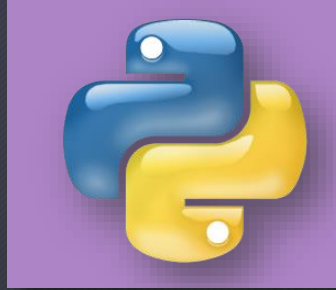


PYTHON - MATRICES

Lic. Mariela Asensio



En la notación del lenguaje Python una matriz es una lista cuyos elementos también son listas de igual longitud y con elementos del mismo tipo.



Se puede asociar una matriz a una representación de un cuadro con datos organizados en filas y columnas. Las filas son horizontales y las columnas verticales.

$$a = \begin{bmatrix} 23 & 45 & 37 \\ 72 & 81 & 91 \\ 58 & 64 & 37 \end{bmatrix}$$

Fila 0	→	00	01	02	03	04	05
Fila 1	→	10	11	12	13	14	15
Fila 2	→	20	21	22	23	24	25
Fila 3	→	30	31	32	33	34	35
Fila 4	→	40	41	42	43	44	45
		↑	↑	↑	↑	↑	↑
		Col 0	Col 1	Col 2	Col 3	Col 4	Col 5

Definición como una lista de dos niveles

```
a=[[23,45,63],[72,81,91],[56,64,37]]  
print(a)
```

```
[[23, 45, 63], [72, 81, 91], [56, 64, 37]]
```



Definición como un arreglo de dos dimensiones con la librería NumPy

```
from numpy import*  
a=array([[23,45,63],[72,81,91],[56,  
64,37]])  
print(a)
```

```
[[23 45 63]  
 [72 81 91]  
 [56 64 37]]
```

Los arreglos se visualizan mejor con su forma tabular.

Se puede especificar el tipo del arreglo: int, float, complex, bool

```
from numpy import*  
a=array([[23,45,63],[72,81,91],[56,64,37]],float)  
print(a)
```

```
[[23. 45. 63.]  
 [72. 81. 91.]  
 [56. 64. 37.]]
```

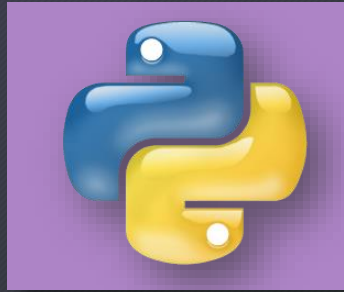
Si no se especifica el tipo, el tipo se toma del contenido.

En el ejemplo anterior la librería NumPy fue cargada con la instrucción:

```
from numpy import*
```

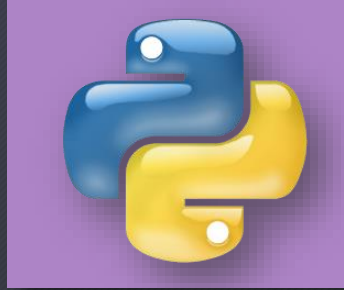
Un inconveniente con esta notación es que en algunos casos entra en conflicto con otras funciones del mismo nombre de la librería estándar de Python. Para evitar esto se prefiere cargar la librería asignándola a una identificación propia.

```
import numpy as np
```



Ahora la notación es:

```
import numpy as np
a=np.array([[23,45,63],[72,81,91],[56,64,37]],float)
print(a)
print(np.max(a))
```



```
[[23. 45. 63.]
 [72. 81. 91.]
 [56. 64. 37.]]
91.0
```

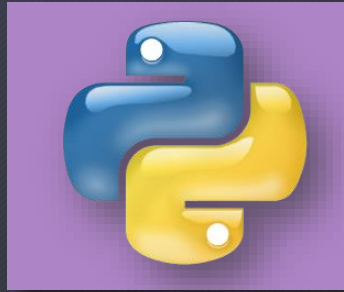
Para referirse a los componentes de una matriz se requieren dos índices y se puede usar la notación de índices separados por corchetes.

Nombre de la matriz[índice de fila][índice de columna]

O bien la notación matemática común

Nombre de la matriz[índice de fila , índice de columna]

Se pueden manejar filas, columnas o componentes individuales del arreglo. El uso de “slicing” (rebanar) para los índices permite referirse con eficiencia a porciones del arreglo.



```
import numpy as np
a=np.array([[23,45,63],[72,81,91],[56,64,37]],float)
print(a[1,:])
```

```
[72. 81. 91.]
```

Imprime la fila 1

```
print(a[:,1:2])
```

```
[[45.]
 [81.]
 [64.]]
```

Imprime la columna 1

```
import numpy as np
a=np.array([[23,45,63],[72,81,91],[56,64,37]],float)
print(a[:,1:])
```

```
[[45. 63.]
 [81. 91.]
 [64. 37.]]
```

