

INTRODUCCIÓN A PYTHON

INTRODUCCIÓN

2026

- ▶ Estructuras de control
- ▶ Listas
- ▶ Funciones integradas
 - Matemáticas
 - Numpy
 - Arreglos bidimensionales (Matrices)



```
self.FidValue = OrderedDict(sorted(self.items(), key=lambda item: item[0]))  
#Read item in dictionary  
for key, value in item.FidValue.items():  
    typeOfFID = mapFidType[key]  
    if (typeOfFID == "DATE"):  
        d = datetime.datetime.strptime(str(value), "%Y-%m-%d")  
        dataCal = datetime.date.strptime(str(value), "%Y-%m-%d")  
        FidAndValue = FidAndValue + value  
    else: FidAndValue = FidAndValue + value
```

```
try:  
    start = date(int(self.start_year.get(self.months.index(self.start_month)),  
                int(self.start_day.get(self.months.index(self.start_month))),  
                int(self.start_year.get(self.months.index(self.start_month))))  
  
    end = date(int(self.end_year.get(self.months.index(self.end_month)),  
              int(self.end_day.get(self.months.index(self.end_month))),  
              int(self.end_year.get(self.months.index(self.end_month))))
```



Al igual que con listas, se puede modificar o recuperar el elemento en una posición usando la notación arreglo[posición]

- Para obtener el tamaño de un arreglo podemos usar `len(arreglo)`, tal como con listas, o el atributo llamado `size`
- Para recorrer un arreglo, podemos utilizar los mismos métodos que con listas:

Python

```
import numpy
a = numpy.array([3, 1, 2])
a[2] = 10

for i in range(0,a.size,1):
    print(a[i],end=" ") #muestra por pantalla: 3 1 10
```



Sin embargo, lo realmente entretenido de numpy son los cálculos sobre los arreglos

`arreglo*num`

retorna un arreglo donde todos los elementos son multiplicados por num (pueden usar /, +, -)

`arreglo.min()`

retorna el valor mínimo de un arreglo

`arreglo.round(decimales)`

retorna un arreglo con todos los elementos redondeados a la cantidad de decimales pasados

`arreglo.sum()`

retorna la suma de los elementos de un arreglo

`arreglo.mean()`

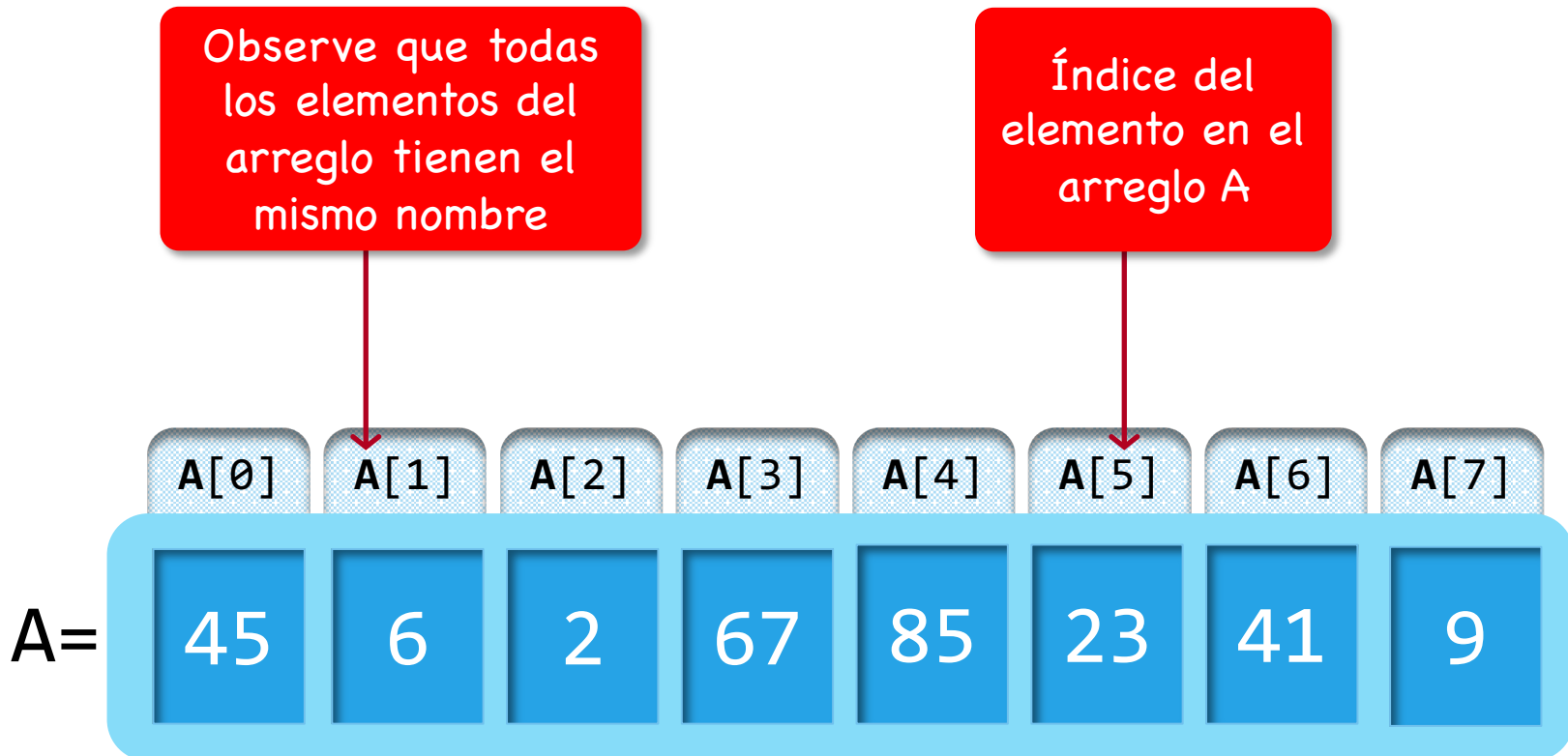
retorna el promedio de los elementos de un arreglo

