

## ÍNDICE

1.	Introducción y objetivos .....	2
1.1.	Objetivos:.....	2
1.2.	¿Qué es Python?.....	2
2.	Instalación .....	4
2.1.	Versiones .....	4
2.2.	Distribuciones de Python .....	5
3.	Herramientas .....	9
3.1.	Entornos de desarrollo .....	9
3.2.	Entornos de desarrollo avanzados .....	9
4.	Python – Ejercicios.....	12
	ANEXO.....	28

## **1. Introducción y objetivos**

Python es un lenguaje de programación de alto nivel ampliamente utilizado en el campo del aprendizaje automático o machine learning. Su popularidad se debe a su facilidad de uso, capacidad de procesamiento y amplia variedad de librerías para el análisis de datos y el aprendizaje automático.

En el campo del aprendizaje automático, Python se utiliza principalmente para la preparación y limpieza de datos, la implementación de modelos de aprendizaje automático y la evaluación de su rendimiento. Existen diversas librerías en Python como Scikit-learn, TensorFlow, Keras, PyTorch, entre otras, que facilitan estas tareas y permiten a los desarrolladores crear modelos de aprendizaje automático precisos y eficientes.

En este tema prepararemos nuestros equipos para poder desarrollar en Python. Para ello, explicaremos cómo instalar las dos distribuciones más populares dentro de Python y cómo instalar nuevos módulos en estas distribuciones para aumentar el número de funcionalidades. Por último, describiremos diferentes herramientas disponibles para desarrollar en este lenguaje y nos centraremos en Jupyter Notebook, que será el entorno de desarrollo que usaremos a lo largo del curso.

### **1.1. Objetivos:**

- Contextualizar Python desde su historia y sus características.
- Comprender el problema de las versiones que ha existido hasta este año.
- Conocer los pasos para instalar Python en nuestro equipo.
- Conocer las distintas herramientas existentes para programar en Python.
- Comprender el entorno de desarrollo de Jupyter Notebook.

### **1.2. ¿Qué es Python?**

Python es un lenguaje de programación versátil, de alto nivel e intuitivo. Es ampliamente utilizado en diversos campos, desde el desarrollo web y científico hasta la inteligencia artificial. Con su sintaxis clara y legible, Python permite a los programadores expresar ideas de manera concisa y eficiente. Su extensa

biblioteca estándar y su comunidad activa brindan acceso a una amplia gama de herramientas y recursos. Python fomenta la programación estructurada y orientada a objetos, lo que facilita el diseño de aplicaciones complejas y su mantenimiento. Aprender Python es una habilidad valiosa en el mundo actual.

## **Ventajas de Python**

Python tiene varias propiedades que lo han convertido en un lenguaje muy potente y fácil de aprender. Estas propiedades son las siguientes:

- **Tipado dinámico:** Python no necesita que definamos el tipo de las variables cuando las inicializamos como pasa, por ejemplo, en Java o C. Cuando inicializamos una variable Python le asigna el tipo del valor que le estamos asignando. Incluso, durante la ejecución, una misma variable podría contener valores con distintos tipos. Esta propiedad hace que sea más sencillo aprender a usarlo, aunque hace que sea más difícil detectar errores asociados con los tipos de datos.
- **Lenguaje multiparadigma:** Python permite aplicar diferentes paradigmas de programación como son la programación orientada a objetos, como Java o C++, programación imperativa, como C, o programación funcional, como Haskell.
- **Interpretado/scripts:** otra ventaja es que podemos ejecutar Python de forma interpretada o usando scripts. Es decir, puedo abrir una consola de Python y escribir y ejecutar las instrucciones una a una o, por otro lado, puedo crear un fichero que almacene todo el programa.
- **Extensible:** por último, Python cuenta con una gran cantidad de módulos y librerías que podemos instalar para incluir nuevas funcionalidades. Sin embargo, como Python está implementado usando C++, podemos crear nuevos módulos en C++ e incluirlo en Python haciendo que el lenguaje sea extensible a nuevos módulos.

## 2. Instalación

### 2.1. Versiones

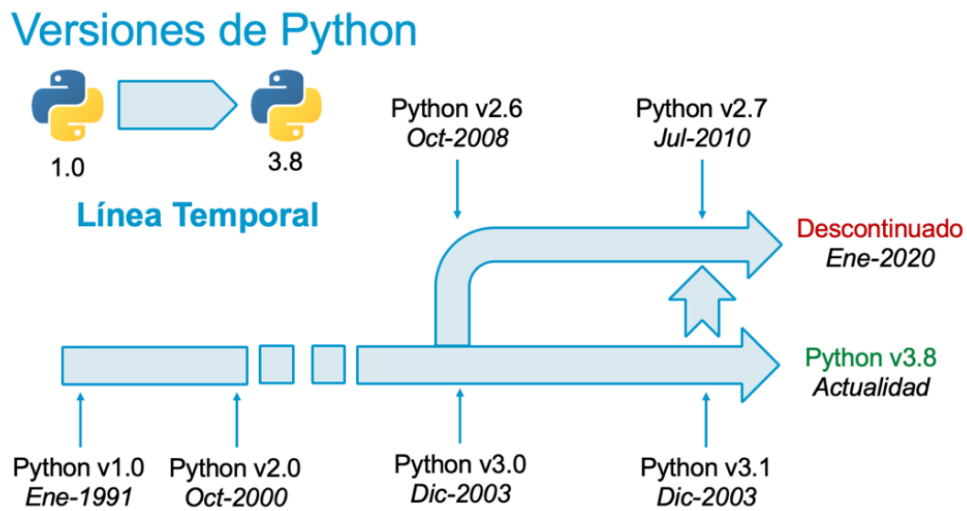


Figura 1 Línea temporal de las versiones de Python

La primera versión de Python se publicó en 1991. Desde entonces se han publicado varias versiones que han seguido siendo más o menos retro compatibles. Es decir, podía usar esas versiones en programas que había creado en versiones anteriores.

Sin embargo, a finales de 2008 se iba a publicar la versión 3.0. Esta nueva versión era un cambio radical con respecto las predecesoras y esto hacía que no fuera compatible con las versiones anteriores de Python.

Por este motivo, se publicó la versión 2.6 como soporte para los desarrollos de la versión 2. En la versión 2.6 se incluyeron nuevas funcionalidades de la versión 3, pero adaptadas a la versión 2. Desde ese momento Python contaba con 2 versiones y las novedades las tenían que duplicar en ambas. Por ejemplo, en 2010 publicaron la versión 3.1 y la 2.7 que incluía las nuevas funcionalidades, pero adaptadas a la versión 2.

Sin embargo, desde el equipo de Python siempre han explicado que las versiones 2.6 o 2.7 eran un parche y que no iban a ser versiones funcionales en un futuro. Este hecho se hizo oficial en enero de 2020 cuando se decidió que la versión 2.7 quedaba descontinuada y que a partir de entonces solo se publicarían nuevas funcionalidades para la versión 3.

## 2.2. Distribuciones de Python

- **Python;** Esta primera opción es una distribución básica de Python, la cual solo incluye los módulos principales de la misma.

descarga	Vínculo	Notas
Windows	<a href="#">Python</a>	Se mantendrá actualizada automáticamente
Windows	<a href="#">Versiones</a>	Todas las versiones de Python para Windows
Linux	<a href="#">Versiones</a>	Todas las versiones de Python para Linux
Mac	<a href="#">Versiones</a>	Todas las versiones de Python para Mac

En esta página tenemos una opción para instalar la última versión estable de Python. En nuestro caso, es la versión 3.11. Para descargarlo, hacemos clic en el botón download Python 3.X y automáticamente empezará la descarga del paquete de instalación.

Ejecute el archivo de instalación de Python descargado, seleccione las características de Python que desea instalar:

1. En la ventana de bienvenida, seleccione "Customize installation" en la parte inferior de la pantalla y haga clic en "Next".