Muestra desde la columna 1 hasta el final.

```
print(a[:,2,:])
```

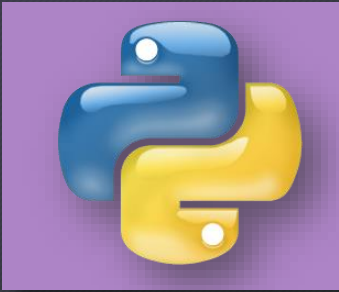
Muestra las filas pares

```
[[23. 45. 63.]
 [56. 64. 37.]]
```

```
print(a[1::2,:])
```

```
[[72. 81. 91.]]
```

Muestra las filas impares



Las formas de los índices permiten asignar porciones de las matrices



```
import numpy as np
a=np.array([[2,3,4,5],[8,3,2,1],[9,2,4,8]])
print(a)
```

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

```
a[1,:]=5
```

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

Asigna el valor 5 a todos los elementos de la fila 1


```
a[:,1]=7
```

Asigna el valor 7 en la columna 1

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

```
x=np.ndim(a)  
print(x)
```

Dimensión del arreglo

2

```
x=np.size(a)
```

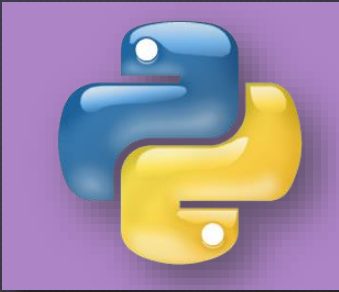
Cantidad total
de elementos
del arreglo

12

```
[i,j]=np.shape(a)  
print(i,j)
```

3 4

Cantidad de filas y
columnas de la matriz



La versión reducida:

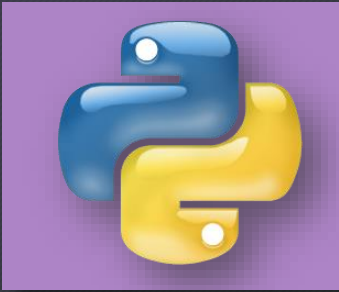
```
import numpy as np
a=np.array([[2,3,4,5],[8,3,2,1],[9,2,4,8]])
x= a.ndim
y= a.size
z = a.shape
print(x,y,z)
```

```
2 12 (3, 4)
```

```
import numpy as np
a=np.arange(12).reshape(3,4)
print(a)
```

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

Genera un arreglo bidimensional con
3 filas y 4 columnas



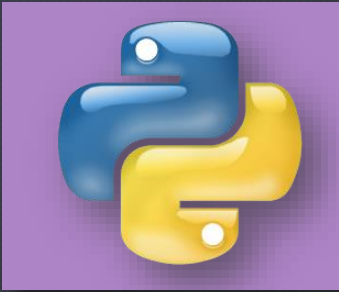

```
import numpy as np
a=np.array([4,7,8,3,5,9,2,4,6])
b=a.reshape(3,3)
print(b)
```

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

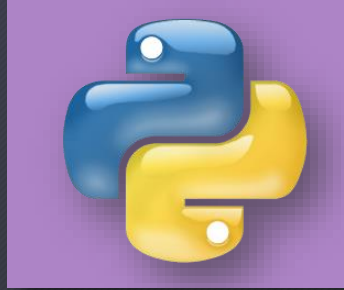
Una matriz se puede convertir en lista

```
x=b.tolist()
print(x)
```

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



Operador de pertenencia en arreglos bidimensionales



```
x = 8 in a  
print(x)
```

True

Las matrices pueden construirse mediante declaraciones implícitas dentro del programa

```
import numpy as np  
a=np.array([[i,2*i]for i in  
range(5)])  
print(a)
```

[0	0]
[1	2]
[2	4]
[3	6]
[4	8]

Genera una matriz de 2 columnas y 5 filas

Generación de arreglos con números aleatorios

```
import numpy as np
a=np.random.choice(range(0,10),[4,5])
print(a)
```

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

Genera una matriz de 4 filas y 5 columnas con valores aleatorios de una cifra



Operaciones con matrices



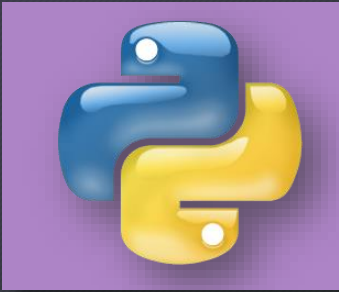
Suma de matrices

```
import numpy as np
a=np.array([[2,3],[4,5]])
b=np.array([[5,2],[1,4]])
c=a+b
print(c)
```

```
[[7 5]
 [5 9]]
```