`arreglo.prod()`

retorna la multiplicación de los elementos de un arreglo



En resumen, gráficamente, un arreglo unidimensional, se puede ver gráficamente:





Aunque los arreglos puedan ayudarnos a resolver varios problemas, el que sean de una sola dimensión reduce su aplicabilidad. Para resolver este problema existen arreglos multidimensionales.

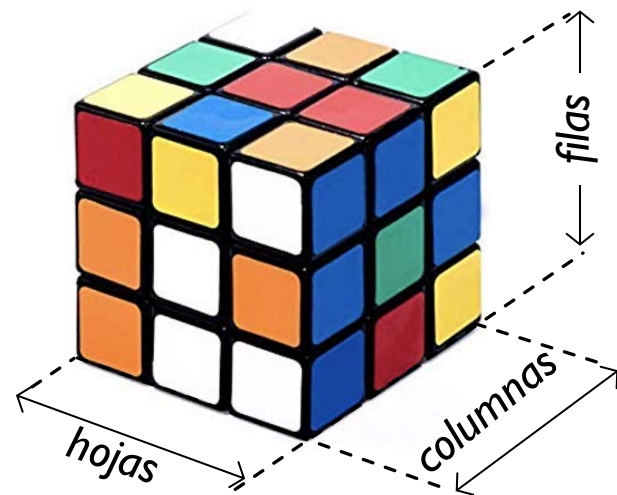
arreglo bidimensional (Matriz)

columns

4	74	3
66	43	2
46	-45	0

filas

arreglo tridimensional





Una matriz (arreglo bidimensional) tiene un número determinado de filas y columnas, los que son determinados en el momento de su creación.

- ▶ Su creación y manejo es similar a la creación unidimensional, pero con dos índices.
- ▶ Para acceder al dato i, j de la matriz llamada `xs`, escribimos `xs[i, j]`, donde i hace referencia a la fila de la matriz y j hace referencia a la columna.

```
import numpy

# creamos una matriz de 3x3 ceros
xs = numpy.zeros([3,3])
xs[0,0] = 10
...
xs[2,2] = -33

#imprime el valor 4
print(xs[0,2])
```

j - columnas

10	74	3
66	43	2
46	-45	-33

i - filas

**arreglo bidimensional
(Matriz)**



Al igual que en el caso unidimensional, existen funciones y operaciones que nos permiten manipular estas matrices de forma más sencilla.