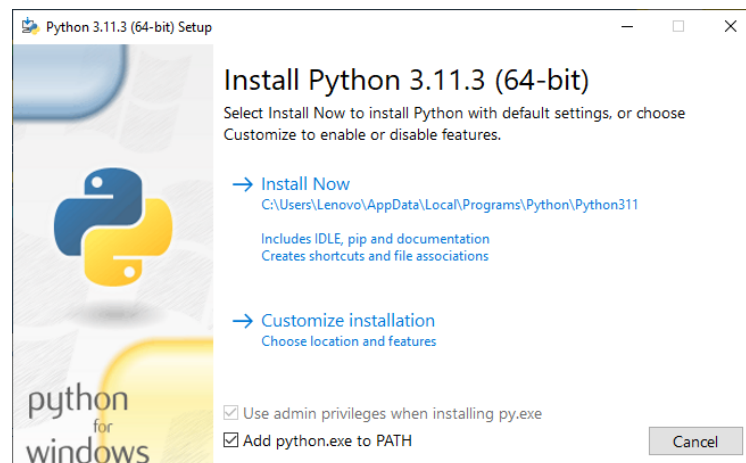
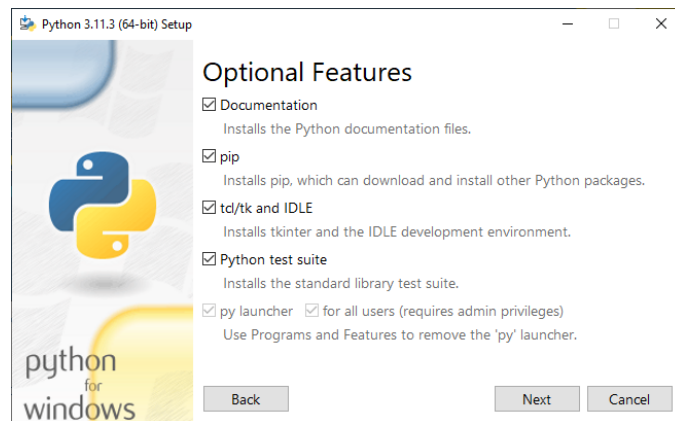


Figura 2 Asistente de configuración de instalación de Python

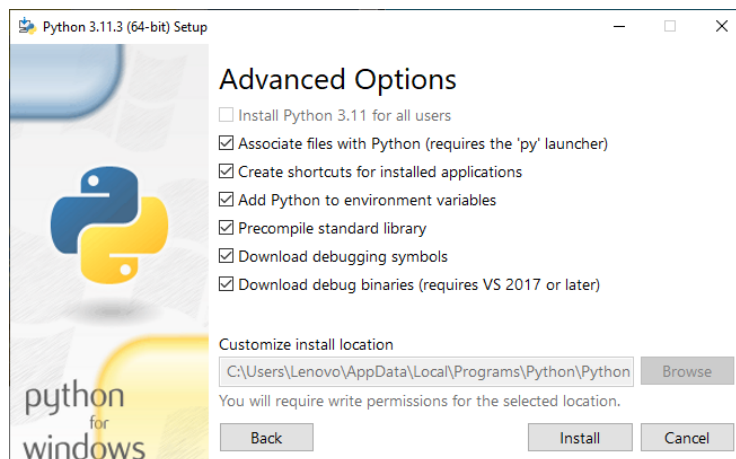
- "Add Python to PATH": Esto agrega la ruta de Python a la variable de entorno PATH de su sistema, lo que le permite ejecutar Python desde cualquier ubicación en su sistema.

2. Seleccionamos todas las casillas en el apartado “Optional Features”



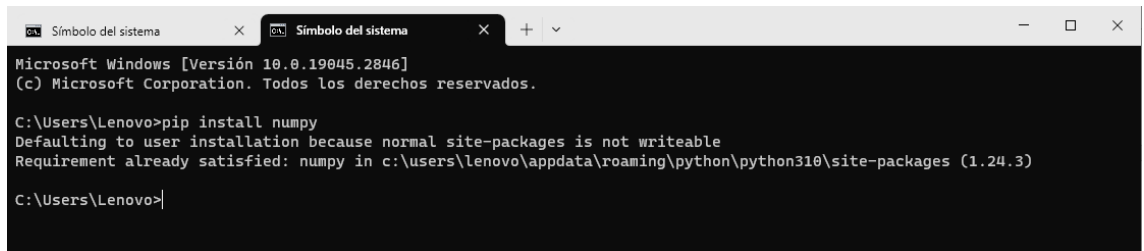
*Figura 3 Selección de características adicionales de Python*

3. Seleccionamos todas las casillas en el apartado “Advanced Options”, a excepción de la opción “for all users” la cual corresponde al ambiente de trabajo o los permisos de este.



*Figura 4 Complementos adicionales de Python*

Ahora puede instalar paquetes y librerías adicionales utilizando pip. Para instalar un paquete, simplemente abra una ventana de terminal o línea de comandos en su sistema y ejecute el comando "pip install <nombre\_del\_paquete>". Por ejemplo, para instalar la librería NumPy, puede ejecutar el comando "pip install numpy".



```
Microsoft Windows [Versión 10.0.19045.2846]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Lenovo>pip install numpy
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in c:\users\lenovo\appdata\roaming\python\python310\site-packages (1.24.3)

C:\Users\Lenovo>
```

*Figura 5 Ejemplo de instalación de un paquete mediante pip*

### Algunos comandos de interés:

- **python --version:** Este comando te mostrará la versión de Python instalada en tu sistema. Es útil para verificar qué versión estás utilizando.
- **where python:** En sistemas Windows, este comando te mostrará la ubicación del ejecutable de Python en tu sistema. Es útil cuando necesitas saber dónde se encuentra instalado Python.
- **pip install nombre\_paquete:** Este comando utiliza la herramienta pip para instalar paquetes adicionales en Python desde el repositorio de PyPI (Python Package Index). Solo necesitas reemplazar nombre\_paquete por el nombre del paquete que deseas instalar.
- **python -m venv nombre\_entorno:** Este comando crea un entorno virtual en Python con el nombre especificado. Los entornos virtuales son útiles para aislar las dependencias de tus proyectos y mantener todo organizado.
- **python script.py:** Utiliza este comando para ejecutar un script de Python desde la línea de comandos. Reemplaza script.py por el nombre del archivo que deseas ejecutar.

- **Anaconda;** Otra distribución con la que podemos instalar Python es Anaconda; la cual, aparte de instalar la versión básica (y estable) de Python, incluye otros módulos importantes dentro del análisis de datos (numpy, pandas, etc.)



Figura 6 Pagina principal para la descarga de las distintas versiones de anaconda.  
Fuente: <https://www.anaconda.com/download#downloads>

Aunque la distribución de Anaconda incluye nuevas funcionalidades a los módulos básicos de Python, no están incluidos todos; por este motivo, de ser necesario instalar nuevos módulos podemos utilizar la opción de símbolo de sistema y ejecutar: *conda install nombre\_modulo*.

**\*\*** Como nota adicional, dada la importancia de aprender a configurar un entorno desde cero, se optará por la instalación de la distribución de Python.



### 3. Herramientas

#### 3.1. Entornos de desarrollo

Como ya hemos explicado, Python es un lenguaje que podemos ejecutarlo de dos formas principalmente: usando el intérprete o usando scripts. En este apartado veremos los diferentes entornos de desarrollo que encontramos para programar en Python.

- **Modo interactivo;** en donde Python al ser un lenguaje interpretado, es capaz de ejecutar las instrucciones según las vamos introduciendo; por este motivo, la instalación de Python incluye el intérprete en el cual podemos realizar o ejecutar instrucciones. Para iniciar el intérprete, solo tenemos que ejecutar ***python*** en la consola. Para salir de la misma, solo debemos ejecutar el comando ***exit()***.

```
Símbolo del sistema - python
Microsoft Windows [Versión 10.0.19045.3086]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Lenovo>python --version
Python 3.11.3

C:\Users\Lenovo>python
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

*Figura 7 Intérprete de Python*

- **IPython;** para mejorar el intérprete de Python, se puede instalar el paquete de IPython (<https://ipython.org/install.html>), el cual añade más funcionalidades al intérprete de Python (resaltado de errores, completado automático, módulos a través del tabulador, etc).

Las dos opciones anteriores nos permiten ejecutar pequeñas instrucciones en Python; sin embargo, para crear programas de mayor complejidad será necesario escribir ***scripts*** que contengan más instrucciones o diferentes bloques de código.

#### 3.2. Entornos de desarrollo avanzados

Entornos orientados a grandes proyectos en Python y se incluyen nuevas funcionalidades como son la gestión de repositorios. Algunos de estos entornos de desarrollo pueden ejecutar en un mismo proyecto scripts y notebooks.

- **Editores de texto plano;** es una de las opciones que se pueden utilizar para la implementación de estos scripts. Existen una gran variedad de editores de texto para cada sistema operativo.
  - Atom
  - Notepad++
  - Visual Code
  - Vim

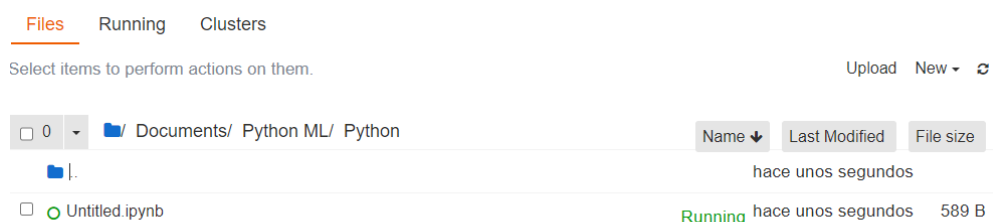
```

