INSTITUTO SUPERIOR TECNOLÓGICO QUITO

Excelencia en Educación Superior

0.1 Tercer Modulo- Python , NumPy

python™

Erick Mejia

1. NumPy nadarray

- NumPy (Num erical Py thon) es la de facto librería estándar para análisis numérico en Python
- Estructura de datos multidimensional eficiente (escrita en C): ndarray
- Colección de funciones para álgebra lineal, estadística descriptiva
- Ayuda al procesamiento de datos

In [23]:
```
#importar libreria
import numpy as np
```

1.1 Performance Sample, NumPy vs Plain Python

In [24]:
```
my_list = list (range(1000000)) #Utilizando listas Python
%time my_list = [x * 2 for x in my_list]
```

```
CPU times: total: 203 ms
Wall time: 189 ms
```

In [25]:
```
my_arr = np.arange(1000000) #Utilizando NumPy Arrays
%time my_arr2 = my_arr * 2
```

```
CPU times: total: 15.6 ms
Wall time: 6.29 ms
```

1.2 Formas de crear ndarrays

In [26]:
```
array = np.array([ [2,71,0,34], [2,171,-35,34] ]) #2D
print(array)
```

```
[[  2  71   0  34]
 [  2 171 -35  34]]
```

In [27]:
```
shopping_list = [ ['onions','carrots', 'celery'], ['apples', 'orange,','grapes'] ]
array = np.array(shopping_list)      #convertir una lista en una ndarray
print(array)
print(type(array))
```

```
[['onions' 'carrots' 'celery']
 ['apples' 'orange,' 'grapes']]
<class 'numpy.ndarray'>
```

In [28]:
```
print(np.array([100,10,1])) # array a partir de una lista
print(np.arange(2,10,2.1)) # array a partir de una secuencia(range) 1D
```

```
[100  10   1]
[2.  4.1 6.2 8.3]
```

In [29]:
```
# Generar arrays con valores fijos
print(np.zeros(2))         #1D
print(np.ones((4,3)))      #matriz 2x2
print(np.empty((2,2,2)))   #matriz 2x2x2, no inicializados
print(np.full((5,2),2))    #matriz 5x2
print(np.eye(3))           #matriz identity
```

```
[0. 0.]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[[[9.88e-324 3.51e-322]
  [0.00e+000 1.68e-322]]

 [[9.88e-324 8.45e-322]
  [      nan 1.68e-322]]]
[[2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

1.3 Obtener información sobre ndarray

In [30]:
```
my_ndarray = np.array([2,71,0,34])
```

```
print(my_ndarray)
```

```
[ 2 71  0 34]
```

In [31]:
```
print(my_ndarray.argmax())         #indice del valor mas alto
print(np.amax(my_ndarray))

print(my_ndarray.argmin())          #indice del valor mas bajo
print(np.amin(my_ndarray))

print(my_ndarray.nonzero())          #retorna los indices de los elementos distinto
```

```
1
71
2
0
(array([0, 1, 3]),)
```

In [32]:
```
my_ndarray = np.full((5,4,2), 2)    #matriz 5x2x2
print(my_ndarray)

print(my_ndarray.size)              #numero de elementos
print(my_ndarray.shape)             #dimensiones
print(my_ndarray.ndim)              #numero de dimensiones
```

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

 [[2 2]
  [2 2]
  [2 2]
  [2 2]]

 [[2 2]
  [2 2]
  [2 2]
  [2 2]]

 [[2 2]
  [2 2]
  [2 2]
  [2 2]]

 [[2 2]
  [2 2]
  [2 2]
  [2 2]]]
40
(5, 4, 2)
3
```

In [33]:
```
my_ndarray = np.array([2,71,0,34])

print(my_ndarray.size)              #numero de elementos
```

```
print(my_ndarray.shape)              #dimensiones
print(my_ndarray.ndim)               #numero de dimensiones
```

```
4
(4,)
1
```

Redimensionar los datos

In [34]:
```
my_ndarray = np.array([[0, 71, 21, 19, 213, 412, 111, 98]])    #matriz 1x8
print(my_ndarray)
print(my_ndarray.shape)
print(my_ndarray.ndim)
```

```
[[  0  71  21  19 213 412 111  98]]
(1, 8)
2
```

In [35]:
```
new_dims = my_ndarray.reshape(2, 4)          #cambiar las dimensiones a 2x4
print(new_dims)
print(new_dims.ndim)
```

```
[[  0  71  21  19]
 [213 412 111  98]]
2
```

In [36]:
```
new_dims = my_ndarray.reshape(4, 2)          #cambiar las dimensiones a 4x2
print(new_dims)
print(new_dims.ndim)
```

```
[[  0  71]
 [ 21  19]
 [213 412]
 [111  98]]
2
```

