OPERACIONES CON PANDA Y NUMPY

¿Qué es NumPy?

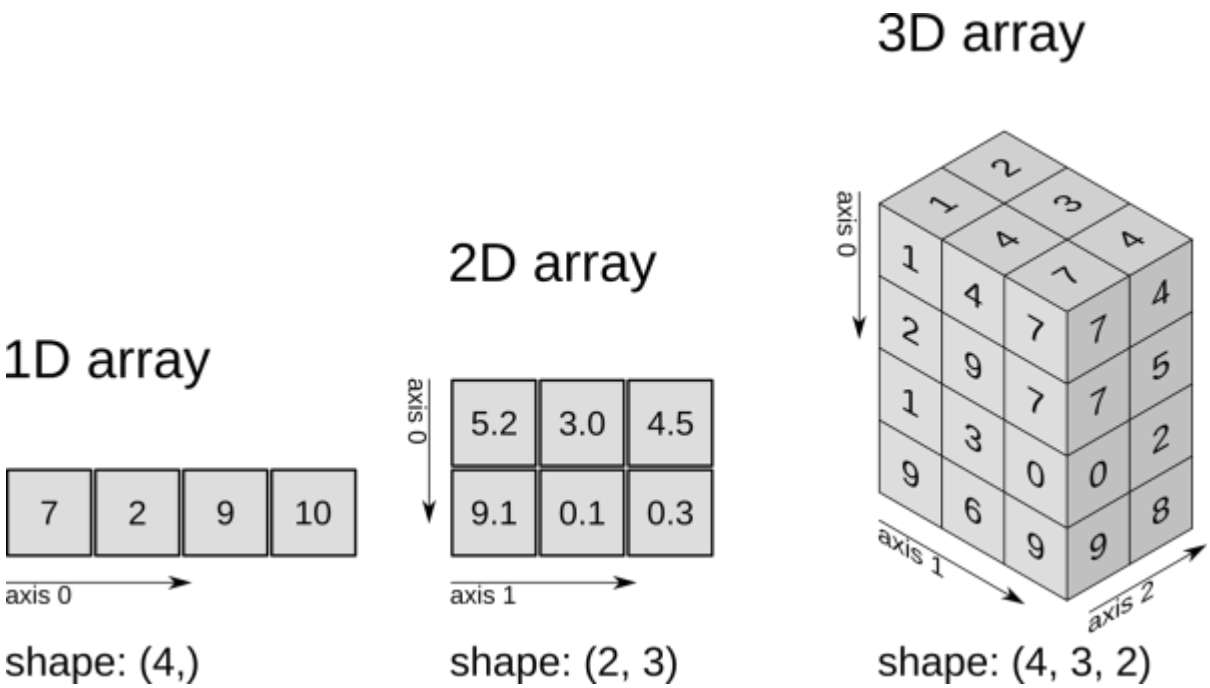
NumPy (Numerical Python) es una librería para trabajar con arreglos y realizar cálculos numéricos de manera eficiente. Proporciona soporte para matrices y arreglos multidimensionales, junto con una colección de funciones matemáticas para realizar operaciones sobre estos arrays de manera eficiente.

Las principales características de NumPy incluyen:

- Rápido para trabajar con grandes volúmenes de datos numéricos.
- Incluye funciones matemáticas avanzadas.
- **Trabaja con Arrays N-dimensionales:** Arrays eficientes y de tamaño fijo que permiten
  realizar operaciones matemáticas y lógicas.
- **Funciones Matemáticas:** Herramientas para realizar operaciones matemáticas avanzadas como álgebra lineal, transformadas de Fourier y generación de números aleatorios.
- **Integración con Otras Bibliotecas:** Se integra perfectamente con otras bibliotecas científicas en Python como Pandas, SciPy y Matplotlib

La clase de objetos array

Un array es una estructura de datos de un mismo tipo organizada en forma de tabla o cuadrícula de distintas dimensiones. Las dimensiones de un array también se conocen como ejes.



