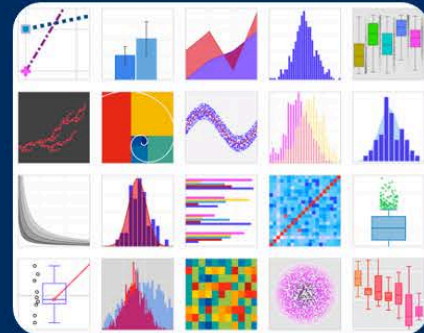


Introducción al Análisis y Visualización de Datos en Python

Día 2 – Almacenamiento de datos (Numpy)



Presentan:

Dr. Ulises Olivares Pinto

Walter André Rosales Reyes

Escuela Nacional de Estudios Superiores Unidad Juriquilla



Contenido



1. Almacenamiento de datos

(~3 horas - 2 bloques)

Diccionarios

Funciones

Numpy

- Arreglos
- Matrices
- Subconjuntos
- Operaciones básicas



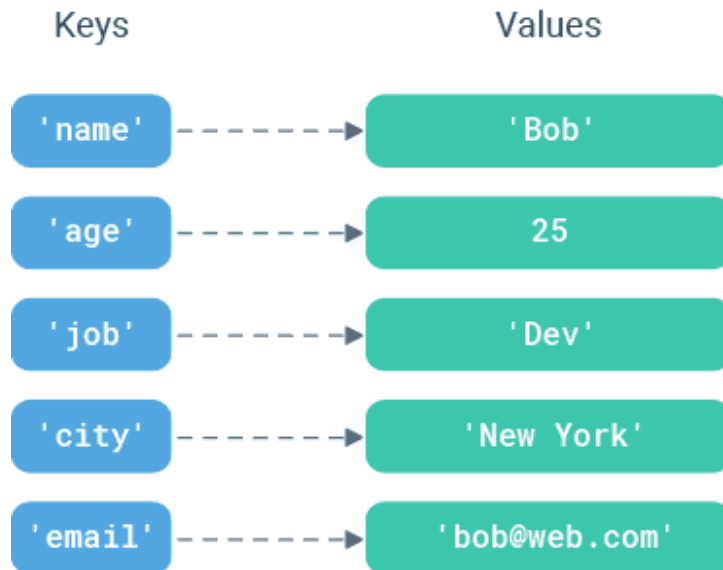
2. Proyecto (30 ~45 minutos)



Diccionarios

- Un diccionario es una **estructura de datos** y un **tipo de dato** en Python que permite almacenar cualquier tipo de dato e identificarlo a través de una llave.

- `dict = {}`
- `dict = {"nombre": "Ulises Olivares"}`
- `dict = {"nombre": "Ulises Olivares", "edad": 32, "cursos": ["Matemáticas", "Programación"]}`
- **Acesso:**
 - `print(dict["nombre"])`
 - `print(dict["cursos"][0])`



Diccionarios (Métodos)

Función	Descripción
clear ()	Elimina todos los elementos del diccionario
copy ()	Devuelve una copia del diccionario
fromkeys ()	Devuelve un diccionario con las claves y el valor especificados
get ()	Devuelve el valor de la clave especificada
items ()	Devuelve una lista que contiene una tupla para cada par clave-valor
keys ()	Devuelve una lista que contiene las claves del diccionario
pop ()	Elimina el elemento con la clave especificada
popitem ()	Elimina el último par clave-valor insertado
setdefault ()	Devuelve el valor de la clave especificada. Si la clave no existe, inserta la clave con el valor especificado
update ()	Actualiza el diccionario con los pares clave-valor especificados
values ()	Devuelve una lista de todos los valores del diccionario.

Diccionarios

Iterar sobre un **diccionario**:

```
1. >>> frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla', 'Guayaba':'rosa'}
2. >>> for nombre, color in frutas.items():
3. ...     print nombre, "es de color", color
4. ...
5. Fresa es de color roja
6. Limon es de color verde
7. Manzana es de color amarilla
8. Papaya es de color naranja
9. Guayaba es de color rosa
```

Ejercicio 1 – (10 minutos)

- Escribir un programa que guarde en un diccionario los precios de algunos artículos de oficina.
- Se deberá preguntar al usuario por un artículo, y cantidad del mismo. Se deberá imprimir el precio total.



Funciones

- Una función es un **bloque de código** con un **nombre asociado**, que recibe ningún o más de un **argumento** como entrada, ejecuta una secuencia de **sentencias** y devuelven un **valor**. Estos bloques pueden ser llamados cuantas veces sea necesario.