Resta de Matrices

```
import numpy as np
a=np.array([[2,3],[4,5]])
b=np.array([[5,2],[1,4]])
c=a-b
print(c)
```

$$\begin{bmatrix} -3 & 1 \\ 3 & 1 \end{bmatrix}$$


Multiplicación (elemento con elemento)

```
import numpy as np
a=np.array([[2,3],[4,5]])
b=np.array([[5,2],[1,4]])
c=a*b
print(c)
```

$$\begin{bmatrix} 10 & 6 \\ 4 & 20 \end{bmatrix}$$

Multiplicación de matrices

```
import numpy as np
a=np.array([[2,3],[4,5]])
b=np.array([[5,2],[1,4]])
c=np.dot(a,b)
print(c)
```

```
[[13 16]
 [25 28]]
```

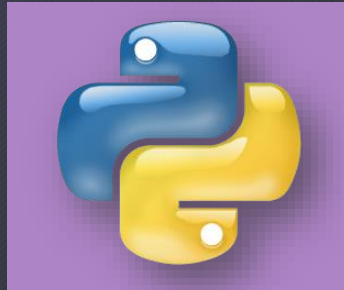
Modo reducido

```
import numpy as np
a=np.array([[2,3],[4,5]])
b=np.array([[5,2],[1,4]])
c=a.dot(b)
print(c)
```

$$A \cdot B = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} =$$

$$= \begin{pmatrix} 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 & 2 \cdot 0 + 0 \cdot 2 + 1 \cdot 1 & 2 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 \\ 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 & 3 \cdot 0 + 0 \cdot 2 + 0 \cdot 1 & 3 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 \\ 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 & 5 \cdot 0 + 1 \cdot 2 + 1 \cdot 1 & 5 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 \end{pmatrix} =$$

$$= \begin{pmatrix} 3 & 1 & 2 \\ 3 & 0 & 3 \\ 7 & 3 & 6 \end{pmatrix}$$



```
[[13 16]
 [25 28]]
```


Funciones matemáticas para matrices



Suma todos los elementos de la matriz

```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
c=np.sum(a)
print(c)
```

45

```
c=np.prod(a)
c=np.mean(a)
c=np.max(a)
c=np.min(a)
```

Producto de todos los elementos

Media aritmética de todos los elementos

El mayor valor del arreglo

El menor valor del arreglo



```
np.argmax(a)  
np.argmin(min)
```

Indica el índice con el mayor y menor valor del arreglo

```
import numpy as np  
a=np.array([[4,2,5],[2,8,4],[6,9,5]])  
b=np.sort(a)  
print(b)
```

Ordena las filas de la matriz

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


Matrices especiales



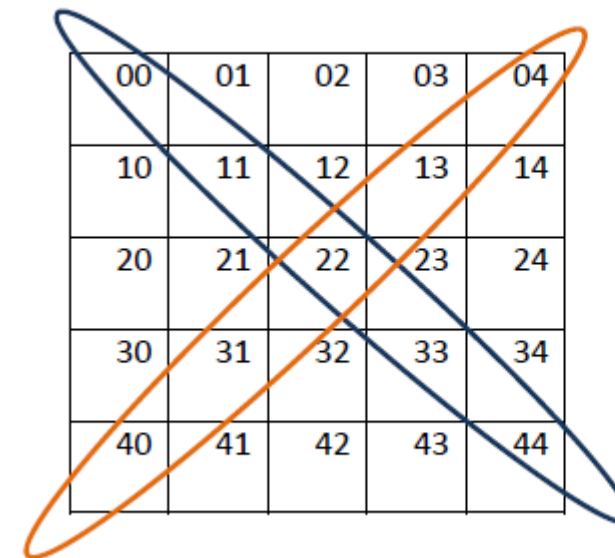
Muestra la diagonal principal de la matriz

```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
b=np.diag(a)
print(b)
```

```
[4 8 5]
```

Suma la diagonal

```
b=np.trace(a)
print(b)
17
```

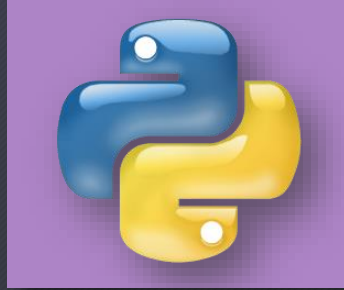


Diagonal Secundaria

Diagonal Principal

Matriz traspuesta

Dada una matriz A, se llama matriz traspuesta de A a la matriz que se obtiene cambiando ordenadamente las filas por las columnas



$$A = \begin{bmatrix} 4 & 2 & 5 \\ 2 & 8 & 4 \\ 6 & 9 & 5 \end{bmatrix} \quad A^t = \begin{bmatrix} 4 & 2 & 6 \\ 2 & 8 & 9 \\ 5 & 4 & 5 \end{bmatrix}$$

