3. Agregaciones de datos
Funciones y paquetes (numpy, matplotlib)

1 Agregaciones de datos

- Cadenas de caracteres
- Listas
- Tuplas
- Diccionarios

1.1 Cadenas de caracteres

Indexación

- El primer carácter de una cadena ocupa la posición 0
- `str[i]` → carácter situado en la posición `i` de la cadena de caracteres `str`
- `str[i:j]` → sub-cadena formada por los caracteres situados entre las posiciones `i` y `j-1` de `str`, ambas incluidas
- `str[i:j:k]` → análogo a `str[i:j]`, utilizando paso `k`
- Se pueden utilizar índices negativos: `-1` corresponde al último carácter de la cadena

[1]:
```
cadena='@Esta es mi cadena de caracteres.#'
```

[2]:
```
cadena[0] # Primer carácter
```

[2]:
```
'@'
```

[3]:
```
cadena[1]
```

[3]:
```
'E'
```

[4]:
```
cadena[len(cadena)-1] # Último carácter
```

[4]:
```
'#'
```

[5]:
```
cadena [-1] # Último carácter
```

[5]:
```
'#'
```

[6]:
```
cadena[-2]
```

[6]:
```
'.'
```

[7]:
```
cadena[0:5]
```

[7]:
```
'@Esta'
```

[8]:
```
cadena[-7:-3]
```

[8]:
```
'tere'
```

[9]:
```
cadena[:] # Todos los elementos de la cadena
```

[9]:
```
'@Esta es mi cadena de caracteres.#'
```

[10]:
```
cadena[:10]
```

[10]:
```
'@Esta es m'
```

[11]:
```
cadena[-7:]
```

[11]:
```
'teres.#'
```

[12]:
```
cadena='ab1cd2ef3gh4ij5kl6mn7op8qr9j'
cadena[2:len(cadena):3] # Corte de cadena utilizando paso 3
```

[12]:
```
'123456789'
```

Recorrido por todos los caracteres de una cadena

[13]:
```
mi_cadena='¡Qué pequeño es el mundo!'
```

[14]:
```
for i in range(len(mi_cadena)):
    print (mi_cadena[i], end='..')
```

```
¡..Q..u..é.. ..p..e..q..u..e..ñ..o.. ..e..s.. ..e..l.. ..m..u..n..d..o..!..
```

[15]:
```
for letra in mi_cadena:  # Forma alternativa de recorrer una cadena
    print (letra, end='..')
```

```
¡..Q..u..é.. ..p..e..q..u..e..ñ..o.. ..e..s.. ..e..l.. ..m..u..n..d..o..!..
```

Ejemplo: inversión del orden de una cadena

[16]:
```
str='desserts'

for i in range(len(str)-1,-1,-1): # valores de i=[7,6,5,...,1,0]
    print(str[i], end='')
```

```
stressed
```

[17]:
```
''' Solución sin bucles for'''
str1='ALI TOMO TILA'
print(str1[-1:-len(str1)-1:-1]) # Invierte los caracteres de cadena
```

```
ALIT OMOT ILA
```

[18]:
```
'''Crear una cadena que contenga el resultado de invertir a str2'''
str2='DABALE ARROZ A LA ZORRA EL ABAD'
reverse_str=''  # La solucion está vacía al principio
for i in range(len(str2)-1,-1,-1):
    reverse_str+=str2[i]  # Concatena a la solución el carácter str[i]

print(reverse_str)
```

```
DABA LE ARROZ AL A ZORRA ELABAD
```

Ejemplo: contar espacios en blanco

[19]:
```
str='Tengo 4 espacios en blanco'

num_blancos=0
for letra in str:
    if letra==' ':
        num_blancos+=1
print (f'Número de blancos: {num_blancos}')
```

```
Número de blancos: 4
```

Ejemplo: contar palabras

[20]:
```
''' Contar palabras:
    Algoritmo:
    El número de palabras es el número de espacios en blanco más uno'''
str=input('Escribe una frase:')

num_blancos=0
```

```
for letra in str:
    if letra==' ':
        num_blancos+=1

num_palabras=num_blancos+1

print (f'Número de palabras: {num_palabras}')
```

```
Escribe una frase: tengo 3

Número de palabras: 2
```

Comprueba cómo se comporta el programa anterior en los casos siguientes:

- La frase comienza con uno o más espacios en blanco
- Hay palabras separadas por más de un espacio en blanco

[21]:
```
''' Contar palabras (versión mejorada)
    Algoritmo:
        1. Contar el número de transiciones 'blanco'->'no blanco'
        2. Si el primer carácter de la cadena es un blanco:
               número de palabras es igual al número de transiciones
           En caso contrario:
               número de palabras es igual al número de transiciones + 1
        '''
str=input('Escribe una frase:')

num_transiciones=0

letra_anterior='' # No hay ningún carácter anterior al primero
for letra_actual in str:
    if letra_anterior==' ' and letra_actual!=' ':
        num_transiciones+=1
    letra_anterior=letra_actual # En la próxima iteración, la letra actual pasa␣
 ↪a ser la anterior

if str[0]==' ':
    num_palabras=num_transiciones
else:
    num_palabras=num_transiciones+1

print (f'Número de palabras: {num_palabras}')
```

```
Escribe una frase: tengo 3

Número de palabras: 2
```

Modificación de cadenas de caracteres

- Las cadenas de caracteres son **inmutables** en Python: una vez creadas, no se puede cambiar su contenido

[22]:
```
str='My new look'
```

[23]:
```
str[7]='b'
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-23-612515c6b50c> in <module>
----> 1 str[7]='b'


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

[24]:
```
str2=str[:7]+'b'+str[8:]
print(str2)
```

```
My new book
```

1.2 Listas

```
lista=[elemento_0, elemento_1, ..., elemento_n]

elemento_i=lista[i]
```

- Las listas pueden contener elementos de tipos no homogéneos
- Las listas son mutables
- Se puede acceder a posiciones negativas (empieza por el final)
- Una lista puede ser elemento de otra lista

[25]:
```
lista1=[1,3,5,7,9]
print(lista1)
```

```
[1, 3, 5, 7, 9]
```

[26]:
```
lista2=['Renault', 'Ford', 'Seat', 'Ferrari', 'Porsche', 'RTULE']
print(lista2)
```

```
['Renault', 'Ford', 'Seat', 'Ferrari', 'Porsche', 'RTULE']
```

[27]:
```
a=5
lista3=['hola',3*a,'@'*8,'adios']
print(lista3)
```

```
['hola', 15, '@@@@@@@@', 'adios']
```

[28]:
```
lista_vacia=[]
print(lista_vacia)
```

