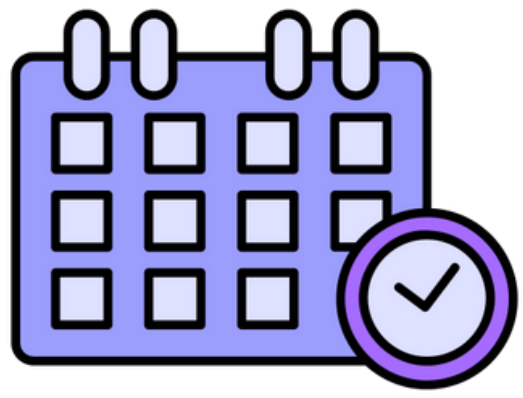
Lección 1: Introducción a Pandas

Planteamiento de la sesión

Tiempo de ejecución: 4 horas



Hasta el momento se ha repasado el trabajo con archivos de texto, pero es conocido que el trabajo más común en el día a día es utilizando hojas de cálculo, algo como tablas en excel o en google sheets.

Python incluye una librería que permite cargar archivos muy similares a una hoja de cálculo y que, generalmente, se generan a partir de ellos, y son los archivos CSV (o archivos de valores separados por comas, por sus siglas en inglés).

Básicamente un archivo .CSV es un listado de datos (con o sin encabezado) en donde cada fila cuenta con los valores de los elementos separados por comas.

Por ejemplo, un archivo csv de la temperatura de la semana se ve así:

```
day,temp,condition
Monday,12,Sunny
Tuesday,14,Rain
Wednesday,15,Rain
Thursday,14,Cloudy
Friday,21,Sunny
Saturday,22,Sunny
Sunday,24,Sunny
```

Recomendación para lectura sencilla, instalar en VS Code Rainbow CSV - Visual Studio Marketplace

La librería de Python que permite trabajar con archivos CSV tiene el mismo nombre y solo es necesario importarla

```
import csv
```

La librería tiene un método reader() que crea un iterable con cada una de las líneas del archivo CSV que puede ser recorrido en un ciclo for…in.

```
import csv

with open('csv_file.csv') as data_file:
 #Se recorre el archivo y se crea un iterable en data
 data = csv.reader(data_file)
 for row in data:
 print(row)
```

Completar la información con el contenido del siguiente archivo hasta la Ejercitación CSV

M2 _integrador_Pandas.ipynb

Materiales:

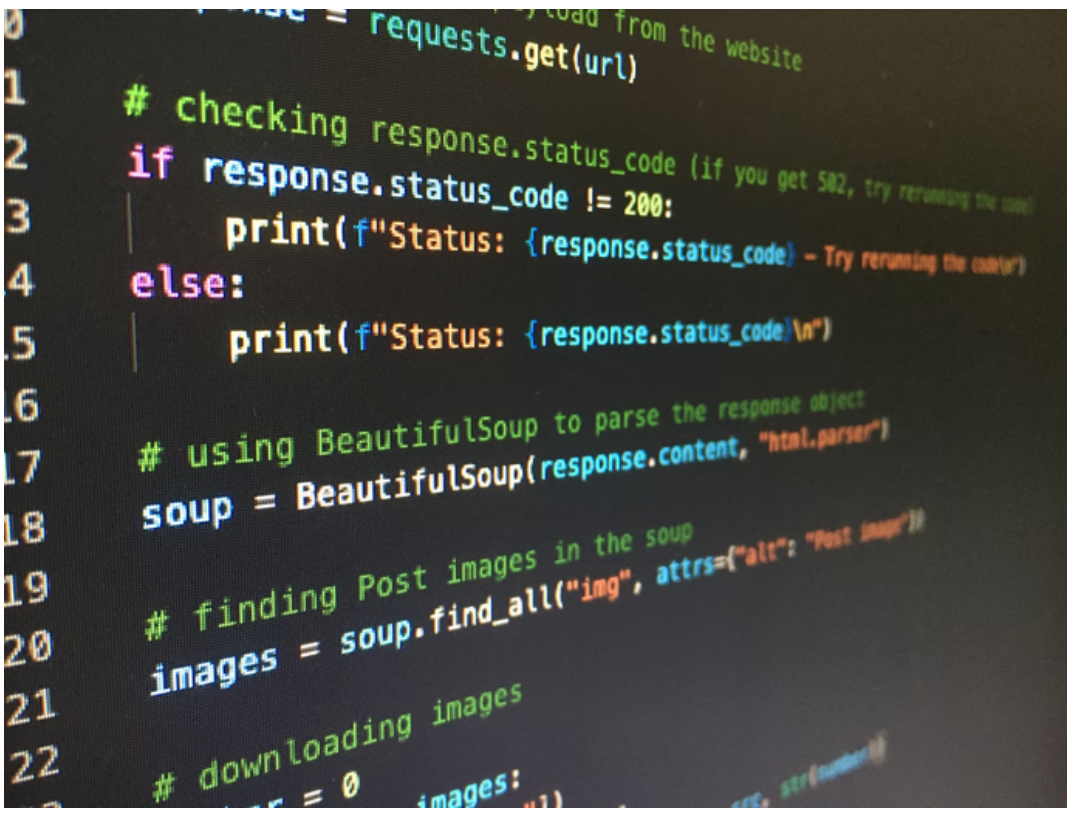
- M2 _integrador_Pandas.ipynb
- Pandas
- Python Tutor
- Data Wrangling - with pandas Cheat Sheet http://pandas.pydata.org
- La librería Pandas | Aprende con Alf
- La librería Matplotlib | Aprende con Alf
- User Guide — pandas 2.2.0 documentation
- pandas.Panel — pandas 0.23.4 documentation
- Getting started — pandas 2.2.0 documentation
- NumPy documentation — NumPy v1.26 Manual
- Creating a Pandas Series - GeeksforGeeks
- Python Pandas Series - GeeksforGeeks
- DataFrame — pandas 2.2.0 documentation
- https://data.cityofnewyork.us/Environment/2018-Central-Park-Squirrel-Census-Squirrel-Data/vfnx-vebw/about_data

¿Qué es Pandas?

