 'Hidráulica', 'Instalacion_5': 'Solar'}
{'Instalacion_1': 'Eólica', 'Instalacion_2': 'Eólica', 'Instalacion_
3': 'Eólica', 'Instalacion_4': 'Hidráulica', 'Instalacion_5': 'Solar'}
```

In [36]:

```
# Añadir
print(dict)
dict["Instalacion_6"]="Solar"
print(dict)
```

```
{'Instalacion_1': 'Eólica', 'Instalacion_2': 'Eólica', 'Instalacion_
3': 'Eólica', 'Instalacion_4': 'Hidráulica', 'Instalacion_5': 'Solar'}
{'Instalacion_1': 'Eólica', 'Instalacion_2': 'Eólica', 'Instalacion_
3': 'Eólica', 'Instalacion_4': 'Hidráulica', 'Instalacion_5': 'Solar',
'Instalacion_6': 'Solar'}
```

Los diccionarios pueden contener todos los tipos (int, floats, listas, otros diccionarios, etc.)

In [37]:

```
dict = {"Instalacion_1": ["Solar", '10 MW'], "Instalacion_2": ["Eólica", '15 MW'], "
dict
```

Out[37]:

```
{'Instalacion_1': ['Solar', '10 MW'],
 'Instalacion_2': ['Eólica', '15 MW'],
 'Instalacion_3': ['Solar', '18 MW'],
 'Instalacion_4': ['Hidráulica', '12 MW'],
 'Instalacion_5': ['Solar', '23 MW'],
 'Instalacion_6': ['Eólica', '8 MW'],
 'Instalacion_7': ['Solar', '11 MW'],
 'Instalacion_8': ['Hidráulica', '11 MW']}
```

In [38]:

```
dict['Instalacion_3'][1]
```

Out[38]:

```
'18 MW'
```

In [39]:

```
############################################################
# Suma 3 MW a la instalación 4 y modifica el diccionario!
dict['Instalacion_4'][1]='15 MW'
print(dict['Instalacion_4'])
```

```
['Hidráulica', '15 MW']
```

In [40]:

```
# SARA
# Ejemplos de dict keys con números

dict = {
    0: 15,
    1: 16,
    2: 55
}

print(dict)
print(dict[2])

dict2 = {

    (0,0): 1,
    (0,1): 2,
    (0,1): 3
}

print(dict2)
print(dict2[0,0])
```

```
{0: 15, 1: 16, 2: 55}
55
{(0, 0): 1, (0, 1): 3}
1
```

Tuples

Tuples son listas **que no permiten ser modificadas!**.

Su sintaxis es igual que en las listas, però para escribir una tupla se usan paréntesis y no corchetes.

In [41]:

```
misupertupla = ('Solar', 'Eólica', 'Hidráulica', 'Gas', 'Nuclear', 'NoQueremosMásCar
print(type(misupertupla), misupertupla[4:])
```

```
<class 'tuple'> ('Nuclear', 'NoQueremosMásCarbón')
```

In [42]:

```
misupertupla[0]='PV'
```

```
---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
 last)
<ipython-input-42-3c72c87d7787> in <module>
----> 1 misupertupla[0]='PV'

TypeError: 'tuple' object does not support item assignment
```

3. Estructuras condicionales (if)

Las estructuras de control condicionales permiten ejecutar una parte del código o otro en función de la evaluación de una o varis condiciones booleanas de SI o NO (**`TRUE`** o **`FALSE`**).

En Python, las estructuras de control condicionales se definen mediante las palabras **`if`** , **`elif`** y **`else`** .

- **`if CONDICION:`** si se cumple la expresión condicional se ejecuta el bloque de código a continuación.
- **`elif CONDICION:`** de lo contrario, si se cumple esta expresión condicional se ejecuta este otro bloque de código.
- **`else:`** de lo contrario, se ejecuta este bloque de código a continuación sin evaluar ninguna condición.

La condición es normalmente evaluada mediante los siguientes operadores relacionales `<, <=, ==, >=, >, !=` .

La parte de código dentro de un `if` no pueden estar vacía, pero si por alguna razón se debe crear una declaración if sin contenido, se puede utilizar la declaración `pass` para evitar de se produzca un error.

In [44]:

```
# Ejemplo

a = 33
b = 200

if b > a:
  pass
```

In [45]:

```
# Ejemplo

celsius = 35
fahrenheit = 9.0 /5.0 * celsius + 32

print("La temperatura en Fahrenheit es", fahrenheit)

if fahrenheit > 90:
    print("Hace calor")
elif fahrenheit < 30:
    print("Hace frío")
else: print("No hace ni frío ni calor")
```

```
La temperatura en Fahrenheit es 95.0
Hace calor
```

Ejercicio

Crear un código para identificar si la variable **`numero`** es un numero positivo, negativo, o de valor 0. Guardar el resultado en una variable llamada **`signo`**

In [46]:

```
numero = 2

##############################
# SARA

if numero > 0:
    signo = 'positivo'
elif numero < 0:
    signo = 'negativo'
else:
    signo = 'cero'


print(signo)
```

```
positivo
```

Tambiém se puede expresar en una sola linea la estructura condicional de la siguiente manera, dependindo de si se trata de una, dos o más condiciones:

In [47]:

```
#Ejemplo

a = 330
b = 330

# con una sola condición:
if a > b: print("a es mayor que b")

#con dos condiciones:
print("A") if a > b else print("B")

#con tres condiciones:
print("A") if a > b else print("=") if a == b else print("B")
```

```
B
=
```

In [48]:

```
# Ejercicio: poner en una sola linea el código para determinar el signo de un número
numero = 2

##########################################
# SARA

#con tres condiciones:
print("Positivo") if numero > 0 else print("Negativo") if numero < 0 else print("Cer
```

```
Positivo
```

4. Listas y loops

4.1 Estructuras de control iterativas (for & while)

Las estructuras de control iterativas (también llamadas bucles o loops), permiten ejecutar un mismo código de manera repetida mientras se cumpla una condición.

Existen dos tipos de estructuras de control iterativas:

- For
- While

La sentencia `for` de Python itera sobre los elementos de cualquier secuencia (una lista o una cadena de caracteres por ejemplo), en el orden que aparecen en la secuencia.

El ciclo `while` permite realizar múltiples iteraciones basándose en el resultado de una expresión lógica que puede tener como resultado un valor `True` o `False` .

In [49]:

```
#Ejemplo de for

for i in range(0,10,1):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [50]:

```
#Ejemplo de while

i=0
while i < 10 :
    print(i)
    i= i+1
```

```
0
1
2
3
4
5
6
7
8
9
```

Además, los bucles con `for` o `while` pueden combinarse con estructuras condicionales `if` dando lugar a estructuras más complejas.

Con la instrucción `break` podemos detener el bucle antes de que haya pasado por todos los elementos.

Con la instrucción `continue` podemos detener la iteración actual del bucle (tanto en `for` como en `while`) y continuar con la siguiente.

In [51]:

```
# Ejemplo

numeros = [-5, 3, 2, -1, 9, 6]
total = 0

for n in numeros:
    if n >= 0:
        total += n

total2 = 0
for n in numeros:
    if n < 0:
        continue
    total2 += n

print(total, total2)
```

```
20 20
```

In [52]:

```
#Ejercicio: calcular el valor medio de una lista con un for

lista=[1,2,3,4]

#################### SARA

suma = 0
for n in lista:
    suma += n

media = suma/len(lista)
print(suma, media)
```

```
10 2.5
```

In [53]:

```
#Ejercicio: rehacer el código anterior para calcular el valor medio con un while en

lista=[1,2,3,4]

# SARA
########################################

media = 0
suma=0
i=0

while i < len(lista):

    print(i)
    suma += float(lista[i])
    i +=1

media = suma/len(lista)
print(suma, media)
```

```
0
1
2
3
10.0 2.5
```

4.2 Lists (o dictionary) comprehensions

Las comprensiones de listas son una forma de ajustar un bucle `for` , una declaración `if` y una asignación, todo en una sola línea.

Una lista de comprensión consta de las siguientes partes:

- Una secuencia de entrada
- Una variable que representa miembros de la secuencia de entrada
- Una expresión opcional
- Una expresión de salida que produce elementos de la lista de salida a partir de miembros de la secuencia de entrada que satisfacen el predicado

In [54]:

```
# Ejemplo
num = [1, 4, -5, 10, -7, 2, 3, -1]
squared = []
for i in num:
     if i > 0:
        squared.append(i**2)
print(type(squared), squared)

# Este for se puede reescribir de la siguiente manera:
# En una iteración, primero comprueba si el if es TRUE y luego realiza el cálculo
squared = [ x**2 for x in num if x > 0]
print(type(squared), squared)
```

```
<class 'list'> [1, 16, 100, 4, 9]
<class 'list'> [1, 16, 100, 4, 9]
```

In [55]:

```
# Ejercicio: calcular el valor medio de una lista con un for de una sola linea

# SARA
########################################################
import numpy

lista=[1,2,3,4]

suma = 0

mean = numpy.mean([x for x in lista])
print(type(mean), mean)
```

```
<class 'numpy.float64'> 2.5
```

Esta forma compacta de expresar los bucles también se puede utilizar para iterar dentro de listas. A contnuación se muestran diferentes maneras de juntar los valores de dos listas usando zip o usando fors anidados.