Atributos y funciones de Numpy

Categoría	Función / Atributo	Descripción
Creación de arrays	np.array()	Crea un array.

	np.zeros()	Crea un array lleno de ceros.
	np.ones()	Crea un array lleno de unos.
	np.arange()	Crea un array con un rango de valores.
	np.linspace()	Crea un array con valores espaciados uniformemente.
	np.eye()	Matriz identidad.
Atributos	.ndim	Número de dimensiones.
	.shape	Tamaño de cada dimensión.
	.size	Número total de elementos.
	.dtype	Tipo de dato de los elementos.
Operaciones matemáticas	np.add()	Suma de arrays.
	np.subtract()	Resta de arrays.
	np.multiply()	Multiplicación.
	np.divide()	División.
	np.sqrt()	Raíz cuadrada.
	np.exp()	Exponencial.
	np.log()	Logaritmo natural.
Estadísticas	np.mean()	Media aritmética.
	np.median()	Mediana.
	np.std()	Desviación estándar.
	np.var()	Varianza.
	np.min()	Valor mínimo.
	np.max()	Valor máximo.
	np.sum()	Suma total.
Manipulación de arrays	np.reshape()	Cambia forma del array.
	np.ravel()	Aplana el array.
	np.transpose()	Transpone el array.
	np.concatenate()	Une arrays.
	np.split()	Divide arrays.
Generación de números aleatorios	np.random.rand()	Aleatorios en [0,1].
	np.random.randint()	Aleatorios enteros.
	np.random.randn()	Distribución normal.
	np.random.choice()	Selección aleatoria de elementos.

¿Qué es Pandas?

Pandas es una biblioteca de código abierto de Python ampliamente **utilizada en el análisis de datos y machine learning**. Proporciona estructuras de datos eficientes y flexibles para manejar datos numéricos y de otro tipo en Python. Los principales componentes de Pandas son los DataFrame y las Series. Un **DataFrame** es una **tabla de datos bidimensional con etiquetas de fila y columna**, mientras que una Series es un arreglo unidimensional de datos etiquetados.

Pandas permite manipular y limpiar datos provenientes de diversas fuentes, como hojas de cálculo, archivos CSV, bases de datos SQL y formatos de datos populares en la web. También ofrece herramientas para el análisis de datos, que incluyen agregación, filtrado y transformación de datos, además de la creación de gráficos y visualizaciones para explorar patrones y tendencias.

Atributos y funciones de Panda

Categoría	Función / Atributo	Descripción
Creación	pd.Series()	Crea una serie unidimensional.
	pd.DataFrame()	Crea un DataFrame (tabla).
Lectura / Escritura	pd.read_csv()	Lee un archivo CSV.
	pd.read_excel()	Lee un archivo Excel.
	df.to_csv()	Guarda DataFrame como CSV.
	df.to_excel()	Guarda DataFrame como Excel.
Información	df.head()	Muestra las primeras filas.
	df.tail()	Muestra las últimas filas.
	df.shape	Número de filas y columnas.
	df.columns	Nombres de columnas.
	df.index	Índices de filas.
	df.info()	Resumen del DataFrame.
	df.describe()	Estadísticas básicas.
Selección de datos	df['columna']	Selección por nombre de columna.
	df.loc[]	Selección por etiquetas.
	df.iloc[]	Selección por índices numéricos.
Filtrado	df[df['col'] > 10]	Filtra filas según condición.
Ordenamiento	df.sort_values()	Ordena por valores.
	df.sort_index()	Ordena por índice.
Agregación	df.sum()	Suma de valores.
	df.mean()	Media aritmética.
	df.max()	Valor máximo.
	df.min()	Valor mínimo.
	df.groupby()	Agrupación de datos.
Limpieza de datos	df.dropna()	Elimina valores nulos.
	df.fillna()	Rellena valores nulos.
	df.drop()	Elimina columnas o filas.
Conversión	df.astype()	Cambia el tipo de datos.

