Ejercicio 1

In [1]:
```
import numpy as np
array = np.arange(4,100,4)
print(array)
```

```
[ 4  8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96]
```

Ejercicio 2

In [2]:
```
import numpy as np
array = np.arange(20).reshape(5,4)
print(array)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

Ejercicio 3

In [3]:
```
import numpy as np
array = np.arange(20)
array = np.flip(array)
print(array)
```

```
[19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0]
```

Ejercicio 4

In [4]:
```
import numpy as np
def invertir_filas(array):
    return np.flip(array,axis=0)
array = np.arange(20).reshape(5,4)
invertir_filas(array)
```

```
Out[4]:  array([[16, 17, 18, 19],
               [12, 13, 14, 15],
               [ 8,  9, 10, 11],
               [ 4,  5,  6,  7],
               [ 0,  1,  2,  3]])
```

Ejercicio 5

```
In [5]:  import numpy as np
         array = np.arange(12).reshape(4, 3)
         medias = np.mean(array, axis=0)
         print(medias)
```

```
[4.5 5.5 6.5]
```

Ejercicio 6

```
In [6]:  import numpy as np
         array = np.arange(12)
         def redimensionar(array):
             return array.reshape(4,3)
         print(redimensionar(array))
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

Ejercicio 7

```
In [7]:  import numpy as np
         def redimensionar(array):
             longitud = len(array)
             if(np.sqrt(longitud) **2 == longitud):
                 seccion = int(np.sqrt(longitud))
                 print(array.reshape(seccion,seccion))
             else:
                 print("No se puede redimensionar")
```

```
array = np.arange(25)
redimensionar(array)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

Ejercicio 8

In [8]:
```
import numpy as np
def maximo_de_fila(array):
    return np.max(array,axis=1)
array = np.arange(20).reshape(4,5)
print(array)
maximo_de_fila(array)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

Out[8]:
```
array([ 4,  9, 14, 19])
```

Ejercicio 9

In [9]:
```
import numpy as np
def contar_valores(array):
    valores, cuenta = np.unique(array, return_counts=True)
    cuenta = zip(valores,cuenta)
    return [a for a in cuenta]
array = np.array([1,1,1,2,2,3,3,3,3])
contar_valores(array)
```

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

Ejercicio 10

```
In [10]: import numpy as np
         def normalizar_array(array):
             print(array)
             media = np.mean(array, axis=0 )
             desviacion = np.std(array, axis=0)
             return array-media/desviacion
         array = np.arange(12).reshape(4,3)
         normalizar_array(array)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
Out[10]: array([[-1.34164079, -0.63978318,  0.06207442],
                [ 1.65835921,  2.36021682,  3.06207442],
                [ 4.65835921,  5.36021682,  6.06207442],
                [ 7.65835921,  8.36021682,  9.06207442]])
```

Ejericicio 11

```
In [11]: import numpy as np
         def normalizar_array(array):
             print(array)
             media = np.mean(array, axis=1, keepdims=1)
             desviacion = np.std(array, axis=0, keepdims=1)
             return array-media/desviacion
         array = np.arange(12).reshape(4,3)
         normalizar_array(array)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
Out[11]: array([[-0.2981424 ,  0.7018576 ,  1.7018576 ],
                [ 1.80743041,  2.80743041,  3.80743041],
                [ 3.91300322,  4.91300322,  5.91300322],
                [ 6.01857603,  7.01857603,  8.01857603]])
```

Ejercicio 12

```
In [12]: import numpy as np
         def get_max_min_idx(array):
             minIdx = np.unravel_index(np.argmin(array), array.shape)
             maxIdx = np.unravel_index(np.argmax(array), array.shape)
             return (minIdx,maxIdx)
         array = np.array([[2,5,4],[5,2,6],[1,2,8]])
         get_max_min_idx(array)
```

```
Out[12]: ((2, 0), (2, 2))
```

Ejercicio 13

```
In [13]: import numpy as np
         def sort_by_first_col(array):
             indices = np.argsort(array[:,0])
             return array[indices]

         array = np.array([[8,5,3],[9,3,7],[4,1,2]])
         sort_by_first_col(array)
```

```
Out[13]: array([[4, 1, 2],
                [8, 5, 3],
                [9, 3, 7]])
```

Ejercicio 14

```
In [14]: import numpy as np
         def generate_matrix(filas, columnas):
             array = np.random.randn(filas,columnas)
             array[array<0] = 0
             return array
         generate_matrix(7,5)
```

```
Out[14]: array([[0.        , 0.77122331, 0.87005724, 0.        , 0.62812327],
                [0.        , 0.        , 0.        , 0.05188529, 0.        ],
                [0.        , 0.09425629, 1.63515165, 0.23874955, 0.        ],
                [0.19782088, 0.        , 0.40593766, 2.65906832, 0.        ],
        [0.        , 0.01199465, 1.26983271, 0.07664429, 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.02564727],
        [0.        , 0.        , 0.        , 1.12085748, 0.4093709 ]])
```

Ejercicio 15

```
In [15]: # import numpy as np
         def max_n_numbers(array, k):
             if k <= 0:
                 print("Valor no valido")
             else:
                 indices = np.argsort(array)[::-1]
                 return indices[:k]

         array = np.array([10,5,7,2,34])
         max_n_numbers(array,4)
```

```
Out[15]: array([4, 0, 2, 1])
```

Ejercicio 16

```
In [16]: import numpy as np
         def generate_matrix(filas, columnas):
             array = np.random.uniform(size=(filas,columnas))
             array[:,:2] = 0
             array[:,-3:] = 1
             return array
         generate_matrix(6,7)
```

```
Out[16]: array([[0.        , 0.        , 0.97054978, 0.90394687, 1.        ,
                 1.        , 1.        ],
                [0.        , 0.        , 0.0568797 , 0.00267126, 1.        ,
                 1.        , 1.        ],
                [0.        , 0.        , 0.45662184, 0.82912969, 1.        ,
                 1.        , 1.        ],
                [0.        , 0.        , 0.30263314, 0.3852039 , 1.        ,
                 1.        , 1.        ],
                [0.        , 0.        , 0.64780273, 0.27337624, 1.        ,
 1.        , 1.        ],
[0.        , 0.        , 0.05832526, 0.49115941, 1.        ,
 1.        , 1.        ]])
```