Sintaxis

```
def NOMBRE (LISTA_DE_PARAMETROS) :  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

Advertencia:

Los bloques de `function` deben estar indentado como otros bloques estructuras de control.

Función main()

In [10]:

```
1  """
2      Función suma
3      Entradas: Dos números (enteros o flotantes)
4      Salida: Sumatoria de dos números
5  """
6  def suma(num1, num2):
7      suma = num1 + num2
8      return suma
9
10 # Función principal
11 def main():
12     a = 2
13     b = 3
14     print("La suma de "+str(a)+"+"+str(b)+" Es: " + str(suma(a,b)) )
15
16 # Función que detecta el interprete de python
17 if __name__ == "__main__":
18     print("Este es el inicio del programa")
19     main()
20
```

Este es el inicio del programa

La suma de 2+3 Es: 5

Ejercicio 2 – 15 minutos (Funciones)

1. Escribir una función para que dado un número entero, calcule su factorial.

$$n! = 1 \times 2 \times 3 \times 4 \times \dots \times (n - 1) \times n.$$

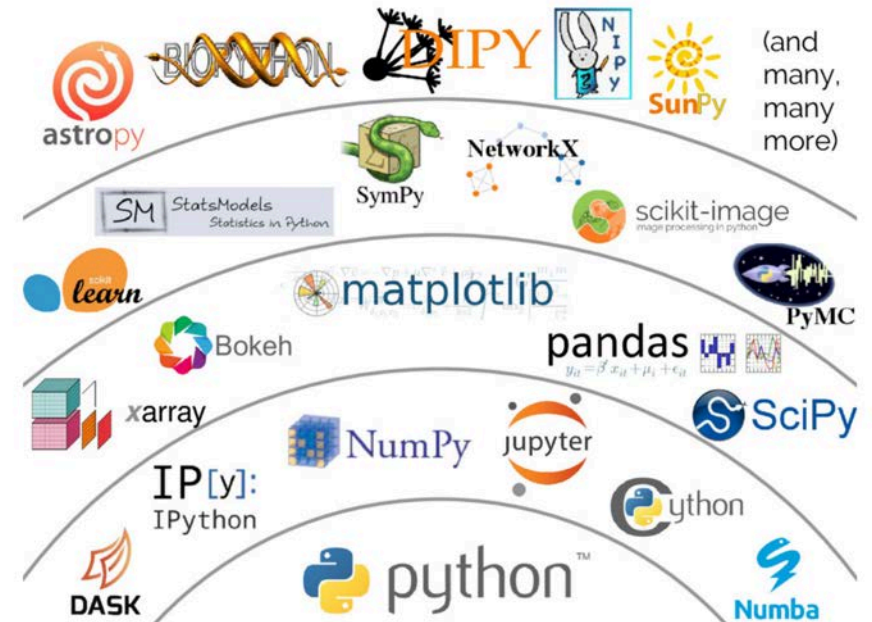
2. Escribir una función para calcular si una cadena es **palíndroma**.

RECONOCER
RECONOCER



Librerías

- Las librerías (módulos) permiten **extender la funcionalidad** de Python.
- Un **módulo** es una porción de programa que realiza una operación específica.
- Una **librería** contiene un conjunto de módulos.



Librerías - Números Aleatorios

```
[18]: 1 # Importar librería random
      2 import random as rn          # Se importa librería con el alias rn
      3 rn.seed(20) # Semilla

[19]: 1 print(random.random())      # Número aleatorio x, 0.0 <= x < 1.0
      0.9056396761745207

[12]: 1 print(random.random())      # Nú x, 0.0 <= x < 1.0
      0.2598274474889769

[13]: 1 rn.uniform(1, 10)           # Número aleatorio x, 1.0 <= x < 10.0
      6.7215328264539025

[13]: 1 rn.randint(0, 10)           # Número aleatorio entero x, 0, 10
      9

[16]: 1 rn.randrange(0, 100, 2)     # Número entero par x, 0, 100
      86

[22]: 1 random.choice('abcdefghij') # Se elige un elemento de forma aleatoria
      'b'

[27]: 1 # Mezcla los elementos de forma aleatoria
      2 elementos = [1, 2, 3, 4, 5, 6, 7]
      3 random.shuffle(elementos)
      4 print(elementos)
      [4, 1, 2, 5, 3, 7, 6]

[28]: 1 random.sample([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 3) # Escoje tres elementos al azar
      [9, 8, 7]
```

Python » English » 3.8.5 » Documentation » The Python Standard Library » Numeric and Mathematical Modules »

Table of Contents

random — Generate pseudo-random numbers

- Bookkeeping functions
- Functions for integers
- Functions for sequences
- Real-valued distributions
- Alternative Generator
- Notes on Reproducibility
- Examples and Recipes

Previous topic

fractions — Rational numbers

Next topic

statistics — Mathematical statistics functions

This Page

Report a Bug
Show Source

random — Generate pseudo-random numbers

Source code: [Lib/random.py](https://lib/random.py)

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available.

Almost all module functions depend on the basic function `random()`, which generates a random float uniformly in the semi-open range [0.0, 1.0). Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of $2^{19937}-1$. The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

The functions supplied by this module are actually bound methods of a hidden instance of the `random.Random` class. You can instantiate your own instances of `Random` to get generators that don't share state.

Class `Random` can also be subclassed if you want to use a different basic generator of your own devising: in that case, override the `random()`, `seed()`, `getstate()`, and `setstate()` methods. Optionally, a new generator can supply a `getrandbits()` method — this allows `randrange()` to produce selections over an arbitrarily large range.

The `random` module also provides the `SystemRandom` class which uses the system function `os.urandom()` to generate random numbers from sources provided by the operating system.

Warning: The pseudo-random generators of this module should not be used for security purposes. For security or cryptographic uses, see the `secrets` module.

See also: M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Transactions on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 1998.

Complementary-Multiply-with-Carry recipe for a compatible alternative random number generator with a long period and comparatively simple update operations.

Librerías – Funciones Matemáticas