Tipos de datos de Pandas

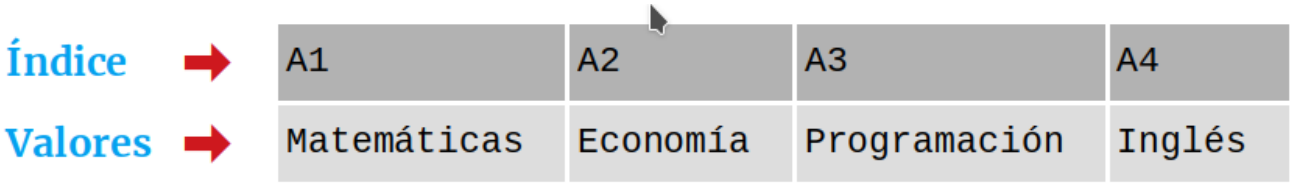
Pandas dispone de tres estructuras de datos diferentes:

A. **Series:** Estructura de una dimensión.
B. **DataFrame:** Estructura de dos dimensiones (tablas).
C. **Panel:** Estructura de tres dimensiones (cubos).

Estas estructuras se construyen a partir de arrays de la librería NumPy, añadiendo nuevas funcionalidades.

A. LA CLASE DE OBJETOS SERIES

Son estructuras similares a los arrays de una dimensión. Son homogéneas, es decir, sus elementos tienen que ser del mismo tipo, y su tamaño es inmutable, es decir, no se puede cambiar, aunque si su contenido.



Ejemplo. *La siguiente serie contiene las asignaturas de un curso.*

Fuente: *https://aprendeconalf.es/docencia/python/manual/pandas/*

Ejemplo

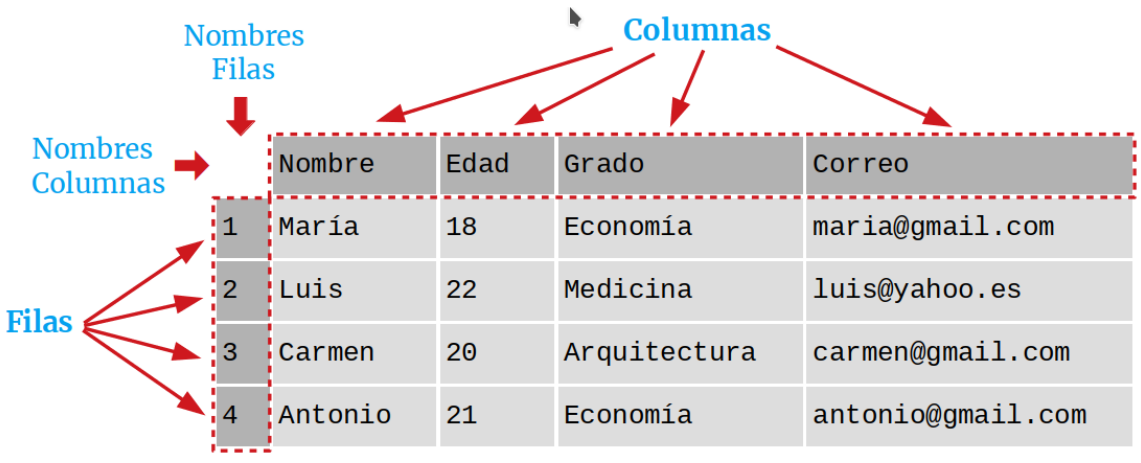
```python
import pandas as pd

# Crear una Serie
numeros = [10, 20, 30, 40, 50]
serie = pd.Series(numeros)

print("Serie básica:")
print(serie)

# --- Atributos ---
print("\nAtributos:")
print("Valores:", serie.values)          # Los datos en forma de arreglo numpy
print("Índices:", serie.index)           # Índice de la serie
print("Tipo de datos:", serie.dtype)     # Tipo de datos de los elementos
print("Tamaño:", serie.size)             # Cantidad total de elementos
print("Forma:", serie.shape)             # Dimensiones de la Serie
print("Dimensiones:", serie.ndim)        # Número de dimensiones
print("¿Tiene valores nulos?:", serie.hasnans) # Si contiene NaN
```

Tabla de atributos de panda.Series

Atributo	Descripción
values	Devuelve los valores de la Serie como un numpy.ndarray.
index	Devuelve los índices o etiquetas de la Serie.
dtype	Tipo de datos de los elementos (int64, float64, object, etc.).
size	Número total de elementos en la Serie.
shape	Devuelve una tupla con la dimensión de la Serie.
ndim	Devuelve el número de dimensiones (siempre 1 para Series).
hasnans	Indica si hay valores nulos (True o False).
name	Nombre asignado a la Serie (por defecto None).

