Tutorial de NumPy en Español

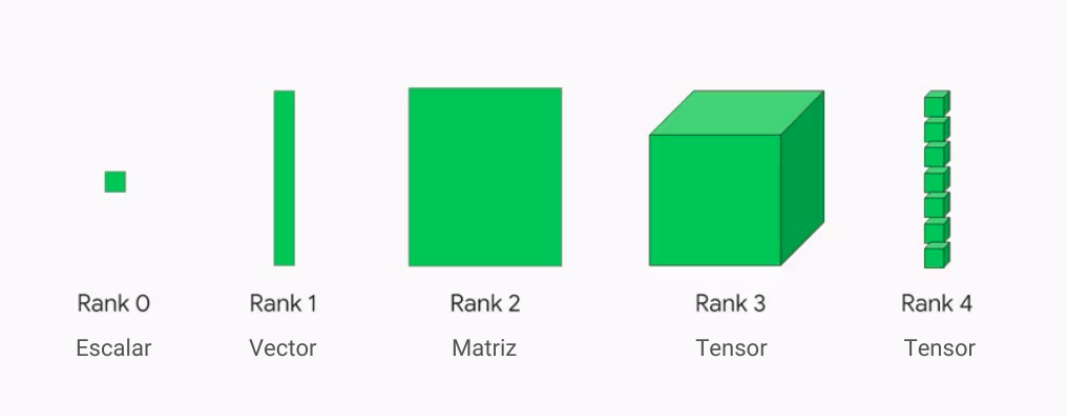
Este tutorial va de la mano con un blog que publiqué en Platzi: Tutorial de NumPy

Se lo importa con la siguiente convención:

```
import numpy as np
```

Arrays de NumPy

- 1D: Vector
- 2D: Matriz
- 3D+: Tensor



¿Cómo se ven en código? *(Ignora cómo lo construí, más adelante te enseñaré a hacerlo)*

```
import numpy as np

vector = np.arange(5)
vector
```

```
array([0, 1, 2, 3, 4])
```

```
matrix = np.arange(9).reshape(3, 3)
matrix
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

```
tensor = np.arange(12).reshape(3, 2, 2)
tensor
```

```
array([[[ 0,  1],
        [ 2,  3]],

       [[ 4,  5],
        [ 6,  7]],

       [[ 8,  9],
        [10, 11]]])
```

Propiedades de los arrays

- Deben tener un único tipo de dato
- Siempre del mismo tamaño "rectangular"

Correcto ✅

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

Tamaño: (3, 3); **tipo**: enteros

Incorrecto ❌

```
[1.7 2.5 5]
[0.0 '3.5']
[7.3 2.3 4 1.1]
```

Tamaño: (???), **tipo**: float, int & str (?)

¿Cómo crear un array en NumPy?

La forma más directa de crearlo es con `np.array(tu_lista)`. Y al igual que las listas, puedes acceder a sus valores por medio de sus índices.

```
my_first_vector = np.array([2, 5, 6, 23])
print(my_first_vector)
```

```
[ 2  5  6 23]
```

```
my_first_matrix = np.array([[2, 4,], [6, 8]])
print(my_first_matrix)
```

```
[[2 4]
 [6 8]]
```

```
my_list = [0, 1, 2, 3, 4]
print(np.array(my_list))
```

```
[0 1 2 3 4]
```

Secuencias

También puedes crear rápidamente un array con secuencias de números:

`np.arange()`

Funciona parecido a `range` en Python, solo que en lugar de regresar un generador, retorna un *ndarray*. Sus argumentos principales son: `start`, `stop` y `step`. Con lo que puedes dar un rango de valores y cada cuánto quieres que aparezcan.

Toma en cuenta que `start` es inclusivo y `stop` es exclusivo. Igual que al hacer slicing en una lista.

```
print(np.arange(start=2, stop=10, step=2))
```

```
[2 4 6 8]
```

```
print(np.arange(11, 1, -2))
```

```
[11  9  7  5  3]
```

`np.linspace()`

Con `np.arange()` puedes decir cada cuánto genere los elementos y con ello te dará un tamaño de array. En cambio con `np.linspace()` puedes decir el tamaño del array y los *steps* se calcularán automáticamente. Y aquí tanto `start` como `stop` son inclusivos.

