

Primeros pasos con Python



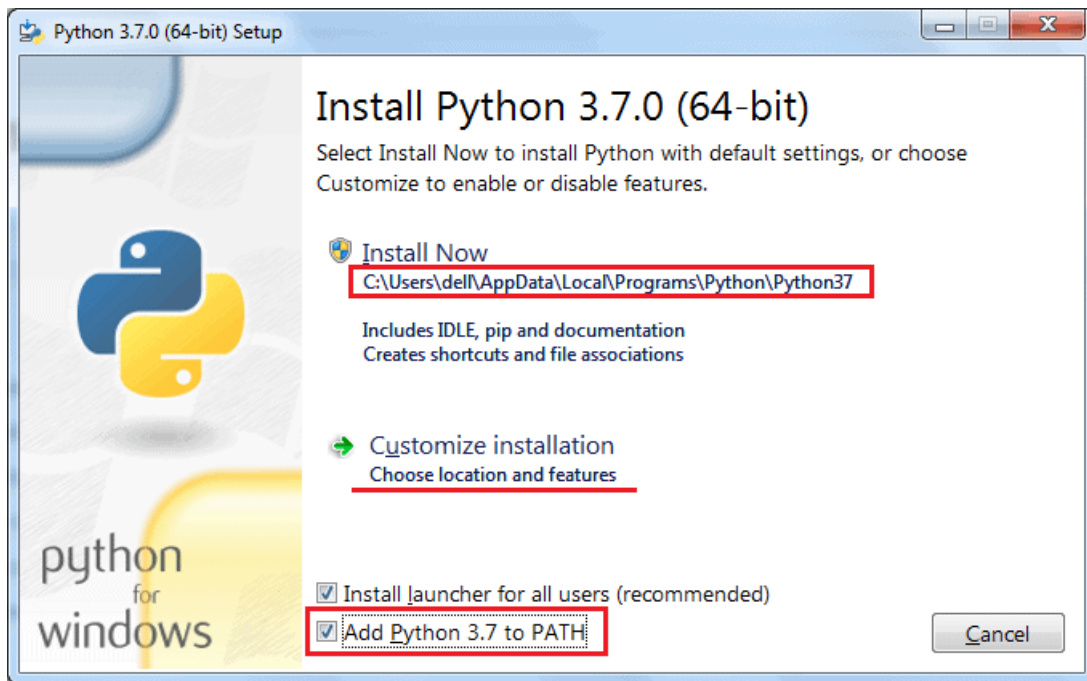
Python es un lenguaje de programación creado a principios de los años 90 por Guido van Rossum. Su nombre es un tributo al grupo de comedia *Monty Python*, y entre sus muchas virtudes, podemos destacar las siguientes:

- Se utiliza a menudo en etapas tempranas de aprendizaje de programación, porque es bastante sencillo de entender.
- Es un lenguaje multiplataforma, con el que podemos desarrollar aplicaciones de todo tipo (escritorio, web, etc) en diferentes sistemas (Windows, Mac, Linux).
- Es un lenguaje interpretado (no compilado), y puede utilizarse como un lenguaje de *script* en terminal, como ocurre con Perl, PowerShell u otros lenguajes de script.
- Utiliza tipado dinámico de datos, es decir, no existen tipos de datos implícitos, ni tenemos que indicarlos al utilizar las variables en el programa. A medida que asignamos valores a las variables, éstas toman el tipo de dato adecuado.
- Podemos utilizar Python tanto desde una perspectiva orientada a objetos (usando clases y objetos) como sin dicha perspectiva (sin necesidad de clases).
- Dispone de multitud de paquetes o librerías adicionales que podemos descargar para construir aplicaciones de todo tipo.

1. Instalación de Python

Para instalar Python en nuestro sistema, basta con ir a la [web oficial](#) y descargar la versión apropiada para nuestro sistema. El intérprete de Python es normalmente un comando llamado `python` o `python3`, dependiendo del sistema operativo en el que estemos. Por ejemplo, en Windows se utiliza el comando `python`, pero en Linux se emplea `python3`.

Cuando instalemos Python en Windows, debemos marcar en el asistente de instalación la casilla para añadir Python directamente al PATH del sistema. De lo contrario, deberemos editar manualmente nosotros la variable de entorno para añadir la carpeta de instalación de Python.