1 def main():
2     """
3     Función principal
4     """
5     print("Hola Mundo")
6
7 if __name__ == '__main__':
8     main()

```

*Figura 8 Ejemplo de script de Python implementado en Notepad++*

- [PyCharm](#), posee una versión limitada (PyCharm community Edition) y una versión de pago (PyCharm Pro).
- [Jupyter Notebook](#), aplicación web, la principal característica es la creación de celdas con objetos específicos, código, texto (markdown) o visualizar graficos.



*Figura 9 Vista del navegador de archivos Jupyter Notebook*

- [Spyder](#), entorno científico gratuito, escrito y orientado para Python, la cual cuenta con funcionalidades avanzadas de edición, análisis, depuración y perfiles de herramienta de desarrollo integral.

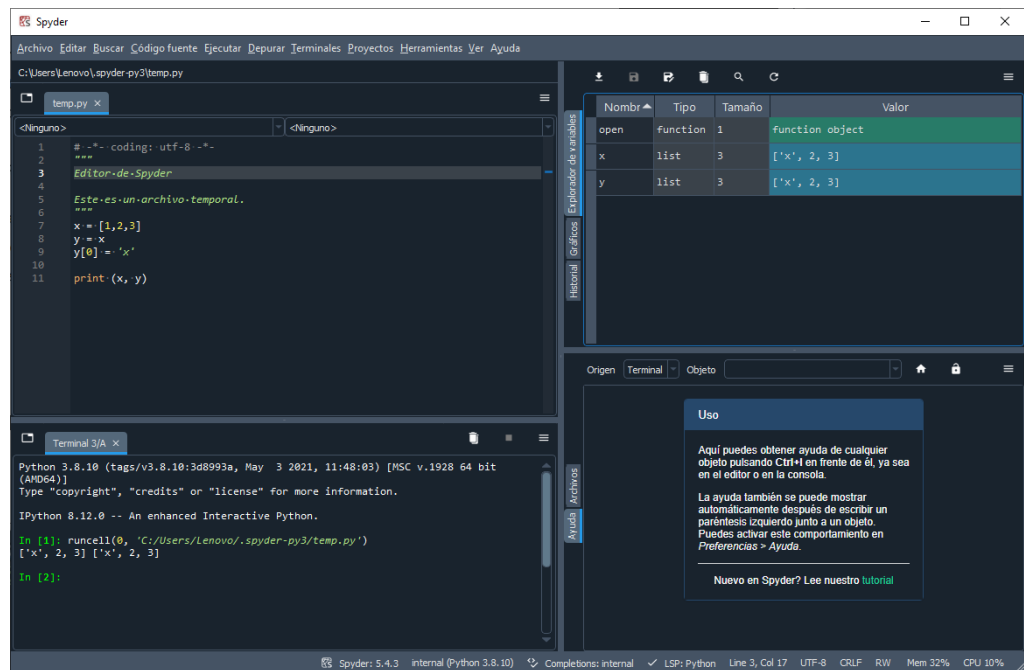


Figura 10 Ventana principal del entorno de desarrollo Spyder.

- 3.3. Visual Studio Code**, proporciona extensiones para Python en donde se admite el autocompletado e IntelliSense, linting, depuración y pruebas unitarias, junto con la capacidad de cambiar fácilmente entre entornos Python, incluidos entornos virtuales.

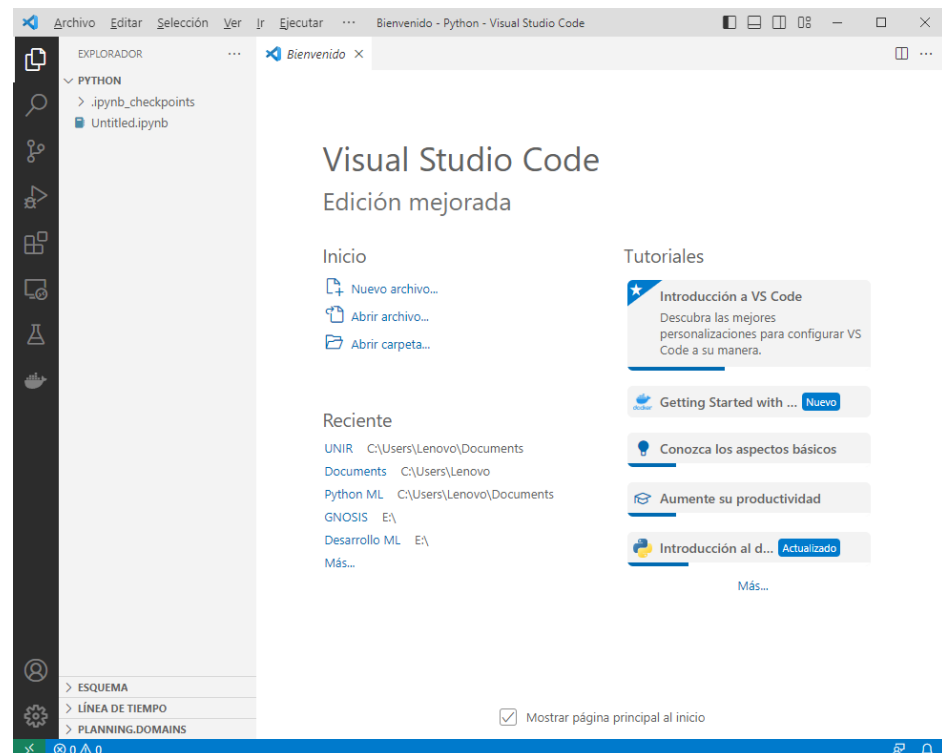


Figura 11 Ventana principal del entorno de desarrollo Visual Studio Code.

#### 4. Python – Ejercicios

**\*\*Recomendación;** si se esta realizando el ejercicio con Windows, se recomienda instalar Windows Terminal (Microsoft Store), la cual ofrece una forma mas eficiente al uso de líneas de comandos (esto beneficia con símbolo de sistema, PowerShell y WSL).



Figura 12 Fuente: <https://www.microsoft.com/store/productId/9N0DX20HK701>

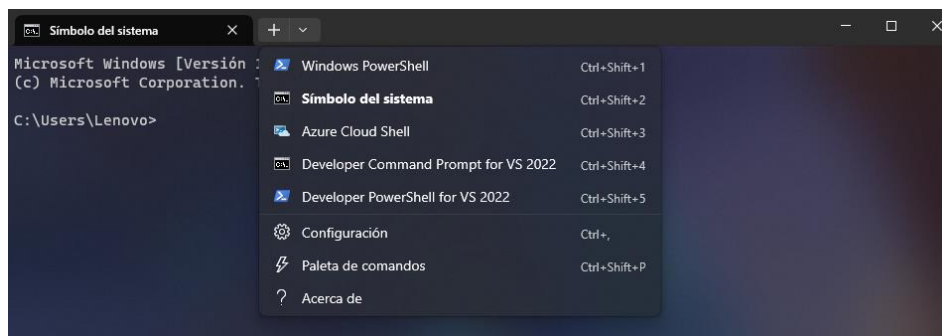
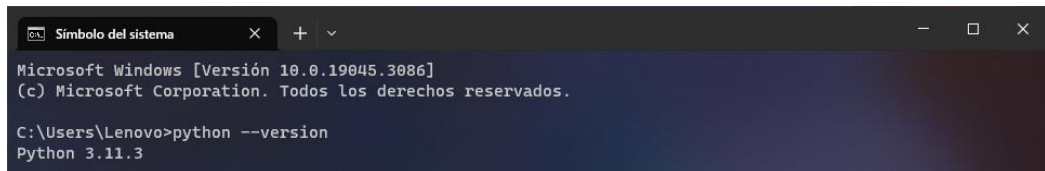


Figura 13 Windows Terminal, Algunas Opciones

## 1. Ejecución de instrucciones básicas mediante CMD.

Este ejercicio tiene como objetivo familiarizarte con la ejecución de instrucciones básicas de Python desde la terminal CMD. A través de ejemplos simples, aprenderás a imprimir mensajes en la pantalla y realizar cálculos utilizando valores ingresados por el usuario.

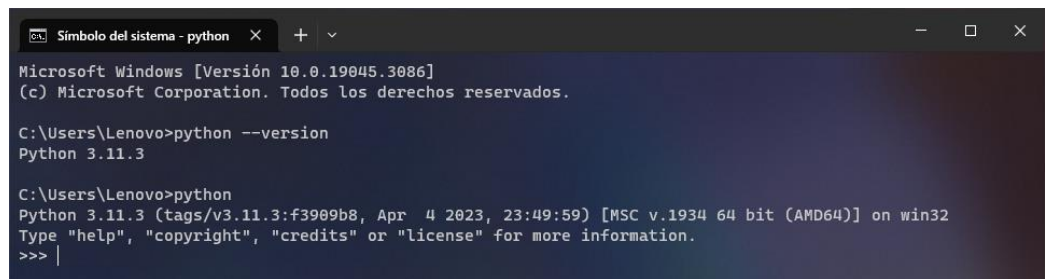
- Abrimos la terminal (CMD o Windows Terminal), en donde ejecutaremos la instrucción `python --version`.