Pandas es una librería de Python usada para trabajar con conjuntos de datos o data sets. Dentro de sus funciones cuenta con análisis, limpieza, búsqueda y manipulación de datos. Su nombre hace referencia a “Panel Data” y “Python Data Analysis”. En resumen, es una librería especializada en el manejo y análisis de estructuras de datos.

Las principales características de esta librería son:

- Define nuevas estructuras de datos basadas en los arrays de la librería NumPy pero con nuevas funcionalidades.
- Permite leer y escribir fácilmente ficheros en formato CSV, Excel y bases de datos SQL.
- Permite acceder a los datos mediante índices o nombres para filas y columnas.
- Ofrece métodos para reordenar, dividir y combinar conjuntos de datos.
- Permite trabajar con series temporales.
- Realiza todas estas operaciones de manera muy eficiente.



Estas características permiten concluir que con pandas es posible obtener conclusiones del análisis de grandes cantidades de datos, a través de métodos que ayudan a limpiar los data sets y obtener de ellos información leíble y relevante. Elementos que son la base para la ciencia de datos **(La ciencia de datos es una rama de las ciencias computacionales que estudia cómo almacenar y analizar datos para obtener información valiosa de ellos).**

Algunas respuestas sencillas que puede dar Pandas son, por ejemplo, encontrar la correlación entre varias columnas, valores promedio, mínimo, máximo o limpiar datos no relevantes, vacíos o nulos.

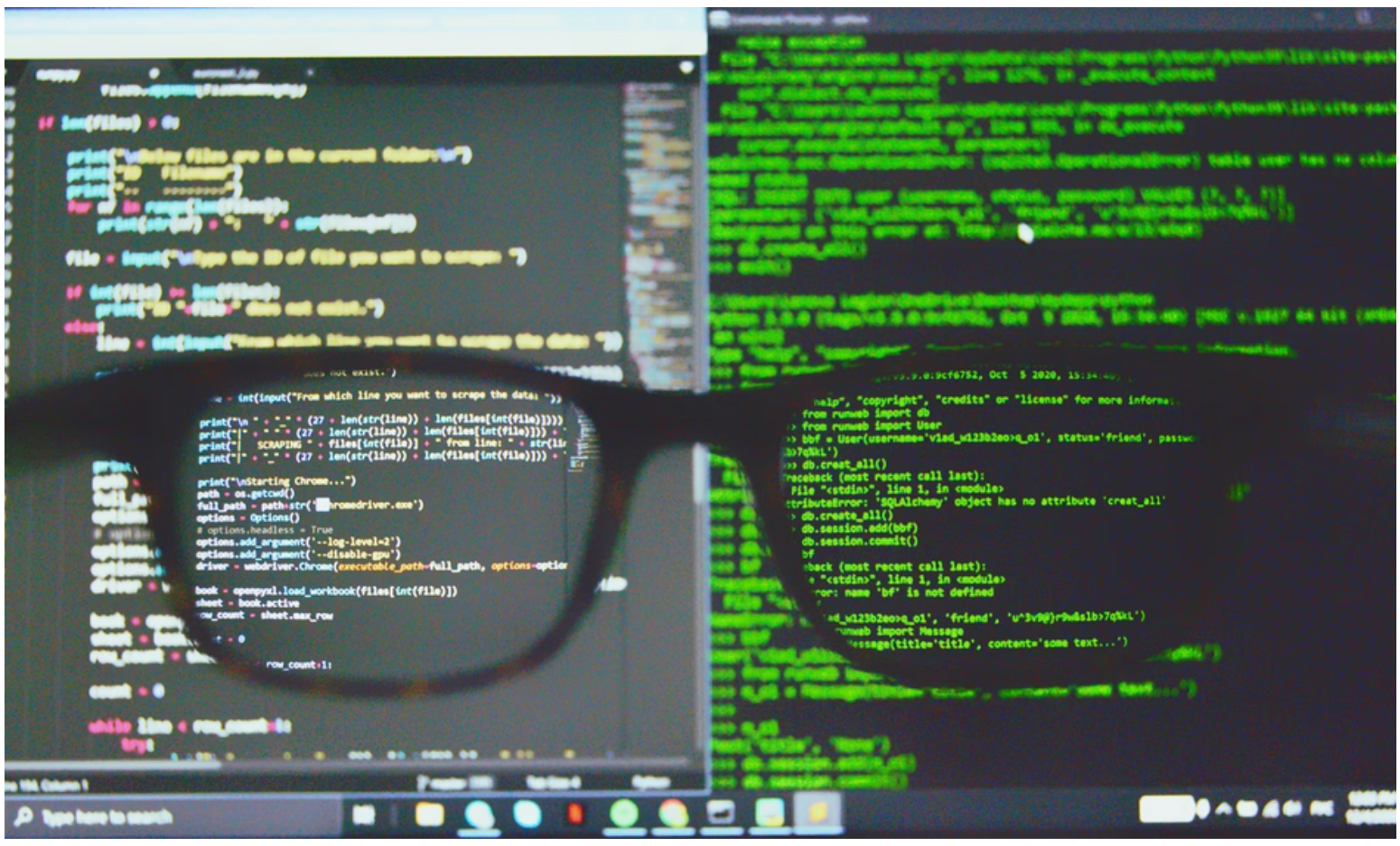
Un excelente ejercicio en este punto es revisar el Getting Started de la documentación oficial de Pandas Getting started — pandas 2.2.0 documentation, allí se pueden revisar ejemplos rápidos para preguntas frecuentes de operaciones comunes.