In [37]:
```
x = np.array([100, 10, 1]).reshape(3,1)
print(x.shape)
print(x)
```

```
(3, 1)
[[100]
 [ 10]
 [  1]]
```

In [38]:
```
#reducir a 1D
new_dims = np.arange(9).reshape(3,3)
print(new_dims)
print(new_dims.flatten())
```

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

In [39]:
```
#reducir
new_dims = np.arange(16).reshape(4,4)
```

```
print(new_dims)
print(new_dims.reshape(2, -1))         #el segundo valor se infiere del primero
```

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

1.4 Manipulación de ndarrays

In [40]:
```
# Cambiar ejes
new_dims = np.arange(16).reshape(8,2)
print(new_dims)
print(new_dims.swapaxes(0,1))
```

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

In [41]:
```
# Flip
new_dims = np.arange(16).reshape(4,4)
print(new_dims)
print(np.flip(new_dims, axis = None))
```

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

In [42]:
```
# ordenar
array1 = np.array([10,2,9,17])
array1.sort()
print(array1)
```

```
[ 2  9 10 17]
```

In [43]:
```
# Juntar dos arrays
a1 = [0,0,0,0,0,0]
a2 = [1,1,1,1,1,1]

print(np.hstack((a1,a2))) # una al lado de la otra
print(np.hstack((a1,a2)).shape)
```

```
print(np.vstack((a1,a2))) # una encima de la otra
print(np.vstack((a1,a2)).shape)
```

```
[0 0 0 0 0 0 1 1 1 1 1 1]
(12,)
[[0 0 0 0 0 0]
 [1 1 1 1 1 1]]
(2, 6)
```

In [44]:
```
array1 = np.array([[2, 4], [6, 8]])
array2 = np.array([[3, 5], [7, 9]])

print(np.concatenate((array1, array2), axis = 0))
```

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

In [45]:
```
# dividir un array
array = np.arange(16).reshape(4,4)
print(array)
print(np.array_split(array, 2, axis = 0)) # divide array en 2 partes iguales
print(np.array_split(array, 3)) # divide array en 3 partes "iguales"
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
[array([[0, 1, 2, 3],
       [4, 5, 6, 7]]), array([[ 8,  9, 10, 11],
       [12, 13, 14, 15]])]
[array([[0, 1, 2, 3],
       [4, 5, 6, 7]]), array([[ 8,  9, 10, 11]]), array([[12, 13, 14, 15]])]
```

In [46]:
```
# dividir un array
array = np.arange(16).reshape(4,4)
print(array)
print(np.array_split(array, [3], axis = 1)) # divide array, por la fila 3
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
[array([[ 0,  1,  2],
       [ 4,  5,  6],
       [ 8,  9, 10],
       [12, 13, 14]]), array([[ 3],
       [ 7],
       [11],
       [15]])]
```

In [47]:
```
split = np.array_split(array, [3], axis = 1)
print(split[0][3][0])
```

```
12
```

In [48]:
```
array = np.arange(64).reshape(8,8)
print(array)
print(np.array_split(array, [1, 3], axis = 0)) # divide array por la fila 1 y 3
print(np.array_split(array, [1, 3], axis = 1)) # divide array por la columna 1␣↪y 3
# axis = n_dimensions - 1
```

```
[[ 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 25 26 27 28 29 30 31]
 [32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47]
 [48 49 50 51 52 53 54 55]
 [56 57 58 59 60 61 62 63]]
[array([[0, 1, 2, 3, 4, 5, 6, 7]]), array([[ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23]]), array([[24, 25, 26, 27, 28, 29, 30, 31],
       [32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47],
       [48, 49, 50, 51, 52, 53, 54, 55],
       [56, 57, 58, 59, 60, 61, 62, 63]])]
[array([[ 0],
       [ 8],
       [16],
       [24],
       [32],
       [40],
       [48],
       [56]]), array([[ 1,  2],
       [ 9, 10],
       [17, 18],
       [25, 26],
       [33, 34],
       [41, 42],
       [49, 50],
       [57, 58]]), array([[ 3,  4,  5,  6,  7],
       [11, 12, 13, 14, 15],
       [19, 20, 21, 22, 23],
       [27, 28, 29, 30, 31],
       [35, 36, 37, 38, 39],
       [43, 44, 45, 46, 47],
       [51, 52, 53, 54, 55],
       [59, 60, 61, 62, 63]])]
```

2 Índices y slicing

- De manera análoga a índices y slicing para listas

In [49]:
```
array = np.arange(10, 16)
print(array)
print(array[-1])       # acceso unidimensional
print(array[:-1])    # slice
```

```
[10 11 12 13 14 15]
15
[10 11 12 13 14]
```