```
[1]: import math
    math.pi

[1]: 3.141592653589793

[2]: math.sin(90)

[2]: 0.8939966636005579

[4]: math.sqrt(9)

[4]: 3.0

[5]: math.ceil(2.5)

[5]: 3

[6]: math.floor(2.9)

[6]: 2

[8]: math.cos(90)

[8]: -0.4480736161291701
```

Python » Spanish » 3.8.5 » Documentation » La Biblioteca Estándar de Python » Numeric and Mathematical Modules »

anterior | siguiente | módulos | índice

Búsqueda rápida

Tabla de contenido

math — Mathematical functions

- Number-theoretic and representation functions
- Power and logarithmic functions
- Trigonometric functions
- Angular conversion
- Hyperbolic functions
- Special functions
- Constants

Tema anterior
numbers — Clase base abstracta numérica

Próximo tema
cmath — Mathematical functions for complex numbers

Esta página

[Reporta un Bug](#)
[Mostrar el código](#)

math — Mathematical functions

This module provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the **cmath** module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

Number-theoretic and representation functions

math.ceil(x)

Return the ceiling of *x*, the smallest integer greater than or equal to *x*. If *x* is not a float, delegates to *x*.`__ceil__()`, which should return an [Integral](#) value.

math.comb(n, k)

Return the number of ways to choose *k* items from *n* items without repetition and without order.

Evaluates to $n! / (k! * (n - k)!)$ when $k \leq n$ and evaluates to zero when $k > n$.