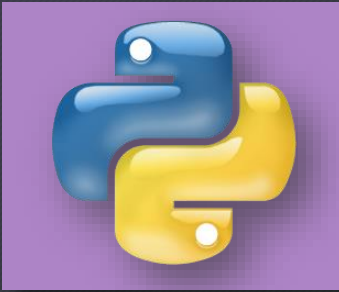
```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]
])
b=np.transpose(a)
print(b)
```

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

`b=a.T` Versión reducida para la traspuesta

Matriz triangular superior

En una **matriz triangular superior** los elementos situados por debajo de la diagonal principal son ceros.



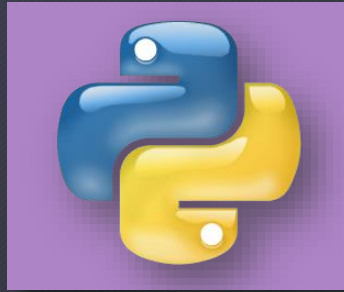
5	7	9
0	1	4
0	0	3

```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
b=np.triu(a)
print(b)
```

```
[[4 2 5]
 [0 8 4]
 [0 0 5]]
```

Matriz triangular inferior

En una **matriz triangular inferior** los elementos situados por encima de la diagonal principal son ceros.



5	0	0
2	1	0
9	8	3

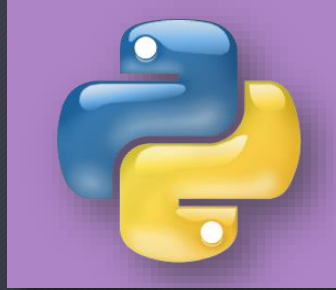
```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
b=np.tril(a)
print(b)
```

```
[[4 0 0]
 [2 8 0]
 [6 9 5]]
```

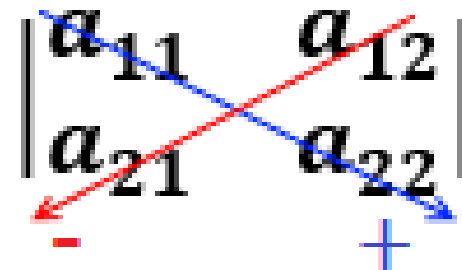
Cálculo del determinante

La matriz cuadrada de dimensión 2 tiene la forma

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

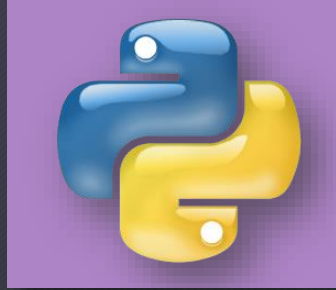


Para calcular el determinante restan el producto de los elementos de las diagonales:

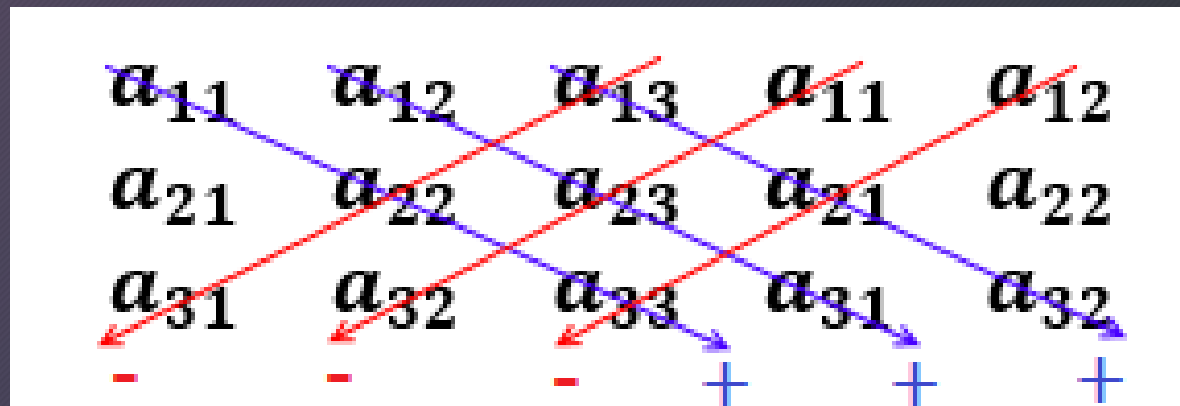
A diagram showing a 2x2 matrix with elements a_{11} , a_{12} , a_{21} , and a_{22} . A blue arrow points from a_{11} to a_{22} (the main diagonal), and a red arrow points from a_{12} to a_{21} (the anti-diagonal). Below the a_{21} position is a red minus sign, and below the a_{22} position is a blue plus sign.
$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}$$

