UNIVERSIDAD NACIONAL DE ASUNCIÓN
FACULTAD DE INGENIERÍA

Cursos Básicos
Primer Ciclo 2024

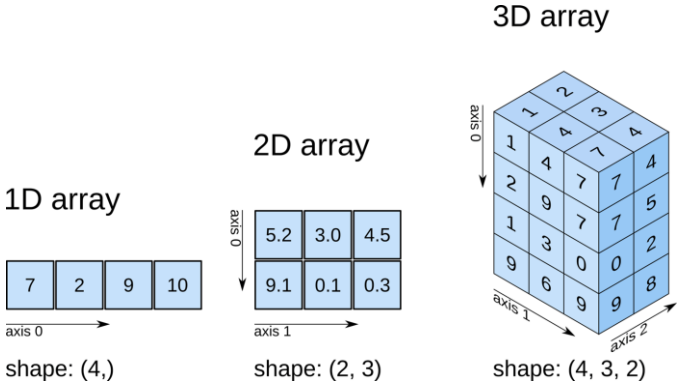
Python- Numpy

NumPy

The fundamental package for scientific computing with Python

LATEST RELEASE: NUMPY 1.26. VIEW ALL RELEASES

NumPy 1.26.0 released 2023-09-16



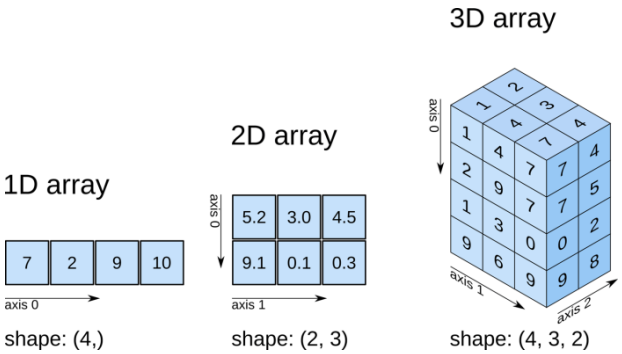
Getting started
- What is NumPy?
- Installation
- NumPy quickstart
- NumPy: the absolute basics for beginners

Fundamentals and usage
- NumPy fundamentals
  - Array creation
  - Indexing on **ndarrays**
  - I/O with NumPy
  - Data types
  - Broadcasting
  - Copies and views
  - Structured arrays
  - Universal functions (**ufunc**) basics

https://numpy.org/

¿Qué veremos hoy?

- Vectores y matrices
  - Numpy: Indexación, slicing, operaciones
  - Generación de datos: zeros(),ones(),empty, arrage()
  - Manipulación: reshape, resize,flatend
  - Concatenación y división
  - Ejercicios
- Manejo Básico de archivos
  - Lectura, Escritura, CSV, txt
  - Funciones de Numpy
- Ejercicios