```
[]
```

[29]:
```
lista2=['Renault', 'Ford', 'Seat', 'Ferrari', 'Porsche', 'RTULE']
lista3=['hola',15,'@'*8,'adios']
```

[30]:
```
len(lista3)
```

[30]: 4

[31]:
```
lista_concatenada=[1,2,3]+['A','B','C']
print(lista_concatenada)
```

```
[1, 2, 3, 'A', 'B', 'C']
```

[32]:
```
lista_triple=['$','=']*3
print(lista_triple)
```

```
['$', '=', '$', '=', '$', '=']
```

[33]:
```
segundo_elemento_lista2=lista2[1]
print(segundo_elemento_lista2)
```

```
Ford
```

[34]:
```
sublista=lista2[2:4]
print(sublista)
```

```
['Seat', 'Ferrari']
```

Comparación de listas

==, != (Dos listas son iguales si tienen el *mismo número de elementos* y todos ellos son iguales)

<, <=, >, >= (funcionamiento análogo a la comparación de cadenas de caracteres)

[35]:
```
[1,2,3,4]==[1,2,3,4]
```

[35]: True

[36]:
```
[1,2,3,4]!=[1,3,4,2]
```

[36]: True

[37]:
```
[1,2,3,4]!=[1,2,3]
```

[37]: True

[38]:
```
['a','b','hola','2']!=['a','b','hola',2]
```

[38]: True

[39]:
```
['a','b','hola','2']==['a','b','hola','2']
```

[39]: True

[40]:
```
[8,9,10,11,12]!=[8,9,10,11]
```

[40]: True

[41]:
```
[11,12,13,14,15]>[11,12,14]
```

[41]: False

[42]:
```
[21,22,23,24,25]>[20,21,22,23,24,25,26,27]
```

[42]: True

[43]:
```
['a','b','c','d']>=['a','b','c','D']
```

[43]: True

[44]:
```
['a','b','c','d']>=['a','b','c','d','e']
```

[44]: False

[45]:
```
['a','b','c','d']>=['a','b','c']
```

[45]: True

[46]:
```
[1,2]<[0]
```

[46]: False

[47]:
```
[1,2]<=[]
```

[47]: False

Recorrido de una lista

[48]:
```
theSimpsons=['Marge','Homer','Bart','Lisa','Maggie']
```

[49]:
```
for i in range(5):
    print(theSimpsons[i], end=' ')
```

```
Marge Homer Bart Lisa Maggie
```

[50]:
```
for i in range(len(theSimpsons)):
    print(theSimpsons[i], end=' ')
```

```
Marge Homer Bart Lisa Maggie
```

[51]:
```
for nombre in theSimpsons:
    print(nombre, end=' ')
```

```
Marge Homer Bart Lisa Maggie
```

Adición de elementos a una lista: concatenación, *append*

[52]:
```
lista=[0,1,2]
```

[53]:
```
lista+=3  # Genera error: no se puede concatenar un número entero a una lista
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-53-fe3bfb7d17d0> in <module>
----> 1 lista+=3  # Genera error: no se puede concatenar un número entero a
↪una lista


TypeError: 'int' object is not iterable
```

[54]:
```
lista+=[3]  # Se concatena la lista [3]
print(lista)
```

```
[0, 1, 2, 3]
```

[55]:
```
lista.append(4)
print(lista)
```

```
[0, 1, 2, 3, 4]
```

[56]:
```
theSimpsons=['Marge','Homer','Bart','Lisa','Maggie']
```

[57]:
```
theSimpsons+=['Krusty','Flanders','Mou']
```

[58]:
```
theSimpsons.append('Milhouse')
```

[59]:
```
print(theSimpsons)
```

```
['Marge', 'Homer', 'Bart', 'Lisa', 'Maggie', 'Krusty', 'Flanders', 'Mou',
'Milhouse']
```

Ejemplo: rellenar una lista desde teclado

[60]:
```
mi_lista=[] # Crea una lista vacía
nombre=input('Nombre?')

while nombre!='': # control por centinela (nombre)
    mi_lista.append(nombre)
    nombre=input('Nombre?')  # se pide el siguiente nombre

print (mi_lista)
```

```
Nombre? Pilar
Nombre? Ticiano
Nombre? Mario
Nombre?
['Pilar', 'Ticiano', 'Mario']
```

Eliminación de elementos de una lista: *del*

[61]:
```
numeros=[2,0,3,0,0,4,6,0,9,-1,-3]
```

[62]:
```
del numeros[1] # Elimina el segundo elemento (la lista queda modificada)
print(numeros)
```

```
[2, 3, 0, 0, 4, 6, 0, 9, -1, -3]
```

[63]:
```
del numeros[-4:-1]
print(numeros)
```

```
[2, 3, 0, 0, 4, 6, -3]
```

[64]:
```
del numeros[:3]
print(numeros)
```

```
[0, 4, 6, -3]
```

[65]:
```
del numeros[:]
print(numeros)
```

```
[]
```

Ejemplo: eliminación de elementos recorriendo una lista

Eliminar todos los valores `0` de la lista `[2,0,3,0,0,4,6,0,9,-1,-3]`

[66]:
```
numeros=[2,0,3,0,0,4,6,0,9,-1,-3]

for i in range(len(numeros)):
    if numeros[i]==0:
        del numeros[i]
print (numeros)
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-66-039cdeeb6516> in <module>
  2
  3 for i in range(len(numeros)):
----> 4     if numeros[i]==0:
  5         del numeros[i]
  6 print (numeros)


IndexError: list index out of range
```

Se produce un error de tipo *índice fuera de rango* porque la lista cambia cada vez que se elimina uno de sus elementos

[67]:
```
numeros=[2,0,3,0,0,4,6,0,9,-1,-3]

for i in range(len(numeros)):
    if numeros[i]==0:
        del numeros[i]
    print ('i=', i, ':', numeros,'-->',len(numeros),'elementos')
```

```
i= 0 : [2, 0, 3, 0, 0, 4, 6, 0, 9, -1, -3] --> 11 elementos
i= 1 : [2, 3, 0, 0, 4, 6, 0, 9, -1, -3] --> 10 elementos
i= 2 : [2, 3, 0, 4, 6, 0, 9, -1, -3] --> 9 elementos
i= 3 : [2, 3, 0, 4, 6, 0, 9, -1, -3] --> 9 elementos
```

```
i= 4 : [2, 3, 0, 4, 6, 0, 9, -1, -3] --> 9 elementos
i= 5 : [2, 3, 0, 4, 6, 9, -1, -3] --> 8 elementos
i= 6 : [2, 3, 0, 4, 6, 9, -1, -3] --> 8 elementos
i= 7 : [2, 3, 0, 4, 6, 9, -1, -3] --> 8 elementos
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-67-50054ba0b4b8> in <module>
  2
  3 for i in range(len(numeros)):
----> 4     if numeros[i]==0:
  5         del numeros[i]
  6     print ('i=', i, ':', numeros,'-->',len(numeros),'elementos')


