

1. En esta práctica usted ejercitará y explorará algunas características del poderoso módulo `Numpy`. Para esto, cargue el módulo usando:

```
import numpy as np
```

y (si no asistió a la clase de esta semana) vea el primero de los videos introductorios disponibles en Canvas [aquí](#).

2. En `Numpy` existe una función llamada `arange` que es muy similar a la ya conocida función `range`. La diferencia es que `range` sirve para generar una *lista* mientras que `arange` genera un *arreglo de Numpy*. Para comprobar esto, ejecute:

```
x = range(10)
y = np.arange(10)
print(x, type(x))
print(y, type(y))
```

En otras palabras, `np.arange(10)` es equivalente a `np.array(range(10))`.

3. Usando arreglos de `Numpy` es posible realizar muchos cálculos en forma rápida y eficiente, sin necesidad de recurrir a ciclos (`for` o `while`). Por ejemplo, puede calcular la misma suma considerada en el problema 3 de la guía 11, es decir,

$$1 + 2 + 3 + 4 + \cdots + 999 + 1000, \quad (1)$$

pero ahora usando las función `sum` de `Numpy` (que suma todos los elementos de un arreglo):

```
n = np.arange(1001)
suma = np.sum(n)
print(suma)
```

o, en una sola línea

```
print(np.sum(np.arange(1001)))
```

Verifique lo anterior y asegúrese de entender qué se está calculando.

4. Adapte la idea del cálculo en el punto anterior para implementar un cálculo alternativo para el factorial de un número n (entero positivo), pero esta vez usando un arreglo de `Numpy` y la función `prod()` que calcula el producto de cada componente de un arreglo de `Numpy` (similarmente a como `sum()` calcula la suma).
5. Usando `Numpy`, calcule el valor de la suma de los primeros 101 términos de la forma

$$1 + \left(\frac{1}{2}\right) + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \cdots + \left(\frac{1}{2}\right)^{100}. \quad (2)$$

6. Verifique que, a diferencia de su pariente `range()`, la función `arange()` también funciona con pasos decimales, por ejemplo

```
print(np.arange(1,10,0.3))
```

7. Otra función muy útil para crear arreglos de valores en un intervalo es `linspace()`, que tiene el formato `linspace(desde,hasta,numerodeelementos)`. Por ejemplo, ejecute los siguientes comandos:

```
x = np.linspace(1,10,20)
y = np.linspace(-np.pi,np.pi,100)
print(x,np.size(x))
print(y,np.size(y))
```

8. Otra propiedad importante de los arreglos es que sus elementos pueden usarse para iterar en un ciclo `for`. Para ver esto, ejecute:

```
x = np.arange(11)
y = x**2
for i in x:
    print ("la componente "+str(i)+" de y es igual a "+str(y[i]))
```

9. Vea el segundo video disponible en Canvas [aquí](#), en el que se explican los aspectos básicos de los arreglos bidimensionales en Numpy.
10. Lea sobre el comandos `shape` y `len` y `size` y sobre el *indexado de arreglos* (tanto uni- como bi-dimensionales) en el [archivo sobre Numpy](#) en el repositorio. Asegúrese de entender los ejemplos ahí discutidos.
11. Descargue el archivo de datos `datos.txt` y guárdelo en la carpeta donde está trabajando. El módulo Numpy contiene una función llamada `genfromtxt`, que lee datos desde un archivo y los asigna a un arreglo, de la dimensión apropiada. Ejecute (en la misma carpeta donde está el archivo `datos.txt`) los siguientes comandos:

```
d = np.genfromtxt("datos.txt")
x = d[:,0]
y = d[:,1]
```

La primera línea carga los datos al arreglo `d`. Las últimas dos líneas asignan la primera columna de datos al arreglo `x` y la segunda columna a `y`. Usando las funciones `shape` y `size` de Numpy, verifique la forma y tamaño de los arreglos `d`, `x` e `y`. Asegúrese de entender qué es lo que realiza exactamente cada comando anterior.

12. Usando lo anterior, calcule e imprima:
- (a) El promedio de los valores de la primera columna. (puede usar la función `sum` y `len` para calcular el promedio, o bien la función `mean` de Numpy).

- (b) El promedio *de los cuadrados* de los valores de la segunda columna.
- (c) La suma de los productos de cada elemento de la primera con la segunda columna (es decir, $0.1 * 0.738 + 0.25 * 0.826 + 0.41 * 0.981 + \dots$).