In [56]:

```
lista1 = ['a','b','c','d']
lista2 = [1,2,3,4]

# crear una lista con fors anidados:
print ([(i,j) for i in lista1 for j in lista2])

# crear una lista usando zip:
print ([(i,j) for i,j in zip(lista1,lista2)])

# crear un diccionario usando zip:
print ( {i:j for i,j in zip(lista1,lista2)} )
```

```
[('a', 1), ('a', 2), ('a', 3), ('a', 4), ('b', 1), ('b', 2), ('b', 3),
('b', 4), ('c', 1), ('c', 2), ('c', 3), ('c', 4), ('d', 1), ('d', 2),
('d', 3), ('d', 4)]
[('a', 1), ('b', 2), ('c', 3), ('d', 4)]
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

5. Diccionarios y tablas de frecuencia (SARA)

Los diccionarios son muy útiles para contar elementos en una lista, y crear una distribución de estos mediante una tabla de frecuencia. A continuación se puede observar un ejemplo donde se quiere saber qué número del 1 al 9 aparece más veces en una lista desordenada de números.

In [57]:

```
mi_lista = [2,3,5,2,5,6,7,8,1,2,3,4,5,2,8,9,5,2,4,6,9,1,2,1,3,6,7,4,3,1,2,9,7,5,4,3,
```

In [58]:

```
# Creamos un diccionario vacío
freq = {}
for numero in mi_lista:
    if (numero in freq):
        freq[numero] += 1
    else:
        freq[numero] = 1

for key, value in freq.items():
    print ("% d : % d"%(key, value))
```

```
 2 :  9
 3 :  8
 5 :  8
 6 :  4
 7 :  5
 8 :  2
 1 :  5
 4 :  7
 9 :  5
```

De una lista de instalaciones, obtenemos estas tipologías de plantas de generación. ¿Qué tecnología tiene más plantas instaladas?

¿Detectas algún problema con los nombres? ¿Cómo lo solucionarías?

In [59]:

```
instalaciones = ['Eólica', 'Solar', 'Hidráulica', 'Solar', 'Eólica', 'Hidráulica', '
instalaciones
```

Out[59]:

```
['Eólica',
 'Solar',
 'Hidráulica',
 'Solar',
 'Eólica',
 'Hidráulica',
 'Gas Natural',
 'Solar',
 'Eólica',
 'Solar',
 'Nuclear',
 'Solar',
 'Hidráulica',
 'Hidráulica',
 'Solar',
 'Carbón',
 'Eólica',
```

In [60]:

```
################################################
# ¿Qué tecnología tiene más plantas instaladas?


dict = {}
count= 0
itm = ''

for item in instalaciones:
    dict[item] = dict.get(item, 0) + 1
    if dict[item] >= count :
        count = dict[item]
        freq = item

print(dict)

print('Tecnología más frecuente: ', freq)  # Most frequent value
```

```
{'Eólica': 10, 'Solar': 17, 'Hidráulica': 11, 'Gas Natural': 5, 'Nucle
ar': 3, 'Carbón': 1, 'hidráulica': 1, 'Eolica': 1, 'eolica': 1, 'eólic
a': 1}
Tecnología más frecuente:  Solar
```

In [61]:

```
# Peeero algo no cuadra. Eolica está esrito de diferentes maneras. Vamos a arreglar

# Vamos a poner la primera letra en mayúscula con capitalize() y además. poner acent
for item in range(len(instalaciones)):
    instalaciones[item] = instalaciones[item].capitalize()

    if instalaciones[item] == 'Eolica':
        instalaciones[item] = 'Eólica'

print(instalaciones)
```

```
['Eólica', 'Solar', 'Hidráulica', 'Solar', 'Eólica', 'Hidráulica', 'Ga
s natural', 'Solar', 'Eólica', 'Solar', 'Nuclear', 'Solar', 'Hidráulic
a', 'Hidráulica', 'Solar', 'Carbón', 'Eólica', 'Eólica', 'Solar', 'Sol
ar', 'Solar', 'Hidráulica', 'Gas natural', 'Hidráulica', 'Nuclear', 'E
ólica', 'Eólica', 'Solar', 'Eólica', 'Gas natural', 'Hidráulica', 'Hid
ráulica', 'Solar', 'Eólica', 'Hidráulica', 'Solar', 'Eólica', 'Solar',
'Eólica', 'Eólica', 'Solar', 'Solar', 'Hidráulica', 'Gas natural', 'So
lar', 'Gas natural', 'Nuclear', 'Eólica', 'Hidráulica', 'Hidráulica',
'Solar']
```

In [62]:

```
# Volvemos a ver que valor es el más frecuente
# Observamos que efectivamente, Eólica está toda en una palabra.

dict = {}
count= 0
itm = ''

for item in instalaciones:
    dict[item] = dict.get(item, 0) + 1
    if dict[item] >= count :
        count = dict[item]
        freq = item

print(dict)
print(freq)  # Most frequent value
```

```
{'Eólica': 13, 'Solar': 17, 'Hidráulica': 12, 'Gas natural': 5, 'Nucle
ar': 3, 'Carbón': 1}
Solar
```

In [63]:

```
from statistics import mode

print(mode(instalaciones))
```

```
Solar
```

6. Funciones

Una función es un bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias donde se ejecutan las operaciones deseadas y devuelve un valor y/o realiza una tarea. Este bloque de código puede ser llamado varias veces cuando se necesite.

En Python existen una serie de funciones integradas por defecto al lenguaje, pero también se pueden crear funciones definidas por el usuario para utilizarlas en sus propios programas.

La lista completa de funciones integradas por defecto puede consultarse en:
www.w3schools.com/python/python_ref_functions.asp
(http://www.w3schools.com/python/python_ref_functions.asp)

In [64]:

```
# Ejemplo de función integrada en Python

# La función input() nos permite asignar a una variable un valor ingresado por el us

a = input()

print('La variable introducida ha sido ' + a)
```

```
8
La variable introducida ha sido 8
```

Para crear una función en python, se debe utilizar la palabra `def` para definir la función. La sintaxis para una definición de función en Python es:

```
def nombre_funcion(lista_de_parametros_entrada):
    sentencias_funcion
    return [expresion_a_devolver]
```

Donde:

- nombre_funcion: es el nombre de la función.
- lista_de_parametros_entrada: es la lista de parámetros que puede recibir una función, separados por coma.
- sentencias_funcion: es el bloque de sentencias en código fuente Python que realizar cierta operación dada.
- expresion_a_devolver: es la expresión o variable que devuelve la sentencia return.

In [65]:

```
# Ejemplo: función para poner a 0 el primer valor de una lista

def change_first_element(list):
    list[0]=0 #it returns none!

numbers=[1,2,3,4]
change_first_element(numbers)
print(numbers)
```

```
[0, 2, 3, 4]
```

Ejemplo: máximo común denominador

El común denominador de dos enteros positivos a y b es el mayor divisor común entre a y b. El algortimo Euclideo es un método iterativo para calcular el máximo común denominador de dos enteros. En pseudocódigo seria:

- Si $a < b$, intercanviar a y b.
- Dividir a entre b y obtener el residuo, r.
- Si $r = 0$, el valor b es el MCD de a y b.

- Si $r \neq 0$, iterar de nuevo reemplazando a por b y reemplazando b por r.

In [66]:

```
def mcd(a,b):
    r = 1
    while r != 0:
        if a<b:
            c=a
            a=b
            b=c
        r = a%b
        if r == 0:
            return b
        else:
            a = b
            b = r

mcd(100,16)
```

Out[66]:

```
4
```

Ejercicio: crear una función para calcular el factorial de un entero

El factorial de un entero no negativo n es el producto de todos enteros positivos menores o iguales a n.

In [67]:

```
# Poner tu código aquí:
# SARA
########################################

def fact_1(n):
    factorial_total = 1
    while n > 1:
        factorial_total *= n
        n -= 1
    return factorial_total

fact_1(5)

# from math import factorial
# factorial(5)
```

Out[67]:

```
120
```

6.1 Argumentos y parámetros de funciones

Al definir una función, los valores que se reciben se denominan parámetros, mientras que durante la llamada, los valores que se envían se denominan argumentos.

Por defecto, se debe llamar a una función con el número correcto de argumentos. Si la función espera 2 argumentos, debe llamar a la función con 2 argumentos.

Argumentos por posición

Cuando se envian argumentos a una función, por defecto estos se reciben por orden en los parámetros definidos, lo que se conoce como argumentos por posición. En el ejemplo anterior, el argumento 100 es la posición 0, que se corresponde al parámetro de la función a, mientras que el argumento 16 es la posición 1, que se corresponde al parámetro de la función b.

Argumentos por nombre

Sin embargo, es posible saltarse el orden de los parámetros si se indica durante la llamada que valor tiene cada parámetro a partir de su nombre.

In [68]:

```
mcd(b=100,a=16)
```

Out[68]:

```
4
```

Parámetros por defecto

Para evitar mensajes de error cuando no se llama a una función con los argumentos necesarios, se pueden definir parámetros por defecto dentro de una función.

In [69]:

```
def mcd(a=None,b=None):
    if a == None or b == None:
        print ("Error, debes enviar dos números a la función")
        return
    r = 1
    while r != 0:
        if a<b:
            c=a
            a=b
            b=c
        r = a%b
        if r == 0:
            return b
        else:
            a = b
            b = r

mcd()
```

```
Error, debes enviar dos números a la función
```

Parámetros indeterminados `*args` y `**kwargs`

En algunas ocasiones, no se sabe con anterioridad cuantos elementos es necesario enviar a una función.

Si no se sabe cuántos argumentos por posición se pasarán a una función, se agrega un * antes del nombre del parámetro en la definición de la función. Por convenio, se suele utilizar el nombre de parámetro `*args` . De esta forma, la función recibirá una tupla de argumentos y podrá acceder a los elementos en consecuencia.

De forma similar, si no se sabe cuántos argumentos por nombre se pasarán a una función, se agregan dos ** antes del nombre del parámetro en la definición de la función. Por convenio, se suele utilizar el nombre de parámetro `**kwargs` . De esta forma, la función recibirá un diccionario de argumentos y podrá acceder a los elementos en consecuencia.

In [70]:

```
def super_funcion(*args,**kwargs):
    total = 0
    for arg in args:
        total += arg
    print ("suma => ", total)
    for kwarg in kwargs:
        print (kwarg, "=>", kwargs[kwarg])

super_funcion(50, -1, 1.56, 10, 20, 300, nombre="Pedro", edad=38)
```

```
suma =>  380.56
nombre => Pedro
edad => 38
```

7. Funciones avanzadas (SARA)

En este apartado se verá como en alguna ocasiones, hay funciones simples que solo queremos usar una vez. Este uso no parece muy apropiado para las funciones de las que hemos hablado hasta ahora... Vamos a ver primero ejemplos dónde este tipo de funciones nos pueden ser de utilidad. Se trata de las funciones `map, filter y reduce .`

7.1 Map

Con un "map" podemos aplicar una función a toda una lista de inputs. Por ejemplo si queremos coger una lista de números y aplicar el cuadrado a todos sus elementos podríamos hacer:

In [71]:

```
items = [1, 2, 3, 4, 5]
squared = []
for i in items:
    squared.append(i**2)
print(items)
print(squared)
```

```
[1, 2, 3, 4, 5]
[1, 4, 9, 16, 25]
```

Usando `Map` , tendríamos:

In [72]:

```
def square_item (x):
    return x**2
squared = list(map(square_item, items))
print(items)
print(squared)
```

```
[1, 2, 3, 4, 5]
[1, 4, 9, 16, 25]
```

Como se puede ver `map` es una función que acepta una función como argumento

```
map(funcion a aplicar, lista de entrada)
```

7.2 Filter

La función filter de una manera similar, devuelve una lista con aquellso elementos de una lista de entrada que cumple una cierta codició (filtra la lista de etrada con una condición determinada). Véamos como ejemplo, si queremos quitar los elementos negativos de una lista y quedarnos solo con los positivos y que sean pares

In [73]:

```
items=[2,5,-6,4,-1,2,7,-8,3,4,7,0,-1,-4,0,4,6,-8,0,1]
positivos=[]
for item in items:
    if item>=0:
        if item % 2 == 0:
            positivos.append(item)
print(items)
print(positivos)
```

