Universidad Autónoma de San Luis Potosí
Facultad de ingeniería
Inteligencia Artificial Aplicada
Practica: 2
Nombre Práctica: Conceptos Básicos de NumPy
Nombre del Alumno

Fecha: 05/02/2025

Procedimiento

Resultados

Como primer paso importe NumPy con el alias *np*

```
[1]  import numpy as np
```

Imagen 1.- Librería NumPy

Convierta la lista de Python, en un arreglo de NumPy y almacenelo en una variable llamada *sudoku_array*.

```
sudoku_array = np.array([
    [0, 0, 4, 3, 0, 0, 2, 0, 9],
    [0, 0, 5, 0, 0, 9, 0, 0, 1],
    [0, 7, 0, 0, 6, 0, 0, 4, 3],
    [0, 0, 6, 0, 0, 2, 0, 8, 7],
    [1, 9, 0, 0, 0, 7, 4, 0, 0],
    [0, 5, 0, 0, 8, 3, 0, 0, 0],
    [6, 0, 0, 0, 0, 0, 1, 0, 5],
    [0, 0, 3, 5, 0, 8, 6, 9, 0],
    [0, 4, 2, 9, 1, 0, 3, 0, 0]
])

print(sudoku_array)
```

```
[[0 0 4 3 0 0 2 0 9]
 [0 0 5 0 0 9 0 0 1]
 [0 7 0 0 6 0 0 4 3]
 [0 0 6 0 0 2 0 8 7]
 [1 9 0 0 0 7 4 0 0]
 [0 5 0 0 8 3 0 0 0]
 [6 0 0 0 0 0 1 0 5]
 [0 0 3 5 0 8 6 9 0]
 [0 4 2 9 1 0 3 0 0]]
```

Imagen 2.- Arreglo de la lista de Python

Para probar alguna de estas funciones cree un arreglo de ceros con 3 renglones y 4 columnas.

```
[4]  c1 = np.zeros((3, 4))
     print(c1)
```

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

Imagen 3.- Arreglo de ceros

Convierta el arreglo *sudoku_array* a un arreglo unidimensional, almacenelo en una variable llamada *sudoku_flatten* e imprimalo.

```
[ ]  sudoku_flatten = sudoku_array.flatten()
     print(sudoku_flatten)
```

```
[0 0 4 3 0 0 2 0 9 0 0 5 0 0 9 0 0 1 0 7 0 0 6 0 0 4 3 0 0 6 0 0 2 0 8 7 1
 9 0 0 0 7 4 0 0 0 5 0 0 8 3 0 0 0 6 0 0 0 0 0 0 1 0 5 0 0 3 5 0 8 6 9 0 0 4
 2 9 1 0 3 0 0]
```

Imagen 4.- Arreglo Unidimensional

Regrese el arreglo *sudoku_flatten* a las dimensiones originales y almacene el resultado en la variable *sudoku_reshaped*.

Imprima la variable sudoku_reshaped para observar que el orden de los elementos no se haya afectado.

```
     sudoku_reshaped = sudoku_flatten.reshape(9, 9)
     print(sudoku_reshaped)
```

```
[[0 0 4 3 0 0 2 0 9]
 [0 0 5 0 0 9 0 0 1]
 [0 7 0 0 6 0 0 4 3]
 [0 0 6 0 0 2 0 8 7]
 [1 9 0 0 0 7 4 0 0]
 [0 5 0 0 8 3 0 0 0]
 [6 0 0 0 0 0 1 0 5]
 [0 0 3 5 0 8 6 9 0]
 [0 4 2 9 1 0 3 0 0]]
```

Imagen 5.- Arreglo Bidimensional

Imprima el tipo de dato del arreglo sudoku_array.

```
     print(sudoku_array.dtype)
```

```
int64
```

Imagen 6.- Tipo de dato del arreglo, int64

Para optimizar memoria cambiemos el tipo de dato del arreglo *sudoku_array* a *uint8*

```
[6]  mem = np.array(sudoku_array, dtype=np.uint8)
     print(mem.dtype)
```

```
uint8
```

Imagen 7.- Tipo de dato del arreglo, unit8

Utilice la función `np.load()` para cargar los datos del archivo `tree_census.npy`.

