



UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES
DEPARTAMENTO DE COMPUTACIÓN Y SIMULACIÓN DE SISTEMAS
TÓPICOS ESPECIALES ITALLER 1



Facilitador: Tomás J. Concepción Miranda

Integrantes:

Gustavo Colucci 8-951-2191

Luis Mejia 8-949-350

Greg Torres, 8-956-675

Objetivos

Aprender las funciones y métodos fundamentales de la librería NumPy

Indicaciones

Se debe realizar un informe en el que se detalle, para cada problema, el desarrollo de la solución. Este trabajo se puede realizar en grupos de 2 o 3 personas. **Envíe su informe en Moodle en formato PDF**, así como el **código fuente**, en el bloque correspondiente **antes de las 11:55 p.m. del día siguiente del taller**.

Rúbrica

Este laboratorio tendrá una puntuación total de 100, donde la evaluación se basada en los aspectos de excelente, bueno, regular, deficiente. Los puntos que se evaluará en la rúbrica se muestran en la tabla:

N.	Aspectos a evaluar				
1	CONTENIDO DE ACUERDO A LO SOLICITADO EN EL ENUNCIADO	Excelente (80)	Bueno (70)	Regular (50)	Deficiente (0)
	Resultados debidamente presentados (capturas de pantalla, explicación) - (80 puntos)				
2	PANTALLA DE PRESENTACIÓN - (10 puntos)	Excelente (10)	Bueno (8)	Regular (5)	Deficiente (0)
	Cumple con todos los parámetros dados en clase				
3	ENTREGA DE TRABAJO EN LA PLATAFORMA MOODLE - (10 puntos)	Excelente (10)	Bueno (8)	Regular (5)	Deficiente (0)
	Entrega a tiempo en la plataforma				

Enunciados

Información General

NumPy es una librería para trabajar con datos numéricos en Python. Esta tiene un objeto “array” que permite varias operaciones con arreglos que resultan convenientes a la hora de trabajar con este tipo de dato. Estos arrays y el API de NumPy son utilizados extensivamente en otras librerías como Pandas, Matplotlib y scikit-learn. Para instalar NumPy, basta con lanzar el comando `pip install numpy`. La documentación oficial se encuentra disponible en numpy.org/doc/stable

Crear arrays

El array es la estructura de datos central de la librería NumPy. Un array es una matriz de valores y contiene información acerca de los datos en bruto, como localizar un elemento, y como interpretar un elemento. La clase `ndarray` es usada para representar vectores y matrices. Cada array NumPy es, por ende, una instancia de la clase `ndarray`. Todos los elementos dentro de un array son del mismo tipo, referidos como `dtype`.

Una manera de inicializar un array NumPy es a partir de una lista Python, usando listas anidadas para datos de dos o más dimensiones. Para crear un array NumPy, puede usar la función `np.array()`. Todo lo que necesita hacer para crear un simple array es pasarle una lista. Si usted elige, puede especificar el tipo de datos en su lista.

Problema 1: Cree una lista `lista_1` que represente el siguiente vector:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Luego cree un array `array_1` a partir de `lista_1`

```
1 import numpy as np
2
3 lista_1=[1,2,3,4,5,6,7,8,9]
4 array_1=np.reshape(lista_1,(3,3))
5
6
7 print(array_1)
8
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TE

```
PS C:\Users\1> & C:/Users/1/AppData/Local/Pro
[[1 2 3]
 [4 5 6]
 [7 8 9]]
PS C:\Users\1>
```

Resolución del problema 1.

Explicación:

Se creo una **lista_1** con los valores de la función, luego se crea un **array_1** el cual mediante la función **np.reshape()** lo cargaremos con la lista con las dimensionone (3x3).
Recreando la matriz pedida por el problema.

Problema 2: Cree un array por cada una de las siguientes características:

- Vacío
- Con 9870 ceros
- Con 10,500 ceros 2
- Con números entre el 0 y el 1 espaciado en 20 números

