Ejercicios de introducción a Numpy

In [1]:
```
import numpy as np
print(np.__version__)
```

```
1.16.2
```

1.- Extrae todos los número impares de un array:

```
In: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
Out: array([1, 3, 5, 7, 9])
```

In [2]:
```
numbers = np.array(np.arange(10))
numbers[numbers % 2 == 1]
```

Out[2]:
```
array([1, 3, 5, 7, 9])
```

2.- Reemplaza los números impares de un array con valores fijos:

```
In: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
Out: array([0, -1, 2, -1, 4, -1, 6, -1, 8, -1])
```

In [3]:
```
# np.where(numbers % 2 == 1, -1, numbers)
numbers[numbers % 2 == 1] = -1
numbers
```

Out[3]:
```
array([ 0, -1,  2, -1,  4, -1,  6, -1,  8, -1])
```

3.- Reemplaza los números impares de un array con valores fijos sin afectar al array original:

```
In: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
Out: array([0, -1, 2, -1, 4, -1, 6, -1, 8, -1])
```

In [3]:
```
numbers = np.array(np.arange(10))
new_numbers = np.where(numbers % 2 == 1, -1, numbers)
new_numbers
```

Out[3]:
```
array([ 0, -1,  2, -1,  4, -1,  6, -1,  8, -1])
```

4.- Convertir un array de 1D en un array de 2D:

```
In: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
Out: array([[0, 1, 2, 3, 4],
            [5, 6, 7, 8, 9]])
```

In [4]:
```
arr1 = np.arange(10).reshape(2, -1)
arr1
```

Out[4]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

5.- Apilar verticalmente dos arrays:

```
In: array([[0, 1, 2, 3, 4],
           [5, 6, 7, 8, 9]]),
    array([[1, 1, 1, 1, 1]
           [1, 1, 1, 1, 1]])
Out: array([[0, 1, 2, 3, 4],
            [5, 6, 7, 8, 9],
            [1, 1, 1, 1, 1],
            [1, 1, 1, 1, 1]])
```

In [6]:
```
arr2 = np.ones_like(arr1)
# arr2 = np.repeat(1, 10).reshape(2,-1)

# Método 1:
np.concatenate([arr1, arr2], axis=0)

# Método 2:
np.vstack([arr1, arr2])

# Método 3:
np.r_[arr1, arr2]
```

Out[6]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

6.- Apilar horizontalmente dos arrays:

```
In: array([[0, 1, 2, 3, 4],
           [5, 6, 7, 8, 9]]),
    array([[1, 1, 1, 1, 1]
           [1, 1, 1, 1, 1]])
Out: array([[0, 1, 2, 3, 4, 1, 1, 1, 1, 1],
            [5, 6, 7, 8, 9, 1, 1, 1, 1, 1]])
```

In [7]:
```
# Método 1:
np.concatenate([arr1, arr2], axis=1)

# Método 2:
np.hstack([arr1, arr2])

# Método 3:
np.c_[arr1, arr2]
```

Out[7]:
```
array([[0, 1, 2, 3, 4, 1, 1, 1, 1, 1],
       [5, 6, 7, 8, 9, 1, 1, 1, 1, 1]])
```

7.- Crea el siguiente patrón sin codificación. Utilice solo las funciones numpy y la siguiente matriz de entrada:

```
In: arr1 = np.array([1,2,3])
Out: array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

In [8]:
```
arr1 = np.array([1,2,3])
np.r_[np.repeat(arr1, 3), np.tile(arr1, 3)]
```

Out[8]:
```
array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

8.- Obtener las posiciones de los elementos que coinciden en dos arrays:

```
In: array([1,2,3,2,3,4,3,4,5,6]),
     array([7,2,10,2,7,4,9,4,9,8])
Out: array([1, 3, 5, 7])
```

In [4]:
```
arr1 = np.array([1,2,3,2,3,4,3,4,5,6])
arr2 = np.array([7,2,10,2,7,4,9,4,9,8])
positions = np.where(arr1 == arr2)
positions[0]
```

Out[4]:
```
array([1, 3, 5, 7])
```

9.- Obtener todos los items entre 5 y 10 del siguiente array:

```
In: array([2, 6, 1, 9, 10, 3, 27])
Out: array([6, 9, 10])
```

