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

```
import numpy as np
```

2. En `Numpy` existe una función llamada `arange` que es muy similar a la ya conocida función `range`. La diferencia es que `range` genera 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,

$$1 + 2 + 3 + 4 + \cdots + 999 + 1000, \tag{1}$$

   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

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

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. 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.

10. 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.

11. 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, $0.1 * 0.738 + 0.25 * 0.826 + 0.41 * 0.981 + \cdots$).

12. Escriba y ejecute el siguiente programa, que hace uso de `Numpy` y del módulo gráfico `Matplotlib`

```
import matplotlib.pyplot as plt
import numpy as np

d = np.genfromtxt("datos.txt")
x = d[:,0]
y = d[:,1]
plt.plot(x,y, marker="o", markersize=5, color="green",
        label="Datos experimentales")
plt.title("Voltaje versus Frecuencia")
plt.xlabel("Frecuencia $f$ [Hertz]")
plt.ylabel("Voltaje $V$ [Volt]")
plt.legend()
plt.savefig("g1.pdf")
```

    Este programa grafica los datos en las listas `x` e `y` usando círculos verdes, que guarda en el archivo `g1.pdf`.

13. Copie el archivo `g1.py` a `g2.py`, que en adelante usará para realizar pruebas.

14. La opción `marker="o"` indica que los puntos son representados por círculos. Note que, por defecto, estos puntos son unidos por rectas. Otros símbolos ("markers") disponibles son listados en la tabla 1. Por ejemplo, la opción `marker="s"` indica al comando `plot` que grafique cuadrados. Además, la opción `color` puede adoptar los valores `blue` (b), `green` (g), `red` (r), `cyan` (c), `magenta` (m), `yellow` (y), `black` (k) y `white` (w). Puede encontrar más colores listados aquí. Cambie los colores y símbolos del grafico en `g2.py` para familiarizarse con estas opciones.

15. Agregue una grilla (malla) a su gráfico usando el comando `plt.grid(True)` antes de `np.savefig`, y vea qué efecto tiene esto sobre el gráfico creado.

16. Cambie los límites del gráfico agregando los comandos

```
plt.xlim(0,90)
plt.ylim(0,15)
```

    y vea el cambio que produce.

17. Exporte el gráfico anterior directamente a formato `.png`, simplemente cambiando `plt.savefig("g2.pdf")` por `plt.savefig("g2.png")` en su programa `g2.py`. Comparta su lindo gráfico personalizado subiendo el archivo `p2.png` al foro de Teams de la práctica.

`"."`	point
`","`	pixel
`"o"`	circle
`"v"`	`triangle_down`
`"^"`	`triangle_up`
`"<"`	`triangle_left`
`">"`	`triangle_right`
`"1"`	`tri_down`
`"2"`	`tri_up`
`"3"`	`tri_left`
`"4"`	`tri_right`
`"8"`	octagon
`"s"`	square
`"p"`	pentagon
`"*"`	star
`"h"`	hexagon1
`"H"`	hexagon2
`"+"`	plus
`"x"`	x
`"D"`	diamond
`"d"`	`thin_diamond`

Cuadro 1: Algunos símbolos disponibles para gráficar puntos con el comando `plot`. Ver este link para más detalles y símbolos.