*Agradecimientos Jose Colbes por los materiales de referencia

Qué es Numpy

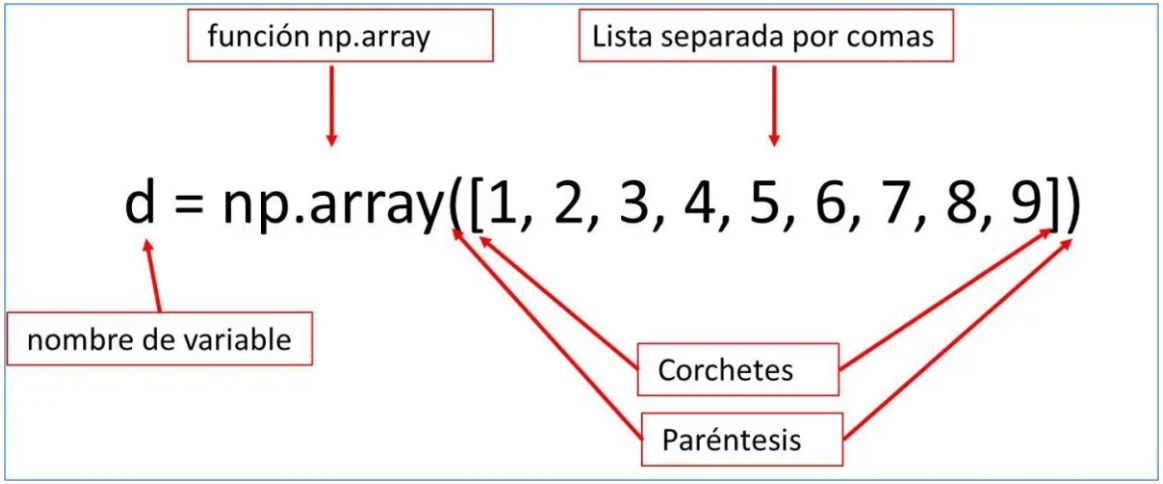
NumPy (abreviatura de Numerical Python), es uno de los paquetes fundamentales más importantes para la computación numérica en Python. La mayoría de los paquetes computacionales que brindan funcionalidad científica utilizan los objetos de matriz de NumPy como lenguaje común para el intercambio de datos.

Algunos características importantes: .

- **ndarray**, un **arreglo multidimensional eficiente** que proporciona operaciones aritméticas orientadas a arreglos rápidos y capacidades de transmisión flexibles.
- Funciones matemáticas para operaciones rápidas en **arreglos completos de datos sin tener que escribir bucles/ciclos**.
- Proporciona formas de **crear, almacenar y/o manipular datos**, lo que le permite integrarse sin problemas y rápidamente con una amplia variedad de bases de datos.
- **Capacidades de álgebra lineal, generación de números aleatorios y transformada de Fourier.**
- Una API de C para conectar NumPy con bibliotecas escritas en C, C++ o FORTRAN.

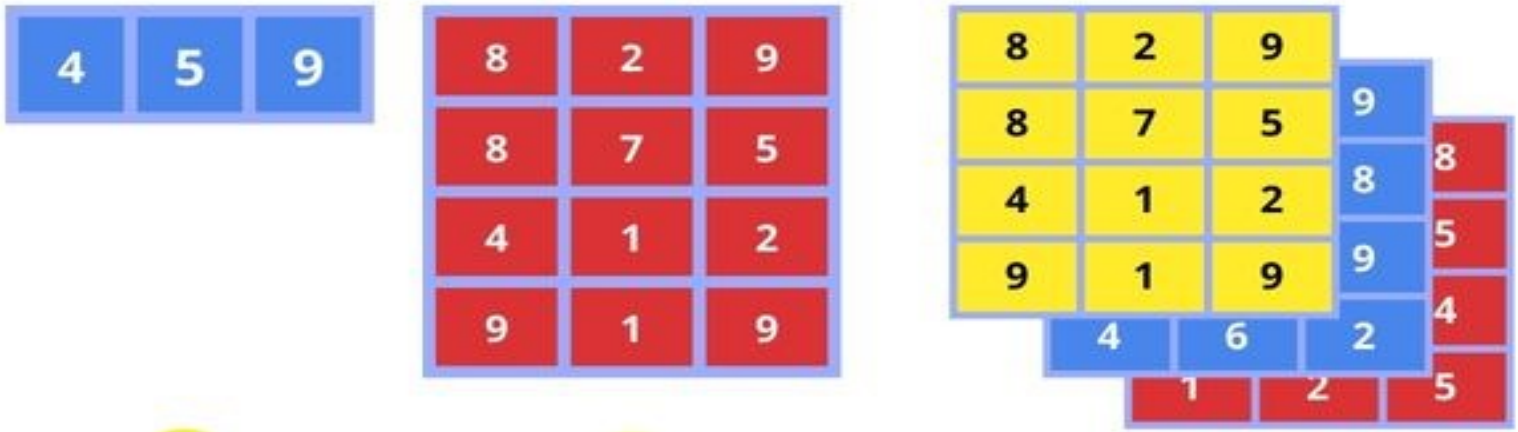
Creando un arreglo en Numpy

```
import numpy as np
print(np.__version__)
```