```
1 import numpy as np
2
3 vacio=np.empty([3, 3])
4 array_9870= np.zeros(9870)
5 array_10500= np.zeros(10500)
6 array_0_1= np.linspace(0,1,20)
7 print("Vacio: ",vacio)
8 print("9870: ",array_9870)
9 print("10500: ",array_10500)
10 print("de 0 a 1: ",array_0_1)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS C:\Users\1> & C:/Users/1/AppData/Local/Programs/Python/Python311/python.exe e:/Ta
Vacio: [[0.000e+000 0.000e+000 0.000e+000]
[0.000e+000 0.000e+000 1.443e-321]
[0.000e+000 0.000e+000 0.000e+000]]
9870: [0. 0. 0. ... 0. 0. 0.]
10500: [0. 0. 0. ... 0. 0. 0.]
de 0 a 1: [0. 0.05263158 0.10526316 0.15789474 0.21052632 0.26315789
0.31578947 0.36842105 0.42105263 0.47368421 0.52631579 0.57894737
0.63157895 0.68421053 0.73684211 0.78947368 0.84210526 0.89473684
0.94736842 1.]

Resolución del problema 2.

Explicación:

- creamos la variable **vacío** la cual mediante la función **empty()** la cargaremos con una matriz 3x3 llena de valores aleatorios (en este caso valores flotantes al no ser especificado).
- Para los puntos 2 y 3 se usó el mismo procedimiento creamos los arreglos **array_9870** y **array_10500** en los cuales mediante la función **np.zeros()** al colocarle la cantidad de datos que queremos, regresara un arreglo con la cantidad de elementos, todos inicializados en 0.
- En el punto 4 creamos un valor **array_0_1** en la cual cargamos valores entre el 0 y el 1 mediante la función **np.linspace()**, función que llenaremos con el valor de inicio (0), el valor final (1) y la cantidad de elementos que debe tener el arreglo (20).

A parte de crear un array a partir de una secuencia de elementos, puede crear fácilmente un array lleno de 0s usando la función `np.zeros` o llenos de 1s con la función `np.ones`. O incluso un array vacío: la función `empty` crea un arreglo cuyos valores iniciales son aleatorios y dependen del estado de la memoria. La razón de usar `empty` sobre `zeros` es velocidad. Puede usar `np.linspace()` para crear un array con valores que están espaciados en un intervalo especificado.

Problema 3: Cree un array para cada una de estas matrices:

$$a = (1 \quad 9 \quad 4 \quad 3 \quad 2)$$

$$b = \begin{pmatrix} 8 \\ 7 \\ 5 \\ 6 \\ 0 \end{pmatrix}$$

- Ordene los array a y b
- Concatene los array a y b en un array c

```

1  import numpy as np
2
3  a=np.array([1,9,4,3,2])
4  b=np.array([[8],[7],[5],[6],[0]])
5
6  a.sort()
7  b=np.sort(b,axis=None)
8  c=np.concatenate((a,b))
9  print("a ordenado<",a)
10 print("b ordenado:",b)
11 print("concatenar:",c)
12

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TER

```

PS C:\Users\1> & C:/Users/1/AppData/Local/Prog
a ordenado< [1 2 3 4 9]
b ordenado: [0 5 6 7 8]
concatenar: [1 2 3 4 9 0 5 6 7 8]

```

Resolución del problema 3.

Explicación:

Cargamos los arreglos **a** y **b** con sus datos, para ordenar **a** usamos la función `sort()` la cual ordenara el arreglo. Para el caso de **b** al ser una matriz bidimensional, la función `np.sort()` la cargamos con el arreglo **b** e indicamos que no tenga eje (`axis=None`) esto volverá al arreglo unidimensional y lo ordenara. Para la concatenación usamos la función `np.concatenate()` la cual introducimos **a** y **b**, cargado la función en una variable **c**, dando como resultado la concatenación de ambos arreglos.

Ordenar un array es sencillo con `np.sort()`. Puede especificar el eje, tipo y orden cuando llama la función. Adicional a ordenar, puede usar `argsort`, `lexsort`, `searchsorted` y `partition`; estas

ofrecen funcionalidades distintas para ordenar un array de modos especiales. La función `np.concatenate()` permite concatenar arrays, juntando todos los elementos de los arrays en uno solo.

Problema 4: Cree un array con la siguiente lista, e imprima su tamaño y su Forma.