In [50]:
```python
# bidimensional
array = np.arange(16).reshape(4,4)
print(array)
print(array[2, 2]) # fila, columna
print(array[3][2]) # fila, columna
```

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

In [51]:
```python
# multidimensional
array = np.arange(36).reshape(2,3,6)
print(array)
print(array[0,2,4])
print(array[1][2][4])
```

```
[[[ 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 25 26 27 28 29]
  [30 31 32 33 34 35]]]
16
34
```

NumPy es row major

In [52]:
```python
row_major = np.array([1, 2, 3, 4, 5, 6])
row_major = row_major.reshape(2,3) # by default, row major
print(row_major)
```

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

In [53]:
```python
row_major = np.array([1, 2, 3, 4, 5, 6])
row_major = row_major.reshape(2,3,order='C') # by default, row major
print(row_major)

col_major = np.array([1, 2, 3, 4, 5, 6])
col_major = col_major.reshape(2,3,order='F') # col major
print(col_major)
```

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

Slice

In [54]:
```
# se puede hacer slice por fila o columna
array = np.arange(81).reshape(9,9)
print(array)
print(array[1:3, 2::2])
```

```
[[ 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 25 26]
 [27 28 29 30 31 32 33 34 35]
 [36 37 38 39 40 41 42 43 44]
 [45 46 47 48 49 50 51 52 53]
 [54 55 56 57 58 59 60 61 62]
 [63 64 65 66 67 68 69 70 71]
 [72 73 74 75 76 77 78 79 80]]
[[11 13 15 17]
 [20 22 24 26]]
```

In [55]:
```
# se puede hacer slice por fila o columna
array = np.arange(9).reshape(3,3)
print(array)
print(array[0,:])
```

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

Diferencias entre índices y slicing en listas y ndarrays:

- ¡Retornan una referencia, no una copia!

In [56]:
```
# uso de slice en listas, retorna copia
lista = [0,1,2,3,4,5,6,7,8,9]
my_copy = lista[0:2] # devuelve una copia
print(my_copy)
my_copy[0] = 222
print(lista)
print(my_copy)
```

```
[0, 1]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[222, 1]
```

In [57]:
```
# slicing retorna una referencia en ndarrays
array = np.arange(10)
my_copy = array[2:5] # devuelve una referencia
print(array)
print(my_copy)
my_copy[0] = 222
print(array)
print(my_copy)
```

```
[0 1 2 3 4 5 6 7 8 9]
[2 3 4]
[  0   1 222   3   4   5   6   7   8   9]
[222   3   4]
```

In [58]:
```
# copiar ndarrays
array = np.arange(10)
my_copy = array[2:5].copy() # hay que hacer la copia explicitamente
my_copy[0] = 222
print(array)
print(my_copy)
```

```
[0 1 2 3 4 5 6 7 8 9]
[222   3   4]
```

- Permite asignar valores a un corte

In [59]:
```
array = np.arange(10)
print(array)

print(array[0:3])

array[0:3] = -1
print(array)
```

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

- Slicing condicionales (mask)

In [60]:
```
# slicing condicional
array = np.arange(-3,4)
print(array)

mask = array %2 == 0
print(mask)
print(array[mask])

print(array[array %2 == 0])

array[mask] = 0
print(array)
```

```
[-3 -2 -1  0  1  2  3]
[False  True False  True False  True False]
[-2  0  2]
[-2  0  2]
[-3  0 -1  0  1  0  3]
```

In [61]:
```
array = np.arange(-3,4)
print(array)

array[array < 0] = 0
print(array)
```

```
[-3 -2 -1  0  1  2  3]
[0 0 0 0 1 2 3]
```

In [62]:
```
# slicing condicional
array = np.arange(-3,4)
print(array)

mask = array % 2 == 0
print(mask)

array_par = array[mask] # devuelve una copia
print(array_par)

array_par[0] = -10
print(array)
print(array_par)
```

```
[-3 -2 -1  0  1  2  3]
[False  True False  True False  True False]
[-2  0  2]
[-3 -2 -1  0  1  2  3]
[-10   0   2]
```

