Nombre de la práctica	Introducción a Pandas			No.	2
Asignatura:	Simulación	Carrera:	Ingeniería en sistemas	Duración de la práctica (Hrs)	

Nombre del alumno: Jesús Navarrete Martínez
Grupo: 3501
II. Lugar de realización de la práctica (laboratorio, taller, aula u otro): Aula
III. Material empleado:

- Equipo de computo

IV. Desarrollo de la práctica:

Introducción a pandas.

Pandas es una biblioteca que proporciona estructuras de datos y herramientas de análisis de datos de alto rendimiento y fáciles de usar.

- La estructura de datos principal es el DataFrame, que puede considerarse como una tabla 2D en memoria (como una hoja de cálculo, nombres de columna y etiquetas de fila).
- Muchas funciones disponibles en Excel están disponibles mediante programación como crear las tablas dinámicas, calcular columnas basadas en otras columnas, trazar gráficos, etc.
- Proporciona un alto rendimiento para manipular (unir, dividir, modificar, etc.) grandes volúmenes de datos.

Import

```
import pandas as pd
```

Estructura de datos en pandas

La biblioteca pandas de manera genérica, contiene las siguientes estructuras de datos:

- **Series**: es un array de una dimensión.
- **DataFrame**: se corresponde con una tabla de dos dimensiones.
- **Panel**: Similar a un diccionario de DataFrames

Creación del objeto series

```
# Creacion del objeto Series.
s=pd.Series([2,4,6,8,10])
print(s)
```

```
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

```
# Creacion de un objeto series e inicializarlo con un diccionario de python
Altura = {"Emilio": 169, "Anel": 145, "Chucho": 170, "Jocelin": 170}
s = pd.Series(Altura)
print(s)
```

```
Emilio     169
Anel       145
Chucho     170
```

```
Jocelin    170
dtype: int64
```

```
# Creacion de un objeto Series e inicializarlo con algunos elementos
de un diccionario de python.
Altura = {"Emilio": 169, "Anel": 145, "Chucho": 170, "Jocelin": 170}
s = pd.Series(Altura, index = ["Jocelin", "Emilio"])
print(s)
```

```
Jocelin    170
Emilio     169
dtype: int64
```

```
# Creacion de un objeto  Series  e inicializarlo con un escalar
s = pd.Series(34,["Num1","Num2", "Num3", "Num4"])
print(s)
```

```
Num1     34
Num2     34
Num3     34
Num4     34
dtype: int64
```

Acceso a los elementos de un array

Cada elemento en un objeto Series tiene un identificador que se denomina **index label**

```
#crear un objeto Series
s= pd.Series([2,4,6,8], index=["Num1","Num2","Num3","Num4"])
print(s)
```

```
Num1     2
Num2     4
Num3     6
Num4     8
dtype: int64
```

```
# Acceder al tercer elemento del objeto
s["Num3"]
```

```
np.int64(6)
```

```
# También se puede acceder por posición.
s[2]
```

```
/tmp/ipykernel_5052/1191963456.py:2: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer
keys will always be treated as labels (consistent with DataFrame
behavior). To access a value by position, use `ser.iloc[pos]`
  s[2]
```

```
np.int64(6)

# loc es la forma estandar de acceder a un elemento de un objeto Series por atributo
s.loc["Num3"]

np.int64(6)

# iloc es la forma estandar de acceder a un elemento de un objeto Series por posicion
s.iloc[2]

np.int64(6)