In [6]:
```
arr1 = np.array([2, 6, 1, 9, 10, 3, 27])
# arr1[(arr1 >= 5) & (arr1 <= 10)]
index = np.where((arr1 >= 5) & (arr1 <= 10))
arr1[index].toList()
```

Out[6]:
```
array([ 6,  9, 10])
```

10.- Intercambia dos columnas de un array (columnas 0 y 1):

```
In: array([[0, 1, 2],
           [3, 4, 5],
           [6, 7, 8]])
Out: array([[1, 0, 2],
            [4, 3, 5],
            [7, 6, 8]])
```

```
In [11]: arr1 = np.arange(9).reshape(3,3)
         # arr1.take([1, 0, 2], axis=1)
         arr1[:, [1,0,2]]
```

```
Out[11]: array([[1, 0, 2],
                [4, 3, 5],
                [7, 6, 8]])
```

11.- Intercambia dos filas de un array (filas 0 y 1):

```
In: array([[0, 1, 2],
           [3, 4, 5],
           [6, 7, 8]])
Out: array([[3, 4, 5],
            [0, 1, 2],
            [6, 7, 8]])
```

```
In [12]: # arr1.take([1, 0, 2], axis=0)
         arr1[[1,0,2], :]
```

```
Out[12]: array([[3, 4, 5],
                [0, 1, 2],
                [6, 7, 8]])
```

12.- Invierte todas las filas de un array 2D:

```
In: array([[0, 1, 2],
           [3, 4, 5],
           [6, 7, 8]])
Out: array([[6, 7, 8],
            [3, 4, 5],
            [0, 1, 2]])
```

```
In [13]: arr1[::-1]
```

```
Out[13]: array([[6, 7, 8],
                [3, 4, 5],
                [0, 1, 2]])
```

13.- Invierte todas las columnas de un array 2D:

```
In: array([[0, 1, 2],
           [3, 4, 5],
           [6, 7, 8]])
Out: array([[2, 1, 0],
            [5, 4, 3],
            [8, 7, 6]])
```

```
In [14]: arr1[:, ::-1]
```

```
Out[14]: array([[2, 1, 0],
                [5, 4, 3],
                [8, 7, 6]])
```

14.- Crear un array 2D con *shape* 5x3 que contenga números decimales aleatorios entre 5 y 10:

```
Out: [[ 8.50061025  9.10531502  6.85867783]
      [ 9.76262069  9.87717411  7.13466701]
      [ 7.48966403  8.33409158  6.16808631]
      [ 7.75010551  9.94535696  5.27373226]
      [ 8.0850361   5.56165518  7.31244004]]
```

```
In [7]: # Fija X decimales en impresión
        np.set_printoptions(precision=6)

        # Método 1:
        rand_arr = np.random.randint(low=5, high=10, size=(5,3)) + np.rando
        m.random((5,3))
        # print(rand_arr)

        # Método 2:
        rand_arr = np.random.uniform(5,10, size=(5,3))
        print(rand_arr)
```

```
[[8.874343 5.938674 8.636287]
 [9.860435 5.852782 9.636226]
 [8.021759 9.386227 5.772033]
 [6.550057 5.321249 9.234174]
 [5.808686 6.658956 6.523695]]
```

15.- indicar el número de elementos nulos, igual a `np.nan` , y su posición:

```
In: array([np.nan,  1,  2,  3, np.nan,  4,  5, np.nan,  7,  8,  9, np.n
an])
Out: Hay 4 valores nulos en las posiciones [0, 4, 7, 11]
```

In [5]:
```
arr1 = np.array([np.nan, 1, 2, 3, np.nan, 4, 5, np.nan, 7, 8, 9, n
p.nan])
print('Hay {0} valores nulos en las posiciones {1}'.format(np.isnan
(arr1).sum(), np.where(np.isnan(arr1))[0].tolist()))
```

```
Hay 4 valores nulos en las posiciones [0, 4, 7, 11]
```

Out[5]:
```
array([ True, False, False, False,  True, False, False,  True, Fal
se,
       False, False,  True])
```

