EJERCICIOS DE LA BIBLIOTECA NUMPY

Ejercicio 1:

Escribe un programa que cree un array unidimensional con los enteros positivos múltiplos de 4 menores que 100.

```
import numpy as np

ejercicio1 = np.arange(4, 1000, 4)

print(ejercicio1)
```

Resultado:

```
[  4   8  12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
  76  80  84  88  92  96 100 104 108 112 116 120 124 128 132 136 140 144
 148 152 156 160 164 168 172 176 180 184 188 192 196 200 204 208 212 216
 220 224 228 232 236 240 244 248 252 256 260 264 268 272 276 280 284 288
 292 296 300 304 308 312 316 320 324 328 332 336 340 344 348 352 356 360
 364 368 372 376 380 384 388 392 396 400 404 408 412 416 420 424 428 432
 436 440 444 448 452 456 460 464 468 472 476 480 484 488 492 496 500 504
 508 512 516 520 524 528 532 536 540 544 548 552 556 560 564 568 572 576
 580 584 588 592 596 600 604 608 612 616 620 624 628 632 636 640 644 648
 652 656 660 664 668 672 676 680 684 688 692 696 700 704 708 712 716 720
 724 728 732 736 740 744 748 752 756 760 764 768 772 776 780 784 788 792
 796 800 804 808 812 816 820 824 828 832 836 840 844 848 852 856 860 864
 868 872 876 880 884 888 892 896 900 904 908 912 916 920 924 928 932 936
 940 944 948 952 956 960 964 968 972 976 980 984 988 992 996]
```

Ejercicio 2:

Escribe un programa que cree un array bidimensional de 5 filas y 4 columnas con los enteros desde 0 hasta 19.

```
import numpy as np

ejercicio2 = np.arange(0, 20, 1).reshape(5, 4)

print(ejercicio2)
```

Resultado:

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

Ejercicio 3:

Escribe un programa que invierte el orden de los elementos de un array unidimensional.

```
import numpy as np

ejercicio3 = np.arange(1,10)

ejercicio3Invertido = np.flip(ejercicio3)

print("Matriz normal:")

print(ejercicio3)

print("Matriz invertida:")

print(ejercicio3Invertido)
```

Resultado:

```
Matriz normal:

[1 2 3 4 5 6 7 8 9]

Matriz invertida:

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

Ejercicio 4:

Escribe un programa que invierte el orden de las filas de un array bidimensional.

```
import numpy as np

ejercicio4 = np.arange(0, 20, 1).reshape(5, 4)

ejercicio4Invertido = np.flip(ejercicio4, axis=1)

print("Matriz normal:")

print(ejercicio4)

print("Matriz invertida:")

print(ejercicio4Invertido)
```

Resultado:

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

Ejercicio 5:

Escribe un programa que calcule la media de los elementos de cada columna de un array bidimensional.

```
import numpy as np

ejercicio5 = np.arange(0, 20, 1).reshape(5, 4)

print("Matriz:")

print(ejercicio5)

print("Media:")

print(np.mean(ejercicio5, axis=0))
```

Resultado:

```
Matriz:

[[ 0  1  2  3]

 [ 4  5  6  7]

 [ 8  9 10 11]

 [12 13 14 15]

 [16 17 18 19]]

Media:

[ 8.  9. 10. 11.]
```

Ejercicio 6:

Escribe un programa que redimensione un array unidimensional a un array bidimensional con 4 filas y 3 columnas.

```
import numpy as np

e6 = np.arange(0,12)

print("Matriz:")

print(e6)

print("Matriz transformada:")

print(e6.reshape(4,3))
```

Resultado:

```
Matriz:

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

Matriz transformada:

[[ 0  1  2]

 [ 3  4  5]

 [ 6  7  8]

 [ 9 10 11]]
```

Ejercicio 7:

Escribe un programa que, dado un array unidimensional, compruebe si se puede redimensionar a un array bidimensional con el mismo número de filas y columnas (matriz cuadrada). En caso afirmativo, debe imprimir la matriz cuadrada. En caso negativo, debe imprimir un mensaje informativo.

```
import numpy as np

ejercicio7 = np.arange(0,16,1)

def arrayABidimensional(array: np.ndarray):
    raiz = np.sqrt(array.size)
    if raiz.is_integer():
        raiz = int(raiz)
        print(array.reshape(raiz, raiz))
    else:
        print("No se puede crear una matriz cuadrada desde ese array")
print("Matriz:")
print(ejercicio7)
print("Matriz transformada:")
arrayABidimensional(ejercicio7)
```

Resultado:

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

Ejercicio 8:

Escribe un programa que, dado un array bidimensional, halle el máximo de cada fila.