```
[2, 5, -6, 4, -1, 2, 7, -8, 3, 4, 7, 0, -1, -4, 0, 4, 6, -8, 0, 1]
[2, 4, 2, 4, 0, 0, 4, 6, 0]
```

In [74]:

```
items=[2,5,-6,4,-1,2,7,-8,3,4,7,0,-1,-4,0,4,6,-8,0,1]
def es_positivo(x):
    return x>=0 and x%2==0
positivos=list(filter(es_positivo, items))
print(items)
print(positivos)
```

```
[2, 5, -6, 4, -1, 2, 7, -8, 3, 4, 7, 0, -1, -4, 0, 4, 6, -8, 0, 1]
[2, 4, 2, 4, 0, 0, 4, 6, 0]
```

Se puede observar como su sitaxis es muy parecisa a la funció `map` , pero usando ahora `filter`

```
filter(funcion a aplicar, lista de entrada)
```

7.3 Reduce

La función `Reduce` será muy útil cuando se quiera hacer un cálculo sobre una lista y obtener el resultado. Por ejemplo si queremos obtener el resultado de multiplicar los números de una lista haríamos:

In [75]:

```
producto = 1
lista = [1,5,7,3,4,9,1,4,7,5,3,9,8,7,2]
for num in lista:
    producto *= num
print(producto)
```

```
1600300800
```

Usando `reduce` :

In [76]:

```
from functools import reduce
lista = [1,5,7,3,4,9,1,4,7,5,3,9,8,7,2]
def multiplicar(x,y):
#     print('X', x)
#     print('Y', y)
    return x*y
producto = reduce(multiplicar, lista)
print(producto)
```

```
1600300800
```

Vemos que para usar `reduce` se tiene que importar de `functools` y que tiene una sintaxis idética a los ateriores casos: `reduce(funcion a aplicar, lista de entrada)`

7.4 Funciones Lambda

En todos estos casos hemos visto como consegumos comprimir el código, pero siempre tenemos que definir fuciones, normalmente muy simple, solo para ser usadas una vez. Para estas ocasiones, existen las `funciones anónimas` o `lambda functions` . Estas se pueden definir y usar en la misma línia, y nos ahorramos un montón de tiempo!!

Su sintaxis es:

```
lambda entradas: salida
```

Vamos a volver a escribir las mismas funciones que antes, perpo usado estas funciones para verlo más claro con ejemplos:

In [77]:

```
items = [1, 2, 3, 4, 5]
squared = list(map(lambda x:x**2, items))
print(items)
print(squared)
```

```
[1, 2, 3, 4, 5]
[1, 4, 9, 16, 25]
```

In [78]:

```
items=[2,5,-6,4,-1,2,7,-8,3,4,7,0,-1,-4,0,4,6,-8,0,1]
positivos=list(filter(lambda x: x>=0 and x%2==0, items))
print(items)
print(positivos)
```

```
[2, 5, -6, 4, -1, 2, 7, -8, 3, 4, 7, 0, -1, -4, 0, 4, 6, -8, 0, 1]
[2, 4, 2, 4, 0, 0, 4, 6, 0]
```

In [79]:

```
from functools import reduce
lista = [1,5,7,3,4,9,1,4,7,5,3,9,8,7,2]
producto = reduce(lambda x,y: x*y, lista)
print(producto)
```

```
1600300800
```

Ejercicio

Dada una lista de nombres de clientes, devolver la lista con todos los nombres con la primera letra en mayúsculas

In [80]:

```
nombres = ['jordi', 'arnau', 'laia', 'pedro', 'juan', 'pau', 'noemi', 'arya', 'marti
# vuestro código aquí!

########################################################################


# filter(funcion a aplicar, lista de entrada)
nombres_capital = list(map(lambda x:x.capitalize(), nombres))
print(nombres_capital)
```

```
['Jordi', 'Arnau', 'Laia', 'Pedro', 'Juan', 'Pau', 'Noemi', 'Arya', 'M
arti', 'Lyanna', 'Miquel', 'Joana', 'Ricard', 'Daenerys', 'Alexandra']
```

Otros usos

Hay más ocasiones donde una función pide otra función como argumeto, dode usar una función lambda será muy útil. Por ejemplo, si miramos la documentación de la función `sorted` que viene en Python por defecto:

https://docs.python.org/3/library/functions.html#sorted
(https://docs.python.org/3/library/functions.html#sorted)

In [81]:

```
desordenado = ['Martí','Anna','daniel','Pau','berta','Roberto','Xavier', 'alexandra'
sorted(desordenado)
```

Out[81]:

```
['Anna', 'Martí', 'Pau', 'Roberto', 'Xavier', 'alexandra', 'berta', 'd
aniel']
```

Si especificamos una función en el apartado `key` opcional:

In [82]:

```
sorted(desordenado, key=lambda x: x.lower())
```

Out[82]:

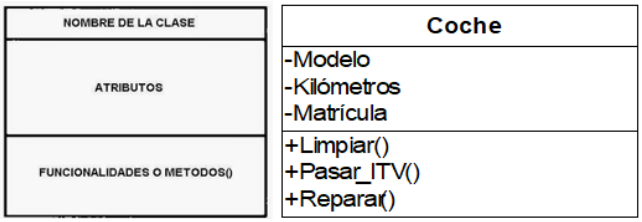
```
['alexandra', 'Anna', 'berta', 'daniel', 'Martí', 'Pau', 'Roberto', 'X
avier']
```

8. Lenguaje orientado a objetos

Python es un lenguaje de programación orientado a objetos. Casi todo en Python es un objeto, con sus propiedades y métodos. Una clase es como un constructor de objetos, o un "plano" para crear objetos.

Entre los conceptos asociados a una clase destacan los siguientes:

- Clase: una especie de "plantilla" en la que se definen los atributos y métodos predeterminados de un tipo de objeto.
- Objeto: instancia de una clase.
- Método: son funciones asociadas a un objeto o a una clase de objetos. Se podria decir que es lo que el objeto puede hacer.
- Atributos: son variables asociadas a un objeto o a una clase de objetos. Se podria decir que son las características que tiene una clase. Hay atributos de instancia para inicializar los atributos mínimos de la clase cuando se crea la clase, y atributos de clase para complmentar la descripción del objeto.



En Python se define una clase con la palabra reservada `class` .

El parámetro `self` es una referencia a la instancia actual de la clase y se usa para acceder a las variables que pertenecen a la clase.

Una vez creado un objeto, se pueden referenciar a sus atributos o invocar a un método por medio del operador `.` .

In [83]:

```
#Ejemplo: Clase de un rectangulo

class Rectangulo:
    """Clase de un rectangulo"""
    # lista de objetos
    x = 0
    y = 0

    # lista de métodos
    def area(self):
        return self.x * self.y

# Crear el objeto 'a' haciendo una instancia a la clase Rectangulo
a = Rectangulo()
# Inicializar x, y
a.x = 10
a.y = 4
# Calcular el area del rectangulo
a.area()
```

Out[83]:

```
40
```

8.1 Atributos de instancia `__init__()`

Todas las clases tienen una función llamada `__init __ ()` , que se ejecuta cuando se inicia la clase y sirve para inicializar los atributos de instancia. Los argumentos que se utilizan en la definición de `__init__()` corresponden a los parámetros que se deben ingresar al instanciar un objeto.

In [84]:

```
#Ejemplo: clase de un rectangulo

class Rectangulo:
    """Clase de un rectangulo"""

    # atributos de instancia
    def __init__(self,x,y):
        self.x = x
        self.y = y

    # lista de métodos
    def area(self):
        return self.x * self.y

a = Rectangulo(10,4)

a.area()
```

Out[84]:

```
40
```

Ejercicio

Añadir a la clase anterior un método para calcular el perímetro del rectángulo y otro método para cambiar el tamaño del rectángulo mediante un factor de escala.

In [85]:

```
# Añadir código aquí
# SARA
###################################

class Rectangulo:
    """Clase de un rectangulo"""

    # atributos de instancia
    def __init__(self,x,y):
        self.x = x
        self.y = y

    # lista de métodos
    def area(self):
        return self.x * self.y

    def perimetro(self):
        return (2*self.x + 2*self.y)

    def escala(self, escala):

        self.x = self.x*escala
        self.y = self.y*escala

        return (self.x, self.y)

a = Rectangulo(10,4)

a.perimetro()
a.escala(2)
```

Out[85]:

```
(20, 8)
```

8.2 Herencia

En programación orientada a objetos, la herencia es la capacidad de reutilizar una clase extendiendo su funcionalidad. Una clase que hereda de otra puede añadir nuevos atributos, ocultarlos, añadir nuevos métodos o redefinirlos.

Ejemplo

En el siguiente ejemplo, la clase alumno hereda de la clase persona para complementar los atributos que puede tener una persona.

In [86]:

```
class Persona:
    def __init__(self, nombre, apellido):
        self.nombre = nombre
        self.apellido = apellido

    def imprimirnombre(self):
        print(self.nombre, self.apellido)

class Estudiante(Persona):
    nota = 0
    clase = 'Clase sin introducir'

    def imprimirnota(self):
        print(self.nota)


x = Persona("Johann Sebastian", "Mastropiero ")
x.imprimirnombre()

y = Estudiante("Pedro", "Sanchez")
y.nota = 4.99
y.imprimirnombre()
y.imprimirnota()
```

```
Johann Sebastian Mastropiero
Pedro Sanchez
4.99
```

9. Introducción a NumPy

NumPy es una librería de Python utilizada para trabajar con matrices. También tiene funciones para trabajar con álgebra lineal o transformadas de Fourier. NumPy significa Numerical Python.

Como hemos visto, en Python ya tenemos listas que sirven al propósito de trabajar con matrices, pero su procesamiento es lento. NumPy tiene como objetivo proporcionar un objeto de matriz que sea hasta 50 veces más rápido que las listas tradicionales de Python, algo importante para la ciencia de datos donde la velocidad y los recursos son muy importantes. Además, se diferencian de las listas en que las matrices de NumPy no se pueden cambiar de tamaño, todos los elementos deben ser del mismo tipo, y se permiten ls operaciones entre matrices.

El código fuente de NumPy se encuentra en este repositorio de github
https://github.com/numpy/numpy (https://github.com/numpy/numpy)

El objeto de matriz en NumPy se llama `ndarray`, proporciona muchas funciones de soporte que hacen que trabajar con `ndarray` sea muy fácil. Se puede crear una ndarray con NumPy con la función `.array()`

In [87]:

```
#Ejemplo
import numpy as np #normalmente se suele importar como np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

Para crear un ndarray, podemos utilizar una lista, tupla o cualquier objeto tipo matriz al método `array()`, y se convertirá en un `ndarray`. Se pueden crear `ndarrays` de diferentes dimensiones. Los 0-D son solo números, los 1-D son vectores, los 2-D son matrices, etc.