La matriz cuadrada de dimensión 3 tiene la forma

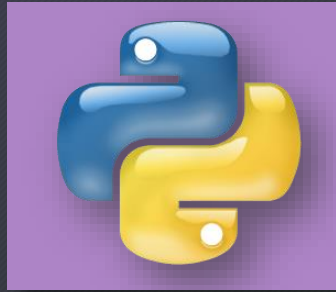
$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$



Calculamos el determinante mediante la llamada **regla de Sarrus**. Una forma de aplicar la regla de Sarrus es escribir las tres columnas de la matriz seguidas de la primer y la segunda columna:



Los elementos de las diagonales con flecha hacia abajo (azul) se multiplican y se suman; los de las otras diagonales (rojo) se multiplican y se restan:

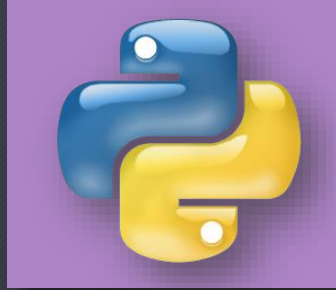


$$\begin{aligned} |A| &= \\ &= a_{11} \cdot a_{22} \cdot a_{33} \\ &\quad + a_{12} \cdot a_{23} \cdot a_{31} \\ &\quad + a_{13} \cdot a_{21} \cdot a_{32} \\ &\quad - a_{13} \cdot a_{22} \cdot a_{31} \\ &\quad - a_{11} \cdot a_{23} \cdot a_{32} \\ &\quad - a_{12} \cdot a_{21} \cdot a_{33} \end{aligned}$$

$$A = \begin{array}{|c|c|c|} \hline 4 & 2 & 5 \\ \hline 2 & 8 & 4 \\ \hline 6 & 9 & 5 \\ \hline \end{array}$$

$$\begin{aligned} \det(A) &= 4 \cdot 8 \cdot 5 \\ &\quad + 2 \cdot 4 \cdot 6 \\ &\quad + 2 \cdot 9 \cdot 5 \\ &\quad - 5 \cdot 8 \cdot 6 \\ &\quad - 2 \cdot 2 \cdot 5 \\ &\quad - 4 \cdot 9 \cdot 4 \\ &= -106 \end{aligned}$$

```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
b=np.linalg.det(a)
print(b)
```



-105.999999999999997

Cálculo de la matriz inversa

Sea A una matriz cuadrada de dimensión n (es decir, $n \times n$), entonces se dice que es **regular** o **invertible** si su determinante es distinto de 0.

Para toda matriz A de dimensión n existe una única matriz B de la misma dimensión tal que $A \cdot B = B \cdot A = I_n$

Es decir, la matriz B es el inverso multiplicativo de A . Como la matriz B es única, la denominamos **matriz inversa de A** y la representamos por A^{-1} .

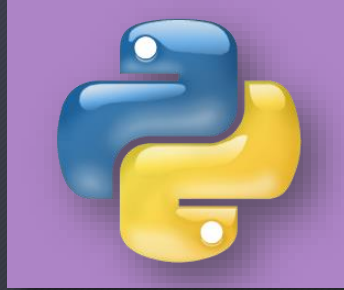
Existen varios procedimientos para calcular la matriz inversa, uno consiste en utilizar la siguiente fórmula:

$$A^{-1} = \frac{(\text{Adj}(A))^T}{|A|}$$

es la matriz traspuesta de la matriz adjunta de A

Determinante de A


```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
b=np.linalg.inv(a)
print(b)
```



```
[[ -0.03773585  -0.33018868   0.30188679]
 [ -0.13207547   0.09433962   0.05660377]
 [  0.28301887   0.22641509  -0.26415094]]
```

Función para comparar matrices

```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
b=np.array([[4,2,5],[2,8,4],[6,9,5]])
c=np.array_equal(a,b)
print(c)
```

True

Salida de arreglos con control de decimales

```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
b=np.linalg.inv(a)
print(b)
np.set_printoptions(precision=4)
print(b)
x=np.around(b,3)
print(x)
```