```
print(np.linspace(start=11, stop=12, num=18))
```

```
[11.         11.05882353 11.11764706 11.17647059 11.23529412 11.29411765
 11.35294118 11.41176471 11.47058824 11.52941176 11.58823529 11.64705882
 11.70588235 11.76470588 11.82352941 11.88235294 11.94117647 12.        ]
```

```
print(np.linspace(0, 1, 11))
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

Arreglos vacíos y predefinidos

Es posible que necesites crear arreglos "vacíos" o con valores pre-definidos.

`np.zeros()` y `np.ones()`

```
print(np.zeros(4))
```

```
print(np.zeros((2, 2)))
```

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

```
print(np.ones(6))
```

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

`np.full()`

Crea un array con un valor en específico. Tiene 2 argumentos principales: `shape` que tiene que ser pasado como una tupla con las dimensiones y `fill_value` con el valor que queramos.

```
print(np.full(shape=(2, 2), fill_value=5))
```

```
[[5 5]
 [5 5]]
```

```
print(np.full((2, 3, 4), 0.55))
```

```
[[[0.55 0.55 0.55 0.55]
  [0.55 0.55 0.55 0.55]
  [0.55 0.55 0.55 0.55]]

 [[0.55 0.55 0.55 0.55]
  [0.55 0.55 0.55 0.55]
  [0.55 0.55 0.55 0.55]]]
```

`np.full_like()`

Sirve si ya tienes un array y quieres tomarlo como base para crear otro con el mismo tamaño, pero con un mismo valor.

```
base = np.linspace(2, 6, 4)
print(np.full_like(base, np.pi))
```

```
[3.14159265 3.14159265 3.14159265 3.14159265]
```

Arreglos aleatorios

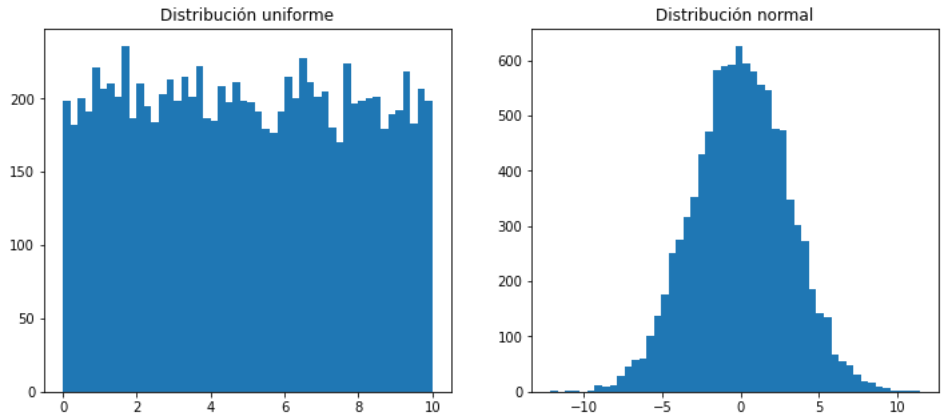
```
import matplotlib.pyplot as plt

uniform = np.random.uniform(0, 10, 10000)
normal = np.random.normal(0, 3, 10000)

plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
plt.hist(uniform, bins=50)
plt.title('Distribución uniforme')

plt.subplot(1, 2, 2)
plt.hist(normal, bins=50)
plt.title('Distribución normal')

plt.show()
```



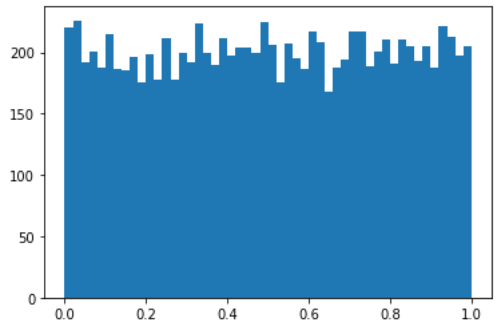
`.rand()` y `.uniform()`

- `.rand()`: números aleatorios en una distribución uniforme. Permite crear ndarrays.
- `.uniform()`: igual que `rand`, pero permite ingresar los límites de la muestra.

```
print(np.random.rand(2, 2))
```

```
[[0.62740202 0.11171536]
 [0.47526728 0.19739417]]
```