Los principales atributos que se pueden utlizar con `ndarrays` son:

Atributo	Descripción
ndim	número de dimensiones de la matriz
shape	número de elementos de cada dimensión de la matriz
size	número de elementos total de la matriz
dtype	tipo de los elementos de la matriz
astype	para cambiar el tipo de los elementos de la matriz

In [88]:

```
#Ejemplo

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
print(d.shape)
print(d.size)
print(d.dtype)

e = d.astype(float)

print(e)
print(e.dtype)
```

```
0
1
2
3
(2, 2, 3)
12
int32
[[[1. 2. 3.]
  [4. 5. 6.]]

 [[1. 2. 3.]
  [4. 5. 6.]]]
float64
```

9.1 Acceder a valores de un ndarray

Se puede acceder a un elemento de la matriz haciendo referencia a su número de índice. Los índices en las matrices NumPy comienzan con 0, lo que significa que el primer elemento tiene índice 0, y el segundo tiene índice 1, etc.

Para acceder a elementos de matrices 2-D podemos usar enteros separados por comas que representan la dimensión y el índice del elemento. Para matrices 3-D o superior, se deben añadir tantos enteros separados por comas como dimensiones tenga la matriz.

In [89]:

```
#Ejemplo

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print(arr[0, 1])
```

```
2
```

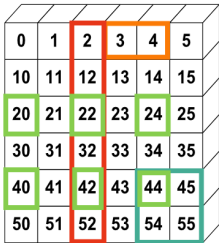
También se puede utilizar el slicing en `ndarrays`. En este caso:
`array[indice_primer_elemento:indice_final:step]`

In [90]:

```
#Ejercicio

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

# devuelve el indice 2 de las dos dimensiones:

arr[:, 2]

# devuelve el intervalo entre el índice 1 y 4 (no incluido) de las dos dimensiones:
arr[:4, 2]

# devuelve todos los elementos de posiciones pares:
```

Out[90]:

```
array([3, 8])
```



In [91]:

```
#Ejercicio: seleccionar los elementos de cada uno de los conjuntos de elementos de l

arr = np.array([[0,1,2,3,4,5],[10,11,12,13,14,15],[20,21,22,23,24,25],[30,31,32,33,3
arr

# Nueva array con selección elementos rojos:
arr2 = arr[:,2]
print(arr2)
# Nueva array con selección elementos naranjas:
arr3 = arr[0, 3:5]
print(arr3)
# Nueva array con selección elementos verdes:
arr4 = arr[2::2, ::2]
print(arr4)
# Nueva array con selección elementos azules:
arr5 = arr[-2:, -2:]
print(arr5)
```

```
[ 2 12 22 32 42 52]
[3 4]
[[20 22 24]
 [40 42 44]]
[[44 45]
 [54 55]]
```

9.2 Importar arrays de archivos .csv

Como los archivos ** Comma-separated values (CSV) ** son un tipop de archivo .TXT, se puede importar un archivo .CSV con la función `np.genfromtxt(ubicacion_archivo)`

Los parametros opcionales de **genfromtxt** pueden consultarse en:
https://numpy.org/doc/stable/reference/generated/numpy.genfromtxt.html
(https://numpy.org/doc/stable/reference/generated/numpy.genfromtxt.html)

Función	Tipo	Descripción
skip_header	opcional	número de líneas a evitar al inicio del fichero
filling_values	opcional	valores a utilizar cunado no haya valores
delimiter	opcional	string utilizado para separar valores (en CSV es ',')
usecols	opcional	seleciona que columnas importar, siendo 0 la primera (e.g. `usecols = (1, 4, 5)`)

Ejercicio:

Observar el archivo **Data/block_13_diario_reducido.csv** y ver como se ha importado a NumPy.

Sustituir valores nan por `99999` y evitar importar la primera fila y las dos primeras columnas.

In [92]:

```
data = np.genfromtxt('Data/block_13_diario_reducido.csv',delimiter=',')
print (data)

######################### SARA

# Sustituir valores nan por 99999
data1 = np.genfromtxt('Data/block_13_diario_reducido.csv',delimiter=',', filling_val
print (data1)


# Evitar importar la primera fila y las dos primeras columnas
data2 = np.genfromtxt('Data/block_13_diario_reducido.csv', delimiter=',', skip_heade
print (data2)
```

```
[[        nan         nan         nan         nan         nan
nan
          nan         nan         nan]
 [        nan         nan  1.091       1.06525     1.674      28.
   0.30539732 29.827       0.262     ]
 [        nan         nan  0.371       0.54652084  1.434      48.
   0.41012023 26.2330002   0.119     ]
 [        nan         nan  0.3205      0.53470833  2.0339999  48.
   0.46443119 25.6659998   0.129     ]
 [        nan         nan  0.257       0.4215      1.068      48.
   0.27438651 20.232       0.115     ]
 [        nan         nan  0.3985      0.56533333  2.2179999  48.
   0.45071169 27.1359999   0.134     ]
 [        nan         nan  0.698       0.74647917  2.204      48.
   0.4886944  35.8310001   0.131     ]
 [        nan         nan  0.2485      0.43635417  1.551      48.
   0.36278131 20.945       0.139     ]
 [        nan         nan  0.205       0.42654167  1.818      48.
   0.47751141 20.4740001   0.119     ]]
[[9.99990000e+04 9.99990000e+04 9.99990000e+04 9.99990000e+04
  9.99990000e+04 9.99990000e+04 9.99990000e+04 9.99990000e+04
  9.99990000e+04]
 [9.99990000e+04 9.99990000e+04 1.09100000e+00 1.06525000e+00
  1.67400000e+00 2.80000000e+01 3.05397325e-01 2.98270000e+01
  2.62000000e-01]
 [9.99990000e+04 9.99990000e+04 3.71000000e-01 5.46520838e-01
  1.43400000e+00 4.80000000e+01 4.10120226e-01 2.62330002e+01
  1.19000000e-01]
 [9.99990000e+04 9.99990000e+04 3.20500000e-01 5.34708329e-01
  2.03399990e+00 4.80000000e+01 4.64431190e-01 2.56659998e+01
  1.29000000e-01]
 [9.99990000e+04 9.99990000e+04 2.57000000e-01 4.21500000e-01
  1.06800000e+00 4.80000000e+01 2.74386511e-01 2.02320000e+01
  1.15000000e-01]
 [9.99990000e+04 9.99990000e+04 3.98500000e-01 5.65333331e-01
  2.21799990e+00 4.80000000e+01 4.50711689e-01 2.71359999e+01
  1.34000000e-01]
 [9.99990000e+04 9.99990000e+04 6.98000000e-01 7.46479169e-01
  2.20400000e+00 4.80000000e+01 4.88694400e-01 3.58310001e+01
  1.31000000e-01]
 [9.99990000e+04 9.99990000e+04 2.48500000e-01 4.36354167e-01
  1.55100000e+00 4.80000000e+01 3.62781306e-01 2.09450000e+01
  1.39000000e-01]
 [9.99990000e+04 9.99990000e+04 2.05000000e-01 4.26541669e-01
  1.81800000e+00 4.80000000e+01 4.77511414e-01 2.04740001e+01
  1.19000000e-01]]
[[ 1.091       1.06525     1.674      28.          0.30539732 29.827
   0.262     ]
 [ 0.371       0.54652084  1.434      48.          0.41012023 26.23300
02
   0.119     ]
 [ 0.3205      0.53470833  2.0339999  48.          0.46443119 25.66599
98
   0.129     ]
 [ 0.257       0.4215      1.068      48.          0.27438651 20.232
   0.115     ]
 [ 0.3985      0.56533333  2.2179999  48.          0.45071169 27.13599
99
   0.134     ]
 [ 0.698       0.74647917  2.204      48.          0.4886944  35.83100
01
   0.131     ]
 [ 0.2485      0.43635417  1.551      48.          0.36278131 20.945
   0.139     ]
 [ 0.205       0.42654167  1.818      48.          0.47751141 20.47400
01
   0.119     ]]
```

9.3 Funciones básicas con NumPy

Aquí podeis ver una lista de las funciones más interesantes a mirar en la documentación, con algun ejemplo:

Función	Descripción
array_name.mean()	Media
array_name.min()	Valor mínimo
array_name.max()	Valor máximo
array_name.sum()	Suma de valores
array_name.std()	Desviación standard
array_name.var()	Varianza
array_name.reshape(nueva_dimension/es)	Permite cambiar las dimensiones

Algunos ejemplos de Methods asociados a NumPy:

Method	Descripción
np.amin(array_name)	valor mínimo
np.amax(array_name)	valor máximo
np.argmax(array_name)	índice del valor máximo
np.argmin(array_name)	índice del valor mínimo
np.isnan(array_name)	identificador de valores Nan

Para una matriz multidimensional, es posible hacer el cálculo a lo largo de una sola dimensión, pasando el parámetro `axis` para indicar el eje. Por ejemplo, en 2-D, axis=0 es la columna y axis=1 es la fila.

Ejercicio

Practicad con alguno de estas funciones o Methods a partir del array `data` .

In [93]:

```
# vuestro código aquí!

# SARAA
################################################

# Encontrar valor y posición del valor máximo
valor_max = data2. max()
print('Valor máximo', valor_max)

# Encontrar valor y posición del valor mínimo de la segunda columna
valor_min = data2. min()
print('Valor mínimo', valor_min)

# Encontrar valor medio de cada fila
medio = list(map(lambda x: x.mean(), data2))
print('Valor medio', medio)

# Encontrar posicion de los valores Nan
np.isnan(data, where=True)
```

```
Valor máximo 48.0
Valor mínimo 0.115
Valor medio [8.889235332075609, 11.016234466169694, 11.02123417410136
6, 10.052555215864444, 11.271792117159945, 12.58559623833339, 10.24037
6496128594, 10.21715045468317]
```

Out[93]:

```
array([[ True,  True,  True,  True,  True,  True,  True,  True,  Tru
e],
       [ True,  True, False, False, False, False, False, False, Fals
e],
       [ True,  True, False, False, False, False, False, False, Fals
e],
       [ True,  True, False, False, False, False, False, False, Fals
e],
       [ True,  True, False, False, False, False, False, False, Fals
e],
       [ True,  True, False, False, False, False, False, False, Fals
e],
       [ True,  True, False, False, False, False, False, False, Fals
e],
       [ True,  True, False, False, False, False, False, False, Fals
e],
       [ True,  True, False, False, False, False, False, False, Fals
e]])
```

9.4 Filtrado de datos con una matriz booleana

Comparaciones booleanas se pueden usar para comparar elementos en matrices de igual tamaño.

La función `where` crea una nueva array a partir de comparar dos arrays, siguiendo la siguiente syntaxis: `where(bool array, true array, false array)`

In [94]:

```
# Ejemplo: evitar error al dividir por cero

a = np.array([1, 3, 0], float)
np.where(a != 0, 1/a, 0)
```

```
C:\Users\Sara\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: Run
timeWarning: divide by zero encountered in true_divide
  after removing the cwd from sys.path.
```

Out[94]:

```
array([1.        , 0.33333333, 0.        ])
```

Esto se puede usar para crear una máscara, extrayendo los índices de una matriz que satisfacen una condición dada.