Also called the binomial coefficient because it is equivalent to the coefficient of *k*-th term in polynomial expansion of the expression $(1 + x)^n$.

Raises [TypeError](#) if either of the arguments are not integers. Raises [ValueError](#) if either of the arguments are negative.

Nuevo en la versión 3.8.

math.copysign(x, y)

Return a float with the magnitude (absolute value) of *x* but the sign of *y*. On platforms that support signed zeros, `copysign(1.0, -0.0)` returns `-1.0`.

math.fabs(x)

Return the absolute value of *x*.

math.factorial(x)

Return *x* factorial as an integer. Raises [ValueError](#) if *x* is not integral or is negative.

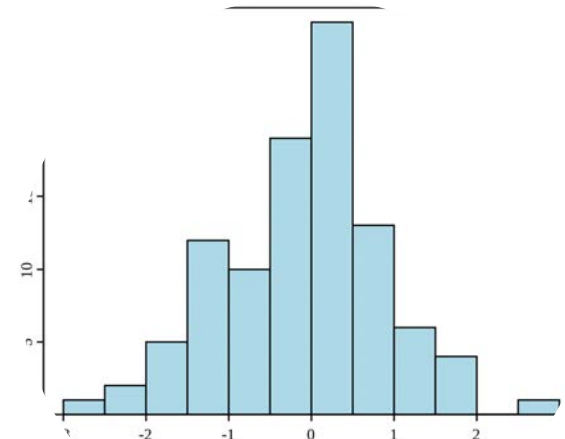
math.floor(x)

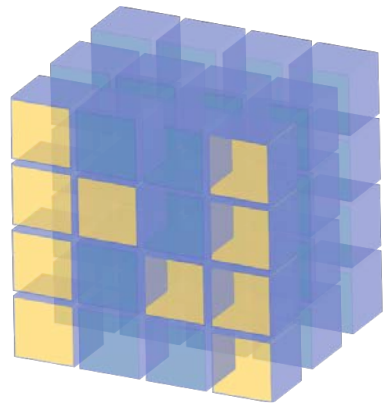
Return the floor of *x*, the largest integer less than or equal to *x*. If *x* is not a float, delegates to *x*.`__floor__()`, which should return an [Integral](#) value.

Ejercicio 3

15 minutos

- Se deberá generar un total de 10,000 números aleatorios en un rango del [1, 100].
1. Se deberá generar una lista en forma de histograma con 10 clases, se deberá contabilizar la totalidad de elementos en cada clase.





NumPy

¿Qué es Numpy?

- **Numpy**, **Scipy** y **Matplotlib**

proporcionan una funcionalidad similar a **MATLAB** en Python.

Características de Numpy:

- Arreglos multidimensionales con un tipo
- Cálculos numéricos **rápidos** (matriciales)
- Funciones matemáticas de **alto nivel**.

¿Por qué necesitamos NumPy?

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$



Python hace cálculos