```
Microsoft Windows [Versión 10.0.19045.3086]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Lenovo>python --version
Python 3.11.3
```

- Ahora ejecutaremos la instrucción `python`, en donde notara que ha accedido al intérprete de Python, a su vez se nos muestran algunos detalles adicionales.



```
Microsoft Windows [Versión 10.0.19045.3086]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Lenovo>python --version
Python 3.11.3

C:\Users\Lenovo>python
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

- **Imprimir tu nombre en la pantalla;** en este caso, simplemente utilizamos la función `print()` para mostrar tu nombre en la pantalla. Puedes reemplazar "Tu nombre" por tu propio nombre.

```
print("hola mundo")
```

- **Uso de variables,** se pueden declarar variables simples con valores estáticos, o se puede incluir la función `input()` para solicitar un ingreso de valor.

```
num1 = 1
num2 = 2
print(num1 * num2)
num1 = float(input("Ingresa el primer número: "))
num2 = float(input("Ingresa el segundo número: "))
print("El resultado de la suma es:", num1 + num2)
```

```
Símbolo del sistema - python
C:\Users\Lenovo>python
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hola muno")
hola muno
>>> num1 = 1
>>> num2= 5
>>> print(num1 * num2)
5
>>> num1 = float(input("Ingresa el primer número: "))
Ingresa el primer número: 5
>>> num2 = float(input("Ingresa el segundo número: "))
Ingresa el segundo número: 6
>>> print("El resultado de la suma es:", num1 + num2)
El resultado de la suma es: 11.0
```

- para finalizar ejecutamos la instrucción `exit()`
- luego ejecutamos la instrucción `cls`, para limpiar la terminal.

```
Símbolo del sistema
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hola muno")
hola muno
>>> num1 = 1
>>> num2= 5
>>> print(num1 * num2)
5
>>> num1 = float(input("Ingresa el primer número: "))
Ingresa el primer número: 5
>>> num2 = float(input("Ingresa el segundo número: "))
Ingresa el segundo número: 6
>>> print("El resultado de la suma es:", num1 + num2)
El resultado de la suma es: 11.0
>>> exit()

C:\Users\Lenovo>cls|
```

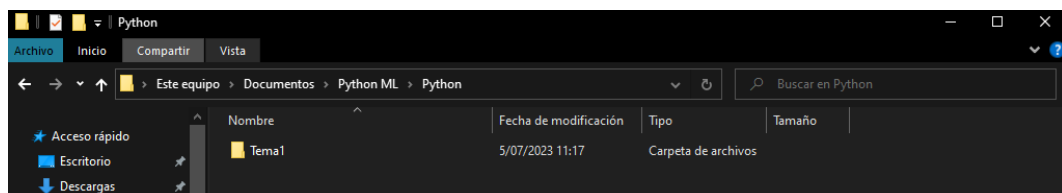
```
Símbolo del sistema
C:\Users\Lenovo>|
```

## 2. Ejecución de scripts usando un editor de texto.

En este ejercicio, aprenderás a crear archivos de script de Python utilizando un editor de texto y ejecutarlos desde la terminal CMD.

Para comenzar, simplemente abre un editor de texto y guarda el código Python en un archivo con extensión ".py". Luego, puedes ejecutar el script desde la terminal CMD utilizando el comando `python nombre_del_archivo.py`, donde "nombre\_del\_archivo.py" es el nombre del archivo que creaste.

- Ahora exploraremos como se puede ejecutar un script; para ello crearemos un espacio de trabajo, nos dirigimos a la carpeta documentos (o a elección) y crearemos una subcarpeta llamada Python, dentro de la misma crearemos un árbol de directorios asociados a cada clase (temas), comenzamos con una carpeta Tema1.

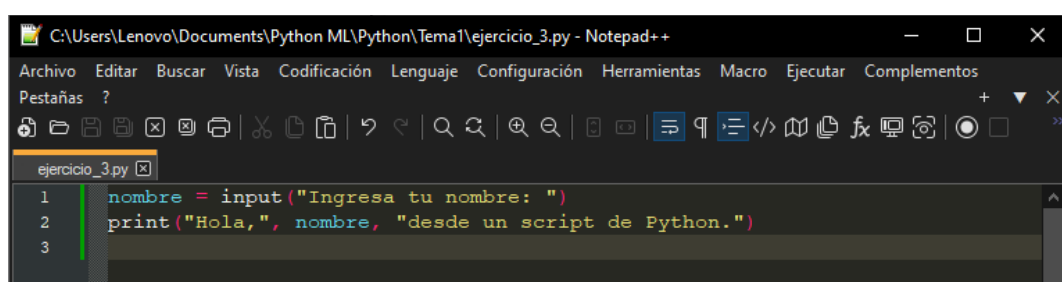


- A continuación, crearemos un documento de texto y lo renombraremos a

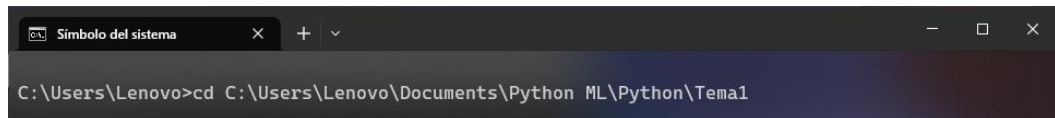


- Podemos abrirlo con cualquier editor de texto (de preferencia Notepad++ si esta en Windows). El código de ejemplo proporcionado en el ejercicio simplemente solicita al usuario que ingrese su nombre utilizando la función `input()`. Luego, utiliza la función `print()` para mostrar un mensaje de saludo en la pantalla, incluyendo el nombre ingresado.

```
nombre = input("Ingresa tu nombre: ")
print("Hola,", nombre, "desde un script de Python.")
```



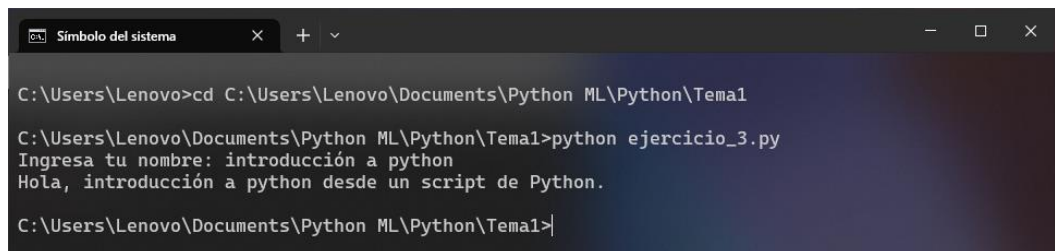
- daremos en guardar las modificaciones del archivo de tipo python (.py), luego copiaremos la ruta donde se encuentra ubicado el archivo y lo pegaremos en el CMD anteponiendo la instrucción cd



```
Símbolo del sistema
C:\Users\Lenovo>cd C:\Users\Lenovo\Documents\Python ML\Python\Tema1
```

- una vez ubicados en la ruta ejecutaremos la siguiente instrucción

```
python ejercicio_3.py
```



```
Símbolo del sistema
C:\Users\Lenovo>cd C:\Users\Lenovo\Documents\Python ML\Python\Tema1
C:\Users\Lenovo\Documents\Python ML\Python\Tema1>python ejercicio_3.py
Ingresa tu nombre: introducción a python
Hola, introducción a python desde un script de Python.
C:\Users\Lenovo\Documents\Python ML\Python\Tema1>
```

- al ejecutar el archivo con python nos pedirá que se ingrese el nombre, para luego imprimirlo en pantalla, tal cual se especifico en la breve instrucción.