IndexError: list index out of range
```

[68]:
```
numeros=[2,0,3,0,0,4,6,0,9,-1,-3]

i=0
while i< len(numeros): # El contador (posición) ha de ser menor que la longitud␣
 ↪de la lista
    if numeros[i]==0:
        del numeros[i]
    print ('i', i, ':', numeros,'-->',len(numeros),'elementos')
    i+=1
```

```
i 0 : [2, 0, 3, 0, 0, 4, 6, 0, 9, -1, -3] --> 11 elementos
i 1 : [2, 3, 0, 0, 4, 6, 0, 9, -1, -3] --> 10 elementos
i 2 : [2, 3, 0, 4, 6, 0, 9, -1, -3] --> 9 elementos
i 3 : [2, 3, 0, 4, 6, 0, 9, -1, -3] --> 9 elementos
i 4 : [2, 3, 0, 4, 6, 0, 9, -1, -3] --> 9 elementos
i 5 : [2, 3, 0, 4, 6, 9, -1, -3] --> 8 elementos
i 6 : [2, 3, 0, 4, 6, 9, -1, -3] --> 8 elementos
i 7 : [2, 3, 0, 4, 6, 9, -1, -3] --> 8 elementos
```

Cuando hay dos elementos `0` consecutivos, uno de ellos no se elimina porque el contador aumenta

Modificar una lista ⇒ controlar la longitud de la lista y la actualización de contadores

[69]:
```
numeros=[2,0,3,0,0,4,6,0,9,-1,-3]

i=0
while i<len(numeros): # El contador (posición) ha de ser menor que la longitud␣
 ↪de la lista
    if numeros[i]==0:
        del numeros[i]
    else:
        i+=1  # Tras eliminar un elemento, no se actualiza el valor del contador
print (numeros)
```

```
[2, 3, 4, 6, 9, -1, -3]
```

Pertenencia de un elemento a una lista: *in*

[70]:
```
juguetes=['aro','pelota','yo-yo','muñeca','coche']
```

[71]:
```
'yo-yo' in juguetes
```

[71]:
```
True
```

[72]:
```
'trompo' in juguetes
```

[72]:
```
False
```

[73]:
```
if 'globo' not in juguetes:
    juguetes.append('globo')
print (juguetes)
```

```
['aro', 'pelota', 'yo-yo', 'muñeca', 'coche', 'globo']
```

Conversión de cadena a lista: *list, split*

[74]:
```
list('letras') # Crea una lista con los caracteres de una cadena
```

[74]:
```
['l', 'e', 't', 'r', 'a', 's']
```

[75]:
```
str='Esta frase    tiene    5 palabras'
```

[76]:
```
str.split() # Convierte una cadena a una lista de palabras
```

[76]:
```
['Esta', 'frase', 'tiene', '5', 'palabras']
```

[77]:
```
str2='9.8, 10, 6.7, 5, 8.2, 9.1'
```

[78]:
```
str2.split(', ') # Se puede especificar la cadena separadora entre palabras
```

[78]:
```
['9.8', '10', '6.7', '5', '8.2', '9.1']
```

Ejemplos:

[79]:
```
str2='9.8, 10, 6.7, 5, 8.2, 9.1'
```

[80]:
```
len(str2.split(', '))
```

[80]:
```
6
```

[81]:
```
str2.split(', ')[0]
```

[81]:
```
'9.8'
```

[82]:
```
float(str2.split(', ')[2])
```

[82]:
```
6.7
```

[83]:
```
c=str2.split(', ')
c.append('11')
c[len(c)-3]
```

[83]:
```
'8.2'
```

[84]:
```
d=[]
for s in c:
    d.append(float(s))
d
```

[84]:
```
[9.8, 10.0, 6.7, 5.0, 8.2, 9.1, 11.0]
```

Conversión de lista a cadena de caracteres: *join*

[85]:
```
lista=['redes neuronales','algoritmos genéticos','minería de datos','HPC']
```

[86]:
```
'; '.join(lista)
```

[86]:
```
'redes neuronales; algoritmos genéticos; minería de datos; HPC'
```

Ejemplos:

[87]:
```
print (''.join(['H','o','l','a']))
```

```
Hola
```

[88]:
```
print ('::'.join(['10','20','30','40','50']))
```

```
10::20::30::40::50
```

[89]:
```
# Eliminación de blancos sobrantes
frase=' Tengo  3     palabras'
lista_aux=frase.split()
frase_limpia=' '.join(lista_aux)
print (frase_limpia)
```

```
Tengo 3 palabras
```

Listas de listas

[90]:
```
matriz=[[11,12,13],[21,22,23],[31,32,33]]
```

[91]:
```
matriz[0] # Primera fila de matriz
```

[91]:
```
[11, 12, 13]
```

[92]:
```
matriz[2][1] # Elemento 1 de la fila 2 de matriz
```

[92]:
```
32
```

[93]:
```
lista=[1,'dos',['tres', 'cuatro'],5]
```

[94]:
```
lista[1]
```

[94]:
```
'dos'
```

[95]:
```
lista[2][1]
```

[95]:
```
'cuatro'
```

1.3 Tuplas

- Las tuplas son **listas inmutables**, una vez creadas no se pueden añadir ni eliminar elementos
- Son más rápidas que las listas
- El acceso a sus elementos es análogo al de las listas
- Se definen delimitadas por paréntesis:

```
variable_tupla=(elemento_0, elemento_1, ..., elemento_n)
```

[96]:
```
transportes=('tierra','mar','aire')
```

[97]:
```
transportes[0]
```

[97]:
```
'tierra'
```

[98]:
```
transportes[-1]
```

[98]:
```
'aire'
```

[99]:
```
transportes[1:3]
```

[99]:
```
('mar', 'aire')
```

1.4 Diccionarios

- Agregaciones de datos 'clave:valor"
- Se accede a los datos utilizando el valor de la clave
- Se definen delimitados entre llaves:

```
variable_diccionario={clave_0:valor_0,
                      clave_1:valor_1,
                      ...
                      clave_n:valor_n
                     }
```

[100]:
```
equipos={'León':'Cultural y Deportiva Leonesa',
        'Ponferrada':'S.D. Ponferradina',
        'Gijón': 'Real Sporting de Gijón',
        'Oviedo': 'Real Oviedo',
        }
```

[101]:
```
equipos
```

[101]:
```
{'León': 'Cultural y Deportiva Leonesa',
 'Ponferrada': 'S.D. Ponferradina',
 'Gijón': 'Real Sporting de Gijón',
 'Oviedo': 'Real Oviedo'}
```

[102]:
```
equipos['Gijón']
```

[102]:
```
'Real Sporting de Gijón'
```

Cualquier elemento inmutable puede actuar como clave de un diccionario:

[103]:
```
diccionario_2={3:'hola', 2:'hello', 1:'hallo'}
```

[104]:
```
diccionario_2[3]  # El valor 3 es una clave, no un índice
```

[104]: 'hola'

[105]:
```
diccionario_3={'UNO':'valor 1', (2,'b'): 'valor 2', 3:'valor 3'}
```

[106]:
```
diccionario_3['UNO']
```

[106]: 'valor 1'

[107]:
```
diccionario_3[(2,'b')]
```

[107]: 'valor 2'

[108]:
```
diccionario_3[3]
```

[108]: 'valor 3'

2 Funciones

2.1 Funciones con nombre (*def*):

```
def nombre_función(argumento_0[=valor_por_defecto_0],
                   argumento_1[=valor_por_defecto_1],
                   ...,
                   argumento_n[=valor_por_defecto_n]):
    # Nota: El número de parámetros puede ser 0
    instrucción_0
    instrucción_1
    ...
    instrucción_K
    [return valor] # La función puede devolver un resultado
```

- **Llamada** a una función:

```
variable_resultado=nombre_función(valor_argumento_1, ..., valor_argumento_n)
```

[109]:
```
'''Función suma'''
def suma(a,b=100):
    resultado=a+b
    return resultado

'''Ejemplos de uso de la función suma'''

print (suma(1.5,2.4)) # a=1.5, b=2.4

print (suma(1)) # Se utiliza el valor por defecto de b

res1=suma(2,9)
res2=suma(4+1j,3+2j)
res3=suma(res1,res2)  # a=res1, b=res2
print (f'{res1}+{res2}={res3}')
```

```
3.9
101
11+(7+3j)=(18+3j)
```

2.2 Módulos

- Un módulo es un archivo que contiene un conjunto de definiciones de funciones y/o v variables
- Para utilizar las funciones y las variables de un módulo en otros programas hay que importarlas.
- Formas de importar un módulo genérico, MOD1:

```
# a. Importar el módulo y acceder a él con el nombre mm
import MOD1 as mm
# Utilizar una función del módulo importado
mm.función_del_módulo()

# b. Importar algunas funciones del módulo para utilizarlas directamente
from nombre_del_módulo import función_1[,función_2,...,función_n])
# Utilizar  las funciones importadas
función_1()

# c. Importar todas las funciones del módulo para utilizarlas directamente
from nombre_del_módulo import *
# Utilizar las funciones importadas
función_1()
función_2()
...
función_n()
```

Ejemplo de módulo: archivo mis_funciones.py

[110]:
```
'''Módulo mis_funciones'''

# Aceleración de la gravedad en m/s^2
g=9.8

'''función suma'''
def suma(a,b):
    return a+b

'''función resta'''
def resta(a,b):
    return a-b

'''función valor absoluto'''
def valor_absoluto(n):
    if n>=0:
        absoluto=n
    else:
        absoluto=-n
```

```
    return absoluto
```

Utilización del módulo mis_funciones.py

[2]:
```
import mis_funciones as mf

a=3
b=-4
c=3.3
d=2.2
```

[112]:
```
mf.suma(a,b)
```

[112]:
```
-1
```

[3]:
```
mf.resta(c,d)
```

[3]:
```
1.0999999999999996
```

[114]:
```
mf.valor_absoluto(b)
```

[114]:
```
4
```

[115]:
```
mf.suma([1,2,3],[4,5,6]) # La operación lista+lista es una concatenación
```

[115]:
```
[1, 2, 3, 4, 5, 6]
```

[116]:
```
print ('F=m*g=3*g=%.3f'%(3*mf.g))
```

```
F=m*g=3*g=29.400
```

2.3 Funciones anónimas (*lambda*)

- Se escriben en una sola línea de código
- Se pueden utilizar inmediatamente

```
lambda argumento_0[,...,argumento_n]:expressión
```

[117]:
```
sumar=lambda x,y: x+y
```

[118]:
```
sumar(1,2)
```

[118]:
```
3
```

Aplicación de una función a todos y cada uno de los elementos de una lista:

[119]:
```
lista=[10,11,12,13,14,15,16]
es_par=lambda x:x%2  # Devuelve True si x es par
```

[120]:
```
[es_par(i) for i in lista] # Lista generada aplicando una función a cada␣
↪elemento de otra
```

[120]:
```
[0, 1, 0, 1, 0, 1, 0]
```

3 Paquetes

- Numpy
- Matplotlib

Nota: La mayoría de paquetes han de instalarse (desde Anaconda, por ejemplo)

3.1 Numpy

Proporciona funcionalidad de alto rendimiento para cálculo numérico: * Trabajar con arrays numéricos * Utilizar herramientas del Álgebra Lineal * Realizar transformadas de Fourier * Realizar operaciones con números aleatorios * etc

[121]:
```
import numpy as np
```

3.1.1 Creación de Arrays

Arrays a partir de listas

[122]:
```
a=np.array([1,2,3]) # Vector de 3 elementos
print(a)
```

```
[1 2 3]
```

[123]:
```
b=np.array([[1.0,2,3],[20,30,40]]) # Matriz de 2 filas y 3 columnas
print(b)
```

```
[[ 1.  2.  3.]
 [20. 30. 40.]]
```

Arrays a partir de sus dimensiones

[124]:
```
# Array de 0s
np.zeros((2,3))  # El argumento es la tupla: (número_filas, número columnas)
```

[124]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

[125]:
```
# Array de 1s
np.ones((2,2))
```

[125]:
```
array([[1., 1.],
       [1., 1.]])
```

[126]:
```
# Array con valores equiespaciados (paso=1, por defecto)
np.arange(1,13,3)
```

[126]:
```
array([ 1,  4,  7, 10])
```

[127]:
```
# Array de k valores equiespaciados
k=5
np.linspace(0,1000,k)
```

[127]:
```
array([   0.,  250.,  500.,  750., 1000.])
```

[128]:
```
# Array con asignación de dimensiones y valor inicial
np.full((2,3),5.6)
```

[128]:
```
array([[5.6, 5.6, 5.6],
       [5.6, 5.6, 5.6]])
```

[129]:
```
# Matriz identidad
np.eye(3)
```

[129]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[130]:
```
np.random.randint(5,10) # Valor entero aleatorio en el intervalor [5,9)
```

[130]:
```
5
```

[131]:
```
# Array de valores aleatorios de tipo entero en el intervalo [0, 10)
np.random.randint(10,size=5)
```

[131]:
```
array([0, 6, 5, 7, 2])
```

[132]:
```
# Array de valores aleatorios de tipo entero en el intervalo [1,5)
np.random.randint(1,5,size=(2,6))  #
```

[132]:
```
array([[4, 3, 1, 3, 3, 1],
       [3, 1, 1, 2, 2, 4]])
```

[133]:
```
np.random.random() # Valor real aleatorio en el intervalo [0,1)
```

[133]:
```
0.33558175215187713
```

[134]:
```
# Array de valores aleatorios en el intervalo [0,1)
np.random.random((2,4))
```

[134]:
```
array([[0.28515763, 0.64138496, 0.7339581 , 0.89582354],
       [0.8574481 , 0.47801444, 0.10068787, 0.8626337 ]])
```

[135]:
```
np.empty((2,2))
```

[135]:
```
array([[ 250.,  500.],
       [ 750., 1000.]])
```

3.1.2 Acceso a los elementos de un array

[136]:
```
a=np.random.random((3,4))
print(a)
```