Almacene el arreglo en una variable llamada `tree_census`

```
[14] from google.colab import drive
     drive.mount('/content/drive')

     tree_census = np.load("/content/drive/MyDrive/Colab Notebooks/Manual/Practica2_CONCEPTOS BÁSICOS DE NUMPY/datasets/tree_census.npy")
     print(tree_census)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
[[     3 501451     24      0]
 [     4 501451     20      0]
 [     7 501911      3      0]
 ...
 [  1198 227387     11      0]
 [  1199 227387     11      0]
 [  1210 227386      6      0]]
```

Imagen 8.- Cargar datos de archivo

Seleccione todos los renglones de la segunda columna (con indice 1) y almacenelos en una variable llamada *block_ids*

```
block_ids = tree_census[:, 1]
print(block_ids)
```

```
344023 344023 342454 342454 342454 342454 342454 342454 349688 349688
 349688 349688 349688 349688 349688 349688 349688 349688 349688 349688
 349688 349688 349688 349688 342455 342455 342455 342455 342455 342456
 342456 342457 342457 342457 342457 342457 342457 342457 324495 324495
 324495 324495 324495 324495 324495 324495 324495 324495 324495 324495
 324495 324495 324495 324495 324495 324495 324495 324495 324495 342458
 342458 342458 342458 342458 342458 342458 400788 400788 400788 400788
 400788 400788 400788 400788 400788 400788 400788 400788 400788 400787
 400787 400787 400787 400787 400787 400787 400787 400787 400787 400787
 400787 344037 344037 344037 344037 344037 344037 344037 344037 344037
 400789 400789 400789 400789 400789 400789 400789 400789 324274 324274
 324274 324274 324274 344033 344033 344033 344033 344033 324275 324275
 324275 344034 344034 344034 324346 324346 324346 324346 324346 506799
 506799 506799 506799 506799 506799 506799 506799 506799 506799 506799
 330413 330413 330413 330413 330413 330413 324345 324345 324345 324345
 506834 506834 506834 506834 506834 506834 342451 342451 342451 342451 342451
 342449 342449 342449 342449 342449 342447 342447 342447 342447 342447
 342450 342450 342450 342450 342450 342448 342448 342448 342448 342448
 342446 342446 342446 342446 342446 506915 506915 506894 506894 506894
 506894 507002 507002 507002 507002 104039 104039 104039 104039 104039
 104039 104039 506912 506912 506912 506912 506912 506912 506912 506912 506912
 104054 104054 104054 506891 506891 506891 506891 506891 506891 506891
 506891 342425 342425 342425 342425 104076 104076 104076 104076 104076
 104076 104076 104076 104076 104000 104000 104000 342424 342424 342424
 342424 342426 342426 342426 342426 104053 104053 104053 104121 104121
 104121 104121 104121 104121 104121 104121 104121 104121 104121 104121
```

Imagen 9.- Selección de datos

De la variable *block_ids* imprima:

- El octavo elemento.
- Todos los elementos hasta el sexto.
- 5 elementos empezando en el indice 10.

```
[10] print(block_ids[7])
     print(block_ids[:6])
     print(block_ids[10:15])
```

```
501911
[501451 501451 501911 501911 501911 501911]
[501911 501911 501909 501909 501909]
```

Imagen 20.- Impresión de datos

Del arreglo *tree_census* selecciones todos los árboles (renglones) con un diametro del tronco menor a 10.

```
mask = tree_census[:] <10
print(mask)
selected_trees = tree_census[mask]

small_trees = tree_census[tree_census[:, 2] < 10]
print(small_trees)
```

```
[[ True False False  True]
 [ True False False  True]
 [ True False  True  True]
 ...
 [False False False  True]
 [False False False  True]
 [False False  True  True]]
[[     7 501911      3      0]
 [     8 501911      3      0]
 [     9 501911      4      0]
 ...
 [  1186 226832      0     20]
 [  1187 226832      0     21]
 [  1210 227386      6      0]]
```

Imagen 31.- Selección de datos

Investigadores descubrieron 2 nuevos árboles, ambos estan almacenados en la variable *new_trees*, agregue sus datos al arreglo *tree_census*

```
[12] new_trees = np.array([[1211, 227386, 20, 0], [1212, 227386, 8, 0]])
```

```
[18] tree_census = np.vstack((tree_census, new_trees))
```

Imagen 42.- Almacenamiento de datos

Utilizando la función `np.load()` lea el archivo `monthly_sales.npy` y almacene el resultado en una variable llamada `monthly_sales`.

```
monthly_sales = np.load("/content/drive/MyDrive/Colab Notebooks/Manual/Practica2_CONCEPTOS BÁSICOS DE NUMPY/datasets/monthly_sales.npy")
print(monthly_sales)
```

```
[[ 4134 23925  8657]
 [ 4116 23875  9142]
 [ 4673 27197 10645]
 [ 4580 25637 10456]
 [ 5109 27995 11299]
 [ 5011 27419 10625]
 [ 5245 27305 10630]
 [ 5270 27760 11550]
 [ 4680 24988  9762]
 [ 4913 25802 10456]
 [ 5312 25405 13401]
 [ 6630 27797 18403]]
```