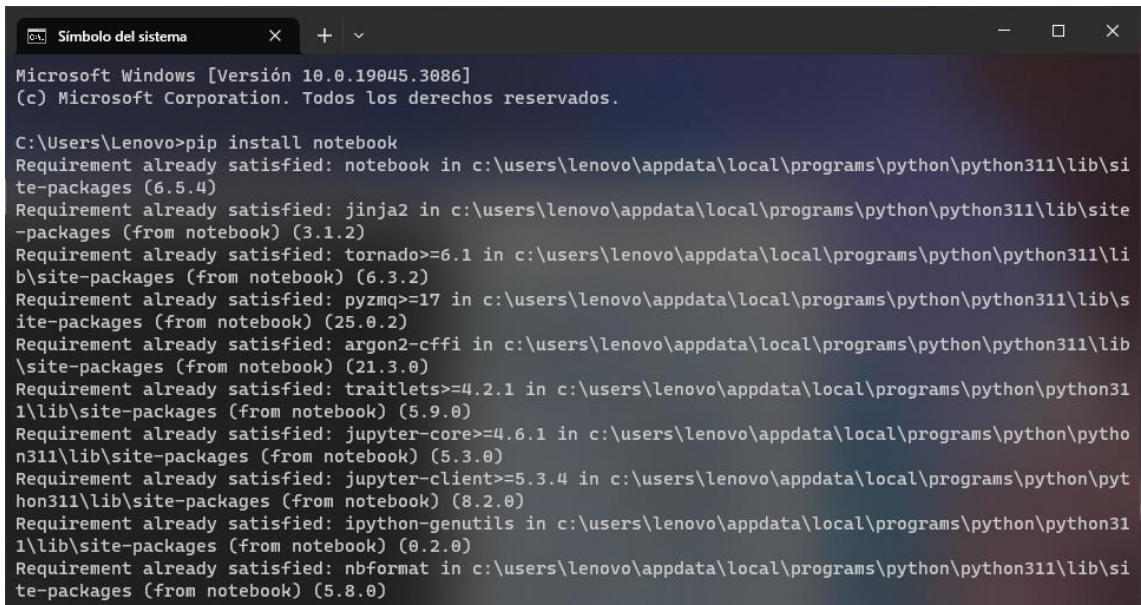
### 3. Introducción a Jupyter

En la tercera parte, nos adentraremos en Jupyter Notebook, una poderosa herramienta de programación interactiva que te permite combinar código, texto y visualizaciones en un único documento llamado "notebook".

#### a) Instalación de Jupyter Notebook

Para instalar Jupyter Notebook (puede acceder al siguiente [enlace](#) para más información), puedes utilizar el administrador de paquetes de Python llamado pip. Abre la terminal CMD y ejecuta el siguiente comando:

```
pip install notebook
```



```
Microsoft Windows [Versión 10.0.19045.3086]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Lenovo>pip install notebook
Requirement already satisfied: notebook in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (6.5.4)
Requirement already satisfied: Jinja2 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from notebook) (3.1.2)
Requirement already satisfied: tornado>=6.1 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from notebook) (6.3.2)
Requirement already satisfied: pyzmq>=17 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from notebook) (25.0.2)
Requirement already satisfied: argon2-cffi in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from notebook) (21.3.0)
Requirement already satisfied: traitlets>=4.2.1 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from notebook) (5.9.0)
Requirement already satisfied: jupyter-core>=4.6.1 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from notebook) (5.3.0)
Requirement already satisfied: jupyter-client>=5.3.4 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from notebook) (8.2.0)
Requirement already satisfied: ipython-genutils in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from notebook) (0.2.0)
Requirement already satisfied: nbformat in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from notebook) (5.8.0)
```

Al ejecutar el código se irán comprobando una serie de requerimientos (así como instalando); una vez terminado podemos ejecutar cls o abrir otra pestaña de la terminal.

#### b) Navegando a través de Jupyter Notebook

- Para iniciar Jupyter notebook solo es necesario ejecutar la siguiente instrucción desde la terminal:

```
jupyter notebook
```

- Tenemos 2 opciones para abrir Jupyter notebook, la primera será colocando la ruta donde queremos que se despliegue para tener un acceso rápido a la ruta de los archivos

```

Microsoft Windows [Versión 10.0.19045.3086]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Lenovo>cd C:\Users\Lenovo\Documents\Python ML\Python\Tema1

C:\Users\Lenovo\Documents\Python ML\Python\Tema1>jupyter notebook|
  
```

- La segunda es ejecutarlo desde la ruta base del sistema

```

Microsoft Windows [Versión 10.0.19045.3086]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Lenovo>jupyter notebook|
  
```

- Luego de ejecutar la instrucción, esto abrirá una nueva pestaña en tu navegador web con la interfaz de Jupyter Notebook. Desde allí, puedes navegar por tus directorios de archivos, crear nuevos notebooks y abrir los existentes.

```

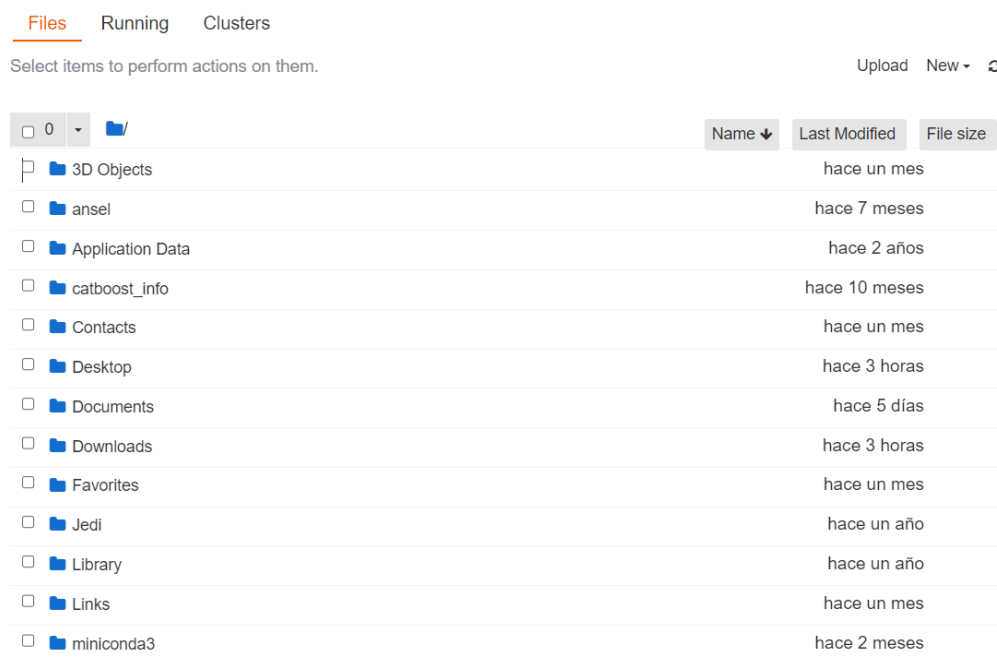
File "C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\jupyter_lsp\serverextension.py", line 76, in load_jupyter_server_extension
  nbapp.io_loop.call_later(0, initialize, nbapp, virtual_documents_uri)
  ^^^^^^^^^^^^^^^^^
AttributeError: 'NotebookApp' object has no attribute 'io_loop'
[I 2023-07-05 12:54:04.295 LabApp] JupyterLab extension loaded from C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\jupyterlab
[I 2023-07-05 12:54:04.295 LabApp] JupyterLab application directory is C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\share\jupyter\lab
[I 2023-07-05 12:54:04.297 LabApp] Extension Manager is 'pypi'.
[I 12:54:04.303 NotebookApp] Serving notebooks from local directory: C:\Users\Lenovo
[I 12:54:04.303 NotebookApp] Jupyter Notebook 6.5.4 is running at:
[I 12:54:04.304 NotebookApp] http://localhost:8888/?token=babf9c616e02a093fbcf4c6a41cbdd02effcdc03e598905d
[I 12:54:04.304 NotebookApp] or http://127.0.0.1:8888/?token=babf9c616e02a093fbcf4c6a41cbdd02effcdc03e598905d
[I 12:54:04.304 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 12:54:04.375 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/Lenovo/AppData/Roaming/jupyter/runtime/nbserver-2152-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=babf9c616e02a093fbcf4c6a41cbdd02effcdc03e598905d
or http://127.0.0.1:8888/?token=babf9c616e02a093fbcf4c6a41cbdd02effcdc03e598905d
0.00s - Debugger warning: It seems that frozen modules are being used, which may
0.00s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off
0.00s - to python to disable frozen modules.
0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to disable this validation.
|
  