# Accediendo al segundo y tercer elemento por posicion
s.iloc[2:4]

Num3    6
Num4    8
dtype: int64
```

Operaciones aritméticas con Series

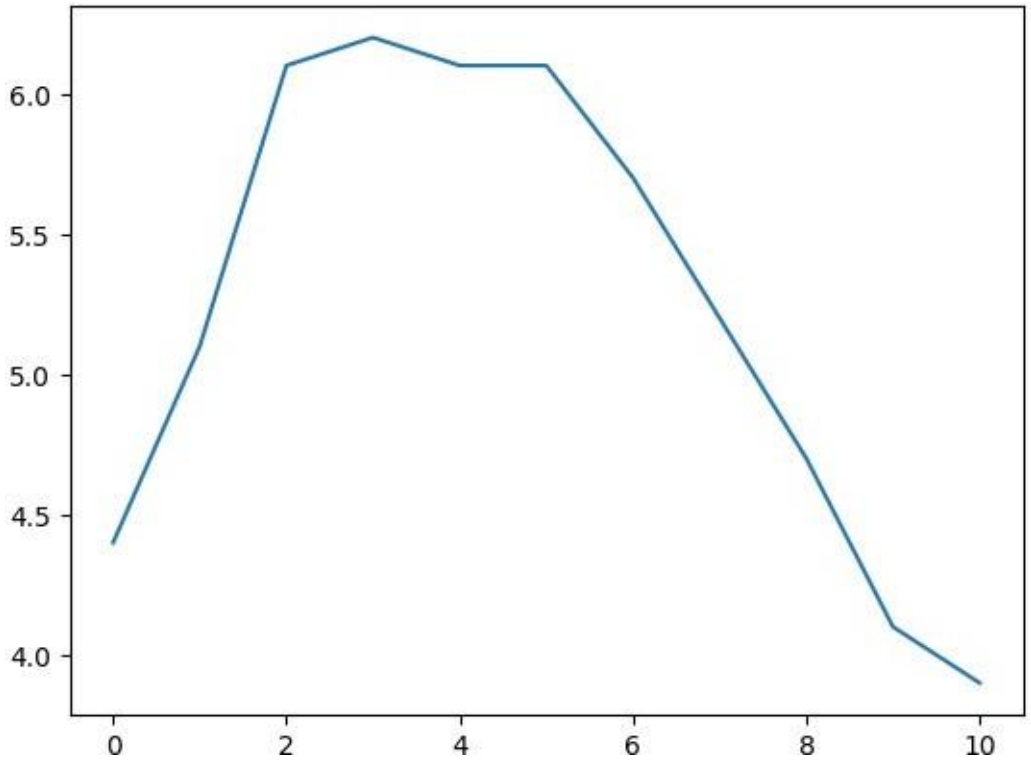
```
# Crear un objeto series
s = pd.Series([2,4,6,8,10])
print(s)

0     2
1     4
2     6
3     8
4    10
dtype: int64

# los objetos series son similares y compatibles con los Arrays de Numpy
import numpy as np
# ufunc de Numpy para sumar los elementos
np.sum(s)

np.int64(30)

s *2

0     4
1     8
2    12
3    16
4    20
dtype: int64
```

Representación grafica de un objeto Series

```
# Crear un objeto Series denominado Temperaturas
Temperaturas = [4.4, 5.1, 6.1, 6.2, 6.1, 6.1, 5.7, 5.2, 4.7, 4.1,3.9]
s = pd.Series(Temperaturas, name="Temperaturas")
s
```

```
0     4.4
1     5.1
2     6.1
3     6.2
4     6.1
5     6.1
6     5.7
7     5.2
8     4.7
9     4.1
10    3.9
Name: Temperaturas, dtype: float64
```

```
# Representación grafica del objeto Series
%matplotlib inline
import matplotlib.pyplot as plt
s.plot()
plt.show()
```



Creación de un objeto DataFrame

```
# Creacion de un DataFrame e inicializarlo con un diccionario de
objetos Series
Personas = {
    "Peso":pd.Series([72,60,74,73],
["Emilio","Anel","Chucho","Jocelin"]),
    "Altura": pd.Series ({"Emilio":169,"Anel":145,"Chucho":170,
"Jocelin":170}),
    "Mascotas": pd.Series([2,9], ["Anel", "Jocelin"])
}

df = pd.DataFrame(Personas)
df
```

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

Es posible forzar el DataFrame a que presente determinadas columnas y el orden determinado

```
# Creación de un DataFrame e inicializarlo con un diccionario de
objetos Series
Personas = {
    "Peso":pd.Series([72,60,74,73],
["Emilio","Anel","Chucho","Jocelin"]),
    "Altura": pd.Series ({"Emilio":169,"Anel":145,"Chucho":170,
"Jocelin":170}),
    "Mascotas": pd.Series([2,9], ["Anel", "Jocelin"])
}
df = pd.DataFrame(
    Personas,
    columns = ["Altura", "Peso"],
    index = ["Chucho", "Emilio"])
df
```

	Altura	Peso
Chucho	170	74
Emilio	169	72

```
# Creación de un DataFrame e inicializarlo con una lista de listas de
python
# Nota: Deben especificarse las columnas e índices por separado.
Valores = [
    [169,3,72],
    [145, 2, 60],
    [170, 1 ,74]
```

```
]
df = pd.DataFrame(
    Valores,
    columns = ["Altura", "Mascotas", "Peso"],
    index = ["Jocelin", "Emilio", "Anel"]
)
df
```

	Altura	Mascotas	Peso
Jocelin	169	3	72
Emilio	145	2	60
Anel	170	1	74