```
[[0.25298079 0.19259176 0.06255713 0.34867772]
 [0.29403863 0.35705405 0.15179017 0.44647549]
 [0.17259901 0.05771529 0.69915521 0.84657434]]
```

[137]:
```
a[0] # fila 0
```

[137]:
```
array([0.25298079, 0.19259176, 0.06255713, 0.34867772])
```

[138]:
```
a[0,1] # fila0, columna 1
```

[138]:
```
0.1925917638516138
```

[139]:
```
a[1,0:2] #fila 1, columnas 0 y 1
```

[139]:
```
array([0.29403863, 0.35705405])
```

[140]:
```
a[0, 2:] # fila 1, columnas 2 y siguientes
```

[140]:
```
array([0.06255713, 0.34867772])
```

[141]:
```
a[:2,1:] # filas 0 y 1, columnas 1 y siguientes
```

[141]:
```
array([[0.19259176, 0.06255713, 0.34867772],
       [0.35705405, 0.15179017, 0.44647549]])
```

3.1.3 Consultar dimensiones de un array

[142]:
```
a=np.random.random((2,4))
```

[143]:
```
a.shape # (filas, columnas) de a
```

[143]:
```
(2, 4)
```

[144]:
```
a.shape[1] # Número de columnas de a
```

[144]:
```
4
```

[145]:
```
len(a) # longitud de a (número de filas)
```

[145]:
```
2
```

3.1.4 Operaciones con arrays

[146]:
```
a=np.array([[10,11,12],[20,21,22]])
b=np.array([[1,1,1],[10,10,10]])
```

[147]:
```
a+b
```

[147]:
```
array([[11, 12, 13],
       [30, 31, 32]])
```

[148]:
```
np.add(a,b)
```

[148]:
```
array([[11, 12, 13],
       [30, 31, 32]])
```

[149]:
```
a-b
```

[149]:
```
array([[ 9, 10, 11],
       [10, 11, 12]])
```

[150]:
```
np.subtract(a,b)
```

[150]:
```
array([[ 9, 10, 11],
       [10, 11, 12]])
```

[151]:
```
a*b  # Producto elemento a elemento
```

[151]:
```
array([[ 10,  11,  12],
       [200, 210, 220]])
```

[152]:
```
np.multiply(a,b)  # Producto elemento a elemento
```

[152]:
```
array([[ 10,  11,  12],
       [200, 210, 220]])
```

[153]:
```
a/b # División elemento a elemento
```

[153]:
```
array([[10. , 11. , 12. ],
       [ 2. ,  2.1,  2.2]])
```

[154]:
```
np.divide(a,b)
```

[154]:
```
array([[10. , 11. , 12. ],
       [ 2. ,  2.1,  2.2]])
```

[155]:
```
np.exp(a)
```

[155]:
```
array([[2.20264658e+04, 5.98741417e+04, 1.62754791e+05],
       [4.85165195e+08, 1.31881573e+09, 3.58491285e+09]])
```

[156]:
```
np.sqrt(a)
```

[156]:
```
array([[3.16227766, 3.31662479, 3.46410162],
       [4.47213595, 4.58257569, 4.69041576]])
```

[157]:
```
valor_pi=np.pi  # Valor del número 'pi' definido en numpy
np.cos(2*valor_pi) # Funciones trigonométricas -> ángulos en radianes
```

[157]:
```
1.0
```

[158]:
```
np.sin(a)
```

[158]:
```
array([[-0.54402111, -0.99999021, -0.53657292],
       [ 0.91294525,  0.83665564, -0.00885131]])
```

[159]:
```
np.cos(a)
```

[159]:
```
array([[-0.83907153,  0.0044257 ,  0.84385396],
       [ 0.40808206, -0.54772926, -0.99996083]])
```

[160]:
```
np.tan(a)
```

[160]:
```
array([[ 6.48360827e-01, -2.25950846e+02, -6.35859929e-01],
       [ 2.23716094e+00, -1.52749853e+00,  8.85165604e-03]])
```

[161]:
```
np.e # Valor del número 'e' definido en numpy
```

[161]:
```
2.718281828459045
```

[162]:
```
np.log(np.array([1,np.e])) # Logaritmo neperiano
```

[162]:
```
array([0., 1.])
```

[163]:
```
np.log10(10) # Logaritmo decimal
```

[163]:
```
1.0
```

[164]:
```
c=np.array([1,2,3])
d=np.array([1,10,100])
```

```
c.dot(d) # Producto escalar
```

[164]: 321

[165]:
```
e=np.array([1,2,3])
f=np.array([3,2,1])
np.cross(e,f) # Producto vectorial
```

[165]: array([-4, 8, -4])

[166]:
```
np.transpose(g) # Transpuesta de una matriz
```

[166]: array(9.8)

[167]:
```
g=np.array([[1,2,3],[10,20,30]])
h=np.array([[1,1],[2,2],[3,0]])
g@h # Producto de dos matrices
```

[167]:
```
array([[ 14,   5],
       [140,  50]])
```

[168]:
```
g.ravel()  # Convertir a matriz unidimensional
```

[168]: array([1, 2, 3, 10, 20, 30])

[169]:
```
g.reshape((3,2)) # Redimensionar una matriz
```

[169]:
```
array([[ 1,  2],
       [ 3, 10],
       [20, 30]])
```

3.1.5 Almacenamiento/Lectura de arrays en disco

Archivos binarios

[170]:
```
a=np.random.random((2,4))
print(a)
```