En lo que respecta a Linux, primero podemos comprobar si ya tenemos Python instalado en nuestra distribución, con este comando:

```
python3 --version
```

De lo contrario, podemos actualizar a Python 3 fácilmente con este comando:

```
sudo apt-get install python3
```

Como hemos visto, podemos utilizar el parámetro `--version` o también `-V` para detectar la versión actualmente instalada de Python.

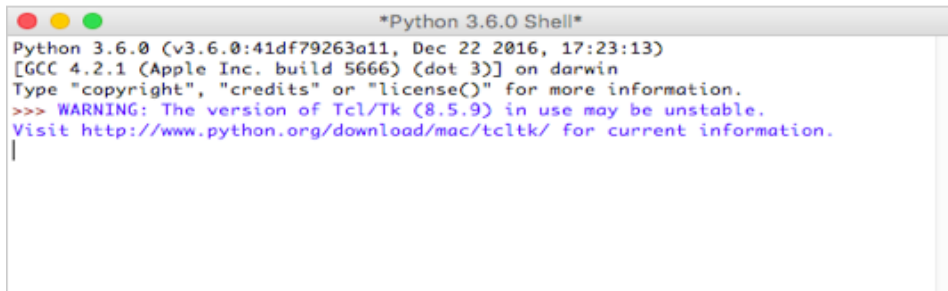
```
python -V
```

2. Elegir un IDE apropiado

Una vez tengamos Python instalado en nuestro sistema, debemos elegir un IDE apropiado para desarrollar nuestras aplicaciones. Podemos encontrar diferentes alternativas en Internet, como por ejemplo IDLE (el entorno que viene integrado en la instalación de Python), Eclipse, Visual Studio Code... En este documento vamos a dar unas breves nociones sobre algunos de los que consideramos más interesantes.

2.1. Utilizando IDLE

IDLE es un entorno que se descarga en la propia instalación de Python. Podemos ejecutarlo escribiendo `idle` o `idle3` en el terminal (dependiendo de nuestro sistema operativo y versión de Python). También podemos elegir el correspondiente acceso directo en el menú de Windows, o en la sección de *Aplicaciones* de Mac. Finalmente, veremos una ventana como esta:



```
*Python 3.6.0 Shell*
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
```

Lo que vemos es un terminal de comandos Python. Podemos escribir comandos sueltos y se ejecutan uno a uno, pero esto no es lo habitual, sino que necesitaremos editar un archivo de código fuente. Para ello, desde el menú *File > New File* podemos crear nuevos archivos de código Python, y guardarlos con el nombre adecuado (los archivos deben tener extensión `.py`, como por ejemplo `prueba.py`). Después, editamos el código fuente y podemos ejecutarlo desde el menú *Run > Run Module* de la ventana de edición, o bien pulsando la tecla F5.

Sin embargo, este entorno se nos puede quedar algo corto o limitado si queremos gestionar varios archivos, y puede resultar algo engorroso estar cambiando entre el terminal y la ventana de edición.

2.2. Utilizando Geany

Si ya tenemos descargado e instalado Python, el intérprete `python` (o `python3`) estará ya correctamente configurado en nuestro sistema. En este caso, podemos emplear un IDE de propósito general como *Geany* para editar nuestros programas. Podemos instalarlo fácilmente desde su web oficial en Windows y en Mac. En el caso de Linux es mejor instalar el paquete *geany* junto con sus dependencias desde el gestor de dependencias del sistema operativo.

La mecánica es similar al caso anterior: con el menú *Archivo > Nuevo* creamos nuevos archivos fuente, que debemos guardar con el nombre adecuado (extensión `.py`) en la carpeta que elijamos. Después, editamos el código fuente y podemos ejecutarlos con el botón de *Ejecutar* en la barra superior de herramientas. Geany se autoconfigura con la instalación de Python, y utiliza automáticamente el comando `python` o `python3` para lanzar los programas. Podemos comprobarlo abriendo el panel *Construir > Establecer comandos de construcción* desde el archivo Python que estemos editando.