```
import numpy as np

ejercicio2 = np.arange(0, 20, 1).reshape(5, 4)

print("Matriz:")

print(ejercicio2)

print("Maximos:")

print(np.max(ejercicio2, axis=1))
```

Resultado:

```
Matriz:

[[ 0  1  2  3]

 [ 4  5  6  7]

 [ 8  9 10 11]

 [12 13 14 15]

 [16 17 18 19]]

Maximos:

[ 3  7 11 15 19]
```

Ejercicio 9:

Escribe un programa que, dado un array unidimensional, halle el número de veces que aparece cada uno de sus elementos únicos.

```
import numpy as np

e9 = np.random.randint(0, 10, size=20)

def contar(array: np.ndarray):
    numeros, valores = np.unique(array, return_counts=True)
    result = {}
    for i in range(len(numeros)):
        result[numeros[i]]=valores[i]
    return result
print("Matriz:")
print(e9)
print("Unicos:")
print(contar(e9))
```

Resultado:

```
Matriz:
[4 3 8 1 4 2 1 3 3 4 3 3 2 4 8 2 5 6 2 3]
Unicos:
{1: 2, 2: 4, 3: 6, 4: 4, 5: 1, 6: 1, 8: 2}
```

Ejercicio 10:

Normaliza un array bidimensional de 4 filas y 3 columnas restando la media y dividiendo por la desviación típica en cada columna.

```
import numpy as np

e10 = np.random.randint(0, 100, size=12).reshape(4,3)

print("Matriz:")

print(e10)

media = np.mean(e10, axis=0)

std = np.std(e10, axis=0)

fil, col  = e10.shape

for i in range(fil):

   for j in range(col):

      e10[i,j] = (e10[i,j]-media[j])/std[j]

print("Matriz normalizada:")

print(e10)
```

Resultado:

```
Matriz:

[[23  6 96]

 [34  8  4]

 [77 14 16]

 [37 19  3]]

Matriz normalizada:

[[ 0 -1  1]

 [ 0  0  0]

 [ 1  0  0]

 [ 0  1  0]]
```

Ejercicio 11:

Normaliza un array bidimensional de 4 filas y 3 columnas restando la media y dividiendo por la desviación típica en cada fila.

```
import numpy as np

e11 = np.random.randint(0, 100, size=12).reshape(4,3)

print("Matriz:")

print(e11)

media = np.mean(e11, axis=1)

std = np.std(e11, axis=1)

fil, col  = e11.shape

for i in range(fil):

   for j in range(col):

      e11[i,j] = (e11[i,j]-media[i])/std[i]

print("Matriz normalizada:")

print(e11)
```

Resultado:

```
Matriz:

[[59 52 84]

 [63 20 53]

 [ 0 76 24]

 [42 86 92]]

Matriz normalizada:

[[ 0  0  1]

 [ 0 -1  0]

 [-1  1  0]

 [-1  0  0]]
```

Ejercicio 12:

Escribe un programa que, dado un array bidimensional, halle los índices (filas y columnas) de los elementos mínimo y máximo del array.

```
import numpy as np

e12 = np.random.randint(1, 50000, size=2500).reshape(50,50)

print("Coordenadas maximo:")

print(np.unravel_index(np.argmax(e12), e12.shape))

print("Coordenadas minimo:")

print(np.unravel_index(np.argmin(e12), e12.shape))
```

Resultado:

```
Coordenadas maximo:

(32, 29)

Coordenadas minimo:

(15, 33)
```

Ejercicio 13:

Escribe un programa que ordene las filas de un array bidimensional según los valores de la primera columna.

```
import numpy as np

e12 = np.random.randint(1, 10, size=25).reshape(5,5)

print("Matriz:")

print(e12)

a = np.argsort(e12[:,0])

e12 = e12[a, :]

print("Matriz ordenada:")

print(e12)
```

Resultado:

```
Matriz:
[[9 3 4 6 1]
 [6 4 5 5 1]
 [1 1 8 7 1]
 [7 8 1 6 6]
 [2 4 4 8 7]]
Matriz ordenada:
[[1 1 8 7 1]
 [2 4 4 8 7]
 [6 4 5 5 1]
 [7 8 1 6 6]
 [9 3 4 6 1]]
```

Ejercicio 14:

Escribe un programa que genere una matriz bidimensional con 7 filas y 5 columnas aleatoriamente de acuerdo a la distribución normal, y a continuación ponga a cero todos los elementos negativos.