Creando un arreglo en Numpy

Las secuencias anidadas, como una lista de listas de igual longitud, se convertirán en una matriz multidimensional:



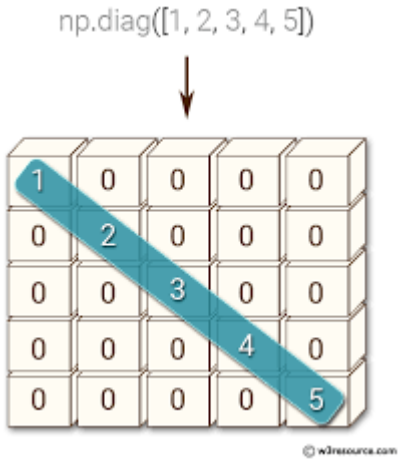
np.array([4,5,9])

np.array([[8,2,9],
[8,7,5],
[4,1,2],
[9,1,9]])

np.array([[[8,2,9],
[8,7,5],[4,1,2],[9,1,9]],
[[4,5,9],[4,2,8],
[2,1,9],[4,6,2]],
[[6,8,8],[2,4,5],
[7,8,4],[1,2,5]]])

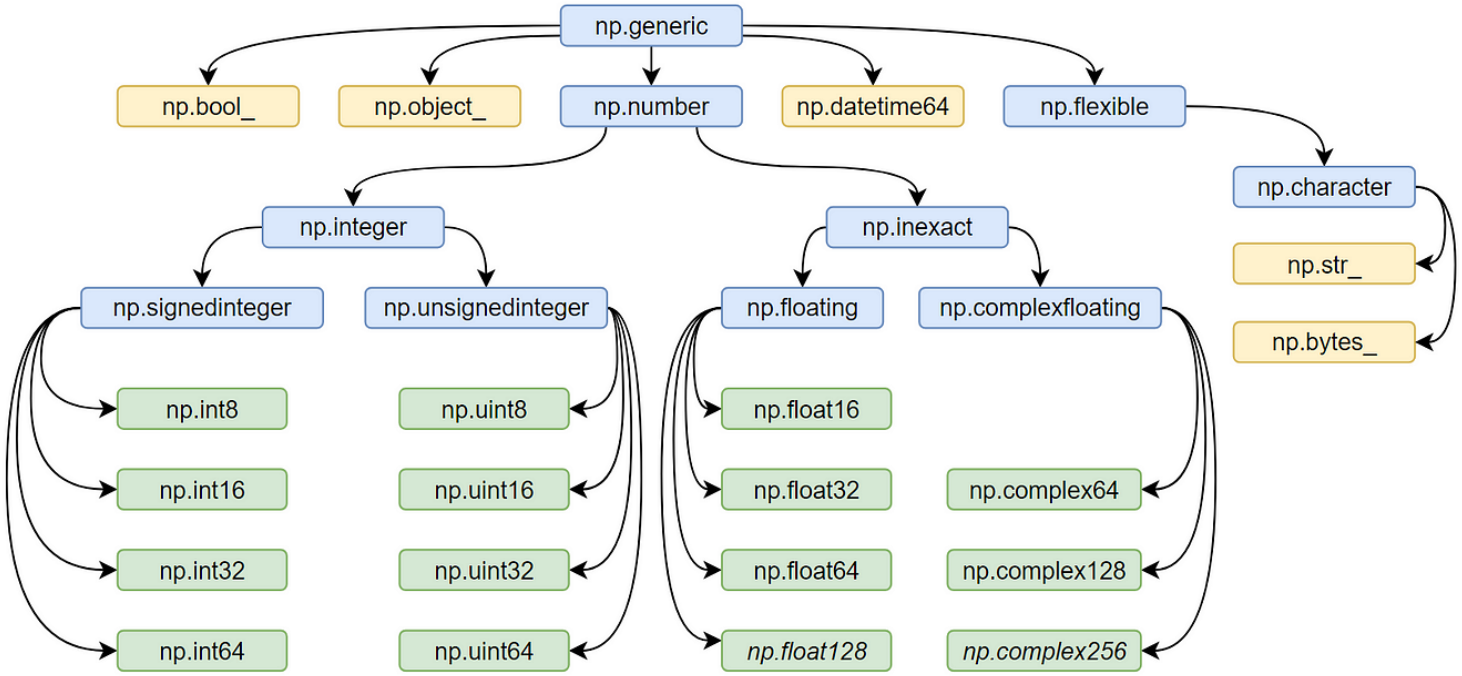
```
a = np.array([1,2,3])
b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
             dtype = float)
```