- ▶ Para obtener el número de elementos de la matriz podemos utilizar el atributo `size`. Sin embargo, el número de elementos por dimensión de la matriz está dado por el atributo `shape`, el que retorna un “arreglo” con el número de filas y columnas.

```
import numpy

# creamos una matriz de 3x3 ceros
xs = numpy.zeros([3,7])
for i in range(0,xs.shape[0],1):
    for j in range(0,xs.shape[1],1):
        xs[i,j]=i*j
print(xs)
```

Resultado:

```
[[ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  2.  3.  4.  5.  6.]
 [ 0.  2.  4.  6.  8. 10. 12.]
```



Cree un código que genere una matriz de 40x3 representando las notas de controles, primera y segunda pruebas de los alumnos del curso. Llene esta matriz con números aleatorios entre 1 y 7. Finalmente, cree una nueva matriz unidimensional de tamaño 40, con los promedios simples de cada alumno y muéstrela por pantalla.

```
import numpy
notas = numpy.random.random([40,3])*6+1
prom = numpy.zeros([40])

for i in range(0,notas.shape[0]):
    suma = 0
    for j in range(0,notas.shape[1]):
        suma = suma + notas[i,j]
    prom[i] = suma/3

print("Los promedios son ", prom.round(2))
```



en el atributo shape podemos determinar las dimensiones de la matriz



Cree un código que genere una matriz de 40x3 representando las notas de controles, primera y segunda pruebas de los alumnos del curso. Llene esta matriz con números aleatorios entre 1 y 7. Finalmente, cree una nueva matriz unidimensional de tamaño 40, con los promedios simples de cada alumno y muéstrela por pantalla.

```
import numpy
notas = numpy.random.random([40,3])*6+1
prom = numpy.zeros([40])

for i in range(0,notas.shape[0]):
    prom[i] = notas[i,:].mean()

print("Los promedios son ", prom.round(2))
```



Como se observa en la última solución, los cálculos se pueden ejecutar sobre la matriz entera, una fila o columna. En los últimos dos casos es necesario el operador :

```
matriz*num
```

retorna una matriz donde todos los elementos son multiplicados por num (puede usar /, +, -)

```
matriz[i,:]*num
```

retorna un arreglo correspondiente a la i-ésima fila multiplicada por num

```
matriz[:,i]*num
```

retorna un arreglo correspondiente a la i-ésima columna multiplicada por num

```
matriz.min()
```

retorna el valor mínimo de la matriz

```
matriz[i,:].min()
```

retorna el valor mínimo de la i-ésima fila

```
matriz[:,i].min()
```

retorna el valor mínimo de la i-ésima columna



En palabras sencillas, el operador `[i,:]` permite obtener/modificar todos los elementos de la i -ésima fila.

En forma similar, el operador `[:, i]` permite obtener/modificar todos los elementos de la i -ésima columna.

Ejemplo:

```
import numpy
ejemplo = numpy.zeros([3,4])
ejemplo[:,2] = 1
print(ejemplo)
```

Resultado:

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



Escriba un programa en Python que solicite al usuario **num** (un número entero), genere una matriz cuadrada de tamaño **num** con la tabla de multiplicar y la muestre por pantalla. Ejemplo:

Ingrese un número: 5

```
[[ 1.  2.  3.  4.  5.]  
 [ 2.  4.  6.  8. 10.]  
 [ 3.  6.  9. 12. 15.]  
 [ 4.  8. 12. 16. 20.]  
 [ 5. 10. 15. 20. 25.]]
```

```
import numpy  
  
num = int(input("Ingresa un número: "))  
tabla=numpy.zeros([num,num])  
  
for i in range(1,num+1):  
    for j in range(1,num+1):  
        tabla[i-1,j-1] = i*j  
  
print(tabla)
```



Escriba un programa que cree una matriz de 10x10 posiciones con contenido numérico aleatorio entero entre 1 y 99 y luego determine si el número 35 se encuentra dentro de dicha matriz.

Si el número 35 se encuentra en la matriz, debe imprimir por consola indicando todas las posiciones de la matriz donde se encontró el número 35. Si el número 35 no está en la matriz, debe imprimir por consola un mensaje indicando que el número 35 no se encuentra en la matriz.

Use la función: `numpy.random.randint(a,b,[dim1,dim2])`

```
import numpy
numeros = numpy.random.randint(1,99,[10,10])
flag = 0
for i in range(0,numeros.shape[0],1):
    for j in range(0,numeros.shape[1],1):
        if numeros[i,j]==35:
            flag = 1
            print("#35 en posicion ",i," ",j)

if flag==0:
    print("el numero no fue encontrado")
```