numéricos lentamente.



Multiplicar dos
matrices de
1000 x 1000

El ciclo triple de
Python tarda **> 10 min.**

Numpy tarda **~ 0.03 segundos**



NumPy Contenido

Arreglos

Forma y transposición

Operaciones matemáticas

Indexación y conjuntos

Arreglos – Numpy

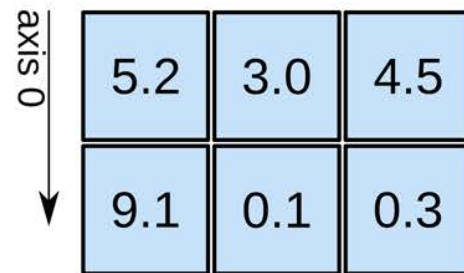
1D array



axis 0

shape: (4,)

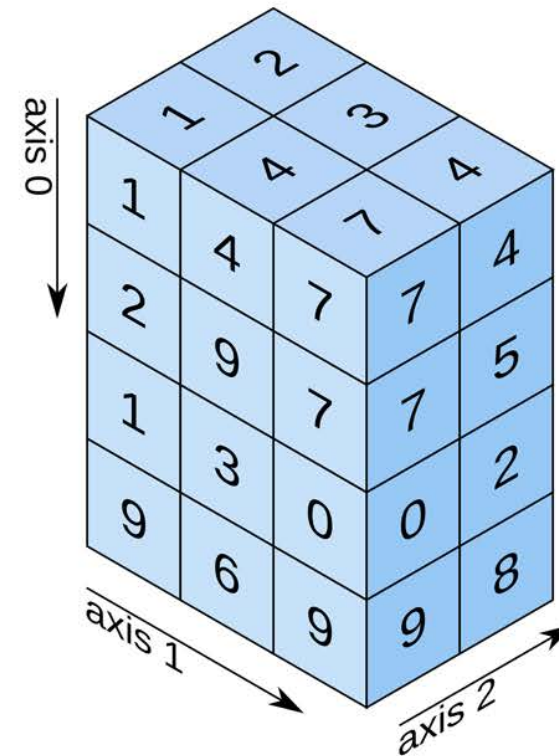
2D array



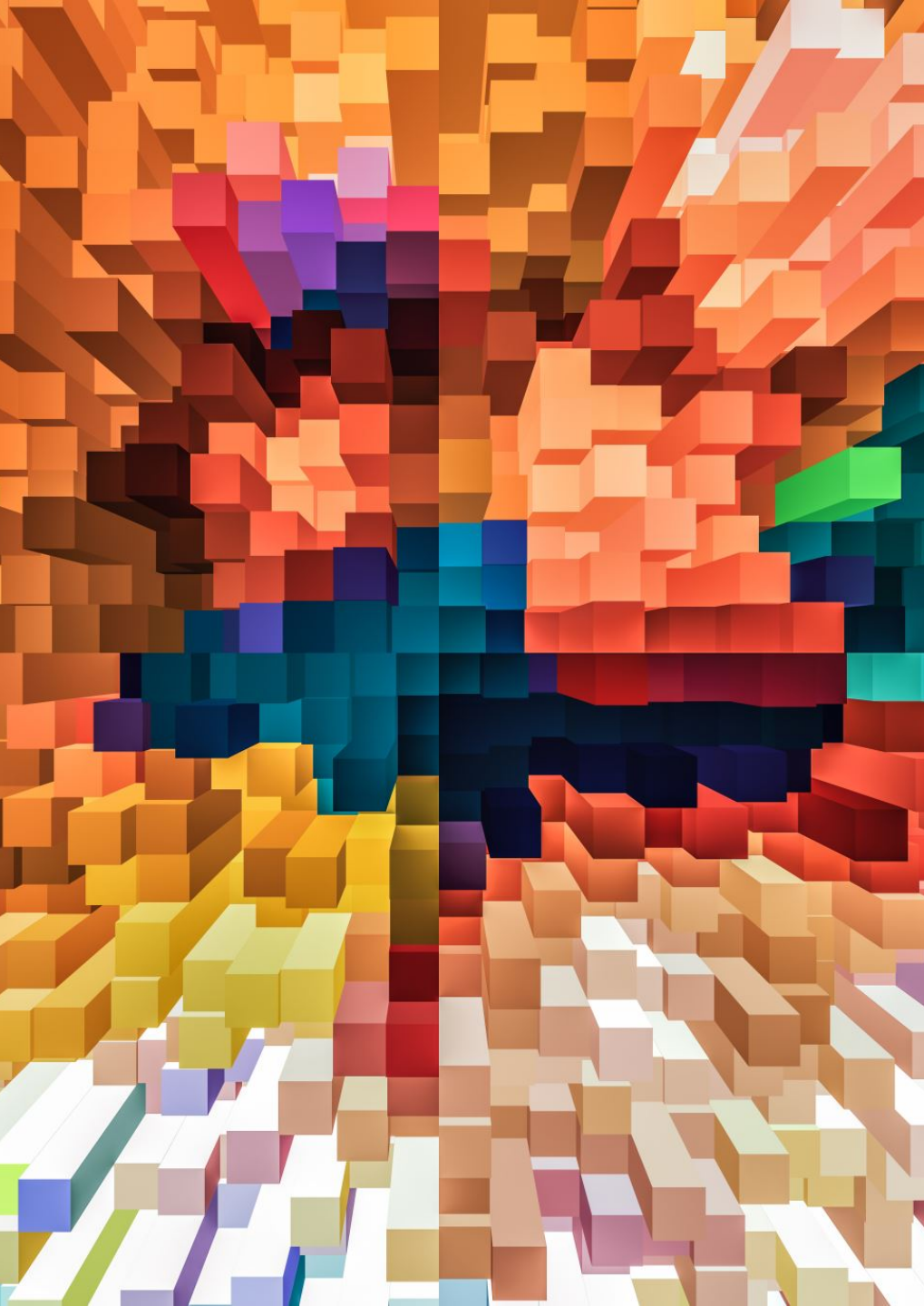
axis 1

shape: (2, 3)

3D array



shape: (4, 3, 2)



Arreglos

Listas estructuradas de números.

- Vectores
- Matrices
- Imágenes
- Tensores
- ConvNets

Arreglos

Listas estructuradas de números.

- **Vectores**
- **Matrices**
- Imágenes
- Tensores
- ConvNets

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

Arreglos

Listas estructuradas de números.

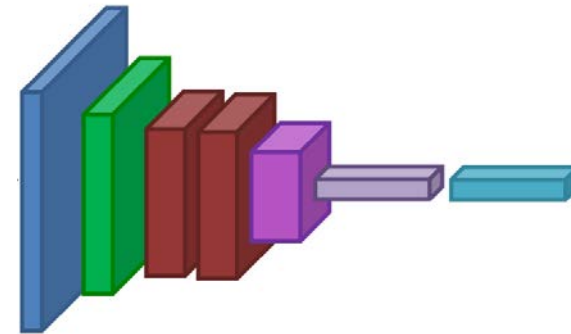
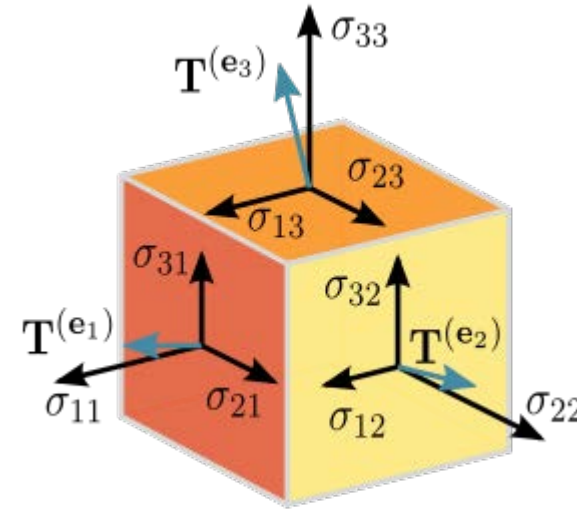
- Vectores
- Matrices
- **Imágenes**
- Tensores
- ConvNets



Arreglos

Listas estructuradas de números.

- Vectores
- Matrices
- Imágenes
- **Tensores**
- **ConvNets**



Arreglos

Listas estructuradas de números.

- Vectores
- Matrices
- Imágenes
- Tensores
- ConvNets

MATRICES

IMAGES

TENSORS

CONVNETS



Arreglos, Propiedades Básicas


1. Las matrices pueden tener cualquier número de dimensiones, incluido cero (un escalar).
2. Las matrices se poseen distintos tipos de datos:
 - np.uint8
 - np.int64,
 - np.float32
 - np.float64
3. Las matrices son densas.

Cada elemento de la matriz existe y tiene el mismo tipo.