```
[[0.05820756 0.64264491 0.79803866 0.08773735]
 [0.23878217 0.70057766 0.72135428 0.13473185]]
```

[171]:
```
np.save('array_a',a) #almacena el array a en el archivo 'array_a.npy'
```

[172]:
```
a_read=np.load('array_a.npy') # lee el array del archivo 'array_a.npy'
print(a_read)
```

```
[[0.05820756 0.64264491 0.79803866 0.08773735]
 [0.23878217 0.70057766 0.72135428 0.13473185]]
```

Archivos de texto

[173]:
```
np.savetxt('array_a.csv',a, delimiter=',')
```

[174]:
```
a_csv=np.genfromtxt('array_a.csv', delimiter=',')
print(a_csv)
```

```
[[0.05820756 0.64264491 0.79803866 0.08773735]
 [0.23878217 0.70057766 0.72135428 0.13473185]]
```

3.2 Matplotlib

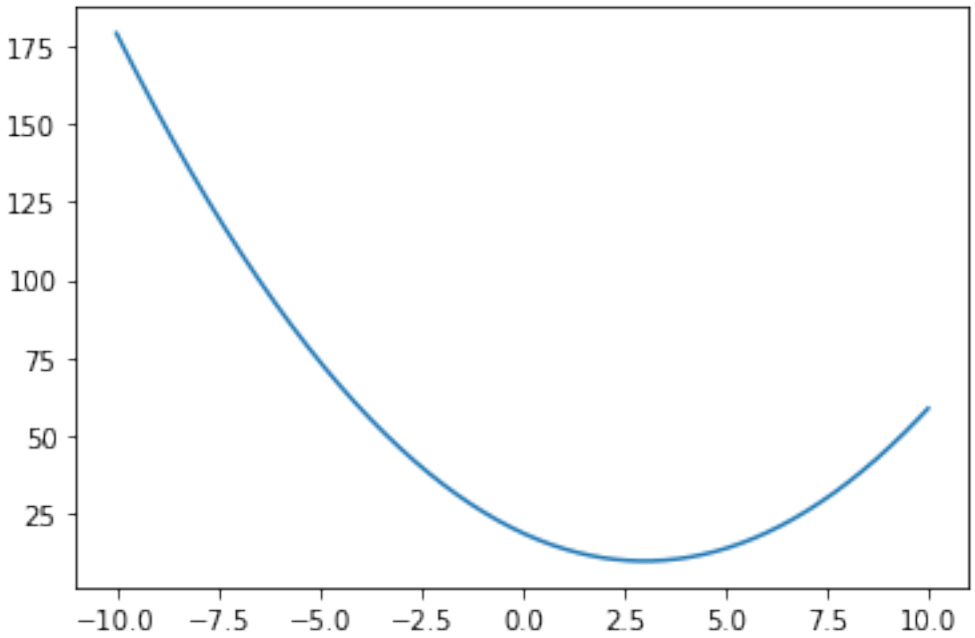
Librería con utilidades para representación gráfica

[175]:
```
import matplotlib.pyplot as plt
```

[176]:
```
import numpy as np
x=np.arange(-10,10,0.01)
y=(x-3)**2+10
z=np.sin(x)
```

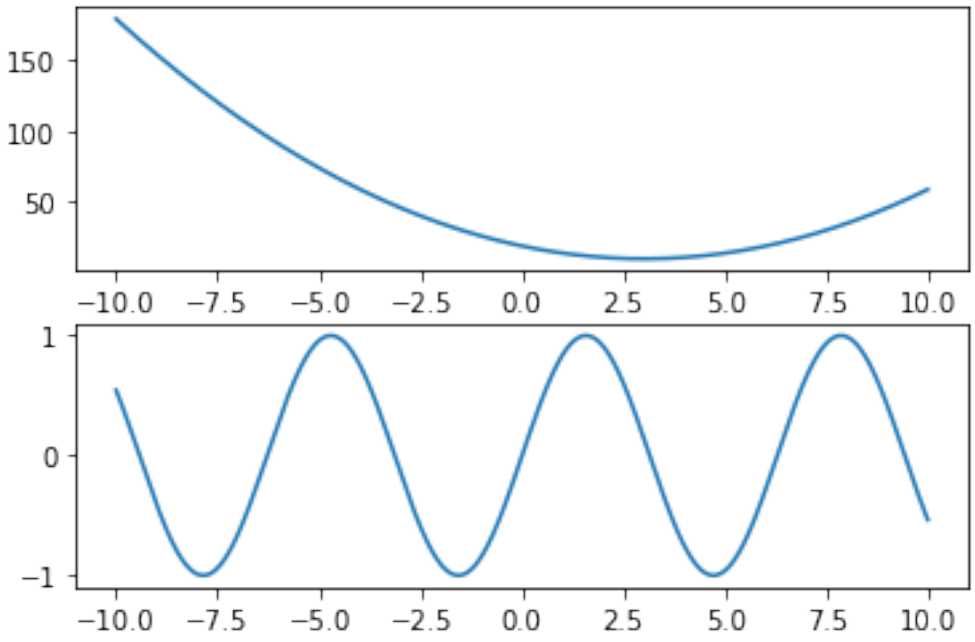
[177]:
```
plt.plot(x,y)  # Curva x-y
```

[177]:
```
[<matplotlib.lines.Line2D at 0x11ada9450>]
```



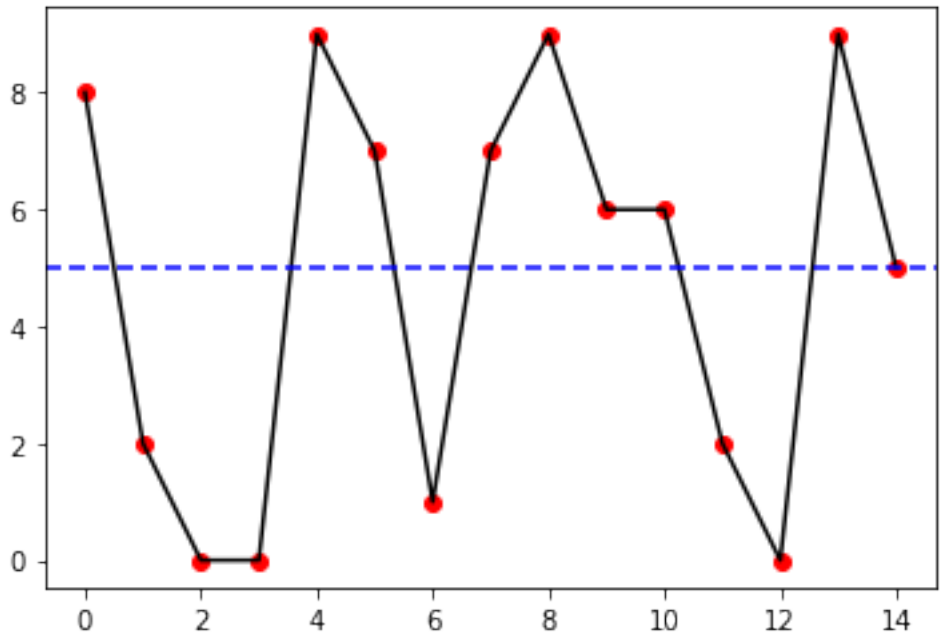
Figuras con uno o más gráficos (*subplots*)

[178]:
```
# 1. Crear una figura
figura=plt.figure()