16.- Escribir un código que devuelva los valores únicos de un array y su número de apariciones:

```
In: array([1, 1, 2, 3, 3, 4, 5, 5, 7, 8, 9, 9])
Out: (array([1, 2, 3, 4, 5, 7, 8, 9]), array([2, 1, 2, 1, 2, 1, 1, 2]))
```

In [17]:
```
arr1 = np.array([1, 1, 2, 3, 3, 4, 5, 5, 7, 8, 9, 9])
np.unique(arr1, return_counts=True)
```

Out[17]:
```
(array([1, 2, 3, 4, 5, 7, 8, 9]), array([2, 1, 2, 1, 2, 1, 1, 2]))
```

17.- Crear una matriz de 5x5 como la siguiente:

```
Out: [[1. 1. 1. 1. 1.]
      [1. 0. 0. 0. 1.]
      [1. 0. 0. 0. 1.]
      [1. 0. 0. 0. 1.]
      [1. 1. 1. 1. 1.]]
```

In [15]:
```
'''
arr1 = np.zeros((5, 5))
arr1[[0, -1], :] = 1
arr1[:, [0, -1]] = 1
'''
'''
arr1 = np.ones((5, 5))
arr0 = np.zeros((3, 3))
arr1[1:-1, 1:-1] = arr0
'''
arr1 = np.ones((5, 5))
arr1[1:-1, 1:-1] = 0
print(arr1)
```

```
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

18.- Crear una matriz de 5x5 como la siguiente:

```
Out: [[1 0 0 0 0]
      [0 2 0 0 0]
      [0 0 3 0 0]
      [0 0 0 4 0]
      [0 0 0 0 5]]
```

In [19]:
```
'''
arr1 = np.zeros((5, 5), dtype="uint8")
for i in range(5):
    arr1[i][i] = i+1
'''
arr1 = np.diag([1, 2, 3, 4, 5])
print(arr1)
```

```
[[1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]
 [0 0 0 0 5]]
```

19.- Crear una matriz de 4x4 como la siguiente:

```
Out: [[0. 1. 0. 1.]
      [1. 0. 1. 0.]
      [0. 1. 0. 1.]
      [1. 0. 1. 0.]]
```

In [18]:
```
'''
arr1 = np.zeros((4, 4))
arr1[::2, 1::2] = 1
arr1[1::2, ::2] = 1
print(arr1)
'''
arr1 = np.ones((4, 4))
arr1[::2, ::2] = 0
arr1[1::2, 1::2] = 0
print(arr1)
```

```
[[0. 1. 0. 1.]
 [1. 0. 1. 0.]
 [0. 1. 0. 1.]
 [1. 0. 1. 0.]]
```

20.- Proporcionar la suma de todos los valores de una matriz y de sus filas y columnas:

```
In: array([[0,1],[2,3]])
Out: Suma total 6, por filas [2, 4] por columnas [1, 5]
```

In [11]:
```
arr1 = np.array([[0,1],[2,3]])
print('Suma total {0}, por filas {1} por columnas {2}'.format(np.su
m(arr1), np.sum(arr1, axis=0), np.sum(arr1, axis=1)))
```

```
Suma total 6, por filas [2 4] por columnas [1 5]
```

21.- Crear un programa que dada una matriz añada un borde de 0's rodeándola:

```
In: array([[ 1.  2.  3.]
           [ 4.  5.  6.]
           [ 7.  8.  9.]])
Out: array([[ 0.  0.  0.  0.  0.]
            [ 0.  1.  2.  3.  0.]
            [ 0.  4.  5.  6.  0.]
            [ 0.  7.  8.  9.  0.]
            [ 0.  0.  0.  0.  0.]])
```

In [21]:
```
arr1 = np.arange(1,10).reshape(3, 3)
'''
arr2 = np.zeros((5, 5))
arr2[1: -1, 1:-1] = arr1
print(arr2)
'''
arr1 = np.pad(arr1, pad_width=1, mode='constant', constant_values=
0)
print(arr1)
```

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

22.- Crear una matriz de 3x5 rellena de '2':

```
Out: array([[2 2 2 2 2]
            [2 2 2 2 2]
            [2 2 2 2 2]])
```

In [19]:
```
# Método 1
arr1 = np.full((3, 5), 2, dtype=np.uint)
print(arr1)
# Método 2
arr1 = np.ones([3, 5], dtype=np.uint) * 2
```

