



INTRODUCCIÓN A PYTHON BIBLIOTECAS Y NUMPY (1)

Background designed by kjpargeter / Freepik

Miguel Carrasco
Junio 2019

- ▶ Estructuras de control
- ▶ Listas
- ▶ Funciones integradas
 - Matemáticas



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



Hasta el momento hemos utilizado funciones básicas en nuestros programas:

```
print("texto ",var)
```



función que recibe múltiples parámetros y lo despliega por pantalla, no retorna ningún valor.

```
int(var o valor)
```



función que recibe un valor, y retorna el mismo valor transformado en un entero

```
input("texto")
```



función que recibe un parámetro, lo despliega por pantalla, recibe un valor desde el teclado y retorna un texto.



Sin embargo, existen miles de módulos en Python disponibles para ser usados por programadores en sus programas.

Para hacer uso de una función de la biblioteca, se llama a la biblioteca y la función correspondiente

Importando módulos

```
import <nombreMódulo>  
nombreMódulo.nombreFunción(<parámetros>)
```



Dentro de los módulos más útiles en Python podemos encontrar:

- math para operaciones matemáticas
- random para generar números aleatorios
- matplotlib para graficar
- numpy y pandas para análisis de datos
- Y muchos más



Veamos algunas funciones importantes del módulo **math**

math.ceil(x) : retorna el entero más pequeño que sea igual o mayor que x

math.exp(x) : retorna el valor exponencial de x

math.fabs(x) : retorna el valor absoluto de x

math.factorial(x) : retorna el factorial de x

math.floor(x) : retorna el entero más grande que sea igual o menor que x

math.log(x,baseLog) : retorna el logaritmo de x en base baseLog

math.pow(x,y) : retorna x elevado y

math.sqrt(x) : retorna la raíz cuadrada de x



Veamos algunas funciones importantes del módulo **time**

- time.sleep(x)** : el programa se pausa por x segundos
- time.time()** : devuelve el número de segundos desde el 1 de Enero de 1970 (año que se utiliza como valor inicial en Unix)
- time.ctime(x)** : convierte el tiempo expresado en segundos en el formato de la fecha actual: Día de la semana, Mes, día, hora, año.



Veamos algunas funciones importantes de la biblioteca **random**

- random.randint (x,y)** : retorna un número aleatorio entero entre x e y.
- random.random ()** : retorna un número aleatorio entre 0 y 1.
- random.uniform (x,y)** : retorna un número aleatorio uniforme entre x e y.

▼ Recordemos

Ya hemos visto en Python el concepto de listas

- 
- Una gran “cajonera” donde podemos almacenar distintos valores
 - Podemos acceder a los valores usando su posición en la lista
 - Una lista puede contener distintos tipos de datos: texto, números, booleanos, entre otros

Python

```
xs = [3, 1, 2]                      # crea una lista
print(xs, xs[2])                    # muestra en pantalla [3, 1, 2] 2
xs[2] = 'hola'                      # listas pueden tener distintos tipos de datos
print(xs)                           # muestra en pantalla [3, 1, 'hola']
xs.append('chao')                   # agrega un elemento al final de la lista
print(xs)                           # muestra en pantalla [3, 1, 'hola', 'chao']
```



Realice un programa que cree una lista con 10 elementos generados al azar y muestre en pantalla el mayor de ellos



Realice un programa que cree una lista con 10 elementos generados al azar y muestre en pantalla el mayor de ellos

saved

share

run



main.py

```
1 import random
2
3 lista = []
4 for i in range(10):
5     lista.append(random.random())
6
7 mayor = 0
8 for i in lista:
9     if i > mayor:
10         mayor = i
11
12 print('El mayor elemento es', mayor)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
```

▶

```
El mayor elemento es 0.919579252100356
```

▶



Problema

Al ser una lista algo tan flexible, se generaliza su uso.

- No puedo realizar las mismas acciones sobre todos los elementos pues algunos pueden ser de distinto tipo
- No puedo ir a una ubicación específica en dos dimensiones en forma inmediata, pues no existe el concepto de tamaño fijo de una lista.

Entre otras cosas

NumPy



- ▶ Estructuras de control
- ▶ Listas
- ▶ Funciones integradas
 - Matemáticas
 - Numpy



```
#Read item in dictionary
for key, value in item.FidValue.items():
    typeOfFID = mapFidType(key)
    if(typeOfFID == "DATE"):
        d = datetime.datetime.strptime(value, "%Y-%m-%d")
        dataCal = datetime.date(d.year, d.month, d.day)
        FidAndValue = FidAndValue + dataCal
    else:FidAndValue = FidAndValue + value
```

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



Numpy

Es un módulo en Python que agrega soporte para arreglos en una o múltiples dimensiones (como matrices), y una gran colección de operaciones matemáticas para operar sobre estos arreglos.