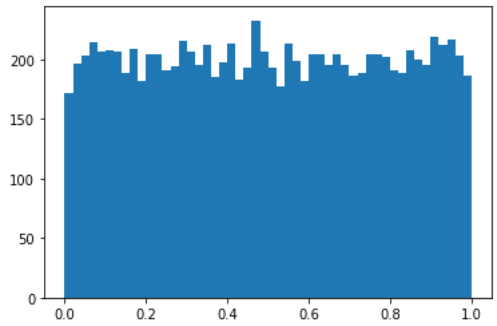
```
rand = np.random.rand(10000)
plt.hist(rand, bins=50)
plt.show()
```




```
print(np.random.uniform(low=0, high=1, size=6))
```

```
[0.7878737  0.3431897  0.77765595 0.60943181 0.30961326 0.60167083]
```

```
uniform = np.random.uniform(low=0, high=1, size=10000)
plt.hist(uniform, bins=50)
plt.show()
```



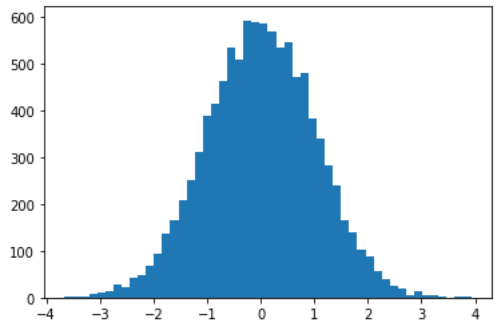
`.randn()` y `.normal()`

- `.randn()`: números aleatorios en una distribución "normal estándar". Permite crear ndarrays.
- `.normal()`: igual que `randn`, pero permite escalar los límites de la muestra.

```
print(np.random.randn(2, 2))
```

```
[[ 0.91140011  1.72792052]
 [-0.84028707 -0.27378577]]
```

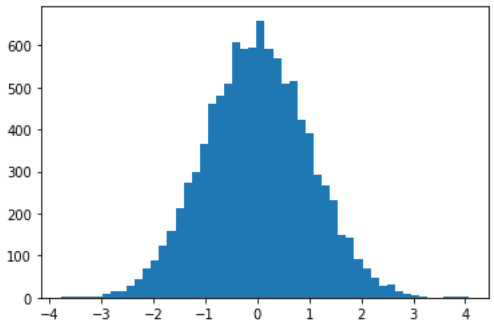
```
normal = np.random.randn(10000)
plt.hist(normal, bins=50)
plt.show()
```




```
print(np.random.normal(loc=0, scale=2, size=6))
```

```
[-2.36743682 -3.12673482 -1.14254395 -3.19805542 -1.11930443 -2.70161226]
```

```
normal2 = np.random.normal(0, 1, 10000)
plt.hist(normal2, bins=50)
plt.show()
```



`.randint()`

Números enteros aleatorios entre un rango dado.

```
print(np.random.randint(low=0, high=10, size=(3, 3)))
```

```
[[1 0 5]
 [5 5 3]
 [7 5 4]]
```

```
print(np.random.randint(1,100,10))
```

```
[61 55  8 95 93 89 27 24  1 38]
```

Tamaño de los arrays

`.reshape()`

Redimensiona un array, es decir, te permite crear matrices o tensores a partir de vectores y viceversa.
Para un array de 2 dimensiones, el primer valor son las filas y el segundo las columnas.

```
a = np.arange(1,10)
B = np.reshape(a, [3,3])
print(B)
```

También se lo puede usar como método:

```
C = np.arange(1, 9).reshape(2, 2, 2)
print(C)
```

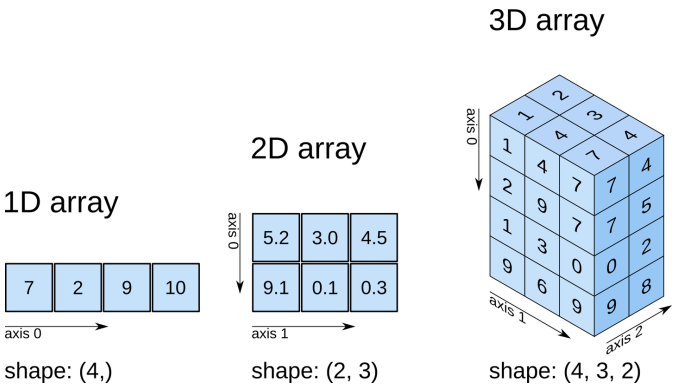
```
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

`.shape`

Devuelve las dimensiones del array.

```
print(B.shape)
```



`.dtype`

Devuelve el tipo de dato de un array.

```
print(B.dtype)
```

```
int64
```

Manipulando arrays

De manera muy parecida a las listas, los ndarrays permiten hacer slicing con `[]`. Sus índices empiezan en `0`.