```
# importar librería de numpy
import numpy as np
# Crear un arreglo bi-dimensional
a = np.array([[1,2,3],
              [4,5,6]],
              dtype = np.float32)
# Imprimir la matriz
print(a)
# Imprimir propiedades de la matriz
print("Dimensiones: " + str(a.ndim))
print("Forma: " + str(a.shape))
print("Tipo de datos: " + str(a.dtype))
```



Creación de arreglos

- `np.ones`, `np.zeros`
 - `np.arange`
 - `np.concatenate`
 - `np.astype`
 - `np.zeros_like`, `np.ones_like`
 - `np.random.random`
- 

Creación de arreglos

- **np.ones, np.zeros**
- np.arange
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

```
# Creación de arreglos
# Arreglo de 1s
ones = np.ones((5,5), dtype=np.float32)
# Arreglo de 0s
zeros = np.zeros((4,4), dtype=np.float32)

#Impresión de matrices
print(ones)
print(zeros)
```

```
# Acceso a la matriz usando índices
for i in range(0, len(ones)):
    for j in range(0, len(ones[0])):
        print(ones[i][j], end= " ")
    print(" ")

# Acceso usando elementos
for i in ones:
    for j in i:
        print(j , end = " ")
    print(" ")
```

Ejercicio 4

- Se deberá generar una matriz de ceros de 10 x 10 y se deberán inicializar la diagonal principal con 1s.
- Se deberá calcular la traza de la matriz anterior.

Traza de una matriz

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$\text{tr } A = a_{11} + a_{22} + a_{33} + a_{44}$$

Creación de arreglos

- np.ones, np.zeros
- **np.arange**
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

```
# Generación de un vector  
np.arange(1, 100)
```

```
array([ 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, 81, 82, 83, 84, 85,  
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

A photograph of a wooden desk with a laptop, a cup of coffee, a notebook, and a smartphone. The desk is made of light-colored wood. A white laptop is partially visible on the left. A white mug with dark liquid is on the left. A notebook with a pen is in the center. A black smartphone is on the right. The background is a dark, textured wall.

Ejercicio 5

- Generar un vector (lineal) de 100 elementos del 0 al 99.
- Se deberá escribir un **-1** en todos aquellos elementos que son "**impares**" y un **-2** en todos los elementos que son "**pares**".

Creación de arreglos

- np.ones, np.zeros
- np.arange
- **np.concatenate**
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

```
# Concatenando arreglos o matrices
A = np.ones((4,4), dtype = np.float32)
B = np.zeros((5,4), dtype = np.float32)

# concatenar matrices
np.concatenate([A,B])

array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]], dtype=float32)
```


Creación de arreglos

- np.ones, np.zeros
- np.arange
- **np.concatenate**
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

```
# Concatenando arreglos o matrices
```

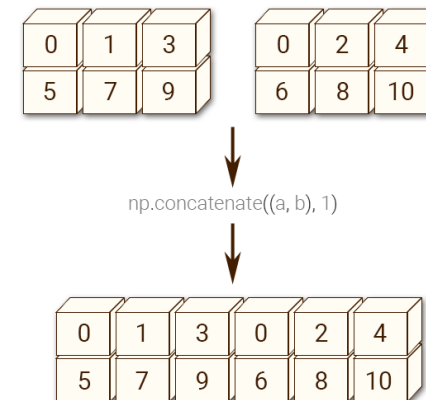
```
A = np.ones((5,3), dtype = np.float32)
```

```
B = np.zeros((5,4), dtype = np.float32)
```

```
# concatenar matrices
```

```
np.concatenate([A,B], axis = 1)
```

```
array([[1., 1., 1., 0., 0., 0., 0.],  
       [1., 1., 1., 0., 0., 0., 0.],  
       [1., 1., 1., 0., 0., 0., 0.],  
       [1., 1., 1., 0., 0., 0., 0.],  
       [1., 1., 1., 0., 0., 0., 0.]], dtype=float32)
```



Creación de arreglos

- np.ones, np.zeros
- np.arange
- np.concatenate
- **np.astype**
- np.zeros_like, np.ones_like
- np.random.random

```
# Uso de Astype para hacer un cast de tipos de datos
A = np.zeros((5,4), dtype = np.float32)
print(len(A[0]))

for i in range(len(A)):
    for j in range(len(A[0])):
        A[i][j] = np.pi
print(A)

print(A.astype(np.int16))
```

```
4
[[3.1415927 3.1415927 3.1415927 3.1415927]
 [3.1415927 3.1415927 3.1415927 3.1415927]
 [3.1415927 3.1415927 3.1415927 3.1415927]
 [3.1415927 3.1415927 3.1415927 3.1415927]
 [3.1415927 3.1415927 3.1415927 3.1415927]]
[[3 3 3 3]
 [3 3 3 3]
 [3 3 3 3]
 [3 3 3 3]
 [3 3 3 3]]
```

Creación de arreglos

- np.ones, np.zeros
- np.arange
- np.concatenate
- np.astype
- **np.zeros_like, np.ones_like**
- np.random.random

```
# Uso de np.zeros_like y np.ones_like
A = np.ones((4,4))
print(A)
# zeros_like genera un arreglo de las
# mismas dimensiones que el original
B = np.zeros_like(A)
print(B)
```