In [95]:

```
arr = np.array([10,8,30,40])
print (arr)
mask = arr < 9 # array booleana con los elementos menores a 9
mask
```

```
[10  8 30 40]
```

Out[95]:

```
array([False,  True, False, False])
```

Ejercicio

Reescribir el siguiente código para obtener un array donde los valores inferiores a 9 se les imponga el valor 10 utilizando el array mak anterior y la función `where`.

In [96]:

```
print ('Resetting all values below 9 to 10...')
print (arr < 9)
arr[arr < 9] = 10
print (arr)
```

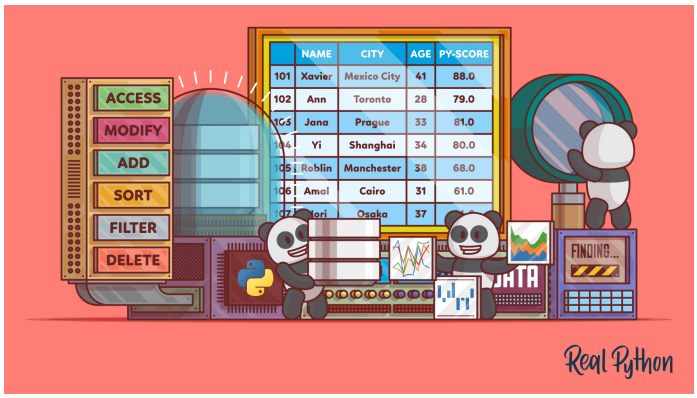
```
Resetting all values below 9 to 10...
[False  True False False]
[10 10 30 40]
```

Para más información sobre NumPy

- http://numpy.scipy.org (http://numpy.scipy.org)
- http://scipy.org/Tentative_NumPy_Tutorial (http://scipy.org/Tentative_NumPy_Tutorial)
- https://numpy.org/learn/ (https://numpy.org/learn/)

10. Introducción a Pandas (SARA)

Padas es de las librerías más importantes en Python para Data Science!!! Básicamente nso permitirá tener `Datasets` abiertos como si se trataran de una hoja de cálculo en el excel, es decir, en forma matricial con dos dimensiones y con columnas e idices indexados.



Pandas sirve para:

- Operar en todo el dataset o a una fila o columna (vectores) con alta eficiencia computacional
- Ayuda a la limpieza de datos y solucionar datos inexistentes
- Acceder a subsets de datos
- Añadir o eliminar columnas con nuevas características.
- Agrupar datos por características.
- Muy eficiente juntando datos de distintas fuentes.
- Trabajar con series temporales.

10.1 Leer datos de un csv

La mayoría de las veces cargaremos los archivos a través de un archivo .csv. A continuación podeis ver el código básico que se usará casi siempre:

In [97]:

```
import pandas as pd #importamos Pandas!
londres = pd.read_csv('Data/block_13_diario.csv')
londres.head(5)
```

Out[97]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	energ
0	MAC000113	2011-12-14	1.0910	1.065250	1.674	28	0.3(
1	MAC000113	2011-12-15	0.3710	0.546521	1.434	48	0.4
2	MAC000113	2011-12-16	0.3205	0.534708	2.034	48	0.4
3	MAC000113	2011-12-17	0.2570	0.421500	1.068	48	0.2
4	MAC000113	2011-12-18	0.3985	0.565333	2.218	48	0.4

10.2 Inspección datos

Si queremos saber qué dimensiones tiene este DataFrame, podemos mirar su 'forma'

In [98]:

```
londres.shape
```

Out[98]:

```
(32992, 9)
```

Hay 32992 entradas y 9 características para cada entrada.

Puede ser muy útil ver las primeras o últimas entradas del dataset para entender mejor estos datos:

In [99]:

```
londres.head(10)
```

Out[99]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	energ
0	MAC000113	2011-12-14	1.0910	1.065250	1.674	28	0.3(
1	MAC000113	2011-12-15	0.3710	0.546521	1.434	48	0.4
2	MAC000113	2011-12-16	0.3205	0.534708	2.034	48	0.4
3	MAC000113	2011-12-17	0.2570	0.421500	1.068	48	0.2
4	MAC000113	2011-12-18	0.3985	0.565333	2.218	48	0.4
5	MAC000113	2011-12-19	0.6980	0.746479	2.204	48	0.4
6	MAC000113	2011-12-20	0.2485	0.436354	1.551	48	0.3
7	MAC000113	2011-12-21	0.2050	0.426542	1.818	48	0.4
8	MAC000113	2011-12-22	0.1830	0.181854	0.259	48	0.0
9	MAC000113	2011-12-23	0.1860	0.185646	0.268	48	0.0

In [100]:

```
londres.tail()
```

Out[100]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	e
32987	MAC005416	2014-02-24	0.0660	0.111437	0.809	48	
32988	MAC005416	2014-02-25	0.0645	0.072521	0.238	48	
32989	MAC005416	2014-02-26	0.0675	0.107917	0.343	48	
32990	MAC005416	2014-02-27	0.0910	0.104250	0.296	48	
32991	MAC005416	2014-02-28	0.0630	0.063000	0.063	1	

Si queremos saber el nombre de las columnas o índices, podemos acceder a ellos:

In [101]:

```
londres.columns
```

Out[101]:

```
Index(['LCLid', 'day', 'energy_median', 'energy_mean', 'energy_max',
       'energy_count', 'energy_std', 'energy_sum', 'energy_min'],
      dtype='object')
```

In [102]:

```
londres.columns[4]
```

Out[102]:

```
'energy_max'
```

In [103]:

```
londres.index
```

Out[103]:

```
RangeIndex(start=0, stop=32992, step=1)
```

Por útlimo, para tener un resumen muy bueno, es muy util el metodo `describe`

In [104]:

```
londres.describe()
```

Out[104]:

	energy_median	energy_mean	energy_max	energy_count	energy_std	ener
count	32992.000000	32992.000000	32992.000000	32992.000000	32865.000000	32992
mean	0.165632	0.223007	0.879596	47.776128	0.182819	10
std	0.158396	0.178484	0.672407	3.035524	0.147394	8
min	0.000000	0.000000	0.000000	1.000000	0.000000	0
25%	0.052000	0.085432	0.348000	48.000000	0.069162	4
50%	0.118500	0.176844	0.775000	48.000000	0.149607	8
75%	0.225000	0.311276	1.248000	48.000000	0.264400	14
max	1.608000	1.407458	6.897000	48.000000	1.104101	67

10.3 Seleccionar datos

Si queremos trabajar o seleccionar solo con una parte del dataset, podemos usar **[]** después del nombre del dataset, e indicar qué columnas se desean seleccionar.

In [105]:

```
londres[['day', 'energy_max']].head()
```

Out[105]:

	day	energy_max
0	2011-12-14	1.674
1	2011-12-15	1.434
2	2011-12-16	2.034
3	2011-12-17	1.068
4	2011-12-18	2.218

Si, por el contrario se quiere seleccionar una parte de las filas, se indicará usando *slicing* como en los strings:

In [106]:

```
londres.iloc[0]
```

Out[106]:

```
LCLid             MAC000113
day              2011-12-14
energy_median         1.091
energy_mean         1.06525
energy_max            1.674
energy_count             28
energy_std          0.305397
energy_sum            29.827
energy_min             0.262
Name: 0, dtype: object
```

loc vs iloc

Para seleccionar partes más específicas (más de una columna, varias columnas y solo unas filas de éstas,...), se usará la indexación con el loc o iloc. La única diferencia entre los dos, es que el primero se basa en el *label* de las filas o columnas, mientras que iloc se basa en el *integer* de estas. Con un ejemplo se entiende mejor:

In [107]:

```
londres.loc[3:5,['day', 'energy_max']]
```

Out[107]:

	day	energy_max
3	2011-12-17	1.068
4	2011-12-18	2.218
5	2011-12-19	2.204

In [108]:

```
londres.iloc[3:5,1:3]
```

Out[108]:

	day	energy_median
3	2011-12-17	0.2570
4	2011-12-18	0.3985

Se debe tener en cuenta que el índice no será siempre un número y puede tratar-se de otro parámetro. Por ejemplo, podemos poner el día como índice.

10.4 Filtrar datos

Otra manera de seleccionar solo una parte del dataset es aplicar *Boolena indexing*, que no es nada más que aplicar un filtro a los datos. Por ejemplo, vamos a seleccionar solo aquellos días en los que la energia máxima ha sido mayor a 1.

In [109]:

```
londres['energy_max'] > 1
```

Out[109]:

```
0         True
1         True
2         True
3         True
4         True
         ...
32987    False
32988    False
32989    False
32990    False
32991    False
Name: energy_max, Length: 32992, dtype: bool
```

Como se puede observar ahora tenemos seleccionadas aquellas columnas que cumplen nuestra condición. ¿Y si lo ponemos directamente dentro del dataset cuando seleccionamos solo una part?

In [110]:

```
londres[londres['energy_max'] > 1]

# SARA
# Aquí vemos que ya no salen todos los días
# londres[londres['energy_max'] > 1.5]
```

Out[110]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	e
0	MAC000113	2011-12-14	1.0910	1.065250	1.674	28	
1	MAC000113	2011-12-15	0.3710	0.546521	1.434	48	
2	MAC000113	2011-12-16	0.3205	0.534708	2.034	48	
3	MAC000113	2011-12-17	0.2570	0.421500	1.068	48	
4	MAC000113	2011-12-18	0.3985	0.565333	2.218	48	
...	...	...	...	...	...	...	
32917	MAC005416	2013-12-16	0.0580	0.162667	1.321	48	
32951	MAC005416	2014-01-19	0.1470	0.210563	1.246	48	
32955	MAC005416	2014-01-23	0.1665	0.207833	1.065	48	
32976	MAC005416	2014-02-13	0.0910	0.134500	1.027	48	
32985	MAC005416	2014-02-22	0.1070	0.260750	1.544	48	

12161 rows × 9 columns

In [111]:

```
######################################################################
# vuestro codigo aquí!

londres[londres['LCLid']=='MAC000113']


# londres['LCLid'].max()
```

Out[111]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	ene
0	MAC000113	2011-12-14	1.0910	1.065250	1.674	28	
1	MAC000113	2011-12-15	0.3710	0.546521	1.434	48	
2	MAC000113	2011-12-16	0.3205	0.534708	2.034	48	
3	MAC000113	2011-12-17	0.2570	0.421500	1.068	48	
4	MAC000113	2011-12-18	0.3985	0.565333	2.218	48	
...	...	...	...	...	...	...	
803	MAC000113	2014-02-24	0.4830	0.624833	2.021	48	
804	MAC000113	2014-02-25	0.4345	0.590542	1.744	48	
805	MAC000113	2014-02-26	0.7615	0.907937	2.106	48	
806	MAC000113	2014-02-27	0.4855	0.567625	1.524	48	
807	MAC000113	2014-02-28	0.8460	0.846000	0.846	1	