```
# Creación de un DataFrame e inicializarlo con un diccionario de
Python
Personas = {
    "Peso":{"Emilio":72,"Anel":60,"Chucho":74,"Jocelin":73},
    "Altura": {"Emilio":169,"Anel":145,"Chucho":170, "Jocelin":170} }
df= pd.DataFrame(Personas)
df
```

	Peso	Altura
Emilio	72	169
Anel	60	145
Chucho	74	170
Jocelin	73	170

Acceso a los elementos de un DataFrame

```
# Creacion de un DataFrame e inicializarlo con un diccionario de
Python.
Personas = {
    "Peso":pd.Series([72,60,74,73],
["Emilio","Anel","Chucho","Jocelin"]),
    "Altura": pd.Series ({"Emilio":169,"Anel":145,"Chucho":170,
"Jocelin":170}),
    "Mascotas": pd.Series([2,9], ["Anel", "Jocelin"])
}
df= pd.DataFrame(Personas)
df
```

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

Acceso a los elementos de las columnas del DataFrame

```
df ["Peso"]
```

```
Anel        60
Chucho      74
Emilio      72
Jocelin     73
Name: Peso, dtype: int64
```

```
df [["Peso", "Altura"]]
```

```
          Peso  Altura
Anel        60     145
Chucho      74     170
Emilio      72     169
Jocelin     73     170
```

```
# Pueden combinarse los elementos anteriores con expresiones
booleanas.
df["Peso"] > 73
```

```
Anel        False
Chucho       True
Emilio      False
Jocelin     False
Name: Peso, dtype: bool
```

```
#Pueden combinarse los metodos anteriores con expresiones boolenas y
mostrar el DataFrame.
df [df["Peso"]>72]
```

```
          Peso  Altura  Mascotas
Chucho      74     170       NaN
Jocelin     73     170       9.0
```

Accediendo a los elementos de las filas del DataFrame

```
# Mostrar el DataFrame
df
```

```
          Peso  Altura  Mascotas
Anel        60     145       2.0
Chucho      74     170       NaN
Emilio      72     169       NaN
Jocelin     73     170       9.0
```

```
df.loc["Emilio"]
```

```
Peso          72.0
Altura       169.0
Mascotas       NaN
Name: Emilio, dtype: float64
```

```
df.iloc[1:3]
```

```
         Peso   Altura   Mascotas
Chucho     74      170        NaN
Emilio     72      169        NaN
```

Consulta avanzada de los elementos de un DataFrame

```
# Mostrar el DataFrame
df
```

```
          Peso   Altura   Mascotas
Anel        60      145        2.0
Chucho      74      170        NaN
Emilio      72      169        NaN
Jocelin     73      170        9.0
```

```
df.query("Altura >= 170 and Peso > 73")
```

```
         Peso   Altura   Mascotas
Chucho     74      170        NaN
```

Copiar un DataFrame

```
# Crear un DataFrame e inicializarlo con un diccionario de objetos
Series
Personas = {
    "Peso":pd.Series([72,60,74,73],
["Emilio","Anel","Chucho","Jocelin"]),
    "Altura": pd.Series ({"Emilio":169,"Anel":145,"Chucho":170,
"Jocelin":170}),
    "Mascotas": pd.Series([2,9], ["Anel", "Jocelin"])
}
df= pd.DataFrame(Personas)
df
```

```
          Peso   Altura   Mascotas
Anel        60      145        2.0
Chucho      74      170        NaN
Emilio      72      169        NaN
Jocelin     73      170        9.0
```

```
# Copia del DataFrame en DataFrame  df en df_copy
#Nota: al modificar un elemento del df_copy no se modifica el df.
df_copy = df.copy()
```

Modificación de un DataFrame

```
# añadir una nueva columna al DataFrame
df["Anio_Nac"]= [2004,2004, 2004, 2004]
df
```

	Peso	Altura	Mascotas	Anio_Nac
Anel	60	145	2.0	2004
Chucho	74	170	NaN	2004
Emilio	72	169	NaN	2004
Jocelin	73	170	9.0	2004

```
# añadir una nueva columna calculada al DataFrame
df["Edad"] = 2024- df["Anio_Nac"]
df
```

	Peso	Altura	Mascotas	Anio_Nac	Edad
Anel	60	145	2.0	2004	20
Chucho	74	170	NaN	2004	20
Emilio	72	169	NaN	2004	20
Jocelin	73	170	9.0	2004	20

```
# Añadir una nueva columna creando un DataFrame nuevo
df_mod = df.assign(Hijos= [2,1,2,1])
df_mod
```

	Peso	Altura	Mascotas	Anio_Nac	Edad	Hijos
Anel	60	145	2.0	2004	20	2
Chucho	74	170	NaN	2004	20	1
Emilio	72	169	NaN	2004	20	2
Jocelin	73	170	9.0	2004	20	1