```
[[2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]]
```

23.- Crear una matriz igual que una dada pero rellena de otro valor:

```
In: array([0, 1, 2, 3])
Out: array([10, 10, 10, 10])
```

```
In [24]: arr1 = np.arange(4, dtype=np.int64)
         # larr1[:] = 10 cambia arr1
         like_arr1 = np.full_like(arr1, 10)
         like_arr1
Out[24]: array([10, 10, 10, 10])
```

24.- Crear una matriz de 8x8 con el siguiente contenido:

```
Out: [[ 0  1  0  2  0  3  0  4]
      [16  0 15  0 14  0 13  0]
      [ 0  5  0  6  0  7  0  8]
      [12  0 11  0 10  0  9  0]
      [ 0  9  0 10  0 11  0 12]
      [ 8  0  7  0  6  0  5  0]
      [ 0 13  0 14  0 15  0 16]
      [ 4  0  3  0  2  0  1  0]]
```

```
In [31]: arr1 = np.zeros((8,8),dtype=int)
         arr1[1::2,::2] = np.arange(16, 0, -1).reshape(4, 4)
         arr1[::2,1::2] = np.arange(1, 17).reshape(4, 4)
         print(arr1)
```

```
[[ 0  1  0  2  0  3  0  4]
 [16  0 15  0 14  0 13  0]
 [ 0  5  0  6  0  7  0  8]
 [12  0 11  0 10  0  9  0]
 [ 0  9  0 10  0 11  0 12]
 [ 8  0  7  0  6  0  5  0]
 [ 0 13  0 14  0 15  0 16]
 [ 4  0  3  0  2  0  1  0]]
```

25.- Indicar el numero de bytes de ocupación en memoria de la matriz anterior:

```
In: [[0 1 0 1 0 1 0 1]
     [1 0 1 0 1 0 1 0]
     [0 1 0 1 0 1 0 1]
     [1 0 1 0 1 0 1 0]
     [0 1 0 1 0 1 0 1]
     [1 0 1 0 1 0 1 0]
     [0 1 0 1 0 1 0 1]
     [1 0 1 0 1 0 1 0]]
Out: 512 bytes
```

In [146]:
```
# print("%d bytes" %(arr1.size * arr1.itemsize))
print("%d bytes" % (arr1.nbytes))
```

```
24 bytes
```

26.- Crear una matriz de 6x5 con 30 números comprendidos entre 2.5. y 6.5, inclusive.

```
Out: [[2.5       , 2.63793103, 2.77586207, 2.9137931 , 3.05172414],
      [3.18965517, 3.32758621, 3.46551724, 3.60344828, 3.74137931],
      [3.87931034, 4.01724138, 4.15517241, 4.29310345, 4.43103448],
      [4.56896552, 4.70689655, 4.84482759, 4.98275862, 5.12068966],
      [5.25862069, 5.39655172, 5.53448276, 5.67241379, 5.81034483],
      [5.94827586, 6.0862069 , 6.22413793, 6.36206897, 6.5       ]]
```

In [27]:
```
np.linspace(2.5, 6.5, 30).reshape(6, 5)
```

Out[27]:
```
array([[2.5     , 2.637931, 2.775862, 2.913793, 3.051724],
       [3.189655, 3.327586, 3.465517, 3.603448, 3.741379],
       [3.87931 , 4.017241, 4.155172, 4.293103, 4.431034],
       [4.568966, 4.706897, 4.844828, 4.982759, 5.12069 ],
       [5.258621, 5.396552, 5.534483, 5.672414, 5.810345],
       [5.948276, 6.086207, 6.224138, 6.362069, 6.5     ]])
```

27.- Obtener el cuarto elemento de la siguiente matriz:

```
In: array([[ 2, 4, 6],
           [ 6, 8, 10]])
Out: 6
```

In [144]:
```
arr1 = np.array([[2, 4, 6], [6, 8, 10]], np.int32)
# arr1[1, 0]
# arr1.take(3)
# arr1.ravel()[3]
# arr1.flatten()[3]
arr1.flat[3]
```

Out[144]:
```
6
```

28.- Cambiar la siguiente matriz según el patrón inidcado:

```
In: array([[1,2,3]])
Out: array([[1],
            [2],
            [3]])
```

In [32]:
```
arr1 = np.array([[1,2,3]])
# np.swapaxes(arr1,0,1)
# arr1.reshape(3, -1)
arr1.T
```

Out[32]:
```
array([[1],
       [2],
       [3]])
```

29.- Obtener la suma de los números múltiplos de 3 o de 5 contenidos entre el 1 y el 100:

```
Out: 2318
```

In [37]:
```
arr1 = np.arange(10000000, dtype=np.float)
%time arr1[list(map(lambda x: x % 3 == 0 or x % 5 == 0, arr1))].sum
()
%time arr1[(arr1 % 3 == 0) | (arr1 % 5 == 0)].sum()
```

```
CPU times: user 10.6 s, sys: 79.3 ms, total: 10.7 s
Wall time: 10.8 s
CPU times: user 295 ms, sys: 54 ms, total: 349 ms
Wall time: 351 ms
```

Out[37]:
```
23333331666668.0
```

30.- Crear una función que haga la raíz cuadrada y sume uno a todos los elementos de cualquier matrix:

```
In: array([1, 4, 9])
Out: array([2, 3, 4])
```

In [143]:
```
def f(x):
    # sqrt(x) + 1
    return np.sqrt(x) + 1 # np.sqrt() es vectorizable, math.sqrt()
no lo sería

vf = np.vectorize(f)

def array_for(x):
    return np.array([f(xi) for xi in x])

def array_map(x):
    return np.array(list(map(f, x)))

def fromiter(x):
    return np.fromiter((f(xi) for xi in x), x.dtype)

def nditer(vector):
    with np.nditer(vector,
                   flags=['external_loop'],
                   op_flags=['readwrite']) as it:
        for x in it:
            x[...] = x**2 + 1
        return it.operands[0]

arr1 = np.array([1, 4, 9])

%time f(arr1)
%time array_for(arr1)
%time array_map(arr1)
%time fromiter(arr1)
%time nditer(arr1)
%time vf(arr1)
```

```
CPU times: user 34 µs, sys: 0 ns, total: 34 µs
Wall time: 41 µs
CPU times: user 58 µs, sys: 0 ns, total: 58 µs
Wall time: 65.1 µs
CPU times: user 52 µs, sys: 1e+03 ns, total: 53 µs
Wall time: 59.1 µs
CPU times: user 39 µs, sys: 1e+03 ns, total: 40 µs
Wall time: 46.3 µs
CPU times: user 53 µs, sys: 1e+03 ns, total: 54 µs
Wall time: 60.1 µs
CPU times: user 150 µs, sys: 1e+03 ns, total: 151 µs
Wall time: 158 µs
```

Out[143]:
```
array([   5,  290, 6725])
```

31.a - Importar el dataset `iris` (../data/iris.data) sin proporcionar un tipo por defecto y mostrar las 3 primeras filas recuperadas:

```
Out: array([[5.1, 3.5, 1.4, 0.2, nan],
            [4.9, 3. , 1.4, 0.2, nan],
            [4.7, 3.2, 1.3, 0.2, nan]])
```

In [148]:
```
# array 2D sin campos de tipo texto
iris = np.genfromtxt('../data/iris.data', delimiter=',', skip_header=True)
iris[:3]
```

Out[148]:
```
array([[5.1, 3.5, 1.4, 0.2, nan],
       [4.9, 3. , 1.4, 0.2, nan],
       [4.7, 3.2, 1.3, 0.2, nan]])
```

31.b - Importar el dataset `iris_2d` (../data/iris.data) manteniendo intactos los valores de tipo texto (**dtype**= `object`) y mostrar las 3 primeras filas recuperadas:

```
Out: array([[b'5.1', b'3.5', b'1.4', b'0.2', b'Iris-setosa'],
            [b'4.9', b'3.0', b'1.4', b'0.2', b'Iris-setosa'],
            [b'4.7', b'3.2', b'1.3', b'0.2', b'Iris-setosa']])
```

In [150]:
```
# array 2D manteniendo los valores de tipo texto
iris_2d = np.genfromtxt('../data/iris.data', delimiter=',', skip_header=True, dtype=object)
iris_2d[:3]
```

Out[150]:
```
array([[b'5.1', b'3.5', b'1.4', b'0.2', b'Iris-setosa'],
       [b'4.9', b'3.0', b'1.4', b'0.2', b'Iris-setosa'],
       [b'4.7', b'3.2', b'1.3', b'0.2', b'Iris-setosa']], dtype=object)
```

31.c - Importar el dataset `iris_1d` (../data/iris.data) manteniendo intactos los valores de tipo texto (**dtype**= `None`) y mostrar las 3 primeras filas recuperadas:

```
Out: array([(5.1, 3.5, 1.4, 0.2, 'Iris-setosa'),
            (4.9, 3. , 1.4, 0.2, 'Iris-setosa'),
            (4.7, 3.2, 1.3, 0.2, 'Iris-setosa')])
```

In [151]:
```
# array 1D manteniendo los valores de tipo texto
iris_1d = np.genfromtxt('../data/iris.data', delimiter=',', skip_header=True, dtype=None, encoding=None)
iris_1d[:3]
```

Out[151]:
```
array([(5.1, 3.5, 1.4, 0.2, 'Iris-setosa'),
       (4.9, 3. , 1.4, 0.2, 'Iris-setosa'),
       (4.7, 3.2, 1.3, 0.2, 'Iris-setosa')],
      dtype=[('f0', '<f8'), ('f1', '<f8'), ('f2', '<f8'), ('f3', '<f8'), ('f4', '<U15')])
```

32.- Recuperar los valores diferentes de columna de texto `species` para los dataset `iris_1d` e `iris_2d` obtenidos en el paso anterior:

```
Out: ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

In [152]:
```
# array 1D
list({row[4] for row in iris_1d})
```

Out[152]:
```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

In [154]:
```
# array 2D
np.unique(iris_2d[:, 4]).astype('str').tolist()
```

Out[154]:
```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

33.- Recuperar los valores diferentes de columna de texto `species` de los dataset `iris_1d` e `iris_2d` obtenidos en el paso anterior y su número de apariciones:

```
Out: (array([b'Iris-setosa', b'Iris-versicolor', b'Iris-virginica'], dt
ype='|S15'),
      array([50, 50, 50]))
```

In [156]:
```
# array 1D
'''
valores = [row[4] for row in iris]
{valor: valores.count(valor) for valor in valores}
'''
np.unique([row[4] for row in iris_1d], return_counts=True)
```

Out[156]:
```
(array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype
='<U15'),
 array([50, 50, 50]))
```

In [158]:
```
# array 2D
np.unique(iris_2d[:, 4], return_counts=True)
```

Out[158]:
```
(array([b'Iris-setosa', b'Iris-versicolor', b'Iris-virginica'],
       dtype=object), array([50, 50, 50]))
```

34.- Obtener la media, la mediana y la desviación estándar de la columna `sepallength` de los dataset `iris_1d` e `iris_2d`.

```
Out: 5.843333333333334 5.8 0.8253012917851409
```

In [159]:
```
# array 1D
sepallength = [x[0] for x in iris_1d]
mu, med, sd = np.mean(sepallength), np.median(sepallength), np.std
(sepallength)
print(mu, med, sd)
```

```
5.843333333333334 5.8 0.8253012917851409
```

In [206]:
```
# array 1D
sepallength = iris_2d[:, 0].astype('float')
mu, med, sd = sepallength.mean(), np.median(sepallength), np.std(se
pallength)
print(mu, med, sd)
```

```
5.843333333333334 5.8 0.8253012917851409
```

35.- Crear una versión normalizada del campo `sepallength` de los dataset `iris_1d` e `iris_2d` cuyos valores estén entre 0 y 1, de modo que el valor mínimo sea 0 y el máximo 1.

```
Out: array([0.22222222, 0.16666667, 0.11111111, ..., 0.61111111, 0.5277
7778, 0.44444444])
```

In [188]:
```
np.set_printoptions(threshold=6)
# sepallength = [x[0] for x in iris_1d]
# Smax, Smin = max(sepallength), min(sepallength)
# S = (sepallength - Smin)/(Smax - Smin)
sepallength = np.array([x[0] for x in iris_1d])
Smax, Smin = sepallength.max(), sepallength.min()
Smax, Smin = sepallength.max(), sepallength.min()
(sepallength - Smin)/(Smax - Smin)
```