El código fuente de pandas se encuentra en un repositorio de github (herramienta que se revisará más adelante en el curso) y puede ser consultado en el enlace GitHub - pandas-dev/pandas: Flexible and powerful data analysis / manipulation library for Python, providing labeled data structures similar to R data.frame objects, statistical functions, and much more.

¿Qué tal realizar la misma ejercitación ahora utilizando Pandas y su método read_csv()?

Complementar aquí con el contenido entre la ejercitación anterior y la ejercitación llamada **Ejercitación CSV ahora con Pandas**.

M2 _integrador_Pandas.ipynb



Tipos de datos de Pandas

Pandas aumenta los tipos de datos o estructuras de datos que se han visto hasta el momento, para apoyarse en el proceso y añadir nuevas funcionalidades, las estructuras se construyen a partir de arrays (“listas”) de la librería NumPy.

Se dispone de tres estructuras de datos diferentes:

- Series: Estructura de una dimensión.
- DataFrame: Estructura de dos dimensiones (tablas).
- Panel: Estructura de tres dimensiones (cubos).

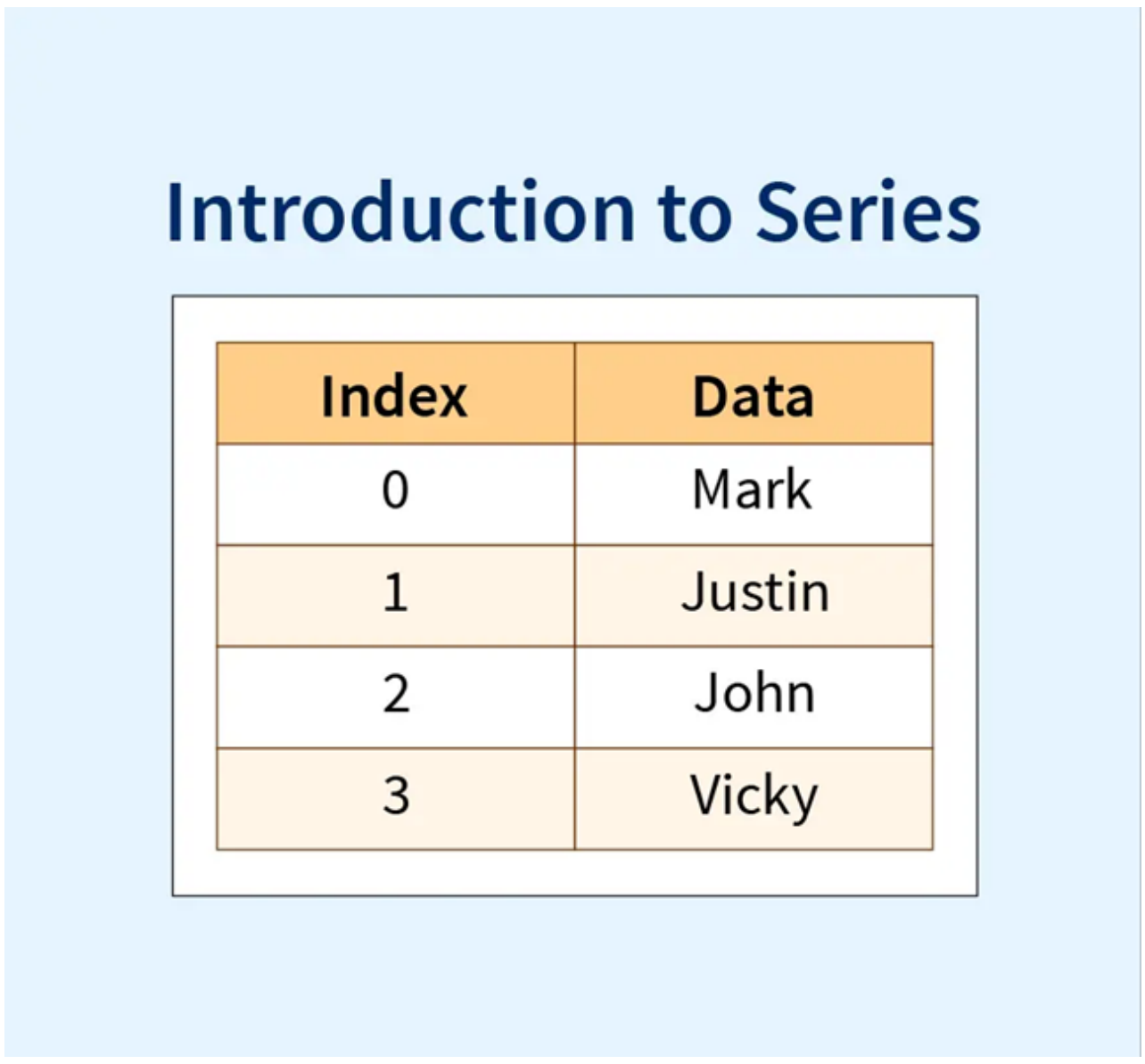
Para el desarrollo del curso, todos los proyectos y explicaciones se basan en los dos primeros tipos de datos del listado, estas son conocidas como las estructuras de datos básicas. Puede encontrar más información sobre los paneles en Pandas en el enlace pandas.Panel — pandas 0.23.4 documentation

Series

Las series en Pandas son similares a los arrays o listas de una sola dimensión, pueden almacenar datos de cualquier tipo pero deben ser homogéneas, es decir que cada uno de sus elementos es del mismo tipo. Se puede asemejar a una columna en una tabla

Las series cuentan con otra característica y es que su tamaño es inmutable, esto quiere decir que la longitud de una serie no se puede modificar, lo que sí es posible modificar es su contenido.