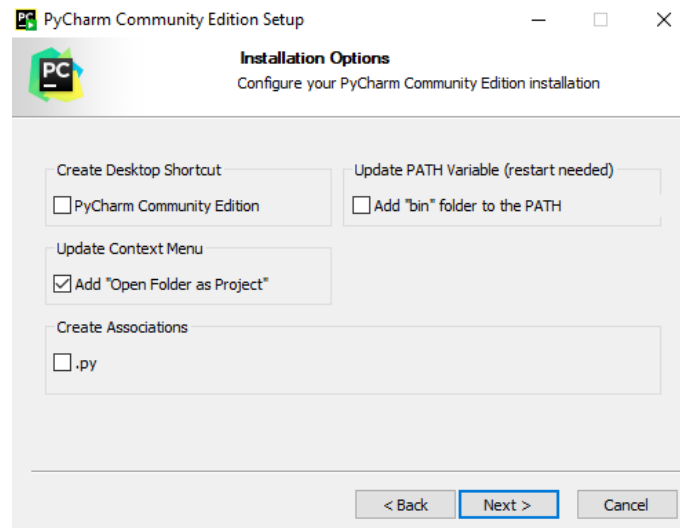
2.3. Intérpretes online

Una opción alternativa para desarrollar programas en Python, especialmente cuando son relativamente simples y cortos, es utilizar algún editor online, como [éste](#). Simplemente debemos escribir el código en el panel correspondiente, y hacer clic en el botón para ejecutar la aplicación, viendo el resultado en la consola derecha.

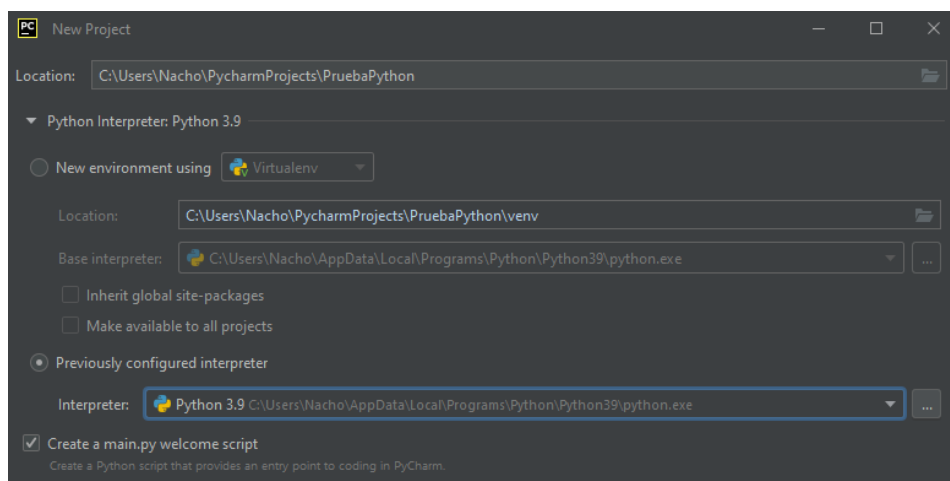
2.4. PyCharm

Pasamos a analizar ahora otro IDE más potente, útil para desarrollar proyectos complejos. Hablamos de **PyCharm**, un IDE desarrollado por la empresa JetBrains, también responsable de otros IDEs populares para otros lenguajes como IntelliJ (Java) o PhpStorm (PHP). Podemos descargar PyCharm gratuitamente (versión *Community*) en su [web oficial](#).

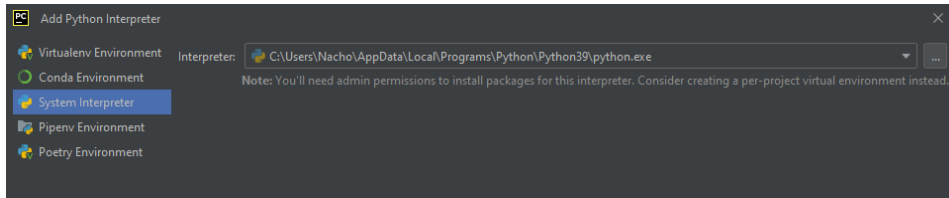
En la instalación hay poco que configurar. En todo caso, para Windows, elegir la casilla para poder abrir carpetas enteras como proyectos con un simple clic derecho:



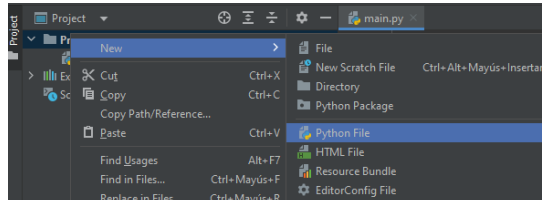
Una vez iniciado, crearemos un nuevo proyecto (*New project*). Por defecto los proyectos se guardarán en la subcarpeta *PyCharmProjects* de nuestra carpeta de usuario del sistema, aunque podemos cambiar la ubicación.