- Acceso a múltiples valores con listados de índices

In [63]:
```
array = np.arange(0,16).reshape(4,4)
print(array)
print(array[[1,2]]) #selecting rows
print(array[:,[1,2]]) #selecting columns

# cross product of indexes
print(array[[1,2],:][:,[1,2]])

print(array[np.ix_([1,2],[1,2])]) # built in ix_

ar = array[[2,3,1]] # slice, copia
print(ar)
ar[0] = 999
print(ar)
print(array)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
[[ 4  5  6  7]
 [ 8  9 10 11]]
[[ 1  2]
 [ 5  6]
 [ 9 10]
 [13 14]]
[[ 5  6]
 [ 9 10]]
[[ 5  6]
 [ 9 10]]
[[ 8  9 10 11]
 [12 13 14 15]
 [ 4  5  6  7]]
[[999 999 999 999]
 [ 12  13  14  15]
 [  4   5   6   7]]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

3 Funciones matemáticas

- Aplicar operaciones entre escalares y arrays o entre arrays

In [64]:
```
# listas y escalares
lista = [0,1,2,3]
lista *= 3
print(lista)

lista += [1]
print(lista)
```

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

In [65]:
```
# ndarray hace las operaciones sobre los elementos
array = np.ones(4, dtype = np.int8)
print(array)
array *= 2
print(array)

array += 10
print(array)

print(array.dtype) # check data type of array
```

```
[1 1 1 1]
[2 2 2 2]
[12 12 12 12]
int8
```

In [66]:
```python
# operaciones entre arrays es como operaciones entre matrices
array_1 = np.ones(4) * 2
array_2 = np.arange(4) + 1

print(array_1)
print(array_2)

print(array_1 + array_2)
print(array_1 - array_2)
print(array_1 * array_2)
print(array_2 / array_1)
```

```
[2. 2. 2. 2.]
[1 2 3 4]
[3. 4. 5. 6.]
[ 1.  0. -1. -2.]
[2. 4. 6. 8.]
[0.5 1.  1.5 2. ]
```

3.1 Broadcasting

Si las arrays no tienen las mismas dimensiones, se hace broadcast del pequeño al grande -> broadcasting

3.2 Operadores unarios y binarios

Tipo	Operación	Descripción
Unario	*abs*	Valor absoluto de cada elemento
	sqrt	Raíz cuadrada de cada elemento
	exp	e^x, siendo x cad elemento
	log, log10, log2	Logaritmos en distintas bases de cada elemento
	sign	Retorna el signo de cada elemento (-1 para negativo, 0 o 1 para positivo)
	ceil	Redondea cada elemento por arriba
	floor	Redondea cada elemento por abajo
	isnan	Retorna si cada elemento es Nan
	cos, sin, tan	Operaciones trigonométricas en cada elemento

Tipo	Operación	Descripción
	arccos, arcsin, arctan	Inversas de operaciones trigonométricas en cada elemento
Binario	*add*	Suma de dos arrays
	substract	Resta de dos arrays
	multiply	Multiplicación de dos arrays
	divide	División de dos arrays
	maximum, minimum	Retorna el valor máximo/mínimo de cada pareja de elementos
	equal, not_equal	Retorna la comparación (igual o no igual) de cada pareja de elementos
	greater, greater_equal, less, less_equal	Retorna la comparación (>, >=, <, <= respectivamente) de cada pareja de elementos

In [67]:
```
# ejemplos de operadores
array = np.arange(5)
print(array)
np.sqrt(array)
```

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

Out[67]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ])
```

In [68]:
```
# ejemplos de operadores
array1 = np.arange(4)
array2 = np.array([0,-1,2,-3])
```

```
print(array1)
print(array2)

print(np.greater(array1,array2))
print(np.less(array2, array1))
```

```
[0 1 2 3]
[ 0 -1  2 -3]
[False  True False  True]
[False  True False  True]
```

4 Estadística descriptiva

- Importante saber la naturaleza de los datos
- Valores máximos, mínimos, distribución, etc.

Función	Descripción
sum(arr)	Suma de todos los elementos de *arr*
mean(arr)	Media aritmética de los elementos de *arr*

Función	Descripción
std(arr)	Desviación estándar de los elementos de *arr*
cumsum(arr)	Devuelve array con la suma acumulada de cada elementos con todos los anteriores
cumprod(arr)	Devuelve array con el producto acumulado de cada elementos con todos los anteriores
min(arr), max(arr)	Mínimo y máximo de *arr*
any(arr)	En array de tipo *bool*, retorna *True* si algún elemento es *True*
all(arr)	En array de tipo *bool*, retorna *True* si todos los elementos son *True* (o <>0 en valores numéricos)
unique(arr)	Devuelve un array con valores únicos
in1d(arr1, arr2)	Devuelve un array con bool indicando si cada elemento de *arr1* está en *arr2*
union1d(arr1, arr2)	Devuelve la unión de ambos arrays
intersect1d(arr1, arr2)	Devuelve la intersección de ambos arrays

In [69]:
```
# sum != cumsum
array1 = np.arange(9)
print(array1)
print(array1.sum())
print(array1.cumsum())
```

```
[0 1 2 3 4 5 6 7 8]
36
[ 0  1  3  6 10 15 21 28 36]
```

In [70]:
```
# any vs all
array1 = np.arange(-8,2).reshape(2,5)
print(array1)
print(array1.any()) # any para saber si hay elementos True o valores <> 0
print(array1.all()) # all para saber si TODOS los elementos son True o valores␣↪<>
```

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

In [71]:
```
# todas las funciones aceptan parametro 'axis'
# 0 para columnas, 1 para filas, 2 para profundidad...
array1 = np.arange(10).reshape(2,5)
print(array1)
print(array1.all(axis=0))
print(array1.all(axis=1))
print(array1.all())
```

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

5 Álgebra lineal

- numpy.linalg contiene funciones para álgebra lineal
- dot product, multiplicación de matrices, cálculo del determinante, factorizaciones...