Out[188]:
```
array([0.22222222, 0.16666667, 0.11111111, ..., 0.61111111, 0.5277
7778,
       0.44444444])
```

In [207]:
```
sepallength = iris_2d[:, 0].astype('float')
(sepallength - sepallength.min())/sepallength.ptp()
```

Out[207]:
```
array([0.22222222, 0.16666667, 0.11111111, ..., 0.61111111, 0.5277
7778,
       0.44444444])
```

36.- Convertir el array 1D `iris_1d` en un array 2D (`iris_2d`) omitiendo el campo `species` .

```
Out: array([[5.1, 3.5, 1.4, 0.2],
            [4.9, 3. , 1.4, 0.2],
            [4.7, 3.2, 1.3, 0.2],
            ...,
            [6.5, 3. , 5.2, 2. ],
            [6.2, 3.4, 5.4, 2.3],
            [5.9, 3. , 5.1, 1.8]])
```

In [196]:
```
# iris_2d = np.genfromtxt('../data/iris.data', delimiter=',', skip_
header=True, dtype='float', usecols=[0,1,2,3])
iris_2d = np.array([row.tolist()[:4] for row in iris_1d])
iris_2d
```

Out[196]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       ...,
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

37.- Insertar el valor `np.nan` en el primer y último elemento del array `iris_2d` y en 18 posiciones aleatorias adicionales.

```
Out: array([[nan, 3.5, 1.4, 0.2],
            [4.9, nan, 1.4, 0.2],
            [4.7, 3.2, 1.3, 0.2],
            ...,
            [6.5, 3. , 5.2, 2. ],
            [6.2, 3.4, 5.4, 2.3],
            [5.9, 3. , 5.1, nan]])
```

In [197]:
```
np.random.seed(100)
iris_2d[0, 0] = np.nan
iris_2d[-1, -1] = np.nan
iris_2d[np.random.randint(150, size=18), np.random.randint(4, size=
18)] = np.nan
iris_2d
```

Out[197]:
```
array([[nan, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       ...,
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, nan]])
```

38.- Obtener todas las filas del array `iris_2db` que tengan algún valor `np.nan` .

```
Out: array([[nan, 3.5, 1.4, 0.2],
            [4.9, nan, 1.4, 0.2],
            [nan, nan, 1.4, 0.2],
            ...,
            [nan, 3.2, 5.9, nan],
            [6.7, 3.3, 5.7, nan],
            [5.9, 3. , 5.1, nan]])
```

In [204]:
```
# [row for row in iris_2d if np.any(np.isnan(row))]
%time iris_2d[np.array([np.any(np.isnan(row)) for row in iris_2d])]
%time iris_2d[[np.any(np.isnan(row)) for row in iris_2d]]
%time iris_2d[np.where(np.isnan(iris_2d))[0]]
```

```
CPU times: user 990 µs, sys: 344 µs, total: 1.33 ms
Wall time: 1.03 ms
CPU times: user 638 µs, sys: 10 µs, total: 648 µs
Wall time: 645 µs
CPU times: user 111 µs, sys: 101 µs, total: 212 µs
Wall time: 133 µs
```

Out[204]:
```
array([[nan, 3.5, 1.4, 0.2],
       [nan, 2.9, 1.4, 0.2],
       [5.8, nan, 1.2, 0.2],
       ...,
       [6. , 3. , 4.8, nan],
       [nan, 3.2, 5.9, 2.3],
       [5.9, 3. , 5.1, nan]])
```

39.- Encontrar el número y la posición de los elementos `np.nan` de la columna `sepallength` del array `iris_2d` .

```
Out: número de posiciones 'nan': 10
     posiciones de los valores 'nan': [  0   8  14 ...  98 107 143]
```

In [201]:
```
print("número de posiciones 'nan':", np.isnan(iris_2d[:, 0]).sum())
print("posiciones de los valores 'nan':", np.where(np.isnan(iris_2d
[:, 0]))[0])
```

```
número de posiciones 'nan': 6
posiciones de los valores 'nan': [  0   8  94  98 107 143]
```