Es importante también destacar, en este paso, que deberemos elegir el entorno de ejecución de nuestro proyecto. Por defecto PyCharm creará un entorno virtual (*Venv*, *Virtual Environment*) para cada proyecto, lo que significará que nos instalará una serie de librerías predefinidas en el proyecto, y podremos añadir otras. Si no queremos hacer eso, deberemos elegir la casilla de *Previously configured interpreter* en la ventana anterior y elegir el intérprete del sistema.



De este modo, se creará una carpeta de código fuente vacía (con un archivo inicial llamado *main.py*), y podremos añadir nuevos archivos al proyecto haciendo clic derecho sobre él, y eligiendo *New > Python file*



Para ejecutar cualquiera de los archivos de nuestro proyecto, hacemos clic derecho sobre él y elegimos la opción de *Run*.

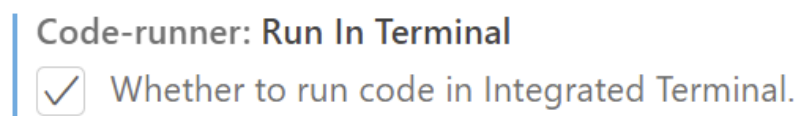
2.5. Visual Studio Code

Visual Studio Code ofrece unas funcionalidades similares a Geany, pero nos será de mayor utilidad cuando queramos gestionar un proyecto con varios archivos fuente. Podemos descargar VS Code de su [web oficial](#). En el caso de Windows y Mac, simplemente ejecutamos el instalador. En el caso de Linux (Ubuntu), desde el terminal, ubicado en la carpeta donde hemos descargado el archivo, ejecutamos este comando:

```
sudo dpkg -i nombre_archivo.deb
```

donde *nombre_archivo.deb* será el archivo con extensión *.deb* que habremos descargado.

Para poder trabajar con Python desde Visual Studio Code, debemos abrir la carpeta donde vayamos a editar los archivos Python (menú *Archivo > Abrir carpeta*), y luego crear los archivos y carpetas que queramos desde el panel izquierdo del explorador. También es recomendable instalar la extensión *Code Runner* y configurarla para poder ejecutar los programas Python en el terminal integrado de Visual Studio Code.



3. Nuestro primer programa Python

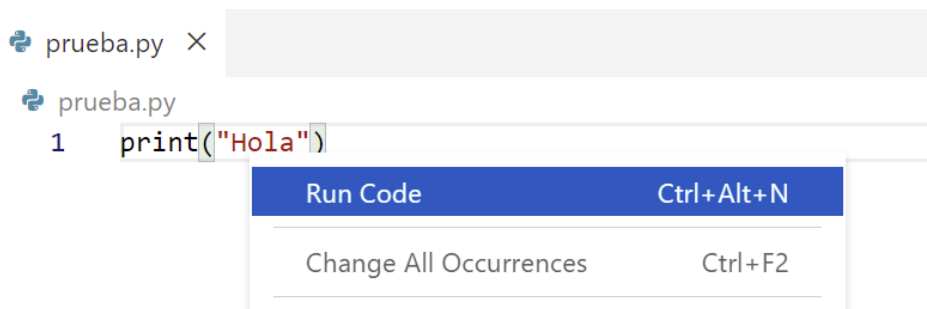
Vamos a crear nuestro primer programa Python que muestre un saludo por pantalla. Podemos emplear cualquiera de los IDEs comentados anteriormente. Creamos un archivo fuente Python llamado, por ejemplo,

`prueba.py` en la carpeta donde queramos trabajar, y lo guardamos con este contenido:

```
print("Hola")
```

A la hora de ejecutarlo:

- Si usamos IDLE, pulsamos F5
- Si usamos Geany, pulsamos el botón de *Ejecutar* de la barra de herramientas
- Si usamos el editor online de *repl.it*, pulsamos el botón de ejecutar o *Run*
- Si usamos PyCharm, hacemos clic derecho sobre el archivo y elegimos la opción de *Run*
- Si utilizamos Visual Studio Code y hemos instalado la extensión *Code Runner*, hacemos clic derecho sobre el código fuente y elegimos la opción *Run Code* que aparecerá en el menú contextual



El resultado en cualquier caso será el mismo, viendo por terminal el texto que hemos indicado en la instrucción `print`:

```
Hola
```

4. Otras herramientas

Además de los IDEs tradicionales para trabajar con Python existen otras posibilidades de trabajo, impulsadas en gran parte por la aplicación que ha tenido este lenguaje en la inteligencia artificial. Son especialmente útiles para trabajo colaborativo online, o para trabajo con *machine learning* y tratamiento de datos.

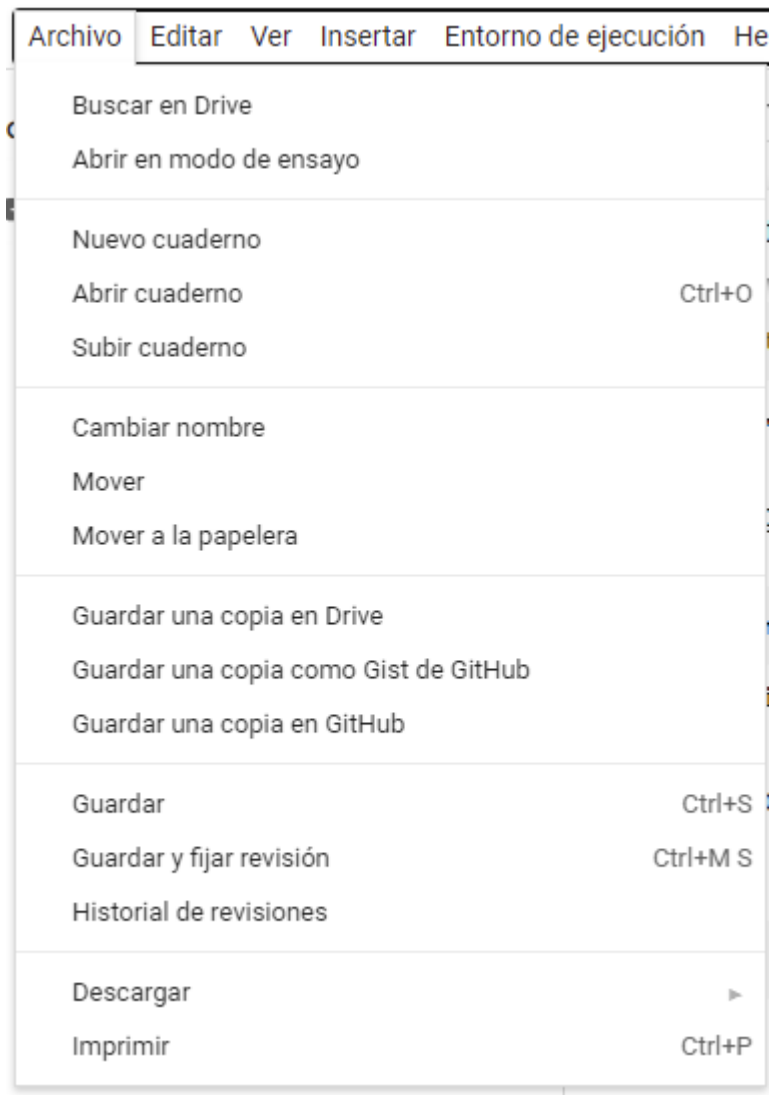
4.1. Google Colab

Google Colab es una de las muchas herramientas que ofrece Google en la nube. Nos permite definir documentos donde intercalar texto con código Python (y algún otro lenguaje, como R o Julia), de forma que podemos hacer tutoriales guiados. Además, facilita la integración con multitud de librerías ya incorporadas en Colab, como *NumPy* o *Matplotlib*, lo que permite trabajar fácilmente con conjuntos de datos y obtener representaciones gráficas en el mismo documento. Estos datos podemos incorporarlos de varias fuentes de forma automática, tales como hojas de cálculo de nuestra cuenta de Google Drive, o incluso repositorios GitHub.