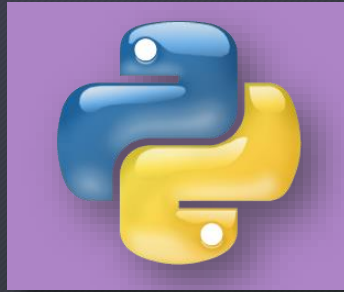
```
[[ -0.03773585  -0.33018868   0.30188679]
 [ -0.13207547   0.09433962   0.05660377]
 [  0.28301887   0.22641509  -0.26415094]]
```

```
[[ -0.0377  -0.3302   0.3019]
 [ -0.1321   0.0943   0.0566]
 [  0.283    0.2264  -0.2642]]
```

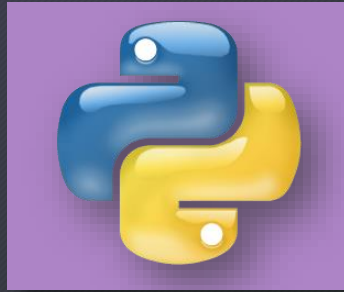
```
[[ -0.038  -0.33   0.302]
 [ -0.132   0.094   0.057]
 [  0.283   0.226  -0.264]]
```

Controlamos la cantidad de decimales
al imprimir el arreglo

Redondeo de decimales en arreglos



La función where es muy útil para sustituir valores de matrices



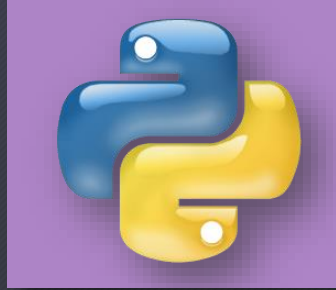
```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
b=np.where(a>4,0,9)
print(b)
```

```
[[9 9 0]
 [9 0 9]
 [0 0 0]]
```

```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
b=np.where(a%2==0,0,1)
print(b)
```

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


En muchas funciones de NumPy se puede especificar mediante la opción **axis** si el recorrido es por columnas axis=0 o por filas axis=1

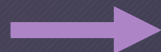


```
import numpy as np
a=np.array([[4,5,6],[2,8,4],[2,5,9]])
print(a)
f=np.sum(a,axis=1)
print(f)
c=np.sum(a,axis=0)
print(c)
```

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

```
[15 14 16]
```

```
[ 8 18 19]
```

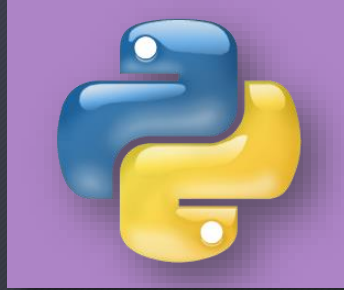


Suma de las filas



Suma de las columnas

```
import numpy as np
a=np.array([[4,5,6],[2,8,4],[2,5,9]])
print(a)
mm=np.amax(a)
print("El mayor de la matriz",mm)
mc=np.amax(a,axis=0)
print("El mayor de cada columna",mc)
mf=np.amax(a,axis=1)
print("El mayor de cada fila",mf)
mem=np.amin(a)
print("El menor de la matriz",mem)
mec=np.amin(a,0)
print("El menor de cada columna",mec)
mef=np.amin(a,1)
print("El menor de cada fila",mef)
```



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

El mayor de la matriz 9

El mayor de cada columna [4 8 9]

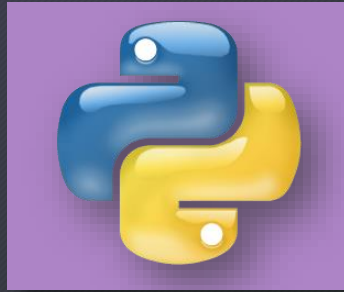
El mayor de cada fila [6 8 9]

El menor de la matriz 2

El menor de cada columna [2 5 4]

El mayor de cada fila [4 2 2]

Existen funciones especiales de NumPy que nos permiten modificar la estructura de las matrices de manera dinámica, es decir, durante la ejecución del programa.