Creando arrays

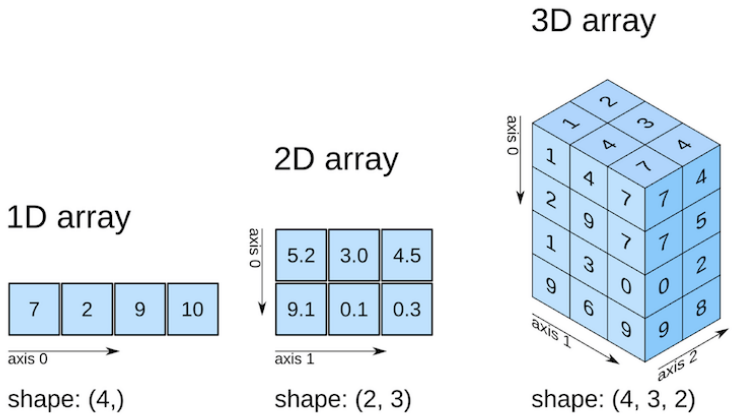


1. **Crear un arreglo de ceros**:
   np.zeros((3,4))
2. **Crear un arreglo de unos**:
   np.ones((2,3,4), dtype=np.int16)
3. **Crear un arreglo de valores equiespaciados (con paso)**:
   d = np.arange(10,25,5)
4. **Crear un arreglo de valores equiespaciados (con número de muestras)**:
   np.linspace(0,2,9)
5. **Crear un arreglo constante**:
   e = np.full((2,2),7)
6. **Crear una matriz identidad**:
   f = np.eye(2)
7. **Crear un arreglo con valores aleatorios**:
   np.random.random((2,2))
8. **Crear un arreglo vacío**:
   np.empty((3,2))

Tipos Numpy arrays



Inspeccionando Elementos



1. a.shape
- •Retorna las dimensiones del array a, indicando el número de filas y columnas en caso de ser un array bidimensional.

2. len(a)
- •Retorna la longitud del array a, es decir, el número total de elementos en el array.

3. b.ndim
- •Retorna el número de dimensiones del array b, es decir, la cantidad de ejes que tiene el array.