808 rows × 9 columns

Ejercicio

Obtener un cuadro resumen de los parámetros para el contador con ID 'MAC000113'

In [112]:

```
######################################################################
# cuadro resumen de los parametros
londres[londres['LCLid']=='MAC000113'].describe()

# Vemos que hay algun missing value en energy_std.
# londres[londres['LCLid']=='MAC000113'].isna().sum()
```

Out[112]:

	energy_median	energy_mean	energy_max	energy_count	energy_std	energy_s
count	808.000000	808.000000	808.000000	808.000000	801.000000	808.000
mean	0.425918	0.519871	1.460573	47.553218	0.316776	24.656
std	0.180692	0.173581	0.558816	4.412963	0.125080	8.470
min	0.154000	0.154458	0.195000	1.000000	0.027048	0.195
25%	0.272375	0.403198	1.088000	48.000000	0.246581	19.162
50%	0.422750	0.525052	1.512000	48.000000	0.326376	25.076
75%	0.537000	0.627714	1.845250	48.000000	0.396441	30.043
max	1.091000	1.098521	3.338000	48.000000	0.916933	52.729

In [113]:

```
############################################################
# vuestro código aquí!

# Valor total de energia para el Id 'MAC000113'

id_MAC000113 = londres[londres['LCLid']=='MAC000113']
print(id_MAC000113['energy_sum'].sum())

# ¿Cuántos contadores distintos hay?

londres['LCLid'].value_counts()

# ¿Cómo se llaman?
```

```
19922.836999400002
```

Out[113]:

```
MAC000113    808
MAC005269    750
MAC005270    750
MAC005317    744
MAC005313    744
MAC005319    744
MAC005367    737
MAC005405    733
MAC005403    733
MAC005392    733
MAC005404    733
MAC005400    733
MAC005411    732
MAC005408    732
MAC005412    732
MAC005407    732
MAC005416    731
MAC005415    731
MAC000301    728
MAC000309    725
MAC005331    703
MAC002662    666
MAC002676    666
MAC002663    666
MAC002702    665
MAC002666    665
MAC002754    660
MAC002757    660
MAC002756    660
MAC002761    660
MAC002764    660
MAC002750    660
MAC002509    602
MAC002512    602
```

10.5 Funciones básicas interesantes

Aquí podeis ver una lista de las funciones más interesantes a mirar en la documentación, con algun ejemplo:

Función	Quantitativas	Qualitativas
count()	Número de observaciones non-null	
sum()	Suma de valores	
mean()	Media de las observaciones	
median()	Mediana de las observaciones	
min()	Valor mínimo	
max()	Valor máximo	
std()	Desviación standard	
var()	Varianza	
value_counts()	Valores unicos en cada columna - tabla de frec	
nunique()	Número de valores distintos	
isnull()	Máscara de valores null o Nan	

Practicad con alguno de estos parámetros

```
MAC002522    600
MAC002519    600
MAC002525    600
MAC002518    600
MAC002514    600
MAC002517    600
MAC002513    600
MAC002558    600
MAC002539    600
MAC002530    600
MAC002605    599
MAC002557    598
MAC005322    531
MAC002510    501
MAC002767    497
MAC005274    286
Name: LCLid, dtype: int64
```

In [114]:

```
#################################################################
# ¿Cuál fue el valor máximo de energía en dia 15 de Mayo de 2013?
londres[londres['day']=='2013-05-15']
```

Out[114]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count
518	MAC000113	2013-05-15	0.7465	0.845771	1.993	48
1246	MAC000301	2013-05-15	0.0805	0.130125	0.879	48
1971	MAC000309	2013-05-15	0.0915	0.271167	1.659	48
2573	MAC002509	2013-05-15	0.1160	0.133000	0.623	48
3175	MAC002510	2013-05-15	0.0425	0.308854	2.345	48
3676	MAC002512	2013-05-15	0.1800	0.319229	1.678	48
4276	MAC002513	2013-05-15	0.0130	0.041854	0.555	48
4876	MAC002514	2013-05-15	0.0915	0.219979	1.353	48
5476	MAC002517	2013-05-15	0.0865	0.095479	0.316	48
6076	MAC002518	2013-05-15	0.1255	0.139625	0.489	48
6676	MAC002519	2013-05-15	0.0445	0.212792	1.898	48
7276	MAC002522	2013-05-15	0.0375	0.094604	0.888	48
7876	MAC002525	2013-05-15	0.1420	0.147542	0.496	48
8476	MAC002530	2013-05-15	0.0140	0.021646	0.047	48
9076	MAC002539	2013-05-15	0.0380	0.035250	0.076	48
9676	MAC002557	2013-05-15	0.1360	0.314708	2.240	48
10274	MAC002558	2013-05-15	0.1215	0.172500	0.498	48
10873	MAC002605	2013-05-15	0.3465	0.491500	2.716	48
11539	MAC002662	2013-05-15	0.1130	0.121333	0.284	48

	LCLid	day	energy_median	energy_mean	energy_max	energy_count
12205	MAC002663	2013-05-15	0.2195	0.233250	1.040	48
12870	MAC002666	2013-05-15	0.3255	0.353396	0.680	48
13536	MAC002676	2013-05-15	0.0370	0.036479	0.066	48
14201	MAC002702	2013-05-15	0.2960	0.331396	0.652	48
14861	MAC002750	2013-05-15	0.0230	0.048500	0.609	48
15521	MAC002754	2013-05-15	0.2735	0.280250	0.547	48
16181	MAC002756	2013-05-15	0.0800	0.184833	1.409	48
16841	MAC002757	2013-05-15	0.2175	0.356896	1.646	48
17501	MAC002761	2013-05-15	0.0195	0.016688	0.021	48
18161	MAC002764	2013-05-15	0.0480	0.109750	0.986	48
18821	MAC002767	2013-05-15	0.0625	0.121729	0.850	48
19408	MAC005269	2013-05-15	0.2835	0.310354	1.117	48
20158	MAC005270	2013-05-15	0.1225	0.170750	1.050	48
21188	MAC005313	2013-05-15	0.0000	0.006771	0.075	48
21932	MAC005317	2013-05-15	0.0280	0.031083	0.097	48
22676	MAC005319	2013-05-15	0.0660	0.117333	0.436	48
23419	MAC005322	2013-05-15	0.0745	0.083458	0.317	48
23941	MAC005331	2013-05-15	0.1935	0.213396	0.772	48
24647	MAC005367	2013-05-15	0.0775	0.115896	0.877	48
25380	MAC005392	2013-05-15	0.2110	0.181708	0.384	48
26113	MAC005400	2013-05-15	0.0405	0.036042	0.047	48
26846	MAC005403	2013-05-15	0.1015	0.275562	1.557	48
27579	MAC005404	2013-05-15	0.1055	0.169542	0.547	48

	LCLid	day	energy_median	energy_mean	energy_max	energy_count
28312	MAC005405	2013-05-15	0.0165	0.035063	0.139	48
29044	MAC005407	2013-05-15	0.3320	0.409625	1.065	48
29776	MAC005408	2013-05-15	0.1185	0.159813	0.724	48
30508	MAC005411	2013-05-15	0.1285	0.163292	0.573	48
31240	MAC005412	2013-05-15	0.2455	0.318729	0.841	48
31971	MAC005415	2013-05-15	0.0650	0.063792	0.126	48
32702	MAC005416	2013-05-15	0.0440	0.043333	0.056	48

In [115]:

```
#################################################################################
# ¿Cuál fue el valor máximo de energía en dia 15 de Mayo de 2013?
londres[londres['day']=='2013-05-15']['energy_sum'].max()
```

Out[115]:

```
40.5969998
```

10.6 Manipular dataframes

Se puede aplicar una función a alguna columna usando el método *apply* y recordando las fuciones lambda!

In [116]:

```
londres['energy_var'] = londres['energy_std'].apply(lambda x: x**2)
londres.head()
```

Out[116]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	energ
0	MAC000113	2011-12-14	1.0910	1.065250	1.674	28	0.3(
1	MAC000113	2011-12-15	0.3710	0.546521	1.434	48	0.4(
2	MAC000113	2011-12-16	0.3205	0.534708	2.034	48	0.4(
3	MAC000113	2011-12-17	0.2570	0.421500	1.068	48	0.2(
4	MAC000113	2011-12-18	0.3985	0.565333	2.218	48	0.4(

Ejercicio

Crea una columna nueva llamada 'ID' con solo las últimas 6 cifras de 'LCLid'

In [117]:

```
############################################################
# vuestro código aquí!

londres['ID']=londres['LCLid'].apply(lambda x: x[-6:])
```

In [118]:

```
londres.head()
```

Out[118]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	energ
0	MAC000113	2011-12-14	1.0910	1.065250	1.674	28	0.3(
1	MAC000113	2011-12-15	0.3710	0.546521	1.434	48	0.4(
2	MAC000113	2011-12-16	0.3205	0.534708	2.034	48	0.4(
3	MAC000113	2011-12-17	0.2570	0.421500	1.068	48	0.2(
4	MAC000113	2011-12-18	0.3985	0.565333	2.218	48	0.4(

10.7 Ordenar

Si se desea ordenar valores, se puede usar `sort_values` . Con un ejemplo se entenderá mejor:

In [119]:

```
londres.sort_values(by='energy_max', ascending= False, inplace=True)
londres.head()
```

Out[119]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	en
3108	MAC002510	2013-03-09	0.7800	0.752292	6.897	48	
3003	MAC002510	2012-11-24	0.1570	0.465229	6.816	48	
3123	MAC002510	2013-03-24	0.9325	0.933542	6.428	48	
3008	MAC002510	2012-11-29	0.0215	0.371896	6.357	48	
3236	MAC002510	2013-07-15	0.0505	0.262250	6.297	48	

10.8 Agrupar

Por último, una función muy útil es la de agrupar (group_by).

Por ejemplo, imagina que quieres agrupar todos los datos por número de contador y sacar algun parámetro de este contador (máximo, mínimo, etc.).

El resultado de esta operación, será otro dataframe con los datos agrupados respeto a una variable. Además también se deberá indicar qué función de *agregación* se desea usar (max, min, mean, count, sum, etc.).

Vamos a implementar el ejemplo comentado!