Dispone de un índice que se asocia con cada elemento y que permite acceder al mismo, en caso de no indicarse el índice, automáticamente se enumeran los elementos con la convención 0-index. Otro de los elementos que se le asigna a la hora de creación de una serie es el tipo de dato o data type, en caso de no ser definido, se infiere basado en los datos incluidos.



Pandas tutorial, consultado en febrero de 2024, disponible en [Python Pandas Series Tutorial (with Examples) - Scaler Topics]

Como se puede apreciar en la imagen anterior, no se le ha indicado un índice, ya que está tomando los valores por defecto, para crear una serie así, solo sería necesario:

1. Importar la librería
2. Invocar a su método de Series()
3. Indicar la data
4. Asignar el data type

```
import pandas as pd
s = pd.Series(['Mark', 'Justin', 'John', 'Vicky'], dtype='string')
 print(s)
```

Salida

```
0   Mark
1  Justin
2   John
3  Vicky
dtype: string
```

Creación de series desde arrays

El ejemplo anterior define la creación de series utilizando listas o “arrays”, como puede recordar, las listas son 0-index, es decir que el primer elemento se almacena en la posición 0 y el último en la posición n-1, siendo n la longitud de la lista.

Tomando como referencia los valores por defecto, la serie asigna dichos índices a no ser que se le indique lo contrario, como por ejemplo:

```
s = pd.Series(['Mark', 'Justin', 'John', 'Vicky'], index=[222,333,444,555],
dtype='string')
 print(s)
```

Salida:

```
222   Mark
333  Justin
444   John
555  Vicky
dtype: string
```

¿O qué tal algo como lo siguiente, qué salida se obtendría?

```
notas = [5.7,8.5,9.1,5.5,8.2,9.0,10,7.0,7.7,9.9]
estudiantes =
["Juan","Jenifer","David","Pablo","Armando","Magdalena","Francesca","Rosmer
y","Vicente","Martin"]

pd.Series(notas, index=estudiantes)
```

Se ha puesto la palabra “arrays” entre comillas, porque como se sabe, Python tiene una estructura array-like que son las listas, sin embargo para utilizar propiamente los arrays es necesario importar numpy.

NumPy es una librería que adiciona arrays de múltiples dimensiones, y hace parte del paquete fundamental para la computación científica. Se conoce como un estándar a la hora de trabajar con datos numéricos en Python.

Puede encontrar toda la documentación de la herramienta en el enlace NumPy documentation — NumPy v1.26 Manual.

Complementar la información con los bloques desde **Revisando las Series** hasta la ejercitación llamada **Ejercitación Pasando a np.**

M2 _integrador_Pandas.ipynb

Creación de series desde diccionarios

Otra de las formas para trabajar con series, es crearlas a partir de diccionarios, aquí aparece una muy buena característica de los diccionarios y es su estructura llave:valor, ya que automáticamente se detectan las llaves como los índices y los valores como la data en sí.

Para crear una serie desde un diccionario solo es necesario crear el diccionario y pasarlo por la función Series().

```
#Listado de notas
new_dict = {
 "Matemáticas" : 8.0,
 "Programación" : 7.7,
 "Inglés" : 8.2
}

#Índices automáticos tomados de las llaves del diccionario
s = pd.Series(new_dict)
print(s)
```

Existen varias otras formas de crear Series utilizando funciones que generen el listado de datos indicado. Para ello, su tarea es ejecutar los siguientes códigos para determinar la salida.

- **Utilizando range**

```
import pandas as pd

s = pd.Series(range(10))
print(s)
```

- **Desde un escalar** (se debe indicar el índice, ya que el valor se repetirá por cada uno)

```
import pandas as pd

s = pd.Series(10, index=[0, 1, 2, 3, 4, 5])
 print(s)
```

- **Utilizando linspace de NumPy**

```
import pandas as pd
import numpy as np

s = pd.Series(np.linspace(1, 100, 10))
print(s)
```

- **Utilizando list comprehension**

```
 import pandas as pd

s = pd.Series(range(1,20,3), index=[x for x in 'abcdefg'])
print(s)
```

¿Hay alguna que no haya quedado clara?

¿Cómo generaría una Serie utilizando el módulo random de np?

Ejercitación generando Series

M2 _integrador_Pandas.ipynb

Ahora hay que revisar la documentación
Series — pandas 2.2.0 documentation

La documentación oficial de Pandas es excelente, cuenta con todos los métodos, atributos , funciones, guías y un gran etc, la idea es a partir de las series que se han creado, se explore la documentación y se prueben diferentes métodos de los disponibles en el listado, atributos como el size, index, dtype, values.

```
s.count()
s.sum()
s.min()
s.max()
s.mean()
s.describe()
```

Operaciones con series

A las series también es posible aplicarles operaciones y funciones, por ejemplo multiplicar cada elemento de una serie por un escalar, obtener el módulo, dividir o sumar, incluso entre series.

```
s * 2
s % 2
```

```
s = pd.Series(['a', 'b', 'c'])
s * 5
```

```
s = pd.Series([2, 4, 6])
p = pd.Series([5, 4, 3])

print(s * p)
```

Aplicar funciones a series

Es posible también aplicar funciones a todos los elementos de una serie utilizando el método apply(f), como por ejemplo obtener el cuadrado de cada elemento en una serie de números, o cambiar entre mayúsculas y minúsculas una serie de texto.

```
s = pd.Series(['a', 'b', 'c'])
s.apply(str.upper)
```

Filtrar una serie

Esta operación permite obtener los valores de una serie que cumplan una condición dada.