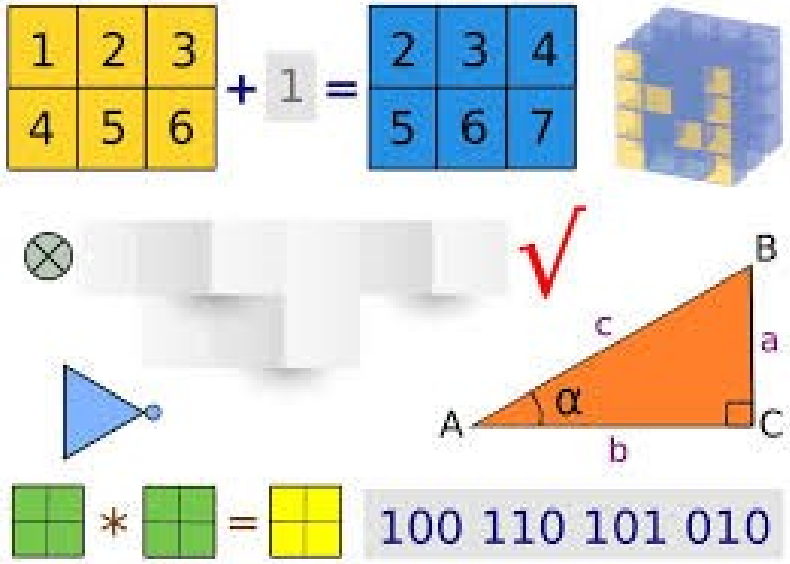
4. e.size
- •Retorna el número total de elementos en el array e, equivalente a la multiplicación de las dimensiones del array.

5. b.dtype
- •Retorna el tipo de datos de los elementos del array b.

6. b.dtype.name
- •Retorna el nombre del tipo de datos de los elementos del array b.

7. b.astype(int)
- •Convierte los elementos del array b a un tipo de datos diferente, en este caso, a un tipo entero.

Operadores



1. **Suma y Resta**: +, -
   - •Se utilizan para sumar y restar elementos de arrays de forma element-wise.
2. **Multiplicación y División**: *, /
   - •Permiten multiplicar y dividir elementos de arrays de forma element-wise.
3. **Exponenciación**: np.exp()
   - •Calcula la exponenciación de cada elemento de un array.
4. **Raíz Cuadrada**: np.sqrt()
   - •Calcula la raíz cuadrada de cada elemento de un array.
5. **Funciones Trigonométricas**: np.sin(), np.cos()
   - •Calculan el seno y coseno de cada elemento de un array.
6. **Logaritmo Natural**: np.log()
   - •Calcula el logaritmo natural de cada elemento de un array.
7. **Producto Punto**: np.dot()
   - •Calcula el producto punto entre dos arrays.

Funciones

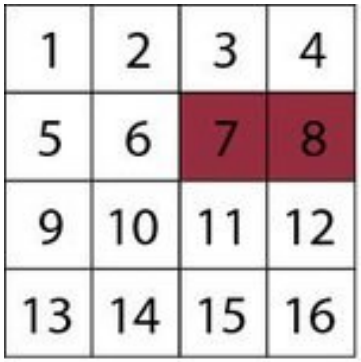
Comparaciones Element-wise

- **a == b**: Comparación elemento a elemento.
  - array([[False, True, True], [False, False, False]], dtype=bool)
- **a < 2**: Comparación elemento a elemento con un valor.
  - array([True, False, False], dtype=bool)
- **np.array_equal(a, b)**: Comparación de arrays completos.
  - True si los arrays son iguales, False de lo contrario.

Funciones Estadísticas Array-wise

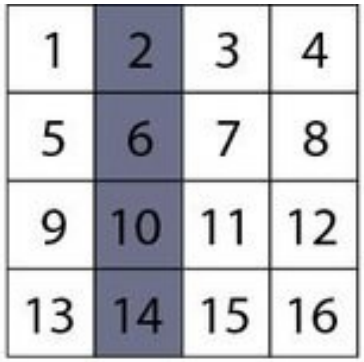
- **a.sum()**: Suma de todos los elementos del array a.
- **a.min()**: Valor mínimo en el array a.
- **b.max(axis=0)**: Valor máximo en cada fila del array b.
- **b.cumsum(axis=1)**: Suma acumulativa de los elementos a lo largo de las filas del array b.
- **a.mean()**: Promedio de todos los elementos del array a.
- **b.median()**: Mediana de los elementos del array b.
- **a.corrcoef()**: Coeficiente de correlación de los elementos del array a.
- **np.std(b)**: Desviación estándar de los elementos del array b.