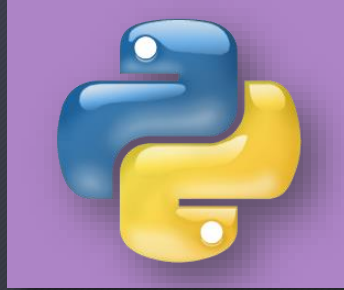
```
import numpy as np
a=np.array([[10,20,30],[40,50,60],[70,80,90]])
print(a)
e=np.insert(a,1,[1,2,3],axis=0)
print(e)
```

```
[[10 20 30]
 [40 50 60]
 [70 80 90]]
```

```
[[10 20 30]
 [ 1  2  3]
 [40 50 60]
 [70 80 90]]
```

Inserta una nueva fila en la posición 1


```
import numpy as np
a=np.array([[10,20,30],[40,50,60],[70,80,90]])
print(a)
e=np.insert(a,1,[1,2,3],axis=1)
print(e)
```

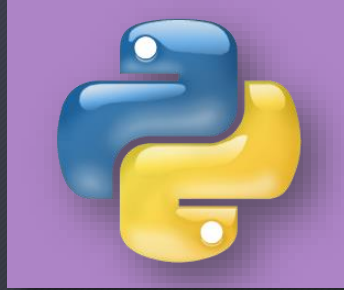


```
[[10 20 30]
 [40 50 60]
 [70 80 90]]
```

```
[[10  1 20 30]
 [40  2 50 60]
 [70  3 80 90]]
```

Inserta una nueva columna en la posición 1

```
import numpy as np
a=np.array([[10,20,30],[40,50,60],[70,80,90]])
print(a)
e=np.delete(a,0,axis=1)
print(e)
```



```
[[10 20 30]
 [40 50 60]
 [70 80 90]]
```

Elimina la columna 0

```
[[20 30]
 [50 60]
 [80 90]]
```

Del mismo modo con la instrucción `e=np.delete(a,0,axis=0)`, elimina la fila 0

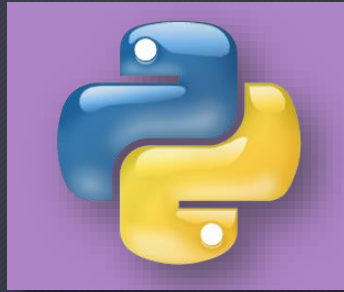
Concatenación de matrices

```
import numpy as np
a=np.array([[10,20],[30,40]])
print(a)
b=np.array([[50,60],[70,80]])
print(b)
c=np.concatenate([a,b])
print(c)
```

```
[[10 20]
 [30 40]]
```

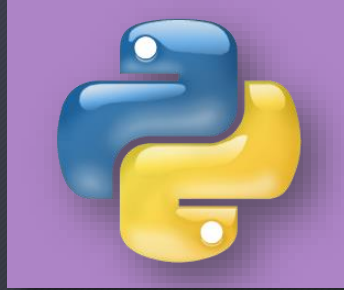
```
[[50 60]
 [70 80]]
```

```
[[10 20]
 [30 40]
 [50 60]
 [70 80]]
```



Si en lugar del código anterior escribimos

```
c=np.concatenate([a,b],axis=1)
```



Obtenemos:

```
[[10 20 50 60]  
 [30 40 70 80]]
```

También podemos escribir:

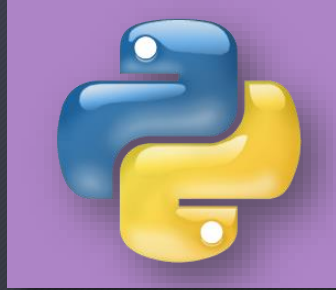
```
c=np.concatenate([a,b],axis=0)
```

Y volvemos a obtener

```
[[10 20]  
 [30 40]  
 [50 60]  
 [70 80]]
```

Almacenamiento y recuperación de arreglos en disco

La función `savetxt('f.txt',c)` almacena un arreglo de contenido numérico en un archivo de texto llamado `f.txt`



```
import numpy as np
a=np.array([[4,2,5],[2,8,4],[6,9,5]])
np.savetxt('ejemplo3.txt',a)
```

`np.loadtxt('f.txt')` recupera un arreglo almacenado en un archivo de texto en el disco

Ejemplo



Suponga un cuadro con las calificaciones de 4 estudiantes en 3 materias

	Álgebra	Física	Química
Carmen	7	6	9
Juan	9	9	7
María	7	8	6
Pedro	7	8	9

Almacenar los datos en arreglos, mostrar el promedio general de calificaciones, mostrar la mayor nota de cada estudiante, mostrar el promedio de cada materia, mostrar de cada estudiante la menor calificación y el nombre de la respectiva materia.

Ejemplo 2



Escribir un programa que permita ingresar por teclado una matriz y muestre un arreglo conteniendo las sumas de las filas de la matriz.

```
import numpy as np
n = int(input('cantidad de filas: '))
m = int(input('cantidad de columnas: '))
a = np.zeros([n,m],dtype=int) #iniciamos la matriz
for i in range(n):
    for j in range(m):
        a[i,j]=int(input('elemento: '))
filas = np.zeros(n,int)
for i in range(n):
    filas[i]=sum(a[i,:]) #suma de las filas
print(a)
print('suma de las filas\n',filas)
```

Tuplas



Una tupla es una colección de datos que pueden tener diferente tipo. Los datos se escriben entre paréntesis, separados por comas.