```
C: > Users > gregt > OneDrive > Documentos > Py_Topicos1 > Taller > Taller1.py
42 array_p4 = np.array([[[[0.30195004, 0.30127072, 0.03042095, 0.74403103, 0.2022509],
43 [0.76041088, 0.79849272, 0.79644947, 0.65030892, 0.55632025]],
44 [0.59384094, 0.29649317, 0.7377125, 0.55018047, 0.44168684]],
45 [0.06494797, 0.33516434, 0.22379763, 0.3428627, 0.3202397 ]],
46 [0.20934204, 0.07305298, 0.87433052, 0.73782305, 0.28831509]],
47 [0.46084052, 0.27598606, 0.05343971, 0.68686768, 0.91083524]]],
48 [[0.70481584, 0.7418082, 0.91056259, 0.03657068, 0.30783487],
49 [0.93055512, 0.61358272, 0.90661768, 0.10189568, 0.29828641]],
50 [0.99889127, 0.75852227, 0.51080695, 0.30961187, 0.62998559]],
51 [0.7817276, 0.02058618, 0.85499711, 0.18267104, 0.69195533]],
52 [0.47014784, 0.49281162, 0.1778547, 0.25006711, 0.12039169]],
53 [0.18558919, 0.41168495, 0.28237535, 0.71745923, 0.58846443]]])
54 print(array_p4)
55 print(array_p4.ndim)
56 print(array_p4.size)
57 print(array_p4.shape)
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** AZURE JUPYTER

```
[[[0.20934204 0.07305298 0.87433052 0.73782305 0.28831509]
[0.46084052 0.27598606 0.05343971 0.68686768 0.91083524]]]

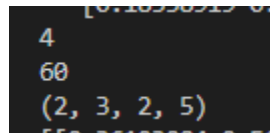
[[[0.70481584 0.7418082 0.91056259 0.03657068 0.30783487]
[0.93055512 0.61358272 0.90661768 0.10189568 0.29828641]]]

[[[0.99889127 0.75852227 0.51080695 0.30961187 0.62998559]
[0.7817276 0.02058618 0.85499711 0.18267104 0.69195533]]]

[[[0.47014784 0.49281162 0.1778547 0.25006711 0.12039169]
[0.18558919 0.41168495 0.28237535 0.71745923 0.58846443]]]]

4
60
(2, 3, 2, 5)
```

Resolución del problema 4.



Explicación:

En este problema primero debemos realizar un arreglo el cual nos ha sido proporcionado por el profesor el cual vamos a manejarlo con las siguientes función al crear este arreglo con el puntero `.ndim` logramos ver la profundidad del arreglo, `.size` nos ayuda a ver el tamaño del arreglo sin importar de cuantas dimensiones sea, y `shape` nos ayuda a comprender la estructura de tus matrices NumPy y realizar operaciones en función de sus dimensiones, lo que es especialmente útil en el procesamiento de datos y cálculos numéricos.