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

Creación de arreglos

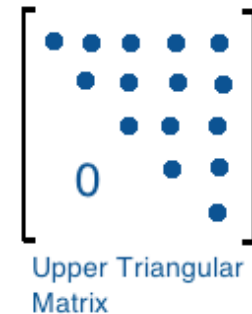
- np.ones, np.zeros
- np.arange
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- **np.random.random**

```
# Generación de arreglos aleatorios
A = np.random.random((5, 5))
print(A)
# Generación de números enteros del 0 al n-1
n = 10
B= np.random.randint(10, size = (7,7))
print(B)
```

```
[[0.15632288 0.73735589 0.77775127 0.77383993 0.51336574]
 [0.53877873 0.69228787 0.06430154 0.50413543 0.32967815]
 [0.4927091  0.14194562 0.8078162  0.19333272 0.49208783]
 [0.0791934  0.86787715 0.81568641 0.97690664 0.82011968]
 [0.90603684 0.64281865 0.14918772 0.15695209 0.82954009]]
[[6 7 1 8 3 3 1]
 [5 6 3 3 1 9 1]
 [5 7 9 9 7 4 6]
 [2 8 5 9 8 9 1]
 [9 5 3 5 4 4 2]
 [0 9 6 2 9 2 5]
 [6 4 8 9 4 0 8]]
```

Ejercicio 6

- Generar una matriz de 10 x 10 elementos e inicializarla con números aleatorios del 0 al 99.
- Se deberá calcular la sumatoria solamente de los elementos en la diagonal superior de la matriz.



Arreglos, zona de peligro

- Deben ser densos, sin espacios vacíos.
- Deben ser de un solo tipo
- No se pueden combinar arreglos de distintos tipos

```
>>> np.ones([7,8]) + np.ones([9,3])  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ValueError: operands could not be broadcast together  
with shapes (7,8) (9,3)
```



```

# Transformaciones de arreglos
a = np.array([1,2,3,4,5,6])
print(a)

# Transformar a 3 filas, 2 columnas
a = a.reshape(3, 2)
print(a)

# Se utiliza -1 para determinar cols
a = a.reshape(2, -1)
print(a)

# Transformar a un arreglo lineal
a = a.ravel()
print(a)

```

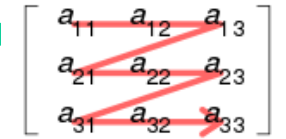
```

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

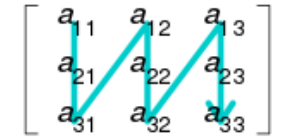
```

Formación

Row-major order



Column-major order



- El número total de elementos no puede cambiar.
- Utilice -1 para inferir la forma de las columnas.
- Row-major por defecto (MATLAB es column-major)

Transposición

```
# Transpuesta de una matriz
A = np.arange(25).reshape(5,5)
print("Matriz Original: \n" + str(A))
# Imprimir matriz transpuesta
print("Matriz Transpuesta: \n" + str(A.T))
# Imprimir matriz
print("Matriz Transpuesta: \n" + str(A.transpose((1,0))))
```

Matriz Original:

```
[[ 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]]
```

Matriz Transpuesta:

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

Matriz Transpuesta:

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



- `a.T` transpone los dos primeros ejes .
- `np.transpose` permuta los ejes.

2	4	-1
-10	5	11
18	-7	6



2	-10	-18
4	5	-7
-1	11	6

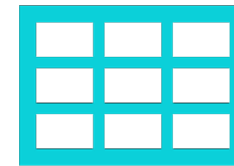
Operadores Matemáticos



Operaciones aritméticas
se realizan por elemento.



Un operador lógico
devuelve una matriz bool



Las operaciones in-situ
modifican la matriz

Operadores Matemáticos

- **Operaciones aritméticas se realizan por elemento.**
- Un operador lógico devuelve una matriz bool
- Las operaciones in-situ modifican la matriz

```
# Operadores Matemáticos
A = np.arange(100).reshape(10,10)
print("Primer Matriz: \n" + str(A))

B = np.arange(100, 200).reshape(10, 10)
print("Segunda Matriz: \n" + str(B))

# Calcular la suma de ambas matrices
print("Suma: " + str(A + B))

# Calcular la resta de ambas matrices
print("Suma: " + str(B - A))

# Calcular la multiplicación de ambas matrices
print("Suma: " + str(A * B))
```

```
Primer Matriz:
[[ 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 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

```
Segunda Matriz:
[[100 101 102 103 104 105 106 107 108 109]
 [110 111 112 113 114 115 116 117 118 119]
 [120 121 122 123 124 125 126 127 128 129]
 [130 131 132 133 134 135 136 137 138 139]
 [140 141 142 143 144 145 146 147 148 149]
 [150 151 152 153 154 155 156 157 158 159]
 [160 161 162 163 164 165 166 167 168 169]
 [170 171 172 173 174 175 176 177 178 179]
 [180 181 182 183 184 185 186 187 188 189]
 [190 191 192 193 194 195 196 197 198 199]]
```

Operadores Matemáticos

- Operaciones aritméticas se realizan por elemento.
- **Un operador lógico devuelve una matriz booleana**
- Las operaciones in-situ modifican la matriz