# 2. Asignar ejes
ax1=figura.add_subplot(211) # Usando 2 filas y 1 columnas, ax1 corresponde al␣
 ↪gráfico 1
ax2=figura.add_subplot(212) # Usando 2 filas y 1 columnas, ax2 corresponde al␣
 ↪gráfico 2
# 3. Dibujar en cada eje
ax1.plot(x,y)
#Segundo eje:
ax2.plot(x,z);
# 4. Mostrar la figura
# figura.show() # Necesario en una aplicación con ventanas gráficas
```



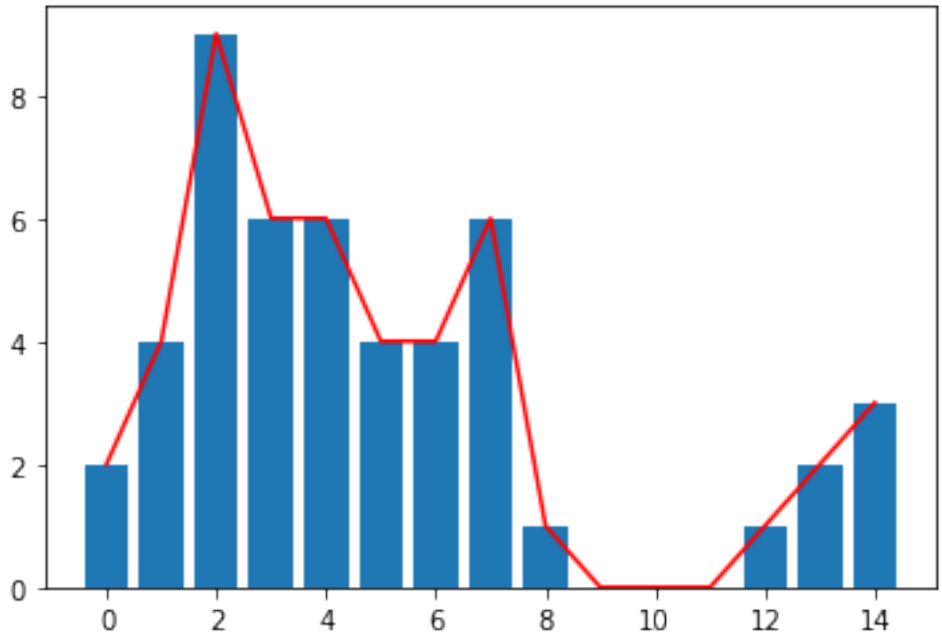
[179]:
```
x2=np.arange(0,15)
y2=np.random.randint(0,10,15)
plt.scatter(x2,y2,color='r') # Muestra los puntos en color rojo (Red) sin␣
 ↪unirlos
plt.plot(x2,y2,'k') # Plot de color negro (blacK)
# Línea horizontal:
    # Cota: 5
    # Color azul (Blue)
    # Discontinua (trazos de 4 puntos separados por 2 puntos)
plt.axhline(5,color='b',dashes=(4,2))
```

[179]:
```
<matplotlib.lines.Line2D at 0x11afa1e50>
```



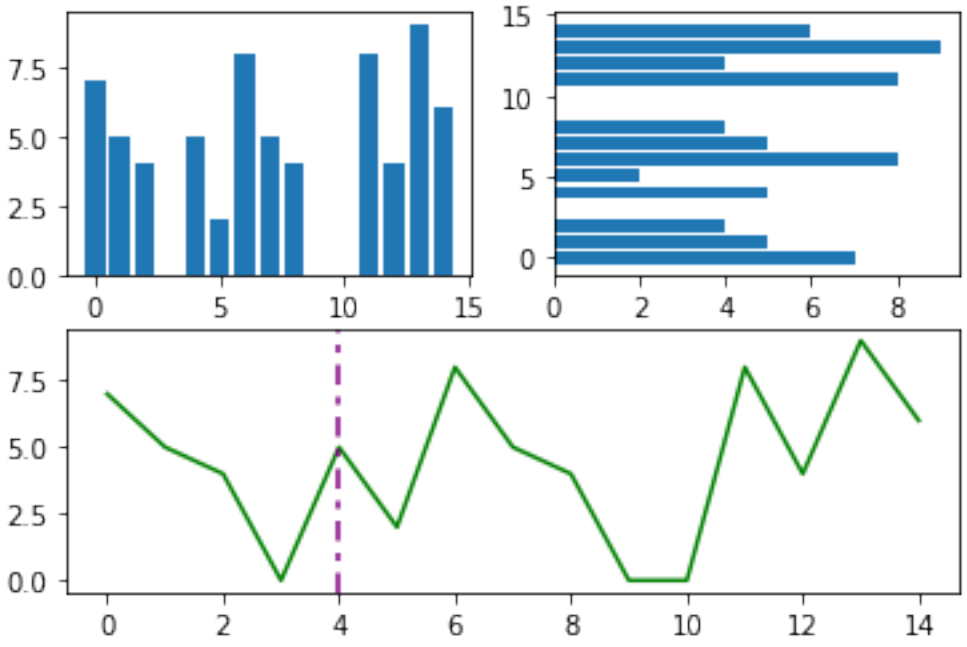
[180]:
```
x=np.arange(0,15)
y=np.random.randint(0,10,15)
plt.bar(x,y) # Diagrama de barras verticales
plt.plot(x,y,'r') # Muestra los puntos y los une mediante una línea
```

[180]: [<matplotlib.lines.Line2D at 0x11b09e250>]



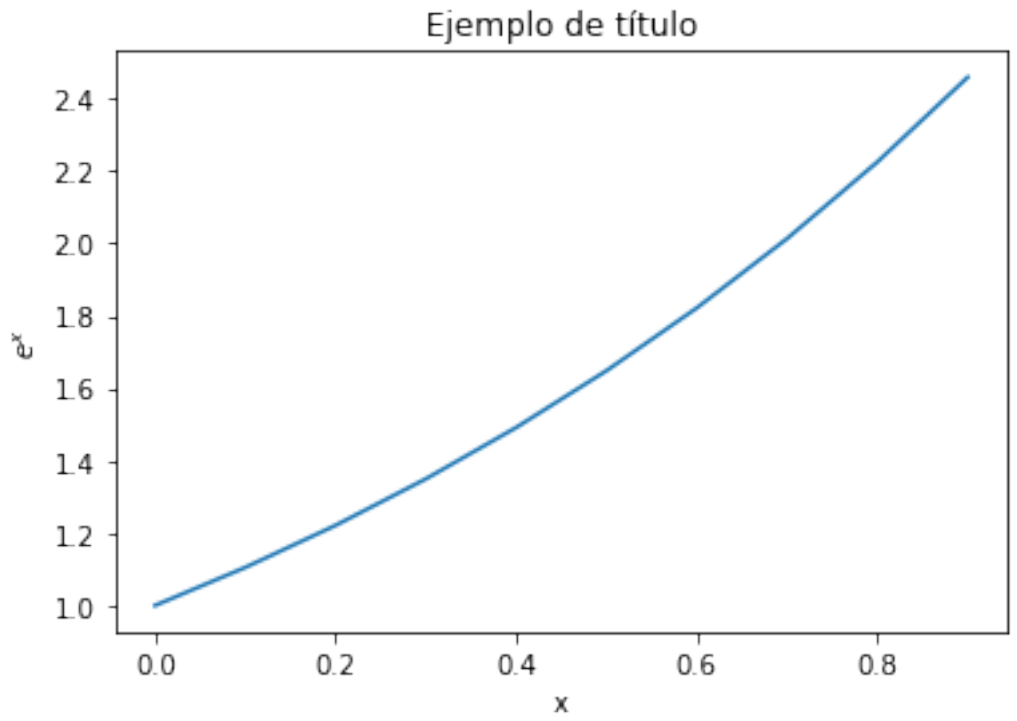
[181]:
```
x=np.arange(0,15)
y=np.random.randint(0,10,15)