Slicing



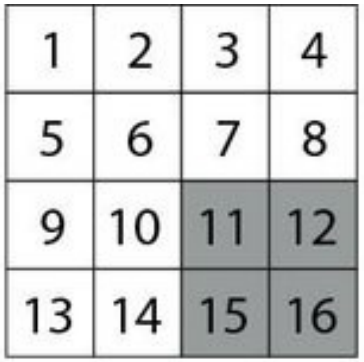
Row one, columns two to four

```
>>> arr[1, 2:4]
array([7, 8])
```



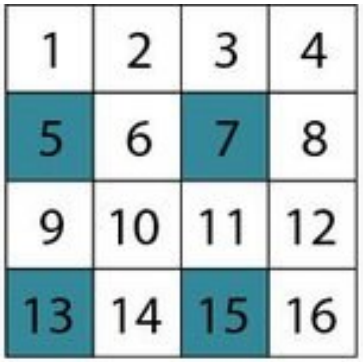
All rows in column one

```
>>> arr[:, 1]
array([2,  6, 10, 14])
```



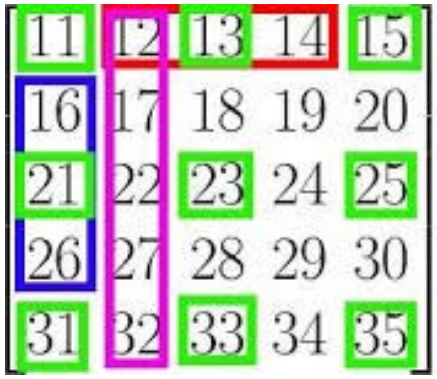
All rows after row two,
all columns after column two

```
>>> arr[2:, 2:]
array([[11, 12],
       [15, 16]])
```



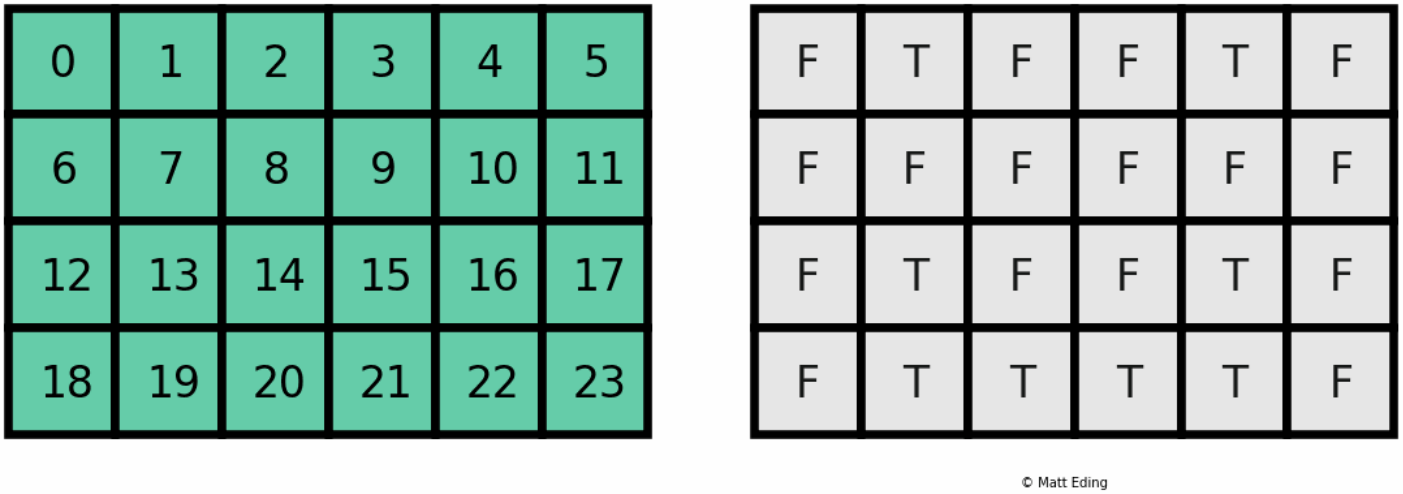
Every other row after row one,
every other column

```
>>> arr[1::2, ::2]
array([[5, 7],
       [13, 15]])
```