Python

```
import numpy # importamos la biblioteca NumPy

xs = numpy.array([3, 1, 2]) # creamos un arreglo de números
print(xs) # muestra en pantalla [3 1 2]
```



▼ ¿Notaron que

... el arreglo en NumPy no separa los valores con una coma al imprimirllo en pantalla?

array()

Los arreglos son el equivalente a una lista en Python, en el sentido que almacena un conjunto de valores. Se crean usando la función array de numpy.

The screenshot shows a repl.it interface. On the left, there's a pink sidebar with the repl.it logo. The main area has tabs for 'saved' and 'share'. Below that is a code editor with a file named 'main.py' containing the following code:

```
1 import numpy
2
3 primer_arreglo = numpy.array([3, 2, 1])
4 segundo_arreglo = numpy.array([3, 'hola', 2])
5 tercer_arreglo = numpy.array([3.2, 2, 1])
6
7 print(primer_arreglo)
8 print(segundo_arreglo)
9 print(tercer_arreglo)
```

To the right is a terminal window showing the execution of the script. The output is:

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
> [3 2 1]
['3' 'hola' '2']
[3.2 2. 1.]
```

El primer arreglo contiene solamente números, sin embargo el segundo, al tener un texto, convierte todos los elementos a texto, y el tercero a decimales



Importante

Los arreglos NO permite mezclar distintos tipos de datos, y solo acepta valores de un mismo tipo convirtiendo a un tipo común.

¿Para qué sirve?



Realice un programa que cree una lista con 10 elementos generados al azar y muestre en pantalla el mayor de ellos

saved

share

run



main.py

```
1 import random
2
3 lista = []
4 for i in range(10):
5     lista.append(random.random())
6
7 mayor = 0
8 for i in lista:
9     if i > mayor:
10         mayor = i
11
12 print('El mayor elemento es', mayor)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
El mayor elemento es 0.919579252100356
>
```

¿Para qué sirve?



Realice un programa que cree **un arreglo** con 10 elementos generados al azar y muestre en pantalla el mayor de ellos

The screenshot shows a Jupyter Notebook interface with the following components:

- Toolbar:** Includes "saved", "share", "run", and other standard notebook controls.
- Code Cell:** Labeled "main.py", containing the following Python code:

```
1 import numpy
2
3 arreglo = numpy.random.random(10,)
4 mayor = numpy.max(arreglo)
5 print('El mayor elemento es', mayor)
```
- Output Cell:** Displays the output of the run command:

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
> El mayor elemento es 0.8914038534619878
>
```



Numpy provee muchas alternativas para crear arreglos:

numpy.zeros(dimensión): Crea un arreglo de tamaño **dimensión** y le asigna a cada elemento valor **0**

```
a = numpy.zeros(5,)  
print(a) # imprime [0. 0. 0. 0. 0.]
```

numpy.ones(dimensión): Crea un arreglo de tamaño **dimensión** y le asigna a cada elemento valor **1**

```
b = numpy.ones((2,))  
print(b)
```

numpy.full(dimensión, valor): Crea un arreglo de tamaño **dimensión** y le asigna a cada elemento valor **valor**

```
c = numpy.full((2,), 7)  
print(c) # muestra [7 7]
```

numpy.random.random(dimensión):
Crea un arreglo de tamaño **dimensión** y le asigna a cada elemento valor **random**

```
d = numpy.random.random(5,)  
print(d)
```



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

Ejemplo:

```
import numpy

a = numpy.random.random(5,)
print(a[3])
```



Para agregar elementos a un arreglo tenemos la función append de la biblioteca Numpy

```
repl.it saved share run Python 3.6.1 (default, [GCC 4.8.2] on linux
main.py
1 import numpy
2
3 a = numpy.array([1,2,3,4,5,6])
4 print(a)
5
6 b = numpy.append(a, [7])
7 print(a)
8 print(b)

Noten que el o los
elementos a agregar deben
ser listas o arreglos
```

[1 2 3 4 5 6]
[1 2 3 4 5 6]
[1 2 3 4 5 6 7]



Importante

Los arreglos NO se actualizan cuando usamos append, sino que se **crea una copia del arreglo con el o los elementos agregados.**



Para obtener el tamaño de un arreglo podemos usar **len(arreglo)**, tal como con listas, o el atributo llamado **size**

The screenshot shows a repl.it interface. On the left, there's a pink sidebar with a downward arrow icon labeled "repl.it". The main area has tabs for "saved", "share", "run", and other file operations. A code editor window titled "main.py" contains the following Python code:

```
1 import numpy
2
3 a = numpy.random.random(5, )
4
5 print(a.size, 'vs', len(a))
6
```