```

- **IMPORTANTE:** Es importante no cerrar la pestaña de la terminal de CMD que ejecuta Jupyter Notebook porque esa instancia de la terminal está ejecutando el servidor de Jupyter Notebook. Si cierras la terminal, se detendrá el servidor y no podrás acceder a tus notebooks ni ejecutar código

en ellos. Aquí hay algunas razones por las cuales es importante mantener abierta la terminal de CMD que ejecuta Jupyter Notebook:

- **Persistencia del servidor:** La terminal de CMD mantiene en ejecución el servidor de Jupyter Notebook, lo que te permite acceder a tus notebooks en cualquier momento. Si cierras la terminal, el servidor se detendrá y tendrás que reiniciarlo nuevamente.
  - **Acceso a notebooks existentes:** Si tienes notebooks abiertos en Jupyter Notebook, mantener abierta la terminal te permite acceder a ellos y continuar trabajando en cualquier momento.
  - **Ejecución de código en tiempo real:** La terminal de CMD está conectada al kernel de Python en ejecución de Jupyter Notebook. Esto significa que puedes ejecutar código en tiempo real en tus notebooks y ver los resultados directamente en la terminal.
  - **Control y monitoreo:** Mantener abierta la terminal de CMD te permite controlar y monitorear el estado del servidor de Jupyter Notebook. Puedes ver los mensajes y registros generados por el servidor, lo que es útil para diagnosticar problemas o errores.
- Continuando con Jupyter notebook, notara que se abrió una pestaña de su navegador predeterminado (de pendiendo de que opción ejecuto podrá ver las carpetas de su sistema)



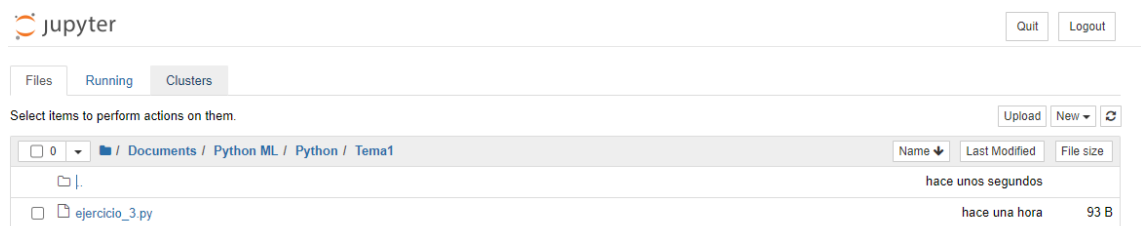
The screenshot shows the Jupyter Notebook file browser interface. At the top, there are tabs for 'Files' (selected), 'Running', and 'Clusters'. Below the tabs, there is a text prompt 'Select items to perform actions on them.' and buttons for 'Upload', 'New', and a refresh icon. The main area displays a list of files and folders. Each item has a checkbox on the left, a folder icon, the name, and the last modified date. The items are: 3D Objects (hace un mes), anseI (hace 7 meses), Application Data (hace 2 años), catboost\_info (hace 10 meses), Contacts (hace un mes), Desktop (hace 3 horas), Documents (hace 5 días), Downloads (hace 3 horas), Favorites (hace un mes), Jedi (hace un año), Library (hace un año), Links (hace un mes), and miniconda3 (hace 2 meses).

	Name	Last Modified	File size
<input type="checkbox"/>	0		
<input type="checkbox"/>	3D Objects	hace un mes	
<input type="checkbox"/>	anseI	hace 7 meses	
<input type="checkbox"/>	Application Data	hace 2 años	
<input type="checkbox"/>	catboost_info	hace 10 meses	
<input type="checkbox"/>	Contacts	hace un mes	
<input type="checkbox"/>	Desktop	hace 3 horas	
<input type="checkbox"/>	Documents	hace 5 días	
<input type="checkbox"/>	Downloads	hace 3 horas	
<input type="checkbox"/>	Favorites	hace un mes	
<input type="checkbox"/>	Jedi	hace un año	
<input type="checkbox"/>	Library	hace un año	
<input type="checkbox"/>	Links	hace un mes	
<input type="checkbox"/>	miniconda3	hace 2 meses	

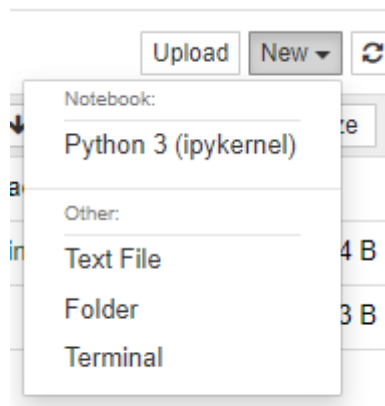
- En primer lugar, para navegar entre las carpetas, solo tendremos que hacer clic en el nombre de la carpeta a la que queremos acceder. Si lo que queremos es volver a la carpeta superior, solo debemos hacer clic en la carpeta que tiene como nombre dos puntos (..)

### c) Creación y exploración del notebook

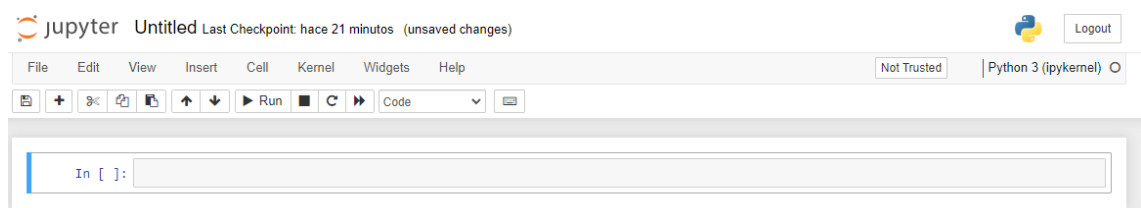
- Ahora navegamos seleccionando las carpetas hasta llegar a la ubicación donde se está desarrollando el tema.



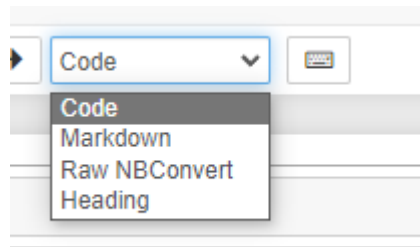
- En la interfaz de Jupyter Notebook, puedes crear un nuevo notebook haciendo clic en el botón "New" y verás la opción "Python 3" bajo "Notebooks"; bajo Other, también podrás crear archivos de texto, carpetas y acceso a la terminal (powerShell).



- Al seleccionar Python3 (ipykernel) se abrirá un nuevo notebook en el que puedes escribir y ejecutar código Python en celdas individuales.



- Cada una de las celdas que se creen dentro del notebook pueden ser del siguiente tipo:
  - Código Python
  - Texto con markdown
  - Formato raw, en donde se muestra el texto con el formato de consola
  - Formato heading, crea una celda con formato título



- Además, existe otro tipo de celda que se da cuando una celda de código devuelve un resultado, estas celdas se crearán de forma automática.
- Justo encima de las celdas, tenemos un conjunto de botones que nos permiten interactuar con las celdas. A continuación, explicaremos cada uno de estos botones siguiendo el orden de izquierda a la derecha:
  - **Guardar:** almacena en el fichero todas las celdas y guarda el estado de ejecución en el que se quedó el notebook.
  - **Nueva celda:** crea una nueva celda inmediatamente después de la celda que tenemos seleccionada.
  - **Cortar:** permite cortar una celda para pegarla en otro punto del notebook.
  - **Copiar:** permite copiar una celda para poder pegarla en otro punto del notebook.
  - **Pegar:** pegamos la celda que hemos copiado o cortado previamente inmediatamente después de la celda que tenemos seleccionada.
  - **Bajar una celda:** desplaza la celda seleccionada una posición hacia abajo.
  - **Subir una celda:** desplaza la celda seleccionada una posición hacia arriba.
  - **Ejecutar celda:** ejecuta el contenido que hay en la celda seleccionada. Si esa celda es de tipo código, ejecutará las instrucciones y devolverá la salida en una celda de salida. Por otro lado, si la celda es de tipo texto, le asignará un formato HTML.