- `print(a[0, 1:4])`
- `print(a[1:4, 0])`
- `print(a[::2,::2])`
- `print(a[:, 1])`

Indexación Boleana




```
import numpy as np
arr = np.array([4, 88, 7, 12, 4, 9, 52, 4])
mascara= (arr > 50)
print(mascara)
                array([False, True, False, False, False, False, True, False])
n=arr[mascara]
print(n)
                    array([88., 52.])
```

Operaciones Matriciales

- **Transponer Array**: np.transpose()
  - Permuta las dimensiones de un array.
- **Cambiar Forma de Array**: ravel(), reshape()
  - ravel(): Aplana el array en una sola dimensión.
  - reshape(): Cambia la forma del array sin cambiar los datos.
- **Añadir/Quitar Elementos**: resize(), append(), insert(), delete()
  - resize(): Cambia la forma del array.
  - append(): Añade elementos al final del array.
  - insert(): Inserta elementos en una posición específica del array.
  - delete(): Elimina elementos de una posición específica del array.
- **Combinar Arrays**: concatenate(), vstack(), hstack(), column_stack()
  - concatenate(): Concatena arrays a lo largo de un eje específico.
  - vstack(): Apila arrays verticalmente (por filas).
  - hstack(): Apila arrays horizontalmente (por columnas).
  - column_stack(): Crea arrays apilados por columnas.
- **Dividir Arrays**: hsplit(), vsplit()
  - hsplit(): Divide el array horizontalmente en una posición específica.
  - vsplit(): Divide el array verticalmente en una posición específica.

Manejo de arquivos

Lectura de Archivos

•**open():** Utilizamos la función open() para abrir un archivo en modo de lectura.

•**Métodos de Lectura:**

- •read(): Lee todo el contenido del archivo.
- •readline(): Lee una línea del archivo.
- •readlines(): Lee todas las líneas del archivo y las devuelve como una lista.

```
with open("archivo.txt", "r") as archivo:
        contenido = archivo.read()
```

Escritura de Archivos

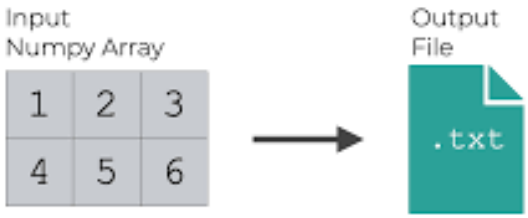
•**open():** Usamos la función open() con el modo de escritura ("w") para abrir un archivo en modo de escritura.

•**Modos de Apertura:**

- •"w": Escritura (sobrescribe el archivo existente).
- •"a": Escritura (añade contenido al final del archivo).

```
# Escritura en archivo
with open("salida.txt", "w") as salida:
      salida.write("Datos procesados:\n")
```

Manejo de arquivos Numpy



savetxt()

•Utilizada para guardar datos de arrays NumPy en un archivo de texto.

•Sintaxis:

python

•np.savetxt(nombre_archivo, datos, delimiter=',', fmt='%d')

•Parámetros:

•nombre_archivo: Nombre del archivo donde se guardarán los datos.

•datos: Array NumPy con los datos a guardar.

•delimiter: Delimitador que separa los valores en el archivo CSV.

•fmt: Formato de los datos en el archivo.