To the right is a terminal window showing the execution results:

```
Python 3.6.1 (default
[GCC 4.8.2] on linux
.
5 vs 5
> |
```



Para recorrer un arreglo, podemos utilizar los mismos métodos que con listas:

The screenshot shows a repl.it interface with a Python script named `main.py`. The code imports numpy and creates an array `a` with values [1, 2, 4, 8, 16, 32]. It then uses two loops to print each element: a standard `for` loop over the array and a `for` loop over a range from 0 to the array's size.

```
repl.it saved share run Python 3.6.1 (default, [GCC 4.8.2] on linux)
main.py
1 import numpy
2
3 a = numpy.array([1,2,4,8,16,32])
4 for i in a:
5     print(i)
6
7 for i in range(0, a.size):
8     print(a[i])
9
```

The output window shows the printed values: 1, 2, 4, 8, 16, 32.

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



Ejemplo

saved

share

run



main.py

```
1 import numpy
2
3 a = numpy.array([1,2,4,8,16,32])
4 print(a * 4)
5
6 print('La suma de los elementos de a es', a.sum())
7 print(' y su multiplicación', a.prod())
8
9 print('El promedio de los elementos de a es', a.mean())
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
```

```
[ 4   8  16  32  64 128]
La suma de los elementos de a es 63
y su multiplicación 32768
El promedio de los elementos de a es 10.5
```

▶



Si los arreglos tienen igual tamaño entonces puedo realizar operaciones de suma, multiplicación, resta, división

The screenshot shows a repl.it interface with a pink sidebar on the left containing the logo and a downward arrow icon. The main area has tabs for 'saved' and 'share' with a 'run' button. On the right, there's a download icon, a cloud icon, a file icon, and a cube icon. The code editor contains 'main.py' with the following content:

```
1 import numpy
2
3 a = numpy.array([1,2,4,8,16,32])
4 b = numpy.array([0,1,1,2,3,5])
5
6 print(a+b)
7 print(a*b)
```

The output window on the right shows the results of running the code in Python 3.6.1:

```
Python 3.6.1 (default, Dec 20 2017, 14:40:14) [GCC 4.8.2] on linux
> [ 1  3  5 10 19 37]
[  0   2   4  16  48 160]
>
```



Incluso podemos usar los arreglos como valores de verdad usando un par de funciones:

- ▶ **arreglo.all():** retorna verdadero si todos los elementos de un arreglo se evaluan como verdadero
- ▶ **arreglo.any():** retorna verdadero si alguno de los elementos de un arreglo se evaluan como verdadero

The screenshot shows a repl.it interface with a pink sidebar on the left containing the text "repl.it". The main area displays a Python script named "main.py" with the following code:

```
1 import numpy
2
3 a = numpy.array([False, False, False, True, False])
4 b = numpy.array([1,1,1,1,1,1])
5
6 print(a.any())
7 print(a.all())
8
9 print(b.any())
10 print(b.all())
11
```

Below the code, there are several buttons: "saved", "share", "run", and download/upload/copy/more icons. To the right, the terminal window shows the execution results:

```
Python 3.6.1 (default
[GCC 4.8.2] on linux
>
True
False
True
True
> █
```



Un científico norteamericano estaba haciendo estudios de la altura de las personas en nuestro país, y guardó sus datos en un arreglo numpy llamado alturas. Sin embargo, utilizó pulgadas como medida (maldito sistema métrico inglés).

Escriba el código para transformar esas mediciones en centímetros





Escriba un programa en Python que, para un **array** con las notas de un curso, calcule el promedio de notas del curso, y extraiga la nota menor y mayor, para luego mostrar estas tres cosas por pantalla

array de notas

[3.4, 4.3, 4.7, 5.1, 5.3, 5.8, 6.1, 6.7, 7.0]



Un científico norteamericano estaba haciendo estudios de la altura de las personas en nuestro país, y guardó sus datos en un arreglo numpy llamado alturas. Sin embargo, utilizó pulgadas como medida (maldito sistema métrico inglés).

Escriba el código para transformar esas mediciones en centímetros

```
import numpy

alturas = numpy.array([59.2, 73., 45.9, 65.7, 80.1])
en_centimetros = alturas * 2.54

print(en_centimetros)
```



Escriba un programa en Python que, para un **arreglo** con las notas de un curso, calcule el promedio de notas del curso, y extraiga la nota menor y mayor, para luego mostrar estas tres cosas por pantalla

```
import numpy

notas = numpy.array([3.4, 4.3, 4.7, 5.1, 5.3, 5.8, 6.1, 6.7, 7.0])

promedio = notas.mean()
menor = notas.min()
mayor = notas.max()

print('El promedio es', promedio)
print('La nota más baja es', menor)
print('La nota más alta es', mayor)
```