- **Stop:** para la ejecución del kernel de Python. Para poder seguir ejecutando nuevas celdas, es necesario reiniciar el kernel de Python.
- **Reiniciar kernel:** reinicia la ejecución del kernel, eliminando de la memoria toda la información del notebook que tuviese almacenada.
- **Reiniciar el kernel y ejecutar todas las celdas:** reinicia el kernel de Python como el botón anterior y, después, ejecuta todas las celdas del notebook.
- **Selección del tipo de celda:** permite seleccionar el tipo de celda que tenemos seleccionado. Los tipos son los que hemos descrito anteriormente.
- A continuación, se muestran los comandos para acceso rápido los cuales facilitan el trabajo dentro del notebook.

Command Mode (press `Esc` to enable)

`F`: find and replace  
`Ctrl-Shift-F`: open the command palette  
`Ctrl-Shift-P`: open the command palette  
`Enter`: enter edit mode  
`P`: open the command palette  
`Shift-Enter`: run cell, select below  
`Ctrl-Enter`: run selected cells  
`Alt-Enter`: run cell and insert below  
`Y`: change cell to code  
`M`: change cell to markdown  
`R`: change cell to raw  
`1`: change cell to heading 1  
`2`: change cell to heading 2  
`3`: change cell to heading 3  
`4`: change cell to heading 4  
`5`: change cell to heading 5  
`6`: change cell to heading 6  
`K`: select cell above  
`Up`: select cell above  
`Down`: select cell below  
`J`: select cell below  
`Shift-K`: extend selected cells above  
`Shift-Up`: extend selected cells above  
`Shift-Down`: extend selected cells below


Edit Shortcuts

`Shift-J`: extend selected cells below  
`Ctrl-A`: select all cells  
`A`: insert cell above  
`B`: insert cell below  
`X`: cut selected cells  
`C`: copy selected cells  
`Shift-V`: paste cells above  
`V`: paste cells below  
`Z`: undo cell deletion  
`D`, `D`: delete selected cells  
`Shift-M`: merge selected cells, or current cell with cell below if only one cell is selected  
`Ctrl-S`: Save and Checkpoint  
`S`: Save and Checkpoint  
`L`: toggle line numbers  
`O`: toggle output of selected cells  
`Shift-O`: toggle output scrolling of selected cells  
`H`: show keyboard shortcuts  
`I`, `I`: interrupt the kernel  
`0`, `0`: restart the kernel (with dialog)  
`Esc`: close the pager  
`Q`: close the pager  
`Shift-L`: toggles line numbers in all cells, and persist the setting  
`Shift-Space`: scroll notebook up  
`Space`: scroll notebook down

Edit Mode (press `Enter` to enable)

<code>Tab</code> : code completion or indent	<code>Ctrl-Right</code> : go one word right
<code>Shift-Tab</code> : tooltip	<code>Ctrl-Backspace</code> : delete word before
<code>Ctrl-]</code> : indent	<code>Ctrl-Delete</code> : delete word after
<code>Ctrl-[</code> : dedent	<code>Ctrl-Y</code> : redo
<code>Ctrl-A</code> : select all	<code>Alt-U</code> : redo selection
<code>Ctrl-Z</code> : undo	<code>Ctrl-M</code> : enter command mode
<code>Ctrl-/</code> : comment	<code>Ctrl-Shift-F</code> : open the command palette
<code>Ctrl-D</code> : delete whole line	<code>Ctrl-Shift-P</code> : open the command palette
<code>Ctrl-U</code> : undo selection	<code>Esc</code> : enter command mode
<code>Insert</code> : toggle overwrite flag	<code>Shift-Enter</code> : run cell, select below
<code>Ctrl-Home</code> : go to cell start	<code>Ctrl-Enter</code> : run selected cells
<code>Ctrl-Up</code> : go to cell start	<code>Alt-Enter</code> : run cell and insert below
<code>Ctrl-End</code> : go to cell end	<code>Ctrl-Shift-Minus</code> : split cell at cursor(s)
<code>Ctrl-Down</code> : go to cell end	<code>Ctrl-S</code> : Save and Checkpoint
<code>Ctrl-Left</code> : go one word left	<code>Down</code> : move cursor down
	<code>Up</code> : move cursor up

- Algunas opciones adicionales, si seleccionamos el nombre del notebook, emergerá una ventana para renombrar el archivo, el cual no es necesario especificar la extensión.

 jupyter **Untitled** Last Checkpoint: hace 11 minutos (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Rename Notebook ✕

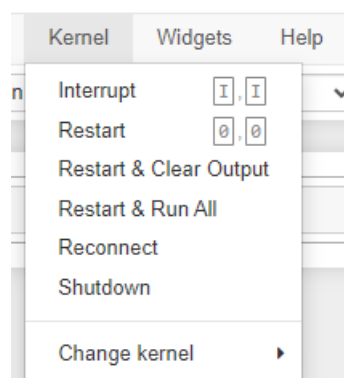
Enter a new notebook name:

Untitled

Cancel


Rename

- Si en algún punto el notebook se traba o no se muestra alguna librería instalada, podemos reiniciar el kernel, interrumpirlo o demás:





#### d) Ejemplo de ejecución de códigos en celdas

- Comenzamos escribiendo la instrucción en una de las celdas, luego podemos seleccionar la opción  **Run** o presionar la combinación de teclas shift + enter

```
print("hola mundo")
```

```
In [1]: print("hola mundo")  
hola mundo
```

- Como podrá notar, la celda pasara a tener un sombreado verde indicando que estamos editando la celda (si el sombreado es azul esto indica que solo estamos posicionados en la celda, pero no editando), a su vez podemos ver que el texto In [ ] que antes estaba vacío ahora posee el numero 1, este indicador nos mostrara el numero de ejecución que se va realizando en el notebook.
- Esto es importante, dado que al poder ejecutar celdas individuales es facil obviar alguna o en el peor de los casos, ejecutar las de un recorrido todas las celdas y no ejecutar algunas, lo que podría sesgar sus resultados.
- También podrá notar que al haber presionado la combinación shift + enter, se creó automáticamente una celda en blanco (si en caso no quiere crear una celda nueva y permanecer sobre la misma solo sebe usar ctrl + enter)
- Ahora ejecutemos la siguiente instrucción

```
nombre = input("Ingresa tu nombre: ")  
print("Hola,", nombre, "desde un script de Python.")
```

```
In [*]: nombre = input("Ingresa tu nombre: ")  
print("Hola,", nombre, "desde un script de Python.")
```

Ingresa tu nombre:

```
In [2]: nombre = input("Ingresa tu nombre: ")  
print("Hola,", nombre, "desde un script de Python.")
```

```
Ingresa tu nombre: ejercicio 3  
Hola, ejercicio 3 desde un script de Python.
```



- En esta instrucción al igual que en el CMD se nos solicita una entrada para poder continuar con la ejecución, esto es debido a que la función input() espera un valor.

```
numerator = 100
denominator = 0
results = numerator / denominator
results
```

```
In [3]: numerator = 100
denominator = 0
results = numerator / denominator
results

-----
ZeroDivisionError                                Traceback (most recent call last)
Cell In[3], line 3
      1 numerator = 100
      2 denominator = 0
----> 3 results = numerator / denominator
      4 results

ZeroDivisionError: division by zero
```

- Cuando se ejecuta este código, se producirá un error en la línea results = numerator / denominator, ya que no es posible dividir un número entre cero. Esta excepción detendrá la ejecución del programa y mostrará un mensaje de error indicando que se produjo una división entre cero.

```
pi = 3.14159
print(pi)
```

```
In [4]: pi = 3.14159
print(pi)

3.14159
```

- Cuando se ejecuta este código, la línea print(pi) mostrará el valor de pi en la pantalla. En este caso, el valor impreso será "3.14159", que es la representación numérica de Pi ( $\pi$ ) con una precisión de cinco decimales.

```
def saludo():
    print("hola mundo")
```

```
In [5]: def saludo():  
        print("hola mundo")
```

```
In [6]: saludo()  
  
hola mundo
```

- Para este caso, En esta celda, se define una función llamada saludo(). Esta función imprime el mensaje "Hola mundo" en la pantalla. Al ejecutar esta celda, la función saludo() se define en la memoria del kernel de Python, pero no se muestra ningún resultado en la salida. En la siguiente celda, se llama a la función saludo() que se definió en la celda anterior. Al ejecutar esta celda, la función saludo() se ejecuta y muestra el mensaje "Hola mundo" en la salida.
- Ahora seleccionando una celda (recuerde que el sombreado tiene que ser azul), presionaremos la tecla M; notaremos que la glosa "In [ ]" a

desaparecido y a su vez que el tipo de celda cambio a

Markdown

```
In [ ]:
```

- Sobre esta celda podremos usar

- código html

```
<h1>Tema 1</h1>
```

- LaTeX

$$V_{\text{sphere}} = \frac{4}{3}\pi r^3$$

- Otros

```
# head1  
## head2  
## head3  
### head 4
```

**\*\*texto en negrita\*\***

\* viñeta

## **ANEXO**

# Python For Data Science Cheat Sheet

## Python Basics

Learn More Python for Data Science [Interactively at www.datacamp.com](https://www.datacamp.com)