B. LA CLASE DATAFRAME

Un objeto del tipo **DataFrame** define un conjunto de datos estructurado en forma de tabla donde cada columna es un objeto de tipo Series, es decir, todos los datos de una misma columna son del mismo tipo, y las filas son registros que pueden contender datos de distintos tipos. Un DataFrame contiene dos índices, uno para las filas y otro para las columnas, y se puede acceder a sus elementos mediante los nombres de las filas y las columnas.



	Nombre	Edad	Grado	Correo
1	María	18	Economía	maria@gmail.com
2	Luis	22	Medicina	luis@yahoo.es
3	Carmen	20	Arquitectura	carmen@gmail.com
4	Antonio	21	Economía	antonio@gmail.com

Ejemplo. *El siguiente DataFrame contiene información sobre los alumnos de un curso. Cada fila corresponde a un alumno y cada columna a una variable.*

Ejemplo

```python
import pandas as pd

# Crear un DataFrame de ejemplo
data = {
    "Nombre": ["Ana", "Luis", "Carla", "Pedro"],
    "Edad": [23, 30, 27, 35],
    "Ciudad": ["Lima", "Cusco", "Arequipa", "Piura"],
    "Puntaje": [85, 90, 88, 92]
}

df = pd.DataFrame(data)

# Mostrar el DataFrame
print("DataFrame:")
print(df)
print()

# Atributos más utilizados
print("Forma (shape):", df.shape)          # Filas y columnas
print("Columnas:", df.columns)            # Nombres de columnas
print("Índices:", df.index)               # Índices
print("Tipos de datos:\n", df.dtypes)     # Tipo de cada columna
print("Valores (array numpy):\n", df.values)  # Solo los valores
print("Número de elementos:", df.size)    # Total de celdas
print("Primeras filas:\n", df.head(2))    # Primeras 2 filas
print("Últimas filas:\n", df.tail(2))     # Últimas 2 filas
```

Tabla de atributos de un DataFrame

Atributo / Método	Descripción	Ejemplo de uso
df.shape	Devuelve una tupla con (número de filas, número de columnas).	(4, 4)
df.columns	Lista con los nombres de las columnas.	Index(['Nombre', 'Edad', 'Ciudad', 'Puntaje'], dtype='object')
df.index	Índice (filas) del DataFrame.	RangeIndex(start=0, stop=4, step=1)
df.dtypes	Tipos de datos de cada columna.	object, int64, ...
df.values	Devuelve un ndarray de NumPy con los datos.	[['Ana', 23, 'Lima', 85], ...]
df.size	Número total de elementos (filas × columnas).	16
df.head(n)	Muestra las primeras n filas.	df.head(3)
df.tail(n)	Muestra las últimas n filas.	df.tail(2)
df.info()	Información resumida del DataFrame.	df.info()
df.describe()	Estadísticas descriptivas (solo numéricas por defecto).	df.describe()

Instalación de las librerías
pip install numpy **pip install pandas**

Algunos comandos básicos para gestión

Comando	Descripción
python --version	Muestra la versión instalada de Python.
python archivo.py	Ejecuta un script de Python llamado archivo.py.
pip install nombre_paquete	Instala un paquete desde **PyPI** (repositorio oficial de Python).
pip install nombre_paquete==1.2.3	Instala una versión específica de un paquete.
pip install --upgrade nombre_paquete	Actualiza un paquete a la última versión.
pip uninstall nombre_paquete	Desinstala un paquete.
pip freeze	Muestra una lista de todos los paquetes instalados y sus versiones.
pip freeze > requirements.txt	Guarda la lista de paquetes y versiones en un archivo requirements.txt.
pip install -r requirements.txt	Instala todos los paquetes listados en requirements.txt.
help(objeto)	Muestra la ayuda/documentación de un objeto, función o módulo en Python.
dir(objeto)	Lista los métodos y atributos disponibles para un objeto.
exit()	Sale de la consola interactiva de Python.