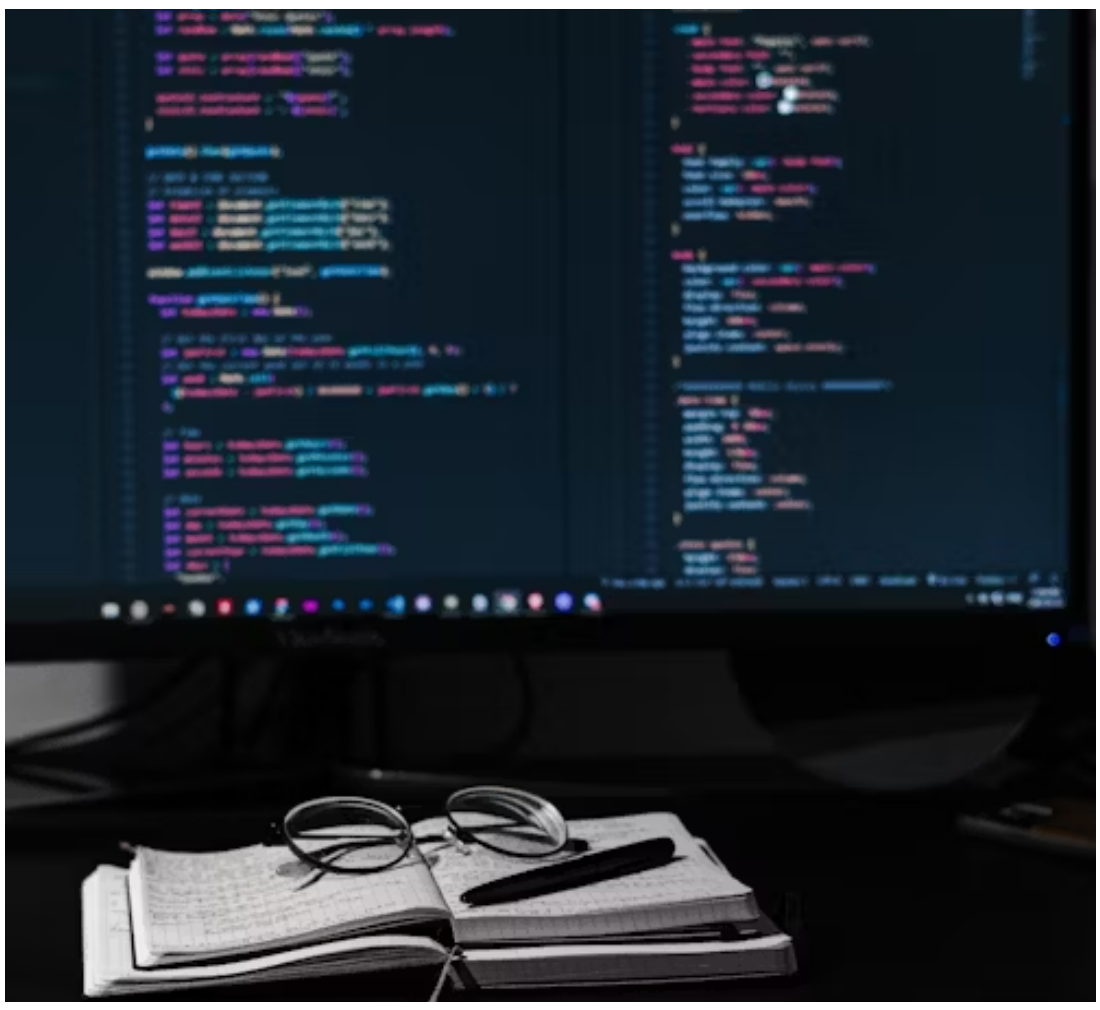
```
s = pd.Series({'Matemáticas': 6.0, 'Economía': 4.5, 'Programación': 8.5})
print(s[s > 5])
```

Ordenar una serie

A diferencia del filtro, es posible ordenar una serie tanto por valores como por índices, puede ser ordenada de forma ascendente o descendente, por defecto se ordena de manera creciente, los métodos son el sort_values() y el sort_index().

```
s = pd.Series({'Matemáticas': 6.0, 'Economía': 4.5, 'Programación': 8.5})
print(s.sort_values())

print(s.sort_index(ascending = False))
```

Eliminar datos nulos y desconocidos



Una serie puede contar con datos nulos None y desconocidos NaN, que a la hora de estar realizando operaciones puede ser un inconveniente. El método que permite hacer dicha limpieza es llamado dropna().