Función	Descripción
dot(mat1, mat2)	Devuelve el producto escalar entre dos arrays. Si son matrices 2D, es equivalente a la multiplicación de ambas
matmul(mat1, mat2)	Devuelve el producto entre dos matrices
trace(mat1, mat2)	Suma de las diagonales de ambas matrices
det(mat)	Devuelve el determinante de la matriz
eig(mat)	Computa los autovalores y autovectores de la matriz cuadrada *mat*
inv(mat)	Devuelve la inversa de la matriz
qr(mat)	Computa la factorización QR de *mat*
solve(A, b)	Resuelve el sistema lineal de ecuaciones $Ax = b$ para b, cuando A es una matriz cuadrada
transpose(mat)	Devuelve la transpuesta de la matrix

In [72]:
```
A = np.arange(12).reshape(4,3)
B = np.arange(6).reshape(3,2)

print(A)
print(B)

print(np.matmul(A, B))
print(np.dot(A, B))
print(A @ B)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
[[0 1]
 [2 3]
 [4 5]]
[[10 13]
 [28 40]
 [46 67]
 [64 94]]
[[10 13]
 [28 40]
 [46 67]
 [64 94]]
[[10 13]
 [28 40]
 [46 67]
 [64 94]]
```

In [73]:
```
import numpy.linalg as lg
import math
```

```
# resolver sistema de ecuaciones
# 3x + 2y + 5z = 12
# x - 3y -z = 2
# 2x -y +12z = 32
# Ax = b --> resolver para x

a = np.array([ [3,2,5], [1,-3,-1], [2,-1,12] ])
b = np.array([12,2,32])

print(a)
print(b)
x,y,z = lg.solve(a,b)
print(f'x={x} y={y} z={z}')

print(math.isclose(3*x + 2*y +5*z, 12))
print(math.isclose(x - 3*y - z, 2))
print(math.isclose(2*x -y + 12*z, 32))
```

```
[[ 3  2  5]
 [ 1 -3 -1]
 [ 2 -1 12]]
[12  2 32]
x=0.7543859649122807 y=-1.2280701754385965 z=2.43859649122807
True
True
True
```

In [74]:
```
#matriz identidad
I = np.identity(4)
print(I)

A = np.arange(16).reshape(4,4)
print(A)

print(np.matmul(A,I))
```

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

6 Filtrado de datos

- Filtrar y modificar datos numéricos con np.where
- Retorna A o B en función de una condición en una array

In [75]:
```
prices = np.array([0.99, 14.49, 19.99, 20.99, 0.49])
mask = prices < 1
print(prices[prices < 1]) #slicing condicional

# mask con los elementos menores de 1
mask = np.where(prices < 1, True, False)
print(mask)
print(prices[mask])

prices[mask] = 1.0
print(prices)

print(np.where(prices < 1, 1.0, prices))
```

```
[0.99 0.49]
[ True False False False  True]
[0.99 0.49]
[ 1.   14.49 19.99 20.99  1.  ]
[ 1.   14.49 19.99 20.99  1.  ]
```

In [76]:
```
# Se puede usar para sustituir datos fuera de rango
bank_transfers_values = np.array([0.99, -1.49, 19.99, 20.99, -0.49, 12.1]).reshape(
print(bank_transfers_values)
clean_data = np.where(bank_transfers_values > 0, bank_transfers_values, 0)
print(clean_data)
```

```
[[ 0.99 -1.49 19.99]
 [20.99 -0.49 12.1 ]]
[[ 0.99  0.   19.99]
 [20.99  0.   12.1 ]]
```

In [77]:
```
# limpiar NaN (sustitución)
data = [10, 12, -143, np.nan, 1, -3] # np.nan --> NotANumber, elemento especial
data_clean = np.where(np.isnan(data), 0, data)
print(data_clean)

print(id(data))
print(id(data_clean))
```

```
[  10.   12. -143.    0.    1.   -3.]
2875610367552
2875636221392
```