```
matrix_cool = np.arange(9).reshape(3, 3)
print(matrix_cool)
```

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

```
print(matrix_cool[1, 2])
```

```
5
```

```
print(matrix_cool[0, :])
```

```
[0 1 2]
```

```
print(matrix_cool[:, 1])
```

```
[1 4 7]
```

```
print(matrix_cool[:, 1:])
```

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

```
print(matrix_cool[0:2, 0:2])
```

```
print(matrix_cool[:, :])
```

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

Copiar un array de NumPy

Debes usar `array1.copy()` o nuevamente `np.array(array1)`

❌ Incorrecto

```
a1 = np.array([2, 4, 6])
a2 = a1
a1[0] = 8
print(a1)
print(a2)
```

```
[8 4 6]
[8 4 6]
```

✅ Correcto

```
a1 = np.array([2, 4, 6])
a2 = a1.copy()
a1[0] = 8
print(a1)
print(a2)
```

```
[8 4 6]
[2 4 6]
```

Funciones matemáticas

Operaciones básicas

```
# Suma
A = np.arange(5, 11)
print(A)
print(A + 10)
```

```
[ 5  6  7  8  9 10]
[15 16 17 18 19 20]
```

```
# Resta
B = np.full(4, 3)
C = np.ones(4, dtype='int')
print(B)
print(C)
print(B - C)
```

```
[3 3 3 3]
[1 1 1 1]
[2 2 2 2]
```

```
# Multiplicación y división
print(A * 10)
print(A / 10)
```

```
[ 50  60  70  80  90 100]
[0.5 0.6 0.7 0.8 0.9 1. ]
```

Estadística

```
height_list = [74, 74, 72, 72, 73, 69, 69, 71, 76, 71, 73, 73, 74, 74, 69, 70, 73, 75, 78, 79, 76, 74
print(np.mean(height_list))
print(np.median(height_list))
print(np.std(height_list))
```

```
73.1923076923077
73.0
2.572326554954764
```

Mínimos y máximos

```
print(np.max(height_list))
print(np.min(height_list))
```

```
79
69
```

Ejercicios

Base de datos

```
import pandas as pd

df = pd.read_csv('Baseball_Players.csv')
height = df['Height(inches)'].to_numpy(dtype='int64')
weight = df['Weight(pounds)'].to_numpy(dtype='int64')
```

```
print(height)
height.shape
```

```
[74 74 72 ... 75 75 73]
```

```
(1034,)
```

```
print(weight)
weight.shape
```

```
[180 215 210 ... 205 190 195]
```

```
(1034,)
```

Retos

Deja en los comentarios del blog tus respuestas y código para cada uno de estos ejercicios:

1. Crea un nuevo array en el que transformes las unidades de las alturas a metros *(pista: multiplica por 0.0254)*.
2. Crea un nuevo array en el que transformes las unidades de los pesos a kilogramos *(pista: multiplica por 0.453592)*.
3. Ahora vas a crear un nuevo array en el que calcules el IMC (Índice de Masa Corporal) de los jugadores a partir de los 2 vectores que creaste *(pista: la fórmula es `peso / altura²`)*.
4. ¿Cuál es el máximo IMC? ¿Y el mínimo?
5. ¿Cuál es la media y la mediana del IMC? ¿Los valores son cercanos o hay un sesgo en los datos?
6. ¿Qué hay de la desviación estándar?
7. ¿Cuál es el peso y altura del jugador #734? ¿Es posible unificar los vectores en una matriz para obtener este resultado?
8. Esto último no lo vimos en el tutorial, así que será un reto para ti. Vas a intentar filtrar los datos para obtener cuántos jugadores tienen un IMC por debajo de 21 *(pista: es posible hacerlo en una línea)*.

```
# Hora de resolver los ejercicios
```

Regalos 🎁

En este punto seguramente estés pensando: *"Rayos, esas fueron demasiadas funciones y argumentos... Y aún me quedan muchas por ver 😱 Estaría genial tener una cheet sheet..."*

Bueno, tus deseos son órdenes 🎖 No te daré una, sino dos:

1. NumPy Cheat Sheet
2. NumPy Cheat Sheet — Python for Data Science

Esto contó como 2 deseos, así que te queda solo 1. Asumiré que quieres poner en práctica lo que aprendiste, los ejercicios que te dejé fueron demasiado sencillos para ti, así que...

1. 100 NumPy excercises

Mis regalos están en inglés 😭

-> **Da clic aquí y sé feliz**

Eso fue largo... Me encantaría leer tus respuestas y comentarios en el blog 💚