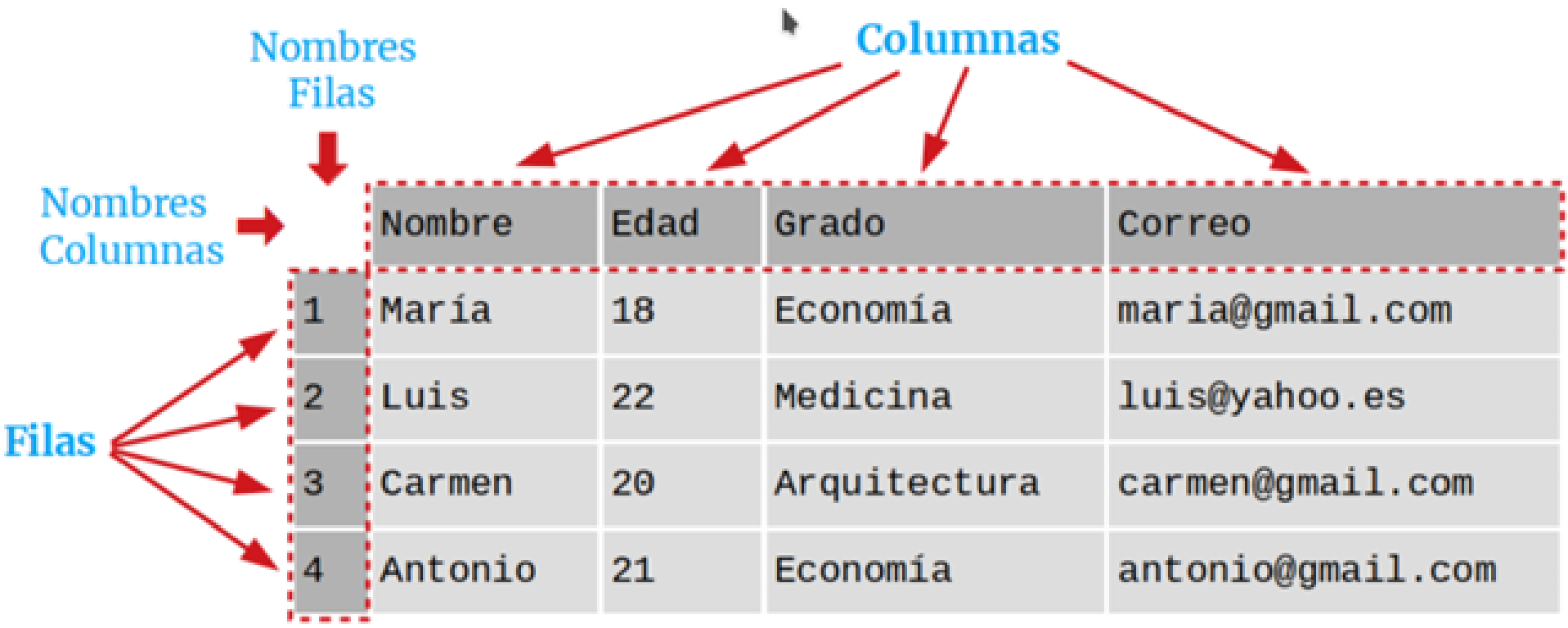
```
s = pd.Series(['a', 'b', None, 'c', np.NaN, 'd'])

s.dropna()
print(s)
```

DataFrame

Hasta el momento se ha trabajado con Series, que son una estructura de una sola dimensión similar a una columna en una tabla, ahora, subiendo un poco la dificultad pero también el uso y aplicaciones en entornos reales, aparecen los DataFrames. Que son conjuntos estructurados de datos en forma de tabla, similar a una tabla de excel.

Las columnas de los Dataframes son de tipo Series, y retomando las características de las Series, los datos de la serie deben ser del mismo tipo, por consiguiente todos los datos de cada columna, deben ser del mismo tipo. Pero si se miran las filas, si pueden contener distintos tipos de datos.

Los Dataframes contienen dos índices, a diferencia del índice único de las Series, un índice para las filas y un índice para las columnas, para acceder a los elementos se utilizan dichos índices.



	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

DataFrame, consultado en febrero de 2024, La librería Pandas | Aprende con Alf.

Creación de un DataFrame

Existen diversidad formas para crear un DataFrame, desde utilizar diccionarios de listas hasta archivos .csv o en excel. Esta característica indica la versatilidad que se puede tener al utilizar la herramienta, y se entiende el por qué los DataFrames son los objetos más comúnmente utilizados de Pandas.

A continuación se presentan las diferentes opciones disponibles para la creación de un DataFrame sencillo.

La estructura base que comparten las diferentes formas de creación de DataFrames es similar y se ve así:

DataFrame(data=estructura_datos, index=filas, columns=columnas, dtype=tipos)

DataFrame desde un diccionario de listas

Utilizando este formato, las claves o llaves del diccionario se convierten en los índices o nombre de las columnas y los datos en la lista se convierten en los valores presentados en las columnas, similar a como se trabaja en las Series, siendo obligatorio que los datos en la lista sean del mismo tipo.

```
import pandas as pd
datos = {'nombre':['María', 'Luis', 'Carmen', 'Antonio'],
'edad':[18, 22, 20, 21],
'grado':['Economía', 'Medicina', 'Arquitectura', 'Economía'],
'correo':['maria@gmail.com', 'luis@yahoo.es', 'carmen@gmail.com',
'antonio@gmail.com']
 }

df = pd.DataFrame(datos)
print(df)
```

En caso de no indicarse los índices de las filas, se inicia igual que con las Series utilizando 0-index, si se utiliza la creación a partir de diccionarios, no se deben indicar los nombres de las columnas ya que son obtenidas de las llaves.

Todas las listas que se utilicen para la creación del DataFrame deben ser del mismo tamaño.