40.- Filtrar las columas del array `iris_2d` cuyo valor de la columna `petallength` sea mayor que 1.5 y cuyo valor de `sepallength` sea menor de 5.0.

```
Out: array([[4.8, 3.4, 1.6, 0.2],
            [4.7, 3.2, 1.6, 0.2],
            [4.8, 3.1, 1.6, 0.2],
            [4.9, 2.4, 3.3, 1. ],
            [4.9, 2.5, 4.5, 1.7]])
```

In [42]:
```
condicion = (iris_2d[:, 2] > 1.5) & (iris_2d[:, 0] < 5.0)
iris_2d[condicion]
```

Out[42]:
```
array([[4.8, 3.4, 1.6, 0.2],
       [4.8, 3.4, 1.9, nan],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [4.9, 2.4, 3.3, 1. ],
       [4.9, 2.5, 4.5, 1.7]])
```

41.- Reemplaza todos los valores `np.nan` del array `iris_2d` con el valor 0.

```
Out: array([[0. , 3.5, 1.4, 0.2],
            [4.9, 0. , 1.4, 0.2],
            [4.7, 3.2, 1.3, 0.2],
            ...,
            [6.5, 3. , 5.2, 2. ],
            [6.2, 3.4, 5.4, 2.3],
            [5.9, 3. , 5.1, 0. ]])
```

```
In [43]: iris_2d[np.isnan(iris_2d)] = 0
         iris_2d
```

```
Out[43]: array([[0. , 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                ...,
                [6.5, 3. , 5.2, 2. ],
                [6.2, 3.4, 5.4, 2.3],
                [5.9, 3. , 5.1, 0. ]])
```

42.- Comprueba que el array `iris_2d` no tiene valores `np.nan` .

```
Out: False
```

```
In [44]: np.isnan(iris_2d).any()
```

```
Out[44]: False
```

43.- Crear una nueva columna en `iris_2d` cuyo valor sea la multiplicación de las columnas `petallength` y `sepallength` .

```
Out: array([[ 0.  ,  3.5 ,  1.4 ,  0.2 ,  0.  ],
            [ 4.9 ,  3.  ,  1.4 ,  0.2 ,  6.86],
            [ 4.7 ,  3.2 ,  1.3 ,  0.2 ,  6.11],
            ...,
            [ 6.5 ,  3.  ,  5.2 ,  2.  , 33.8 ],
            [ 6.2 ,  3.4 ,  5.4 ,  2.3 , 33.48],
            [ 5.9 ,  3.  ,  5.1 ,  1.8 , 30.09]])
```

In [45]:
```
sepallength = iris_2d[:, 0]
petallength = iris_2d[:, 2]
new_row = petallength * sepallength
# Introduce una nueva dimension para encajar con iris_2d
# new_row = new_row[:, np.newaxis]
new_row.shape = (len(iris_2d), 1)
new_iris_2d = np.hstack([iris_2d, new_row])
new_iris_2d
```

Out[45]:
```
array([[ 0.  ,  3.5 ,  1.4 ,  0.2 ,  0.  ],
       [ 4.9 ,  3.  ,  1.4 ,  0.2 ,  6.86],
       [ 4.7 ,  3.2 ,  1.3 ,  0.2 ,  6.11],
       ...,
       [ 6.5 ,  3.  ,  5.2 ,  2.  , 33.8 ],
       [ 6.2 ,  3.4 ,  5.4 ,  2.3 , 33.48],
       [ 5.9 ,  3.  ,  5.1 ,  0.  , 30.09]])
```

44.- Encontrar el segundo valor más grande de la columna `petallength` para la especie `Iris-setosa` .

Out: 1.7

In [205]:
```
iris_2d = np.genfromtxt('../data/iris.data', delimiter=',', skip_header=True, dtype='object')
petal_len_setosa = iris_2d[iris_2d[:, 4] == b'Iris-setosa', [2]].astype('float')
np.unique(np.sort(petal_len_setosa))[-2]
```

Out[205]:
```
1.7
```

45.- Calcular los percentiles 5 y 95 del campo `sepallength` del dataset `iris` .

Out: array([0. , 7.2])

In [208]:
```
np.percentile(sepallength, q=[5, 95])
```

Out[208]:
```
array([4.6  , 7.255])
```