In [78]:
```
# np.where puede seleccionar elementos
bank_transfers_values = np.array([0.99, -1.49, 19.99, 20.99, -0.49, 12.1]).reshape(
print(bank_transfers_values)
credits = np.where(bank_transfers_values > 0) # array de índices para los que la co

print(credits)
print(bank_transfers_values[credits]) # es igual que bank_transfers_values[bank_tra
print(bank_transfers_values[bank_transfers_values > 0])
```

```
[[ 0.99 -1.49 19.99]
 [20.99 -0.49 12.1 ]]
(array([0, 0, 1, 1]), array([0, 2, 0, 2]))
[ 0.99 19.99 20.99 12.1 ]
[ 0.99 19.99 20.99 12.1 ]
```

In [79]:
```
# valor de array dependiendo del valor en array referencia
rewards_default = np.arange(6)
print(rewards_default)

rewards_upgrade = np.arange(6) * 10
print(rewards_upgrade)

daily_points = np.array([0, 0, 1, 6, 0, 2])
print(daily_points)

final_rewards = np.where(daily_points > 1, rewards_upgrade, rewards_default)
print(final_rewards)
```

```
[0 1 2 3 4 5]
[ 0 10 20 30 40 50]
[0 0 1 6 0 2]
[ 0  1  2 30  4 50]
```

7 Números aleatorios

- Python módulo random

In [80]:
```
import random

print(random.random()) # número entre 0 y 1
print(random.randint(0,5)) # integral entre dos valores
print(random.uniform(0,5)) # real entre dos valores
```

```
0.62286439410579
1
0.5504765085663343
```

In [81]:
```
# elegir un elemento al azar dentro de una colección
names = ['Pikachu','Eevee','Charmander']
random.choice(names)
```

Out[81]:
```
'Eevee'
```

7.1 NumPy.random

- Generar fácilmente listas con valores aleatorios

In [82]:
```
print(np.random.rand()) # número entre 0 y 1
print(np.random.randint(0,5)) # integral entre dos valores
print(np.random.randint(5, size=(2, 4))) # parametro 'size' para indicar las dimens
```

```
0.23370279934604943
1
[[4 1 3 0]
 [1 1 3 0]]
```

In [83]:
```
# array de elementos aleatorios (distribucion normal gaussiana, media 0, desviación
print(np.random.randn(4,2))
```

```
[[-0.37782477 -1.97323686]
 [ 0.91758353 -1.34850194]
 [ 0.88227466 -1.29521228]
 [ 1.55899706  0.3214769 ]]
```

In [84]:
```python
# otras distribuciones
print(np.random.binomial(n=5, p=0.3)) # 'n' intentos, 'p' probabilidad
print(np.random.uniform(low=0, high=10)) # distribución uniforme
print(np.random.poisson(lam=2, size=(2,2))) # 'lam' número de ocurrencias esperadas
```

```
1
5.0381145506744005
[[1 1]
 [2 3]]
```

7.2 Random seed

- Números pseudo-aleatorios
- Ordenadores generan números a partir de ecuaciones
- Basadas en un número inicial (seed)
- Bueno para reproducibilidad de experimentos

In [85]:
```python
i = 5
for _ in range(5):
    np.random.seed(i) # misma semilla
    print(np.random.rand())
    print(np.random.rand())

print()

for i in range(6):
    np.random.seed(i) # diferentes semillas
    print(np.random.rand())
    print(np.random.rand())
```

```
0.22199317108973948
0.8707323061773764
0.22199317108973948
0.8707323061773764
0.22199317108973948
0.8707323061773764
0.22199317108973948
0.8707323061773764
0.22199317108973948
0.8707323061773764

0.5488135039273248
0.7151893663724195
0.417022004702574
0.7203244934421581
0.43599490214200376
0.025926231827891333
0.5507979025745755
0.7081478226181048
0.9670298390136767
0.5472322491757223
0.22199317108973948
0.8707323061773764
```

8 Escritura y lectura de ndarrays a archivos

- Para futuro acceso
- Compartir datos con otros
- Formato *.npy

In [86]:
```python
import os
import numpy as np

# Aseguramos que la carpeta "res" exista antes de guardar
# Si no existe, la crea. Si ya existe, no hace nada.
os.makedirs("res", exist_ok=True)

ruta = os.path.join("res", "o_values_array.npy")

values = np.random.randint(9, size=(3,3))
np.save(ruta, values)

print("Archivo guardado en:", ruta)
print(values)
```

```
Archivo guardado en: res\o_values_array.npy
[[6 6 0]
 [8 4 7]
 [0 0 7]]
```

In [87]:
```python
# Leer de archivo
old_values = np.load(ruta)
print(old_values)
```

```
[[6 6 0]
 [8 4 7]
 [0 0 7]]
```