```
import numpy as np

e14 = np.random.normal(size=(7,5))

negativos = np.where(e14>=0, 1,0)

print("Matriz:")

print(e14)

print("Matriz sin negativos:")

print(np.abs(e14*negativos))
```

Resultado:

```
Matriz:
[[-0.99325127 -0.74515676 -0.89479471  0.46517403  0.64216198]
 [-1.2971965   1.03479678  0.5968923  -0.11927694  0.45933731]
 [ 0.34844704  1.16073831 -1.76587735  0.51356564  1.0666899 ]
 [-0.17731229  1.41823253 -0.39875018  1.52403641 -0.83585507]
 [-1.58467163  1.18340697  0.41834347 -0.14199437  0.69946849]
 [ 0.00574507  0.99952862 -0.3693723  -0.95560521  1.67231706]
 [-1.30660199 -0.06333281 -0.60811742 -0.18464734 -0.34839303]]
Matriz sin negativos:
[[0.         0.         0.         0.46517403 0.64216198]
 [0.         1.03479678 0.5968923  0.         0.45933731]
 [0.34844704 1.16073831 0.         0.51356564 1.0666899 ]
 [0.         1.41823253 0.         1.52403641 0.        ]
 [0.         1.18340697 0.41834347 0.         0.69946849]
 [0.00574507 0.99952862 0.         0.         1.67231706]
 [0.         0.         0.         0.         0.        ]]
```

Ejercicio 15:

Escribe un programa que, dado un array unidimensional y un número entero positivo k, halle los índices de los k mayores valores del array.

```
import numpy as np

e15 = np.random.randint(0, 50, size=100)

k = 7

mayores = []

for i in range(e15.shape[0]):

   if(e15[i]>k):

      mayores.append(i)


print("Matriz:")

print(e15)

print("Numeros mayores que "+str(k)+":")

print(e15[mayores])
```

Resultado:

```
Matriz:
[45 34  2 19 14 16  1 37 18  2 38 40 36 18 45 11 19  8 23  6 15 42 34 30
 29 45 24  5 23  2 37 10 30  8 36 46  6 26 38 44 18 45 39 31 44  8 39 10
 15 45  7 39 34 21  0 33 12 38 34 29  1 44 46 22 45 12 49 45 34 37 21 28
 17 17  3 24 13 34  8 46 10  6 30  6 32 38 31 36 28 11  9 37  2 16 42 37
 21  8 21 26]
Numeros mayores que 7:
[45 34 19 14 16 37 18 38 40 36 18 45 11 19  8 23 15 42 34 30 29 45 24 23
 37 10 30  8 36 46 26 38 44 18 45 39 31 44  8 39 10 15 45 39 34 21 33 12
 38 34 29 44 46 22 45 12 49 45 34 37 21 28 17 17 24 13 34  8 46 10 30 32
 38 31 36 28 11  9 37 16 42 37 21  8 21 26]
```

Ejercicio 16:

Escribe un programa que genere una matriz bidimensional con 6 filas y 7 columnas aleatoriamente de acuerdo a la distribución uniforme, y a continuación ponga a cero las dos primeras columnas y a uno las tres últimas columnas.

```
import numpy as np

e16 = np.random.uniform(size=(6,7))

print("Matriz:")

print(e16)

e16[:,0]=0

e16[:,1]=0

e16[:,-1]=0

e16[:,-2]=0

e16[:,-3]=0

print("Matriz con 0:")

print(e16)
```

Resultado:

```
Matriz:
[[0.83991885 0.26895518 0.66102868 0.79245001 0.71865919 0.93753159
  0.30222735]
 [0.59040761 0.74507763 0.86586371 0.95830016 0.58175665 0.83485619
  0.25991628]
 [0.56258912 0.45863763 0.14673885 0.11076467 0.04974238 0.89146687
  0.72116698]
 [0.78557923 0.24274514 0.88409494 0.75450405 0.14067182 0.15103905
  0.64475932]
 [0.76131701 0.97420561 0.51906141 0.62941552 0.97140125 0.3184503
  0.87517131]
 [0.93645108 0.04267828 0.2142031  0.7369168  0.43918128 0.09674237
  0.67248038]]
```

Matriz con 0:

```
[[0.         0.         0.66102868 0.79245001 0.         0.
  0.        ]
 [0.         0.         0.86586371 0.95830016 0.         0.
  0.        ]
 [0.         0.         0.14673885 0.11076467 0.         0.
  0.        ]
 [0.         0.         0.88409494 0.75450405 0.         0.
  0.        ]
 [0.         0.         0.51906141 0.62941552 0.         0.
  0.        ]
 [0.         0.         0.2142031  0.7369168  0.         0.
  0.        ]]
```