EJERCICIOS

Ejercicio 1

Implementa el siguiente algoritmo. Tipos de arrays

```python
import numpy as np

# Array de una dimensión (1D)
a1 = np.array([1, 2, 3])
print("Array 1D:")
print(a1)
print()

# Array de dos dimensiones (2D)
a2 = np.array([[1, 2, 3], [4, 5, 6]])
print("Array 2D:")
print(a2)
print()

# Array de tres dimensiones (3D)
a3 = np.array([
    [[1, 2, 3], [4, 5, 6]],
    [[7, 8, 9], [10, 11, 12]]
])
print("Array 3D:")
print(a3)
```

Ejercicio 2

Implementa el siguiente algoritmo. Operaciones matemáticas.

```python
import numpy as np

# Creamos dos arrays 1D
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print("Array A:", a)
print("Array B:", b)
print()

# Suma
print("Suma (A + B):", a + b)

# Resta
print("Resta (A - B):", a - b)

# Multiplicación elemento a elemento
print("Multiplicación (A * B):", a * b)

# División elemento a elemento
print("División (A / B):", a / b)

# Potencia elemento a elemento
print("Potencia (A ** 2):", a ** 2)

# Operaciones con escalares
print("A + 10:", a + 10)
print("B * 3:", b * 3)

# Operaciones estadísticas
print("Suma total de A:", np.sum(a))
print("Promedio de B:", np.mean(b))
print("Valor máximo de A:", np.max(a))
print("Valor mínimo de B:", np.min(b))
```

Ejercicio 3

Implementa el siguiente algoritmo. Operaciones estadisticas

```python
import numpy as np

# Creamos un array
datos = np.array([10, 20, 30, 40, 50])

print("Array:", datos)
print()

# Funciones estadísticas básicas
print("Suma total:", np.sum(datos))
print("Promedio:", np.mean(datos))
print("Mediana:", np.median(datos))
print("Desviación estándar:", np.std(datos))
print("Varianza:", np.var(datos))
print("Valor máximo:", np.max(datos))
print("Valor mínimo:", np.min(datos))
print("Índice del valor máximo:", np.argmax(datos))
print("Índice del valor mínimo:", np.argmin(datos))
```

LEER DATOS DE UN ARCHIVO CSV

Ejercicio 4

Implementa el siguiente algoritmo. Leer datos de un archivo CSV

```python
import numpy as np

# Leer la columna 'promedio' (segunda columna → índice 1)
ruta = r"C:\Users\DELL\Desktop\python\senati\1_numpy_panda\notas.csv"

notas = np.genfromtxt(ruta, delimiter=';', skip_header=1, usecols=1)

print(notas)

promedio = np.mean(notas)
desviacion = np.std(notas)
desaprobadas = notas[notas < 11]

print("Promedio general:", promedio)
print("Desviación estándar:", desviacion)
print("Notas desaprobadas:", desaprobadas)
```

Ejercicio 5

Implementa el siguiente algoritmo. Leer datos de un archivo en Google Sheets

```
import numpy as np
import urllib.request #Permite trabajar con enlaces web

# URL del CSV exportado desde Google Sheets
url = 'https://docs.google.com/spreadsheets/d/e/2PACX-1vSNtdDRtpjj4b4INDL3nb

# Descargar y leer datos directamente desde la web
response = urllib.request.urlopen(url)
data = np.genfromtxt(response, delimiter=',', skip_header=1, usecols=1)

# Procesar los datos
promedio = np.mean(data)
desviacion = np.std(data)
desaprobadas = data[data < 11]

print("Promedio general:", promedio)
print("Desviación estándar:", desviacion)
print("Notas desaprobadas:", desaprobadas)
```

IMPLEMENTACIÓN DE UN CRUD

¿Qué es un CRUD?

CRUD es un acrónimo en informática que significa **Crear, Leer, Actualizar y Eliminar** (Create, Read, Update, Delete en inglés). Se refiere a las cuatro operaciones básicas que se pueden realizar sobre los datos almacenados, típicamente en una base de datos o sistema de gestión de datos.