```
# Operadores lógicos
```

```
A = np.random.random((5,5))  
print("Matriz Original: \n" + str(A))
```

```
# Operador lógico  
print("Operador lógico a > 0.5 \n" + str(a>0.5))
```

Matriz Original:

```
[[0.0124458  0.75318949 0.51012785 0.14677449 0.38131653]  
 [0.63214844 0.49240062 0.72302018 0.5913991  0.34027848]  
 [0.52156944 0.39808869 0.46779844 0.03746587 0.8252981 ]  
 [0.8354298  0.7554512  0.53322169 0.09566736 0.7134824 ]  
 [0.6791975  0.23094997 0.0116702  0.10245596 0.39682071]]
```

Operador lógico a > 0.5

```
[[False  True False  True  True  True  True  True False  True]  
 [ True False  True False False  True  True  True  True False]  
 [ True  True  True False  True  True  True False False  True]  
 [ True False  True  True False False  True False False False]  
 [False  True False  True False  True  True False  True  True]  
 [ True False  True False False False  True False False  True]  
 [ True False  True  True False False False False False False]  
 [False False  True  True False  True False  True  True False]  
 [ True False  True False False  True False  True False False]  
 [False False False False  True  True  True False  True False]]
```

```
# Operaciones in-situ
A = np.random.randint(10, size= (5,5))
print("Matriz A Original: \n" + str(A))

B = A = np.random.randint(10, size= (5,5))
print("Matriz B Original: \n" + str(B))

# Operación in-situ
A += B
print("Matriz A Modificada: \n" + str(A))
```

Matriz A Original:

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

Matriz B Original:

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

Matriz A Modificada:

```
[[ 4 16  6 14  2]
 [ 6  2  2 14  0]
 [14 10  0 10  2]
 [ 4 18 10 18 10]
 [ 6  8 12 16  0]]
```

Operadores Matemáticos

- Operaciones aritméticas se realizan por elemento.
- Un operador lógico devuelve una matriz booleana
- **Las operaciones in-situ modifican la matriz**

Inversa de una Matriz

- Una matriz A es invertible si existe una matriz cuadrada tal que:

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$$

- La matriz inversa de A se denota como: A^{-1}

```
# Inversa de una matriz
import numpy as np

# Inicializar una matriz
A = np.random.randint(10, size= (5,5))
print("Matriz A Original: \n" + str(A))

# Calcular matriz inversa
B = np.linalg.inv(A)
print("Matriz Inversa A^-1: \n" + str(B))
```

Matriz A Original:

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

Matriz Inversa A⁻¹:

```
[[-0.0137931  0.08390805  0.06781609 -0.02873563 -0.04367816]
 [-0.04396552 -0.06587644  0.11199713 -0.00826149  0.08994253]
 [-0.09094828  0.17826868 -0.14658764  0.06052443 -0.00675287]
 [ 0.06810345  0.00237069 -0.06400862 -0.02478448  0.06982759]
 [ 0.11767241 -0.17417385  0.0360273  0.0784842 -0.05445402]]
```




Ejercicio 7

- Generar una matriz de 10×10 elementos e inicializarla con números aleatorios del 0 al 99.
1. Se deberá diseñar una función para calcular el cuadrado de la matriz.
 2. Se deberá diseñar una función para realizar la multiplicación de matrices.
 3. Se deberá diseñar una función para calcular la inversa de una matriz.

Funciones matemáticas universales

Elemento por elemento

Examples:

- np.exp
- np.sqrt
- np.sin
- np.cos
- np.isnan

```
# Funciones universales
A = np.random.randint(10, size= (5,5))
print("Matriz A Original: \n" + str(A))

# Calcular Raiz cuadrada
print("Raiz Cuadrada: \n" + str(np.sqrt(A)))
```

Matriz A Original:

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

Raiz Cuadrada:

```
[[0.          2.64575131 0.          0.          0.          ]
 [2.82842712  2.64575131 1.41421356 1.          2.          ]
 [1.73205081  2.64575131 2.44948974 3.          3.          ]
 [1.73205081  3.          1.41421356 1.73205081 2.          ]
 [1.73205081  2.64575131 1.41421356 1.73205081 2.23606798]]
```

Indexación

```
x[0,0]      # Elemento superior izquierdo  
x[0,-1]     # Primera fila, última columna  
x[0,:]      # Primera fila (many entries)  
x[:,0]      # Primera columna (many entries)
```

Notas:

- Indexación cero
- Los índices multidimensionales están separados por comas.

Subconjuntos en python

Sintaxis: inicio:fin:pasos

```
a = list(range(10))
```

```
a[:3] # indices 0, 1, 2
```

```
a[-3:] # indices 7, 8, 9
```

```
a[3:8:2] # indices 3, 5, 7
```

```
a[4:1:-1] # indices 4, 3, 2 (this one is tricky)
```