Imagen 53.- Cargar datos de archivo

Obtenga el total de ventas de las 3 empresas por mes.

```
[22] total_sales = np.sum(monthly_sales, axis=1)

     for month, total_sales in enumerate(total_sales, start=1):
         print(f"Mes {month}: Total de ventas = {total_sales}")
```

```
Mes 1: Total de ventas = 36716
Mes 2: Total de ventas = 37133
Mes 3: Total de ventas = 42515
Mes 4: Total de ventas = 40673
Mes 5: Total de ventas = 44403
Mes 6: Total de ventas = 43055
Mes 7: Total de ventas = 43180
Mes 8: Total de ventas = 44580
Mes 9: Total de ventas = 39430
Mes 10: Total de ventas = 41171
Mes 11: Total de ventas = 44118
Mes 12: Total de ventas = 52830
```

Imagen 64.- Obtención de ventas por mes

Obtenga el total de ventas a final de año para cada empresa

```
total_sales = np.sum(monthly_sales, axis=0)

for company, total_sales in enumerate(total_sales, start=1):
    print(f"Empresa {company}: Total de ventas = {total_sales}")
```

```
Empresa 1: Total de ventas = 59673
Empresa 2: Total de ventas = 315105
Empresa 3: Total de ventas = 135026
```

Imagen 75.- Obtención de ventas por año

```
[25] cumulative_sales = np.cumsum(monthly_sales, axis=0)

     for company in range(len(cumulative_sales[0])):
         print(f"Empresa {company + 1}: Ventas acumuladas = {cumulative_sales[-1, company]}")
```

```
Empresa 1: Ventas acumuladas = 59673
Empresa 2: Ventas acumuladas = 315105
Empresa 3: Ventas acumuladas = 135026
```

Imagen 86.- Obtención de ventas acumuladas

Se desean calcular los impuestos de las ventas de cada mes por cada industria.

Para calcular el impuesto solo se necesita multiplicar el valor de las ventas por 0.05.

Obtenga los impuestos mensuales y alamcenelos en un arreglo llamado *tax_collected*

```
[26] tax_collected = monthly_sales * 0.05
```

Imagen 97.- Obtención de impuestos

Ahora obtenga el total de ingresos sumando el total de ventas de cada mes y el impuesto calculado de ese mes.

```
total_income = monthly_sales + tax_collected

for month, total_income in enumerate(total_income, start=1):
    print(f"Mes {month}: Total de ingresos = {total_income}")
```

```
Mes 1: Total de ingresos = [ 4340.7  25121.25  9089.85]
Mes 2: Total de ingresos = [ 4321.8  25068.75  9599.1 ]
Mes 3: Total de ingresos = [ 4906.65 28556.85 11177.25]
Mes 4: Total de ingresos = [ 4809.   26918.85 10978.8 ]
Mes 5: Total de ingresos = [ 5364.45 29394.75 11863.95]
Mes 6: Total de ingresos = [ 5261.55 28789.95 11156.25]
Mes 7: Total de ingresos = [ 5507.25 28670.25 11161.5 ]
Mes 8: Total de ingresos = [ 5533.5 29148.  12127.5]
Mes 9: Total de ingresos = [ 4914.  26237.4 10250.1]
Mes 10: Total de ingresos = [ 5158.65 27092.1  10978.8 ]
Mes 11: Total de ingresos = [ 5577.6  26675.25 14071.05]
Mes 12: Total de ingresos = [ 6961.5  29186.85 19323.15]
```

Imagen 108.- Obtención de total de ingresos

En esta última sección se aplicarán diferentes técnicas de transformación a la pintura *Paseo por el acantilado en Pourville* de Monet

Daremos permisos a Colab para acceder a Drive

Cargue el archivo llamado *rbg_array* y almacenelo en una variable con el mismo nombre.

```
import numpy as np
```

```
[33] from google.colab import drive
     drive.mount('/content/drive')
     from PIL import Image

     file_path = "/content/drive/MyDrive/Colab Notebooks/Manual/Practica2_CONCEPTOS BÁSICOS DE NUMPY/datasets/paseo.jpg"
     image = Image.open(file_path)

     # Convert the image to a NumPy array
     rgb_array = np.array(image)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Imagen 119.- Carga de imagen

La siguiente celda de código permite visualizar la imagen