Problema 5: Remodele el array del problema anterior para que sea de 5×12

```

59  nueva_forma=(5,12)
60  #nuevamatriz_512 = array_p4.reshape(nuevaforma)
61  print(array_p4.reshape(nueva_forma))
62

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** AZURE JUPYTER

```

[[[0.47014784 0.49281162 0.1778547 0.25006711 0.12039169]
  [0.18558919 0.41168495 0.28237535 0.71745923 0.58846443]]]]
4
60
(2, 3, 2, 5)
[[0.36193084 0.56127072 0.83642095 0.74463163 0.2822369 0.76041008
  0.79849272 0.79644947 0.65030892 0.55632025 0.59384094 0.29649317]
 [0.7377125 0.55018047 0.44168684 0.06494797 0.33516434 0.22379763
  0.3428627 0.3202397 0.20934204 0.07305298 0.87433052 0.73782305]
 [0.28831509 0.46084052 0.27598606 0.05343971 0.68686768 0.91083524
  0.70481584 0.7418082 0.91056259 0.03657068 0.30783487 0.93055512]
 [0.61358272 0.90661768 0.10189568 0.29828641 0.99889127 0.75852227
  0.51080695 0.30961187 0.62998559 0.7817276 0.02058618 0.85499711]
 [0.18267104 0.69195533 0.47014784 0.49281162 0.1778547 0.25006711
  0.12039169 0.18558919 0.41168495 0.28237535 0.71745923 0.58846443]]
PS C:\Users\gregt>

```

Resolución del problema 5.

Explicación:

Bueno agarrando el arreglo anterior que es array_p4 seguimos con esta actividad la cual nos pide realizar una forma y usamos la función. reshape nos ayuda a cambiar la forma de nuestro arreglo sin modificar sus datos subyacentes y redimensionarlas.

Problema 6: Cree un array de una dimensión y conviértalo en uno de dos dimensiones

```

63  #P6
64  list_d1 = np.array([1,2,3,4,5,6,7,8,9])
65  print(list_d1)
66  list_d2 = list_d1[:, np.newaxis]
67  print(list_d2)
68

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** AZURE JUPYTER

```

0.70481584 0.7418082 0.91056259 0.03657068 0.30783487 0.93055512]
[0.61358272 0.90661768 0.10189568 0.29828641 0.99889127 0.75852227
 0.51080695 0.30961187 0.62998559 0.7817276 0.02058618 0.85499711]
[0.18267104 0.69195533 0.47014784 0.49281162 0.1778547 0.25006711
 0.12039169 0.18558919 0.41168495 0.28237535 0.71745923 0.58846443]]
[1 2 3 4 5 6 7 8 9]
[[1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
PS C:\Users\gregt>

```

Resolución del problema 6.

Explicacion: Con este problema primero debemos realizar una lista nueva y utilizamos un arreglo horizontal unidimensional que cuenta del 1 al 9 luego usamos la función .newaxis para convertir este arreglo unidimensional a un arreglo bidimensional ahora en vez de tener 9 columnas y una fila tiene 1 columna con 9 filas.

Problema 7: Cree un array con números del 1 al 50. Extraiga los números mayores o iguales a 3 divisibles por 7.

```
69 #P7
70 array_50n = np.arange(1, 51)
71 print(array_50n[(array_50n >= 3) & (array_50n % 7 == 0)])
72
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL AZURE JUPYTER

```
[ 7 14 21 28 35 42 49]
PS C:\Users\gregt>
```

Resolución del problema 7.

Explicación: Primero diseñamos nuestro arreglo de uno a 51 pues esto sigue siendo 50 elementos con la función. `arange` y luego procedemos a extraer los números mayores o iguales a 3 y divisibles a 7.

Problema 8: Cree un array a partir de [1.3, 2.4] y súmele 1. Luego multiplíquela por 2.

Explicación: Primero, importé la librería NumPy para tener acceso a sus funciones. Luego, creé un array con los elementos [1.3, 2.4] y lo imprimí para saber su estado inicial. Después, sumé 1 a cada elemento del array usando `+=`, lo cual actualiza el array en su lugar. Volví a imprimir el array para verificar que la suma se había realizado correctamente. Finalmente, multipliqué cada elemento del array por 2 usando `*=` y mostré el array resultante. Todo funcionó como se esperaba, y el array final fue el resultado de aplicar las operaciones matemáticas requeridas.