In [120]:

```
contadores = londres[['LCLid','energy_max', 'energy_min', 'energy_sum']].groupby('LC
contadores
```

Out[120]:

LCLid	energy_max	energy_min	energy_sum
MAC000113	808	808	808
MAC000301	728	728	728
MAC000309	725	725	725
MAC002509	602	602	602
MAC002510	501	501	501
MAC002512	602	602	602
MAC002513	600	600	600
MAC002514	600	600	600
MAC002517	600	600	600
MAC002518	600	600	600
MAC002519	600	600	600
MAC002522	600	600	600
MAC002525	600	600	600
MAC002530	600	600	600
MAC002539	600	600	600
MAC002557	598	598	598
MAC002558	600	600	600
MAC002605	599	599	599
MAC002662	666	666	666
MAC002663	666	666	666
MAC002666	665	665	665
MAC002676	666	666	666
MAC002702	665	665	665
MAC002750	660	660	660
MAC002754	660	660	660
MAC002756	660	660	660
MAC002757	660	660	660
MAC002761	660	660	660
MAC002764	660	660	660
MAC002767	497	497	497

LCLid	energy_max	energy_min	energy_sum
MAC005269	750	750	750
MAC005270	750	750	750
MAC005274	286	286	286
MAC005313	744	744	744
MAC005317	744	744	744
MAC005319	744	744	744
MAC005322	531	531	531
MAC005331	703	703	703
MAC005367	737	737	737
MAC005392	733	733	733
MAC005400	733	733	733
MAC005403	733	733	733
MAC005404	733	733	733
MAC005405	733	733	733
MAC005407	732	732	732
MAC005408	732	732	732
MAC005411	732	732	732
MAC005412	732	732	732
MAC005415	731	731	731
MAC005416	731	731	731

10.9 Merge

Muchas veces tendremos la información deseada en distintas bases de datos, archivos o datasets. Pandas nos ofrece la opción de juntar y fusionar datasets con *merge*.

Veamos un ejemplo con el dataset *londres*.

Si cargaos otro dataset con la información de cada casa obtenemos:

In [121]:

```
import pandas as pd

londres = pd.read_csv('Data/block_13_diario.csv')
londres.head(5)
```

Out[121]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	energ
0	MAC000113	2011-12-14	1.0910	1.065250	1.674	28	0.3(
1	MAC000113	2011-12-15	0.3710	0.546521	1.434	48	0.4
2	MAC000113	2011-12-16	0.3205	0.534708	2.034	48	0.4(
3	MAC000113	2011-12-17	0.2570	0.421500	1.068	48	0.2
4	MAC000113	2011-12-18	0.3985	0.565333	2.218	48	0.4

In [122]:

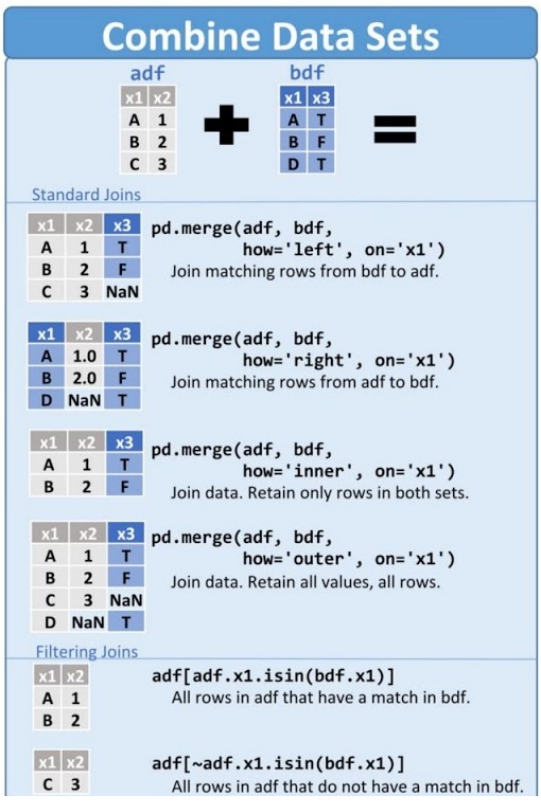
```
casas = pd.read_csv('Data/informations_households.csv')
casas.head(5)
```

Out[122]:

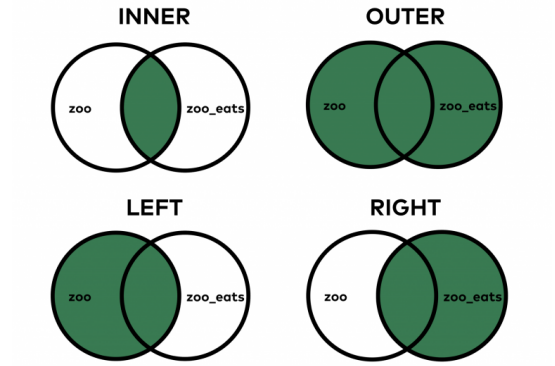
	LCLid	stdorToU	Acorn	Acorn_grouped	file
0	MAC005492	ToU	ACORN-	ACORN-	block_0
1	MAC001074	ToU	ACORN-	ACORN-	block_0
2	MAC000002	Std	ACORN-A	Affluent	block_0
3	MAC003613	Std	ACORN-A	Affluent	block_0
4	MAC003597	Std	ACORN-A	Affluent	block_0

Podemos ver como por cada contador, nos da información de dónde podemos encontrar más parámetros de aquel contador y los agrupa por unos rupos llamados Acron. Para saber más sobre estos, ir directamente al repositorio de este dataset e inspeccionar con detalle.

De todos modos, si queremos juntar toda la información en un mismo dataset, tendríamos que hacer un merge:



Donde los modos *how* posibles se refieren a:



In [123]:

```
combinado = pd.merge(londres, casas, how='left', on='LCLid')
combinado.head(5)
```

Out[123]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	energ
0	MAC000113	2011-12-14	1.0910	1.065250	1.674	28	0.3(
1	MAC000113	2011-12-15	0.3710	0.546521	1.434	48	0.4
2	MAC000113	2011-12-16	0.3205	0.534708	2.034	48	0.4(
3	MAC000113	2011-12-17	0.2570	0.421500	1.068	48	0.2
4	MAC000113	2011-12-18	0.3985	0.565333	2.218	48	0.4

10.10 Otros

Hay muchas más funciones que se pueden usar con Pandas. Os animamos a mirar en su documentación, foros, cheat sheets, cursos,...

https://pandas.pydata.org/pandas-docs/stable/index.html (https://pandas.pydata.org/pandas-docs/stable/index.html)

11. Creación de gáficos

En Python existen diversas librerías para crear gráficos como Bokeh, altair, Pandas o Matplotlib

En este curso aprenderemos las funciones básicas de matplotlib, ya que las otras librerías siempre estan basadas en esta y nos ayudará a luego saber usarlas también.

11.1 Matplotlib.pyplot

El módulo que se usa es el pyplot de matplotlib. Toda su documentación se puede encontrar aquí:

https://matplotlib.org/api/pyplot_api.html#module-matplotlib.pyplot (https://matplotlib.org/api/pyplot_api.html#module-matplotlib.pyplot)

Vaamos a importarla!

In [124]:

```
import matplotlib.pyplot as plt
```

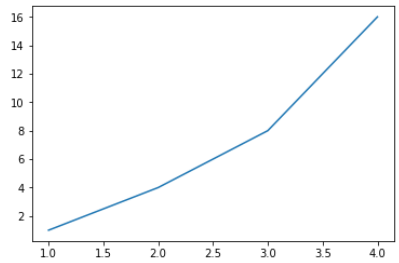
Las dos instancias más importntes en pyplot son

- figure: en matplotlib.figure.Figure, una *figure* ser dónde iran los gráficos. Es el lienzo donde puede haber uno o varios gráficos.
- axes: eb matplotlib.axes.Axes, econtramos dode iran los gráficos en sí

Vamos a crear un primer gráfico de prueba:

In [125]:

```
import matplotlib.pyplot as plt
import numpy as np

plt.plot([1,2,3,4],[1,4,8,16])
plt.show()
```

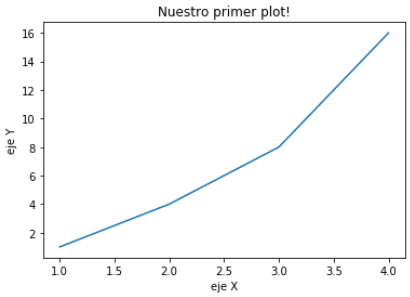


En este caso hemos creado un gráfico usando *plot*, donde el primer argumento es el eje de las x y el segundo el de las y (deben ser de la smismas dimensiones).

Podemos añadir otra información como título y nombre de los ejes:

In [126]:

```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4],[1,4,8,16])
plt.title('Nuestro primer plot!')
plt.xlabel('eje X')
plt.ylabel('eje Y')
plt.show()
```



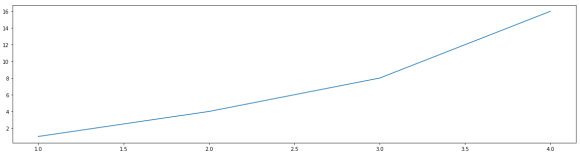
11.2 Crear ventanas

Si se quiere modificar el tamaño del gráfico, es dóde tendremos que usar el término *figure* comentdo anteriormente. Recordamos que *figure* era el lienzo donde poníamos uno o más gráficos. En este caso, solo ponemos uno.

Aquí podemos ver como se puede configurar una *figure* con unas medidas predeterminadas:

In [127]:

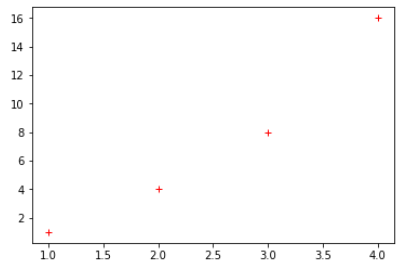
```
plt.figure(figsize=(20,5))
plt.plot([1,2,3,4],[1,4,8,16])
plt.show()
```



Para cada gráfico también se puede determinar el estilo del gráfico con un tercer argumento:

In [128]:

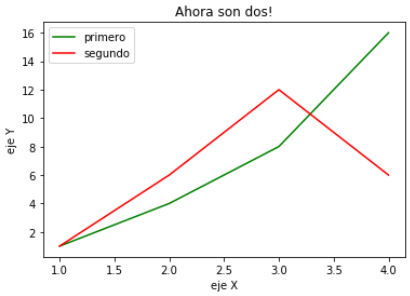
```
plt.plot([1,2,3,4],[1,4,8,16],'r+')
plt.show()

# https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

# character description
# '.'   point marker
# ','   pixel marker
# 'o'   circle marker
# 'v'   triangle_down marker
# '^'   triangle_up marker
# '<'   triangle_left marker
# '>'   triangle_right marker
# '1'   tri_down marker
# '2'   tri_up marker
# '3'   tri_left marker
# '4'   tri_right marker
# '8'   octagon marker
# 's'   square marker
# 'p'   pentagon marker
# 'P'   plus (filled) marker
# '*'   star marker
# 'h'   hexagon1 marker
# 'H'   hexagon2 marker
# '+'   plus marker
# 'x'   x marker
# 'X'   x (filled) marker
# 'D'   diamond marker
# 'd'   thin_diamond marker
# '|'   vline marker
# '_'   hline marker
```