```
# Eliminar una columna existente del DataFrame
del df["Peso"]

df
```

	Altura	Mascotas	Anio_Nac	Edad
Anel	145	2.0	2004	20
Chucho	170	NaN	2004	20
Emilio	169	NaN	2004	20
Jocelin	170	9.0	2004	20

```
# Eliminar una columna existente, devolviendo una copia del DataFrame resultante.
df_mod= df_mod.drop(["Hijos"], axis=1)
df_mod
```

	Peso	Altura	Mascotas	Anio_Nac	Edad
Anel	60	145	2.0	2004	20
Chucho	74	170	NaN	2004	20
Emilio	72	169	NaN	2004	20
Jocelin	73	170	9.0	2004	20

Evaluación de expresiones sobre un DataFrame.

```
# Crear un DataFrame e inicializarlo con un diccionario de objetos
Series.
Personas = {
    "Peso":pd.Series([72,60,74,73],
["Emilio","Anel","Chucho","Jocelin"]),
    "Altura": pd.Series ({"Emilio":169,"Anel":145,"Chucho":170,
"Jocelin":170}),
    "Mascotas": pd.Series([2,9], ["Anel", "Jocelin"])
}
df= pd.DataFrame(Personas)
df
```

```
          Peso  Altura  Mascotas
Anel        60     145       2.0
Chucho      74     170       NaN
Emilio      72     169       NaN
Jocelin     73     170       9.0
```

```
# Evaluar una función sobre una columna del DataFrame.
df.eval("Altura /2")
```

```
Anel        72.5
Chucho      85.0
Emilio      84.5
Jocelin     85.0
Name: Altura, dtype: float64
```

```
# Evaluar una función utilizando una variable local
max_altura  =165
df.eval("Altura > @max_altura")
```

```
Anel        False
Chucho       True
Emilio       True
Jocelin      True
Name: Altura, dtype: bool
```

```
# Aplicar una función a una columna del DataFrame.
def func(x):
    return x+2
df["Peso"].apply(func)
```

```
Anel        62
Chucho      76
Emilio      74
Jocelin     75
Name: Peso, dtype: int64
```

Guardar y cargar el DataFrame

```
# Crear un DataFrame e inicializarlo con un diccionario de objetos Series.
Personas = {
    "Peso":pd.Series([72,60,74,73],
["Emilio","Anel","Chucho","Jocelin"]),
    "Altura": pd.Series ({"Emilio":169,"Anel":145,"Chucho":170,
"Jocelin":170}),
    "Mascotas": pd.Series([2,9], ["Anel", "Jocelin"])
}
df= pd.DataFrame(Personas)
df
```

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

```
# Guardar el DataFrame como CSV,HTML,JSON.
df.to_csv("DataFrame_Personas.csv")
df.to_html("DataFrame_Personas.html")
df.to_json("DataFrame_Personas.json")

# Cargar el DataFrame en Jupyter
df2=pd.read_csv("DataFrame_Personas.csv")

df2
```

	Unnamed: 0	Peso	Altura	Mascotas
0	Anel	60	145	2.0
1	Chucho	74	170	NaN
2	Emilio	72	169	NaN
3	Jocelin	73	170	9.0

```
# Cargar el DataFrame con la primera columna correctamente asignada
df2 =pd.read_csv("DataFrame_Personas.csv", index_col=0)
df2
```

	Peso	Altura	Mascotas
Anel	60	145	2.0
Chucho	74	170	NaN
Emilio	72	169	NaN
Jocelin	73	170	9.0

V. Conclusiones:

En esta práctica, hemos explorado los conceptos fundamentales de **Pandas**, una herramienta esencial para el manejo y análisis de datos en Python. Aprendimos a crear y manipular **DataFrames** y **Series**, las estructuras principales que permiten organizar y analizar datos de manera eficiente. También vimos cómo aplicar funciones para limpiar, transformar y filtrar datos, lo cual es crucial en cualquier flujo de trabajo de análisis.

Pandas ofrece una amplia gama de funcionalidades que facilitan la exploración y el procesamiento de grandes conjuntos de datos. Su capacidad para integrarse con otras bibliotecas como NumPy y su flexibilidad en la manipulación de datos lo convierten en una herramienta indispensable en campos como el análisis de datos, la ciencia de datos y la ingeniería de datos. Con estas habilidades, estamos mejor preparados para abordar desafíos más complejos y realizar análisis más avanzados en futuros proyectos.