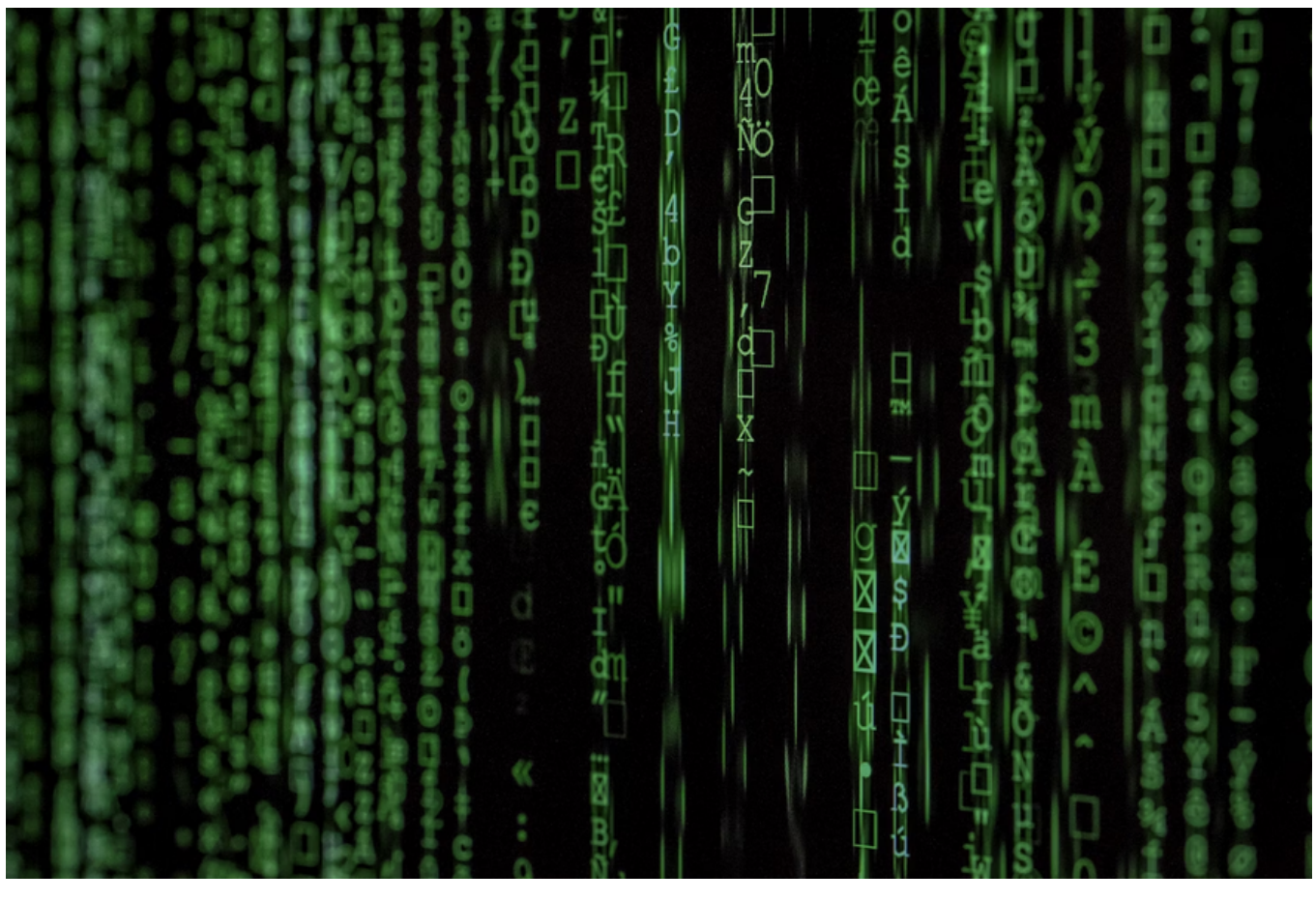
DataFrame desde una lista de listas

Para crear el DataFrame utilizando lista de listas se comparten algunas características, como que deben tener la misma longitud, sin embargo aquí es necesario, si se quiere nombrar, indicar los nombres de las columnas y los índices de las filas.

Los datos de cada una de la lista de listas representan cada una de las columnas que tendrá el DataFrame.

En caso de no indicarse los índices, se utiliza la notación 0-index, tanto para las columnas como para las filas.

```
import pandas as pd
df = pd.DataFrame([['María', 18, 'maria@gmail.com'], ['Luis', 22, 'luis@gmail.com'], ['Carmen', 20, 'carmen@gmail.com']], columns=['Nombre', 'Edad', 'Email'])
print(df)
```



DataFrame desde una lista de diccionarios

La lista de diccionarios es una de las estructuras más utilizadas en muchos lenguajes de programación, ya que permiten almacenar datos de una forma estructurada, que pueden o no compartir una plantilla o forma.

Para la creación de un DataFrame, si las estructuras comparten la forma, es decir las mismas llaves del diccionario, el resultante tendrá todos los valores no nulos, en caso de que alguno de los diccionarios no cuente con las mismas llaves que el resto, es decir que no se comparta la forma, los datos bajo esa llave serán marcados como desconocidos o NaN.

Los índices de las columnas serán tomadas de las llaves de cada diccionario y los índices de las filas, deben ser indicados en la lista de índices.

```
import pandas as pd
df = pd.DataFrame([{'Nombre':'María', 'Edad':18}, {'Nombre':'Luis', 'Edad':22}, {'Nombre':'Carmen'}])
print(df)
```

Como se puede apreciar en el último diccionario de la lista, no cuenta con los datos bajo la llave ‘edad’, por lo que al presentar la información aquí aparecerá un NaN.

DataFrame desde un Array

Al utilizar un array para generar un DataFrame, sus datos se convierten en las columnas de la tabla, el nombre de los índices deben ser especificados utilizando la lista index y columns, en caso de no hacerlo, se asignara según la nomenclatura 0-index.