In [88]:
```python
import os
import numpy as np

ruta = os.path.join("res" ,"o_array_group.npz")

values1 = np.random.randint(9, size=(3,3))
values2 = np.random.randint(4, size=(2,2))
values3 = np.random.randint(16, size=(4,4))

# Al guardar con savez, si no les das nombre, numpy los llama arr_0, arr_1, etc.
np.savez(ruta, values1, values2, values3)

print("¡Listo! Archivo .npz creado en:", ruta)
```

```
¡Listo! Archivo .npz creado en: res\o_array_group.npz
```

In [89]:
```python
# Cargar el archivo
npzfile = np.load(ruta)

print("Archivos dentro del .npz:", npzfile.files) # Debería salir ['arr_0', 'arr_1'

print("\n--- Imprimiendo arrays ---")
for key in npzfile.files:
    # OJO AQUÍ: Este print tiene que tener un TAB a la izquierda
    print(f"Array {key}:")
    print(npzfile[key])
    print("-" * 20)
```

```
Archivos dentro del .npz: ['arr_0', 'arr_1', 'arr_2']

--- Imprimiendo arrays ---
Array arr_0:
[[1 5 7]
 [0 1 4]
 [6 2 1]]
--------------------
Array arr_1:
[[2 3]
 [0 2]]
--------------------
Array arr_2:
[[ 5 10  0  0]
 [ 4 14  4  9]
 [ 3 11 15  2]
 [ 4  6  9  3]]
--------------------
```

8.0.1 ¿Por qué molestarse con .npy?

- Bases de datos suelen estar en CSV o txt, que requieren lectura con streams (parsing)
- ¿Para qué escribir los datos de nuevo?

In [90]:
```python
import numpy as np
import os

# Creamos la carpeta si no existe
os.makedirs("res", exist_ok=True)
ruta_csv = os.path.join("res", "fdata.csv")

# Generamos 1 millón de datos (1000x1000) aleatorios entre 0 y 100
# Usamos int8 para que coincida con lo que pide el profe
datos = np.random.randint(0, 100, 1000*1000)

# Los guardamos como texto separados por comas (formato CSV manual)
cadena_datos = ",".join(map(str, datos))

with open(ruta_csv, "w") as f:
    f.write(cadena_datos)

print("¡Archivo 'fdata.csv' creado! Ahora pasa a la siguiente celda.")
```

```
¡Archivo 'fdata.csv' creado! Ahora pasa a la siguiente celda.
```

In [91]:
```python
# Esta celda NO lleva %%timeit
ruta_csv = os.path.join("res" ,"fdata.csv")

# 1. Leer CSV (lento)
with open(ruta_csv,'r') as f:
    data_str = f.read()

data = data_str.split(',')
data_array = np.array(data, dtype = np.int8).reshape(1000,1000)

# 2. Guardar como NPY (rápido)
ruta_npy = os.path.join("res" ,"o_fdata.npy")
np.save(ruta_npy, data_array)

print("Conversión terminada. Archivos listos para la carrera de velocidad.")
```

```
Conversión terminada. Archivos listos para la carrera de velocidad.
```

In [92]:
```python
%%timeit
# Todo lo que está en esta celda se ejecutará varias veces para medir el tiempo
with open("res/fdata.csv",'r') as f:
    data_str = f.read()
data = data_str.split(',')
data_array = np.array(data, dtype = np.int8).reshape(1000,1000)
```

```
436 ms ± 28.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [93]:
```python
%%timeit
# Cargar directamente el binario
ruta_npy = os.path.join("res" ,"o_fdata.npy")
data = np.load(ruta_npy)
```

```
1.07 ms ± 63 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

8.0.2 velocidad + simplicidad vs memoria

8.1 Ejercicios

- Ejercicios para practicar NumPy: https://github.com/rougier/numpy100/blob/master/100_Numpy_exercises.ipynb