```
Taller#1-P8.py > ...
1  import numpy as np
2
3  array = np.array([1.3,2.4])
4  print("array= ",array)
5  print("\n")
6  array += 1
7  print("array + 1= ",array)
8  print("\n")
9  array *= 2
10 print("array * 2= ",array)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\gusta\Desktop\Topicos especiales>
array= [1.3 2.4]

array + 1= [2.3 3.4]

array * 2= [4.6 6.8]
PS C:\Users\gusta\Desktop\Topicos especiales>
```

Resolución del problema 8.

Problema 9: Cree un array a partir de la siguiente lista:

```
[[0.65890249, 0.47548884, 0.98114111, 0.39234078],  
 [0.42884138, 0.84663185, 0.75058791, 0.14837213],  
 [0.64171933, 0.06110044, 0.96310199, 0.30722254]]
```

Calcule e imprima las siguientes informaciones:

- El número más grande
- El número más pequeño
- La suma del array
- El promedio del array
- La desviación estándar del arreglo

Explicación: Primero, creé un array a partir de una lista de listas que contiene números flotantes. Luego, utilicé diversas funciones para calcular y mostrar estadísticas clave del array. Encontré el valor máximo con `np.max`, el mínimo con `np.min`, la suma total de los elementos con `np.sum`, el promedio con `np.mean` y la desviación estándar con `np.std`. Cada cálculo me dio una visión más completa de los datos del array.

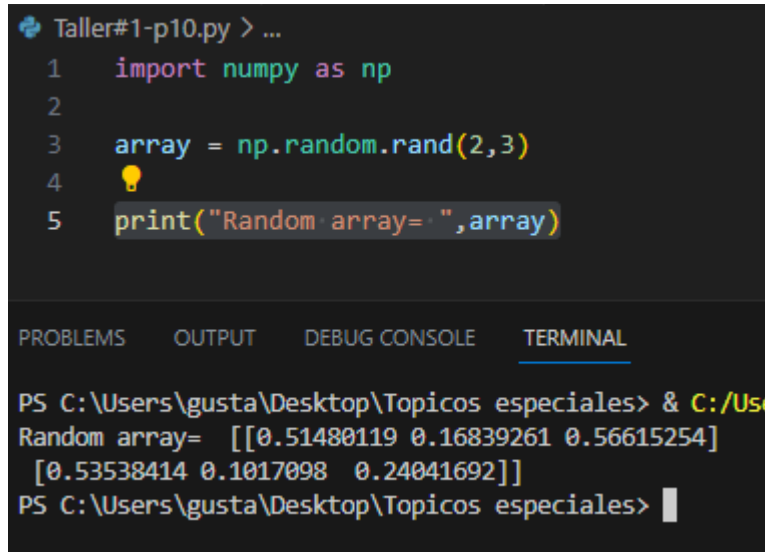
```
Taller#1-P9.py > ...  
4 lista = [[0.65890249, 0.47548884, 0.98114111, 0.39234078],  
5 [0.42884138, 0.84663185, 0.75058791, 0.14837213],  
6 [0.64171933, 0.06110044, 0.96310199, 0.30722254]]  
7  
8 array = np.array(lista)  
9 print("array= ",array)  
10 print("\n")  
11 print("maximo = ", np.max(array))  
12 print("minimo = ", np.min(array))  
13 print("suma = ", np.sum(array))  
14 print("media = ", np.mean(array))  
15 print("desviacion estandar = ", np.std(array))  
16  
17
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\gusta\Desktop\Temas especiales> & C:/Users/gusta/AppData/Local/Programs/Python/Python39-64/Python.exe  
array= [[0.65890249 0.47548884 0.98114111 0.39234078]  
 [0.42884138 0.84663185 0.75058791 0.14837213]  
 [0.64171933 0.06110044 0.96310199 0.30722254]]  
  
maximo = 0.98114111  
minimo = 0.06110044  
suma = 6.65545079  
media = 0.5546208991666667  
desviacion estandar = 0.2899905465255104  
PS C:\Users\gusta\Desktop\Temas especiales>
```


Problema 10: Cree un array aleatorio de 2×3

Explicación: Generé un array de dimensiones 2x3 lleno de números aleatorios entre 0 y 1 utilizando la función `np.random.rand`. Luego, imprimí este array para visualizar los números aleatorios generados.