Y podemos poner más de un gráfico en el mismo plot, indicando en la leyenda que es cada cosa:

In [129]:

```
plt.plot([1,2,3,4],[1,4,8,16],'g-',label='primero')
plt.plot([1,2,3,4],[1,6,12,6],'r-', label='segundo')
plt.title('Ahora son dos!')
plt.xlabel('eje X')
plt.ylabel('eje Y')
plt.legend()
plt.show()
```

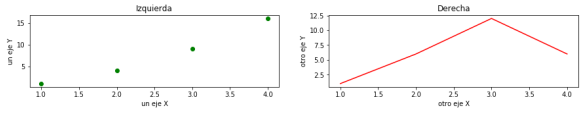


In [130]:

```
plt.figure(figsize=(15,2))

plt.subplot(1,2,1)
plt.plot([1,2,3,4],[1,4,9,16],'go')
plt.title('Izquierda')
plt.xlabel('un eje X')
plt.ylabel('un eje Y')

plt.subplot(1,2,2)
plt.plot([1,2,3,4],[1,6,12,6],'r-')
plt.title('Derecha')
plt.xlabel('otro eje X')
plt.ylabel('otro eje Y')
```

Out[130]:

```
Text(0, 0.5, 'otro eje Y')
```



O, por ejemplo:

11.3 Subplots

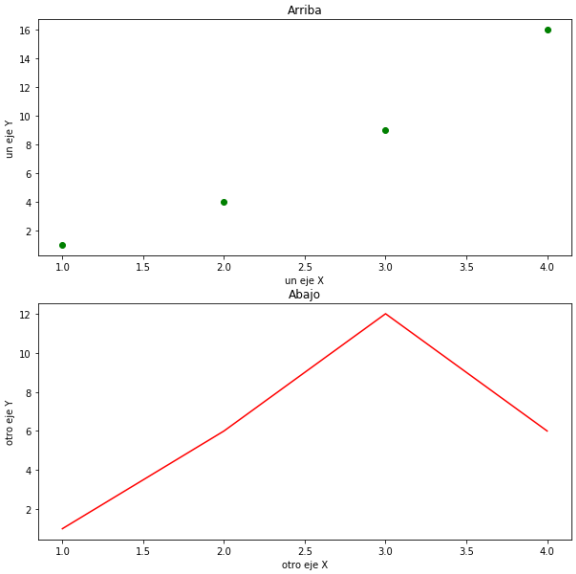
Para poner distintos gráficos con distintos ejes se puede usar Subplots. El métdodo `subplot` tiene 3 argumentos: número de filas, número de columnas e índice. Veamos con un ejemplo como se usa:

In [131]:

```
plt.figure(figsize=(10,10))

plt.subplot(2,1,1)
plt.plot([1,2,3,4],[1,4,9,16],'go')
plt.title('Arriba')
plt.xlabel('un eje X')
plt.ylabel('un eje Y')

plt.subplot(2,1,2)
plt.plot([1,2,3,4],[1,6,12,6],'r-')
plt.title('Abajo')
plt.xlabel('otro eje X')
plt.ylabel('otro eje Y')
```

Out[131]:

```
Text(0, 0.5, 'otro eje Y')
```



11.4 Gráfico de barras

Para hacer un gráfico de barras en vez de `plot` usaremos `bar` , siguendo una estructura muy parecida.

Recordemos el dataset de antes:

In [132]:

```
contadores = londres[['LCLid','energy_max', 'energy_min', 'energy_sum']].groupby('LC
contadores.head(10)
```

Out[132]:

LCLid	energy_max	energy_min	energy_sum
MAC000113	1.460573	0.157256	24.656976
MAC000301	1.085894	0.027124	9.521648
MAC000309	1.997796	0.077452	19.015297
MAC002509	0.628286	0.051095	7.462581
MAC002510	2.144932	0.027697	12.070756
MAC002512	1.556296	0.121689	18.047101
MAC002513	0.806083	0.016887	4.796607
MAC002514	0.806902	0.033100	7.510262
MAC002517	1.050635	0.044790	11.447748
MAC002518	0.364883	0.032975	6.818670

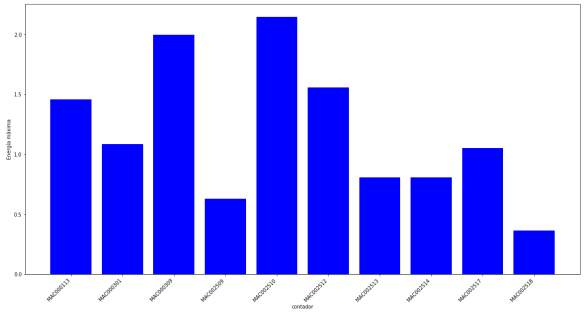
Podemo hacer una gràfica de barras:

In [133]:

```
divisiones=contadores.index[:10]
marcas=contadores['energy_max'][:10]

plt.figure(figsize=(20,10))
plt.bar(divisiones, marcas, color = 'blue')
plt.xlabel('contador')
plt.ylabel('Energía máxima')
plt.xticks(rotation=45, ha='right')
```

Out[133]:

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], <a list of 10 Text xticklabel objects
>)
```



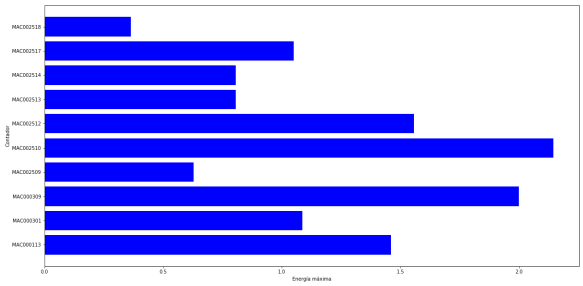
O en barras horizontales, usando `barh` :

In [134]:

```
divisiones=contadores.index[:10]
marcas=contadores['energy_max'][:10]

plt.figure(figsize=(20,10))
plt.barh(divisiones, marcas, color = 'blue')
plt.ylabel('Contador')
plt.xlabel('Energía máxima')
```

Out[134]:

```
Text(0.5, 0, 'Energía máxima')
```

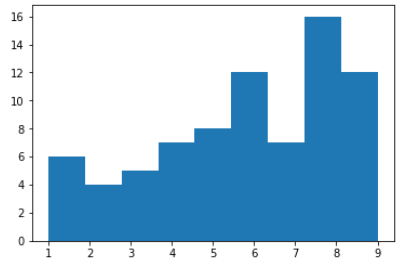


In [135]:

```
plt.hist([1,5,8,7,4,1,2,4,3,6,5,4,6,3,2,3,6,9,8,5,4,1,6,9,8,7,8,8,8,7,4,7,8,8,9,9,8,
```

Out[135]:

```
(array([ 6.,  4.,  5.,  7.,  8., 12.,  7., 16., 12.]),
 array([1.        , 1.88888889, 2.77777778, 3.66666667, 4.55555556,
        5.44444444, 6.33333333, 7.22222222, 8.11111111, 9.        ]),
 <a list of 9 Patch objects>)
```



11.5 Histograma

Veamos con un ejemplo como hacer un histograma:

In [136]:

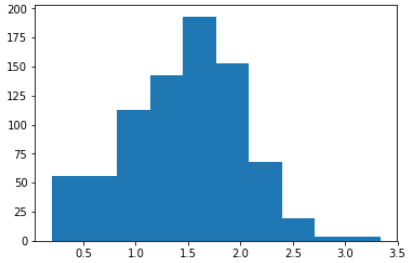
```
## Ejercicio
## Generar un histograma de la energía maxima del contador MAC000113

#dataset londres filtrando solo un LCLid del MAC000113
contador_113 = londres[londres['LCLid']=='MAC000113']

#energía màxima de este contador
e_max_113= contador_113[['energy_max']]
```

In [137]:

```
plt.hist(e_max_113.values)
```

Out[137]:

```
(array([ 56.,  56., 113., 142., 193., 153.,  68.,  19.,   4.,   4.]),
 array([0.195     , 0.50930001, 0.82360002, 1.13790003, 1.45220004,
        1.76650005, 2.08080006, 2.39510007, 2.70940008, 3.02370009,
        3.3380001 ]),
 <a list of 10 Patch objects>)
```

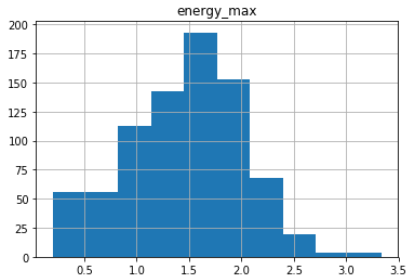


In [138]:

```
#plot
e_max_113.hist()
```

Out[138]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000024EC71
E8438>]],
      dtype=object)
```

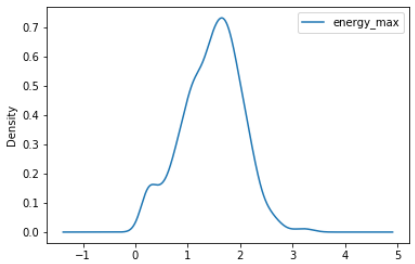


In [139]:

```
e_max_113.plot.density()
```

Out[139]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24ec743ebe0>
```



In [140]:

```
import plotly.express as px
```

In [141]:

```
!pip install plotly
```

```
Requirement already satisfied: plotly in c:\users\sara\anaconda3\lib\s
ite-packages (4.14.3)
Requirement already satisfied: six in c:\users\sara\anaconda3\lib\site
-packages (from plotly) (1.12.0)
Requirement already satisfied: retrying>=1.3.3 in c:\users\sara\anacon
da3\lib\site-packages (from plotly) (1.3.3)
```

Plots interactivos: Plotly

Existen plots interactivos, donde se puede hacer zoom, seleccionar datos,etc.

In [142]:

```
import plotly.graph_objects as go

# Create traces
fig = go.Figure()
fig.add_trace(go.Scatter(x=contador_113['day'], y=contador_113['energy_max'],
                         mode='lines',
                         name='Energy max - lines'))
fig.add_trace(go.Scatter(x=contador_113['day'], y=contador_113['energy_min'],
                         mode='lines+markers',
                         name='Energy min - lines+markers'))
fig.add_trace(go.Scatter(x=contador_113['day'], y=contador_113['energy_mean'],
                          mode='markers', name='energy_mean - markers'))


# Edit the layout
fig.update_layout(title='Resumen consumos ID MAC000113',
                  xaxis_title='Month - Year',
                  yaxis_title='Energy (kWh)')


fig.show()
```