```
from matplotlib import pyplot as plt
plt.imshow(rgb_array)
```

```
<matplotlib.image.AxesImage at 0x7993258dee10>
```

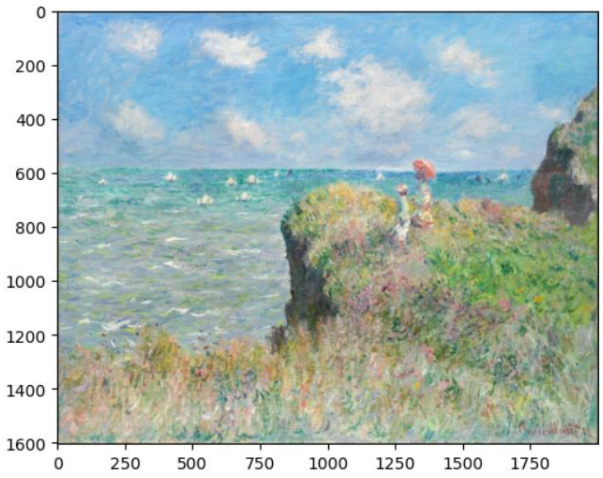

Imagen 20.- Visualización de imagen

Utilizando el método *flip*, haga que el oceano se encuentre en la parte derecha de la imagen

```
f = np.flip(rgb_array, axis=1)
plt.imshow(f)
```

```
<matplotlib.image.AxesImage at 0x7993248752d0>
```

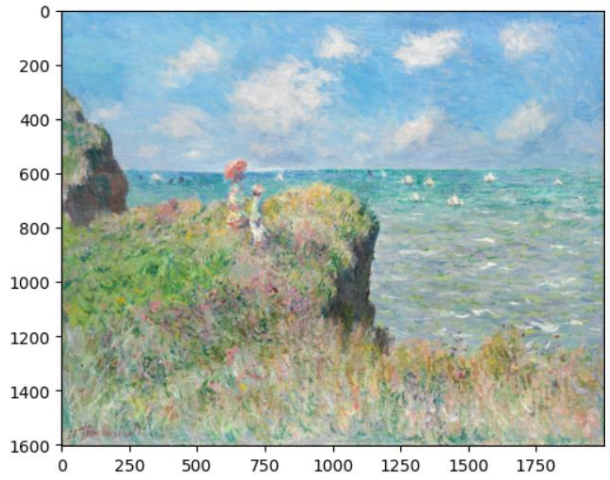

Imagen 21.- Darle vuelta a la imagen

utilizando el metodo *transpose* cambie el orden de renglones y columnas para girar la imagen 90°

```
[39] t = np.transpose(rgb_array, (1, 0, 2))
     plt.imshow(t)
```

```
<matplotlib.image.AxesImage at 0x7993247068d0>
```

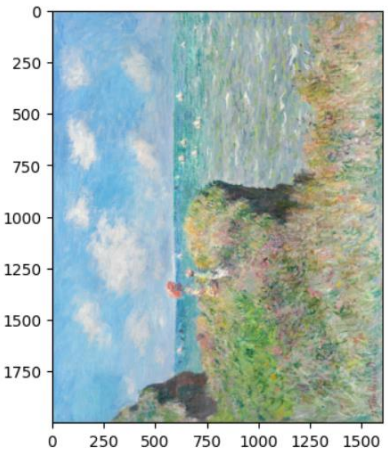

Imagen 12.- Girar la imagen 90°

Comprensión

1. ¿Qué es Numpy?

NumPy (Numerical Python) es una biblioteca de Python para computación científica que proporciona soporte para matrices y arreglos multidimensionales, junto con una gran colección de funciones matemáticas de alto rendimiento para operar con estos datos.

Es ampliamente utilizado en el análisis de datos, la inteligencia artificial, el procesamiento de imágenes y la simulación científica.

2. **¿Cuál es la diferencia entre una lista de Python y un arreglo Numpy?**

La lista de Python puede almacenar diferentes tipos de datos en una misma lista y el arreglo NumPy solo puede almacenar datos de un mismo tipo, haciéndolo mas eficiente porque almacena los datos en un bloque de memoria y usa operaciones vectorizadas y la lista de Python almacena referencias a objetos en la memoria.

Además la lista de Python solo tiene funciones básicas y el arreglo de NumPy puede hacer operaciones más avanzadas.

3. **¿Qué es el broadcasting en Numpy y por qué es importante?**

Es una técnica que permite realizar operaciones entre arreglos de diferentes formas sin necesidad de repetir o expandir datos manualmente. Esto hace que las operaciones sean más eficientes. NumPy "extiende" automáticamente los arreglos más pequeños para que tengan la misma forma que los más grandes, sin necesidad de hacer copias de datos.

4. **¿Cuál es la importancia de Numpy en el ámbito del aprendizaje automático y la ciencia de datos?**

Es importante ya que proporciona estructuras de datos optimizadas y funciones matemáticas eficientes para manejar grandes volúmenes de datos numéricos. Facilita la implementación de modelos de Machine Learning al manejar datos matriciales y cálculos matemáticos de manera optimizada.

Conclusiones

El conocer la librería NumPy ayuda a poder manejar grandes cantidades de datos, además de poder hacer diversas operaciones con ellos, sobre todo matrices.

Esta librería es muy útil para el aprendizaje automático y la ciencia de datos.