Si el array es de una sola dimensión se creará un DataFrame similar a una Serie, para array de varias dimensiones, cada array corresponderá con las columnas del DataFrame

```
import pandas as pd
import numpy as np

arr = np.array([1,2,3,4,5], dtype='int')

df = pd.DataFrame((arr, [5,4,3,2,1]), )

print(df)
```

DataFrame desde un archivo CSV o Excel

Para la creación de los DataFrames desde archivos como csv y excel, es necesario utilizar el módulo read_csv() o read_excel(), según sea el caso del tipo de documento.

La estructura de los métodos para la lectura de los archivos es la siguiente:

read_csv("path_to_CSV_file", sep="separator_simbol",
header="number_of_the_column_names".
index_col="number_of_the_rows_id", na_values="NaN_values",
decimal="decimal_deparator")

El único atributo obligatorio a la hora de definir la lectura desde un archivo CSV es el documento, los demás tienen valores por defecto, como por ejemplo utilizar coma(,) como separador, utilizar la primera fila del documento como el header, utilizar notación 0-index para los índices de las filas, los valores no válidos se asignan como NaN y los separadores decimales van con punto(.), para el caso de los documentos tipo excel, el método es similar, solo que añadiendo la opción de definir la hoja con la cual se requiere trabajar.

ead_excel("path_to_CSV_file", sheet_name="name_of_the_sheet",
header="number_of_the_column_names".
index_col="number_of_the_rows_id", na_values="NaN_values",
decimal="decimal_deparator")

import pandas as pd

df = pd.read_csv("weather_data.csv")
print(df.head());

Ahora que ya se conocen diversas formas de crear un DataFrame, es hora de algunas ejercitaciones.

Recuerda siempre tener la documentación a mano, DataFrame — pandas 2.2.0 documentation.

El enunciado de cada ejercitación se encuentra en el documento:
M2 _ integrador Pandas.ipynb

- **Ejercitación Atributos DataFrames**
- **Ejercitación Obteniendo el promedio**
- **Ejercitación Creando un DataFrame**

Para la **Ejercitación contando ardillas,** debe descargar el dataset correspondiente al censo de ardillas del Central Park desde el siguiente enlace
https://data.cityofnewyork.us/Environment/2018-Central-Park-Squirrel-Census-Squirrel-Data/vfnx-vebw/about_data

Para la **Ejercitación List & Dict Comprehension,** es necesario iniciar con el listado en formato CSV de las letras y sus códigos, puede revisar el listado completo en el enlace: Alfabeto radiofónico - Wikipedia, la enciclopedia libre

Más acciones sobre columnas en un DataFrame

Existen más operaciones que son comunes a la hora de trabajar con DataFrames, entre ellos, la actualización de los nombres de las columnas y el cambio del índice de la tabla.

Cambiando el nombre de las columnas

Se debe entregar un diccionario donde se tenga el nombre anterior y el nombre nuevo de cada columna a modificar:

```
# df.rename(columns=columnas, index=filas)

import pandas as pd
df = pd.read_csv(
'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

print(df.rename(columns={'nombre':'nombre y apellidos', 'altura':'estatura'},
index={0:1000, 1:1001, 2:1002}).head())
```

Cambiando el índice

Cuando no se asigna un índice a la hora de la creación del DataFrame, se ha evidenciado que se asigna la notación 0-index, sin embargo, es posible cambiar ese índice a una columna diferente, verificando que hayan o no valores repetidos, en su propiedad verify_integrity.

```
#df.set_index(keys = columnas, verify_integrity = bool)

import pandas as pd
df = pd.read_csv(
'https://raw.githubusercontent.com/asalber/manual-
python/master/datos/colesterol.csv')

print(df.set_index("nombre").head())
```

Acceder a un elemento en el DataFrame

Existen diferentes métodos que permiten obtener los valores de una posición en particular, o de una fila o de una columna, algunos como iloc() ya se encuentra depreciado pero .at o .loc, pueden realizar la función.

```
#df.iloc[fila, columna]

import pandas as pd
df = pd.read_csv(
'https://raw.githubusercontent.com/asalber/manual-
python/master/datos/colesterol.csv')

#print(df)

print(df.loc[1], end='\n\n')

print(df.loc[2, 'edad'])
print(df.loc[[2, 5]])
```

Añadir una nueva columna al DataFrame

Puede que en algún momento, sea necesario agregar más información a un DataFrame, como una nueva columna, puede ser creada por el usuario o resultado de otra consulta u otro DataFrame.

#df['new_column_name'] = [lista, de, elementos, a, agregar]

```
import pandas as pd
 df = pd.read_csv(
 'https://raw.githubusercontent.com/asalber/manual-
 python/master/datos/colesterol.csv')

 df['diabetes'] = pd.Series([False, False, True, False, True])

 print(df)
```

Eliminar una columna

#df['new_column_name'] = [lista, de, elementos, a, agregar]

```
import pandas as pd
df = pd.read_csv(
'https://raw.githubusercontent.com/asalber/manual-
python/master/datos/colesterol.csv')


df['diabetes'] = pd.Series([False, False, True, False, True])

print(df)

df.pop("diabetes")

print(df)
```

Acciones con filas

Ahora debe buscar en la documentación como agregar nueva filas y removerlas en un DataFrame, como pista, puede trabajar con .append() y .drop()

Ahora, ¿cómo puede ordenar? puede probar los métodos .sort_values() y .sort_index().

¿Puede ordenar por varias columnas?