### Variables and Data Types

#### Variable Assignment

```
>>> x=5
>>> x
5
```

#### Calculations With Variables

>>> x+2	Sum of two variables
>>> x-2	Subtraction of two variables
>>> x*2	Multiplication of two variables
>>> x**2	Exponentiation of a variable
>>> x%2	Remainder of a variable
>>> x/float(2)	Division of a variable

#### Types and Type Conversion

str()	'5', '3.45', 'True'	Variables to strings
int()	5, 3, 1	Variables to integers
float()	5.0, 1.0	Variables to floats
bool()	True, True, True	Variables to booleans

### Asking For Help

```
>>> help(str)
```

### Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

#### String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

### Lists

Also see [NumPy Arrays](#)

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

#### Selecting List Elements

Index starts at 0

**Subset**

```
>>> my_list[1]
Select item at index 1
>>> my_list[-3]
Select 3rd last item


Slice



```
>>> my_list[1:3]
Select items at index 1 and 2
>>> my_list[1:]
Select items after index 0
>>> my_list[:3]
Select items before index 3
>>> my_list[:]
Copy my_list


Subset Lists of Lists



```
>>> my_list2[1][0]
my_list[list[itemOfList]]
>>> my_list2[1][:-2]
```


```


```

#### List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

#### List Methods

```
>>> my_list.index(a)
Get the index of an item
>>> my_list.count(a)
Count an item
>>> my_list.append('!')
Append an item at a time
>>> my_list.remove('!')
Remove an item
>>> del my_list[0:1]
Remove an item
>>> my_list.reverse()
Reverse the list
>>> my_list.extend('!')
Append an item
>>> my_list.pop(-1)
Remove an item
>>> my_list.insert(0, '!')
Insert an item
>>> my_list.sort()
Sort the list
```

#### String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

#### String Methods

```
>>> my_string.upper()
String to uppercase
>>> my_string.lower()
String to lowercase
>>> my_string.count('e')
Count String elements
>>> my_string.replace('e', 'i')
Replace String elements
>>> my_string.strip()
Strip whitespace from ends
```

### Libraries

**Import libraries**

```
>>> import numpy
>>> import numpy as np
Selective import
>>> from math import pi
>>> from math import pi
```

**pandas** Data analysis

**Machine learning**

**NumPy** Scientific computing

**matplotlib** 2D plotting

### Install Python

**ANACONDA**

Leading open data science platform powered by Python

**Free IDE that is included with Anaconda**

**spyder**

**jupyter**

Create and share documents with live code, visualizations, text, ...

### NumPy Arrays

Also see [Lists](#)

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3], [4,5,6]])
```

#### Selecting NumPy Array Elements

Index starts at 0

**Subset**

```
>>> my_array[1]
Select item at index 1
```

**Slice**

```
>>> my_array[0:2]
array([1, 2])
Select items at index 0 and 1
```

**Subset 2D NumPy arrays**

```
>>> my_2darray[:, 0]
array([1, 4])
my_2darray[rows, columns]
```

#### NumPy Array Operations

```
>>> my_array > 3
array([False, False, False, True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

#### NumPy Array Functions

```
>>> my_array.shape
Get the dimensions of the array
>>> np.append(other_array)
Append items to an array
>>> np.insert(my_array, 1, 5)
Insert items in an array
>>> np.delete(my_array, [1])
Delete items in an array
>>> np.mean(my_array)
Mean of the array
>>> np.median(my_array)
Median of the array
>>> my_array.corrcoef()
Correlation coefficient
>>> np.std(my_array)
Standard deviation
```



**DataCamp**  
Learn Python for Data Science [Interactively](https://www.datacamp.com)



## sys Variables

argv	Command line args
builtin_module_names	Linked C modules
byteorder	Native byte order
check_interval	Signal check frequency
exec_prefix	Root directory
executable	Name of executable
exitfunc	Exit function name
modules	Loaded modules
path	Search path
platform	Current platform
stdin, stdout, stderr	File objects for I/O
version_info	Python version info
winver	Version number

## sys.argv for \$ python foo.py bar -c qux --h

sys.argv[0]	foo.py
sys.argv[1]	bar
sys.argv[2]	-c
sys.argv[3]	qux
sys.argv[4]	--h

## os Variables

altsep	Alternative sep
curdir	Current dir string
defpath	Default search path
devnull	Path of null device
extsep	Extension separator
linesep	Line separator
name	Name of OS
pardir	Parent dir string
pathsep	Patch separator
sep	Path separator

**Note** Registered OS names: "posix", "nt", "mac", "os2", "ce", "java", "riscos"

## Class Special Methods

__new__(cls)	__lt__(self, other)
__init__(self, args)	__le__(self, other)
__del__(self)	__gt__(self, other)
__repr__(self)	__ge__(self, other)
__str__(self)	__eq__(self, other)
__cmp__(self, other)	__ne__(self, other)
__index__(self)	__nonzero__(self)
__hash__(self)	
__getattr__(self, name)	
__getattribute__(self, name)	
__setattr__(self, name, attr)	
__delattr__(self, name)	
__call__(self, args, kwargs)	

## String Methods

capitalize() *	lstrip()
center(width)	partition(sep)
count(sub, start, end)	replace(old, new)
decode()	rfind(sub, start, end)
encode()	rindex(sub, start, end)
endswith(sub)	rjust(width)
expandtabs()	rpartition(sep)
find(sub, start, end)	rsplit(sep)
index(sub, start, end)	rstrip()
isalnum() *	split(sep)
isalpha() *	splitlines()
isdigit() *	startswith(sub)
islower() *	strip()
isspace() *	swapcase() *
istitle() *	title() *
isupper() *	translate(table)
join()	upper() *
ljust(width)	zfill(width)
lower() *	

**Note** Methods marked \* are locale dependant for 8-bit strings.

## List Methods

append(item)	pop(position)
count(item)	remove(item)
extend(list)	reverse()
index(item)	sort()
insert(position, item)	

## File Methods

close()	readlines(size)
flush()	seek(offset)
filen()	tell()
isatty()	truncate(size)
next()	write(string)
read(size)	writelines(list)
readline(size)	

## Indexes and Slices (of a=[0,1,2,3,4,5])

len(a)	6
a[0]	0
a[5]	5
a[-1]	5
a[-2]	4
a[1:]	[1,2,3,4,5]
a[:5]	[0,1,2,3,4]
a[:-2]	[0,1,2,3]
a[1:3]	[1,2]
a[1:-1]	[1,2,3,4]
b=a[:]	Shallow copy of a

## Datetime Methods

today()	fromordinal(ordinal)
now(timezoneinfo)	combine(date, time)
utcnow()	strptime(date, format)
fromtimestamp(timestamp)	
utcfromtimestamp(timestamp)	

## Time Methods

replace()	utcoffset()
isoformat()	dst()
__str__()	tzname()
strftime(format)	

## Date Formatting (strftime and strptime)

%a	Abbreviated weekday (Sun)
%A	Weekday (Sunday)
%b	Abbreviated month name (Jan)
%B	Month name (January)
%c	Date and time
%d	Day (leading zeros) (01 to 31)
%H	24 hour (leading zeros) (00 to 23)
%I	12 hour (leading zeros) (01 to 12)
%j	Day of year (001 to 366)
%m	Month (01 to 12)
%M	Minute (00 to 59)
%p	AM or PM
%S	Second (00 to 61*)
%U	Week number <sup>1</sup> (00 to 53)
%w	Weekday <sup>2</sup> (0 to 6)
%W	Week number <sup>3</sup> (00 to 53)
%x	Date
%X	Time
%y	Year without century (00 to 99)
%Y	Year (2008)
%Z	Time zone (GMT)
%%	A literal "%" character (%)

1. Sunday as start of week. All days in a new year preceding the first Sunday are considered to be in week 0.
2. 0 is Sunday, 6 is Saturday.
3. Monday as start of week. All days in a new year preceding the first Monday are considered to be in week 0.
4. This is not a mistake. Range takes account of leap and double-leap seconds.