In [104…

```
import numpy as np

# Ver la versión de numpy
print("Versión de NumPy:", np.__version__)

# Ver la configuración
print("\nConfiguración de NumPy:")
np.show_config()
```

```
Versión de NumPy: 2.4.0

Configuración de NumPy:
Build Dependencies:
  blas:
    detection method: pkgconfig
    found: true
    include directory: C:/Users/runneradmin/AppData/Local/Temp/cibw-run-5tdgda_u/cp3
11-win_amd64/build/venv/Lib/site-packages/scipy_openblas64/include
    lib directory: C:/Users/runneradmin/AppData/Local/Temp/cibw-run-5tdgda_u/cp311-w
in_amd64/build/venv/Lib/site-packages/scipy_openblas64/lib
    name: scipy-openblas
    openblas configuration: OpenBLAS 0.3.30  USE64BITINT DYNAMIC_ARCH NO_AFFINITY
      Haswell MAX_THREADS=24
    pc file directory: D:/a/numpy-release/numpy-release/.openblas
    version: 0.3.30
  lapack:
    detection method: pkgconfig
    found: true
    include directory: C:/Users/runneradmin/AppData/Local/Temp/cibw-run-5tdgda_u/cp3
11-win_amd64/build/venv/Lib/site-packages/scipy_openblas64/include
    lib directory: C:/Users/runneradmin/AppData/Local/Temp/cibw-run-5tdgda_u/cp311-w
in_amd64/build/venv/Lib/site-packages/scipy_openblas64/lib
    name: scipy-openblas
    openblas configuration: OpenBLAS 0.3.30  USE64BITINT DYNAMIC_ARCH NO_AFFINITY
      Haswell MAX_THREADS=24
    pc file directory: D:/a/numpy-release/numpy-release/.openblas
    version: 0.3.30
Compilers:
  c:
    commands: cl
    linker: link
    name: msvc
    version: 19.44.35222
  c++:
    commands: cl
    linker: link
    name: msvc
    version: 19.44.35222
  cython:
    commands: cython
    linker: cython
    name: cython
    version: 3.2.3
Machine Information:
  build:
    cpu: x86_64
    endian: little
    family: x86_64
    system: windows
  host:
    cpu: x86_64
    endian: little
    family: x86_64
    system: windows
Python Information:
```

```
  path: C:\Users\runneradmin\AppData\Local\Temp\build-env-20ylnh9g\Scripts\python.ex
e
  version: '3.11'
SIMD Extensions:
  baseline:
  - X86_V2
  found:
  - X86_V3
```

In [95]:
```python
import numpy as np

# Crear un array
mi_array = np.array([1, 2, 3, 4, 5])

# Ver cuánta memoria usa
print(mi_array.nbytes)
```

```
40
```

In [96]:
```python
# Importar numpy
import numpy as np

# Crear un vector con 10 ceros
vector = np.zeros(10)

# Poner un 1 en la quinta posición
vector[4] = 1

# Mostrar resultado
print(vector)
```

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

In [97]:
```python
# Importar numpy
import numpy as np

# Crear un vector con números
vector = np.array([1, 2, 3, 4, 5])

# Darle la vuelta al vector (invertirlo)
vector_invertido = vector[::-1]

# Mostrar el resultado
print(vector_invertido)
```

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

In [98]:
```python
# Traer numpy para trabajar
import numpy as np

# Lista de números donde queremos buscar
numeros = np.array([1, 2, 0, 0, 4, 0])

# Buscar en qué posiciones están los números que NO son cero
posiciones = np.nonzero(numeros)
```

```
# Imprimir las posiciones encontradas
print(posiciones)
```

```
(array([0, 1, 4]),)
```

In [99]:
```
# Traer numpy para trabajar
import numpy as np

# Crear un cubo de 3x3x3 con números enteros al azar (del 0 al 9)
cubo = np.random.randint(0, 10, (3, 3, 3))

# Mostrar el cubo
print(cubo)
```

```
[[[2 1 7]
  [4 1 3]
  [8 4 7]]

 [[3 3 9]
  [9 0 5]
  [0 6 3]]

 [[6 5 4]
  [9 5 4]
  [3 5 3]]]
```

In [100…
```
# Traer numpy para trabajar
import numpy as np

# Crear un vector con 30 números enteros al azar (del 1 al 100)
vector = np.random.randint(1, 101, 30)

# Calcular el promedio
promedio = vector.mean()

# Mostrar el vector y el promedio
print("Vector:", vector)
print("El promedio es:", promedio)
```

```
Vector: [ 60  37  49  15  16  92  12  45  62  69  28  85  26  61  34  27  33  33
  55  91  28  10  54  54  85  71  51  91  74 100]
El promedio es: 51.6
```

In [101…
```
# Traer numpy
import numpy as np

# Crear una tabla de números
tabla = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])

# Ponerle un marco de ceros alrededor
tabla_con_marco = np.pad(tabla, pad_width=1, constant_values=0)

# Mostrar
print(tabla_con_marco)
```

```
[[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 [102…

```python
# Traer numpy
import numpy as np

# Crear una tabla de 5x5 con números debajo de la línea del medio
tabla = np.diag([1, 2, 3, 4], k=-1)

# Mostrar
print(tabla)
```

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

In [103…

```python
# Traer numpy
import numpy as np

# Encontrar la posición del elemento número 100 en una tabla de tamaño (6,7,8)
posicion = np.unravel_index(99, (6, 7, 8))

# Mostrar la posición
print("El elemento 100 está en la posición:", posicion)
```

```
El elemento 100 está en la posición: (np.int64(1), np.int64(5), np.int64(3))
```

9 Github

https://github.com/Erick-305/machine-learning.git