Manejo de arquivos Numpy

genfromtxt()
•Utilizada para cargar datos desde un archivo de texto en un array NumPy.
•Sintaxis:
python
•datos = np.genfromtxt(nombre_archivo, delimiter=',')
•Parámetros:
•nombre_archivo: Nombre del archivo del que se cargarán los datos.
•delimiter: Delimitador que separa los valores en el archivo CSV.
•La función genfromtxt() maneja automáticamente datos faltantes y tipos de datos mixtos

loadtxt()
•loadtxt() es más simple y eficiente que genfromtxt() cuando se trabaja con archivos de texto que contienen
datos uniformemente estructurados (por ejemplo, todos los valores son del mismo tipo y no hay datos faltantes).
•No maneja automáticamente valores faltantes o tipos de datos mixtos.
•Se puede utilizar para cargar datos desde archivos de texto donde todas las filas y columnas tienen el mismo número de elementos.

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

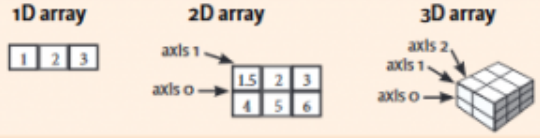
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((2,4))                  Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)  Create an array of ones
>>> d = np.arange(10,25,5)           Create an array of evenly
                                     spaced values (step value)
>>> np.linspace(0,2,9)               Create an array of evenly
                                     spaced values (number of samples)
>>> e = np.full((2,2),7)             Create a constant array
>>> f = np.eye(2)                    Create a 2X2 identity matrix
>>> np.random.random((2,2))          Create an array with random values
>>> np.empty((3,2))                  Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape          Array dimensions
>>> len(a)           Length of array
>>> b.ndim           Number of array dimensions
>>> e.size           Number of array elements
>>> b.dtype          Data type of array elements
>>> b.dtype.name     Name of data type
>>> b.astype(int)    Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                        Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                 Subtraction
>>> b + a                            Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                      Addition
>>> a / b                            Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                   Division
>>> a * b                            Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                 Multiplication
>>> np.exp(b)                        Exponentiation
>>> np.sqrt(b)                       Square root
>>> np.sin(a)                        Print sines of an array
>>> np.cos(b)                        Element-wise cosine
>>> np.log(a)                        Element-wise natural logarithm
>>> e.dot(f)                         Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                           Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                            Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)             Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                          Array-wise sum
>>> a.min()                          Array-wise minimum value
>>> b.max(axis=0)                    Maximum value of an array row
>>> b.cumsum(axis=1)                 Cumulative sum of the elements
>>> a.mean()                         Mean
>>> b.median()                       Median
>>> a.corrcoef()                     Correlation coefficient
>>> np.std(b)                        Standard deviation
```

Copying Arrays

```
>>> h = a.view()     Create a view of the array with the same data
>>> np.copy(a)       Create a copy of the array
>>> h = a.copy()     Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()         Sort an array
>>> c.sort(axis=0)   Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                             Select the element at the 2nd index
  3
>>> b[1,2]                           Select the element at row 1 column 2
  6.0                                (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                           Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]                         Select items at rows 0 and 1 in column 1
  array([ 2. ,  5.1])
```

```
>>> b[:1]                            Select all items at row 0
  array([[1.5, 2., 3.]])             (equivalent to b[0:1, :])
>>> c[1,...]                         Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]                        Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                           Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]    Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. ,  2. ,  6. ,  1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]    Select a subset of the matrix's rows
  array([[ 4. ,  5. ,  6. ,  4. ],   and columns
         [ 1.5,  2. ,  3. ,  1.5],
         [ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)              Permute array dimensions
>>> i.T                              Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                        Flatten the array
>>> g.reshape(3,-2)                  Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))                  Return a new array with shape (2,6)
>>> np.append(h,g)                   Append items to an array
>>> np.insert(a, 1, 5)               Insert items in an array
>>> np.delete(a,[1])                 Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)     Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                 Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                       Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                 Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))           Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                       Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                   Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])]  index
>>> np.vsplit(c,2)                   Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```