Ejercicio 6

Implementa el siguiente algoritmo. Crud

```
import csv
import os

archivo = "notas.csv"

# Crear archivo con encabezados si no existe
if not os.path.exists(archivo):
    with open(archivo, mode="w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["estudiante", "promedio"])

def mostrar_notas():
    print("\n LISTA DE NOTAS")
    with open(archivo, mode="r", newline="") as f:
        reader = csv.reader(f, delimiter=';')
        for i, row in enumerate(reader):
            if len(row) < 2: continue  # Evita filas vacías o corruptas
            print(f"{i}. {row[0]} - {row[1]}") if i != 0 else print("Encabezados:", row)

```

```python
def agregar_nota():
    nombre = input("Nombre del estudiante: ")
    promedio = input("Nota promedio: ")
    with open(archivo, mode="a", newline="") as f:
        writer = csv.writer(f, delimiter=';')
        writer.writerow([nombre, promedio])
    print("Nota agregada correctamente.")

def eliminar_nota():
    mostrar_notas()
    fila = int(input("Número de fila a eliminar (no incluyas encabezado = fila 0): "))

    with open(archivo, mode="r", newline="") as f:
        lines = list(csv.reader(f, delimiter=';'))

    if 1 <= fila < len(lines):
        eliminado = lines.pop(fila)
        with open(archivo, mode="w", newline="") as f:
            writer = csv.writer(f, delimiter=';')
            writer.writerows(lines)
        print(f"Registro eliminado: {eliminado[0]} - {eliminado[1]}")
    else:
        print("Índice inválido. Intenta nuevamente.")


# Menú principal
while True:
    print("\n====== CRUD DE NOTAS ======")
    print("1. Ver notas")
    print("2. Agregar nota")
    print("3. Eliminar nota")
    print("4. Salir")

    opcion = input("Elige una opción: ")

    if opcion == "1":
        mostrar_notas()
    elif opcion == "2":
        agregar_nota()
    elif opcion == "3":
        eliminar_nota()
    elif opcion == "4":
        print("Adiós.")
        break
    else:
        print("Opción no válida. Intenta nuevamente.")
```

DETECCIÓN DE UN COLOR DETERMINADO

Ejercicio 7

Implementa el siguiente algoritmo. Detección del color rojo sobre una imagen

pip install opencv-python - pip install opencv-contrib-python - pip show opencv-python(Comprobar version)

```python
import cv2
import numpy as np
import pandas as pd

ruta_csv = r"C:\Users\DELL\Desktop\python\senati\1_numpy_panda\colores_detectados.csv"
ruta_imagen = r"C:\Users\DELL\Desktop\python\senati\1_numpy_panda\pepe2.jpg"

# 1. Cargar imagen
imagen = cv2.imread(ruta_imagen)
if imagen is None:
    print("No se pudo cargar la imagen.")
    exit()

# 2. Convertir la imagen a espacio de color HSV
hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)

# 3. Definir rango para color rojo (en HSV)
rojo_bajo = np.array([170, 200, 70])    # límite inferior
rojo_alto = np.array([180, 255, 255])  # límite superior

# 4. Crear máscara usando NumPy
mask = cv2.inRange(hsv, rojo_bajo, rojo_alto)

# 5. Encontrar contornos (áreas rojas)
contornos, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Lista para guardar resultados
resultados = []

 # 6. Recorrer contornos detectados
 for cnt in contornos:
     area = cv2.contourArea(cnt)
     if area > 300:  # evitar puntos muy pequeños
         x, y, w, h = cv2.boundingRect(cnt)
         cv2.rectangle(imagen, (x, y), (x+w, y+h), (0, 0, 255), 2)
         resultados.append({"Posición_X": x, "Posición_Y": y, "Área": area})

 # 7. Guardar en CSV con Pandas
 df = pd.DataFrame(resultados)
 df.to_csv(ruta_csv, index=False)

 # 8. Mostrar imagen y máscara
 cv2.imshow("Imagen Original", imagen)
 cv2.imshow("Máscara Rojo", mask)
 cv2.waitKey(0)
 cv2.destroyAllWindows()

 print("Datos guardados en 'colores_detectados.csv'")

 # Esquema HSV
 #https://omes-va.com/wp-content/uploads/2019/09/gyuw4.png
```