fig2=plt.figure()
ax21=fig2.add_subplot(221)
ax22=fig2.add_subplot(222)
ax23=fig2.add_subplot(212)

ax21.bar(x,y) # Diagrama de barras verticales
ax22.barh(x,y) # Diagrama de barras horizontales
ax23.plot(x,y,'g') # plot x-y
#línea vertical:
    # Posición: 4
    # Color verde
    # Línea de ejes (trazos de 5 puntos y puntos de 2 puntos; separaciones de 2␣
 ↪puntos)
ax23.axvline(4 ,color='purple',dashes=(5,3,2,3))
```

[181]: <matplotlib.lines.Line2D at 0x11b2a67d0>



[182]:
```
x3=np.arange(0,1.0,0.1)
y3=np.exp(x3)

fig3=plt.figure()
ax3=fig3.add_subplot(111)

'''Asignar títulos a los ejes'''
ax3.set(title='Ejemplo de título',
        ylabel= r'$e^x$',   # Formato LaTeX para Ecuaciones
        xlabel='x')
ax3.plot(x3,y3) # plot x-y
```

[182]:
```
[<matplotlib.lines.Line2D at 0x11b3a1c10>]
```

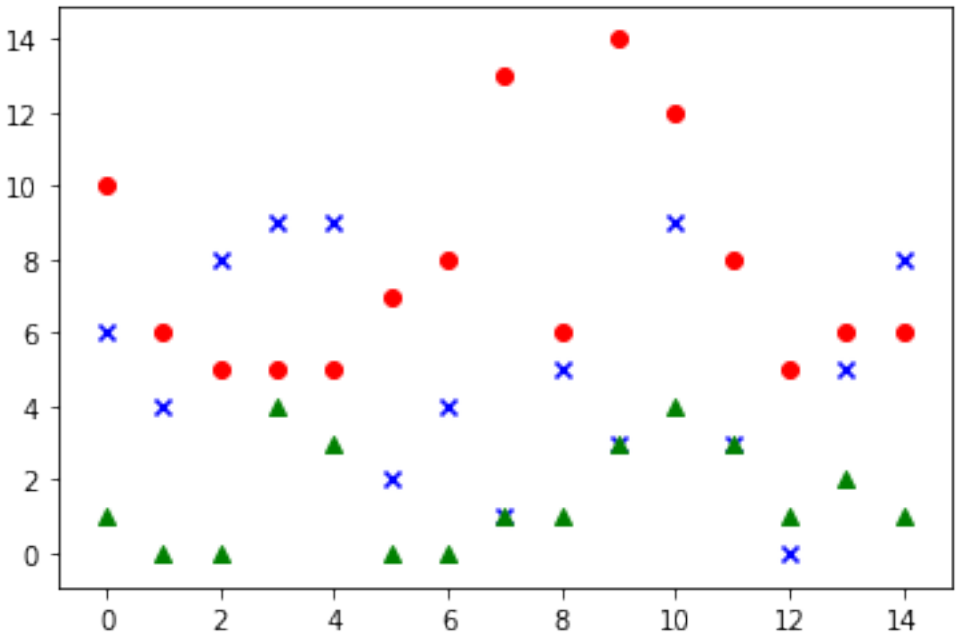


Mostrar imágenes

[183]:
```
'''Configuración de Jupyter Lab para mostrar imágenes y gráficos de matplotlib:
%matplotlib inline -> (opción por defecto) En la ventana de Jupyter Lab
%matplotlib qt  ->  En una ventana diferente con opciones de 'zoom' y 'grabar',␣
 ↪entre otras
'''
%matplotlib qt
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg  # Adecuada para imágenes en formato png

# Abrir archivo con una imagen
img = mpimg.imread('http://ingenierias.unileon.es/wp-content/uploads/2019/10/
 ↪logo_ingenierias_retina.png')
# No mostrar los ejes del gráfico
plt.axis('off')
# Mostrar la imagen
plt.imshow(img);
```

Guardar figuras

[184]:
```
x=np.arange(0,15)
y=np.random.randint(0,10,15)
z=np.random.randint(5,15,15)
p=np.random.randint(0,5,15)

fig=plt.figure()
ax=fig.add_subplot(111)
ax.scatter(x,y, marker='x', color='blue')
ax.scatter(x,z, marker='o', color='red')
ax.scatter(x,p, marker='^', color='green')

# Grabar la imagen:
    # dpi (puntos por pulgada): valor por defecto, el de la visualización
    # transparent: por defecto el fondo no es transparente (False)
fig.savefig('mi_imagen.png') # Por defecto, el dpi de la gráfica  (dpi)
fig.savefig('mi_imagen_fondo_transparente.png', dpi=150, transparent=True)
```



Nota: También se pueden guardar las figuras desde las ventanas de la aplicación