'abc'	73	5.28	'rs'	5
-------	----	------	------	---

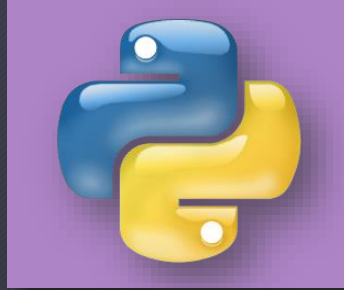
Los componentes de una tupla no se pueden modificar después de haber sido creados. Las celdas son enumeradas desde el cero.

Definición de una tupla:

```
>>> x= 'abc', 45, 8.5, 'sr', 75
>>> x
('abc', 45, 8.5, 'sr', 75)
```


Una tupla se puede convertir a lista

```
>>> x = list(x)
>>> x
['abc', 45, 8.5, 'sr', 75]
```



Una lista se puede convertir en tupla

```
>>> x= tuple(x)
>>> x
('abc', 45, 8.5, 'sr', 75)
```

Los valores de una tupla se pueden desempacar asignándole variables

```
>>> a,b,c,d,e = x      >>> b
>>> a                  45
'abc'                  >>> e
                        75
```

ZIP una función para formar tuplas entre dos listas



```
>>> materia=['Cálculo I','Cálculo II','Química General']
>>> codigo=['001','002','003']
>>> t=zip(codigo,materia)
>>> list(t)
[('001', 'Cálculo I'), ('002', 'Cálculo II'), ('003', 'Química General')]
```

A los elementos de una tupla se los puede sumar, calcular el mayor y el menor e incluso eliminarlos.

Conjuntos



Los conjuntos se construyen como una lista de valores encerrados entre llaves. Por definición, los componentes de un conjunto no están ordenados ni contienen elementos repetidos.

Definición de un conjunto:

```
>>> u={1,3,8,5,2}
>>> u
{1, 2, 3, 5, 8}
```

También se pueden definir con la instrucción set(c)

```
>>> r=set([1,3,6,7,9,8,5])
>>> r
{1, 3, 5, 6, 7, 8, 9}
```

Operación con conjuntos:

El resultado de la operación entre dos conjuntos es un objetos que no se puede indexar. Pero se lo puede convertir nuevamente a un objeto indexable.



Sean: **a**, **b**: conjuntos

Símbolo	Operación	Resultado	Función equivalente
a b	Unión de conjuntos	Conjunto	a.union(b)
a & b	Intersección de conjuntos	Conjunto	a.intersection(b)
a - b	Diferencia de conjuntos	Conjunto	a.difference(b)
a ^ b	Diferencia simétrica de conjunts	Conjunto	a.symmetric_difference(b)
a < b	Subconjunto	Valor lógico	a.issubset(b)

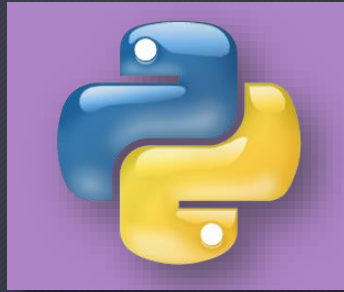

```
>>> a={3,4,9,6,7}
>>> b={4,6,2,9,7,6}
>>> c=a|b
>>> c
{2, 3, 4, 6, 7, 9}
>>> d=a.intersection(b)
>>> d
{9, 4, 6, 7}
```

Propiedad de pertenencia

```
>>> 5 in a
True
>>> 7 not in a
False
```

Conversión de un
conjunto a una lista

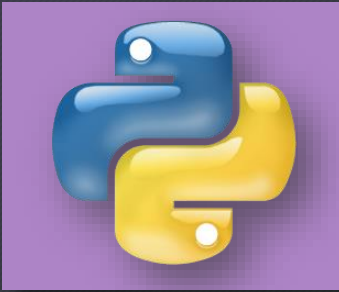
```
>>> l=list(a)
>>> l
[1, 3, 5, 7, 9]
```



Cadenas de caracteres convertidas a conjuntos

```
>>> x='adivinanza'  
>>> c=set(x)  
>>> c  
{'i', 'v', 'n', 'd', 'a', 'z'}
```

Sin repeticiones y los elementos no mantienen el orden



Uso de conjuntos para encontrar caracteres comunes

```
>>> a='programa'  
>>> b='panorama'  
>>> c=set(a)&set(b)  
>>> c  
{'p', 'm', 'r', 'a', 'o'}
```