```
Taller#1-p10.py > ...
1  import numpy as np
2
3  array = np.random.rand(2,3)
4  ⚡
5  print("Random array= ",array)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\gusta\Desktop\Topicos especiales> & C:/Use
Random array= [[0.51480119 0.16839261 0.56615254]
               [0.53538414 0.1017098  0.24041692]]
PS C:\Users\gusta\Desktop\Topicos especiales> |
```

Resolución del problema 10.

Problema 11: Cree un arreglo a partir de la siguiente lista:

[52, 49, 68, 40, 56, 34, 80, 74, 69, 53, 73, 51, 65, 68, 47, 34, 44, 40, 40, 64]

Obtenga las siguientes informaciones del array:

- Los valores únicos.
- Los índices de los valores únicos.
- La frecuencia de los valores únicos.

Explicación: Primero, creé un array con una lista de 20 números enteros. Luego, utilicé la función `np.unique` para encontrar y mostrar los valores únicos en el array. Además, utilicé las opciones `return_index=True` y `return_counts=True` con `np.unique` para obtener los índices de los valores únicos y la frecuencia con la que aparecen, respectivamente.

```
Taller#1-p11.py > ...
1  import numpy as np
2
3  array = np.array([52, 49, 68, 40, 56, 34, 80, 74, 69, 53, 73, 51, 65,
4  print("array= ",array)
5  print("\n")
6  print("Valores unicos = ", np.unique(array))
7  print("Indices de valores unicos = ", np.unique(array, return_index=True))
8  print("Frecuencia de valores unicos = ", np.unique(array, return_counts=True))

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\gusta\Desktop\Temas especiales> & C:/Users/gusta/AppData/Local/Microsoft/Windows/PowerShell/PowerShell.exe -Command "python Taller#1-p11.py"
array= [52 49 68 40 56 34 80 74 69 53 73 51 65 68 47 34 44 40 40 64]

Valores unicos = [34 40 44 47 49 51 52 53 56 64 65 68 69 73 74 80]
Indices de valores unicos = [ 5  3 16 14  1 11  0  9  4 19 12  2  8 10  7  6]
Frecuencia de valores unicos = [2 3 1 1 1 1 1 1 1 1 1 2 1 1 1]
PS C:\Users\gusta\Desktop\Temas especiales> SDS
```

Resolución del problema 11.

Problema 12: Lea el archivo medidas.csv para las columnas cyl y hp, transponga la matriz, y guarde la matriz modificada como medidas_T.csv.

Explicación:

Primero, leí un archivo CSV llamado 'medidas.csv' en un DataFrame de Pandas. Luego, seleccioné solo las columnas 'cyl' y 'hp' para crear un nuevo DataFrame. Tras esto, transpuse el DataFrame, intercambiando filas por columnas, para obtener una nueva estructura de datos. Finalmente, guardé este DataFrame transpuesto en un nuevo archivo CSV llamado 'medidas_T.csv', conservando los índices. Este código resuelve el Problema 12 al manipular y guardar datos de un archivo CSV.

```
import pandas as pd
import numpy as np

df = pd.read_csv('medidas.csv')

df_S = df[['cyl', 'hp']]

df_T = df_S.T

df_T.to_csv('medidas_T.csv', index=True)
```

Resolución del problema 12.

```
medidas_T.csv
1  ,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
2  cyl,6,6,4,6,6,8,6,8,4,4,6,6,8,8,8,8,8,4,4,4,4,8,8,8,8,4,4,4,8,6,8,4
3  hp,110,110,93,110,175,105,245,62,95,123,123,180,180,180,205,215,230,66,52,65,97,150,150,245,175,66,91,113,264,175,335,109
4
```

Resultados del problema 12.