4.1.1. Primeros pasos

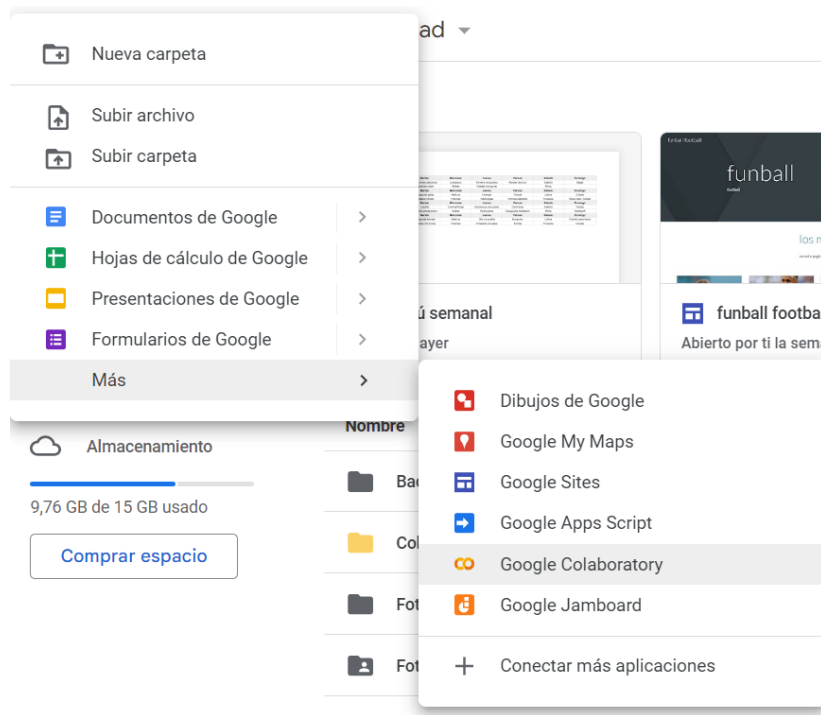
Para comenzar a trabajar con Colab, debemos acceder a su [web oficial](#) con nuestra cuenta de Google (lo que asociará Colab con el espacio en Google Drive de esa cuenta). Una vez dentro, veremos que Colab trabaja en base a unos documentos llamados **cuadernos**, que se guardan en un formato especial *ipynb*, compatible con otras herramientas como Jupyter, que veremos a continuación.

Desde el menú *Archivo* podemos abrir un cuaderno que tengamos previamente guardado en Drive, crear nuevos cuadernos, o subir archivos *ipynb* nuevos a nuestra cuenta.



NOTA: si accedemos a un cuaderno Colab de una cuenta que no es la nuestra, es recomendable copiarlo a nuestra cuenta de Drive, usando el menú *Archivo* > *Guardar una copia en Drive*, ya que de lo contrario los cambios que hagamos no se van a guardar.

También es posible crear nuevos documentos Colab desde nuestro panel principal de Google Drive, con el botón izquierdo *Nuevo*. Si no aparece la opción de *Colab*, podemos añadirla desde el apartado de *Conectar más aplicaciones*.



4.1.2. Trabajo con cuadernos. Operaciones básicas

Dentro de un cuaderno abierto, desde el menú *Insertar* podemos insertar encabezados de sección para dividir la estructura de nuestro documento, y en cada sección podemos incluir bloques de texto explicativo, o bloques de código que podemos ejecutar pulsando en la flecha a la izquierda del bloque o la combinación de teclas *Control + Intro*. El código se ejecuta en una máquina virtual remota gestionada por Google Colab.

```
+ Código + Texto
```

Para mostrar información por pantalla, usamos la instrucción `print`:


```
0.5 ✓ ▶ print("Hola, buenas")
    Hola, buenas
```

Para recoger información del usuario, usamos la instrucción `input`:

```
3.5 ✓ [2] nombre = input("Escribe tu nombre:")
    Escribe tu nombre:Nacho
```

Escribe a continuación el código para pedirle al usuario que escriba su nombre y mostrar un mensaje de saludo por pantalla. Por ejemplo "Hola, Nacho".

```
[ ]
```

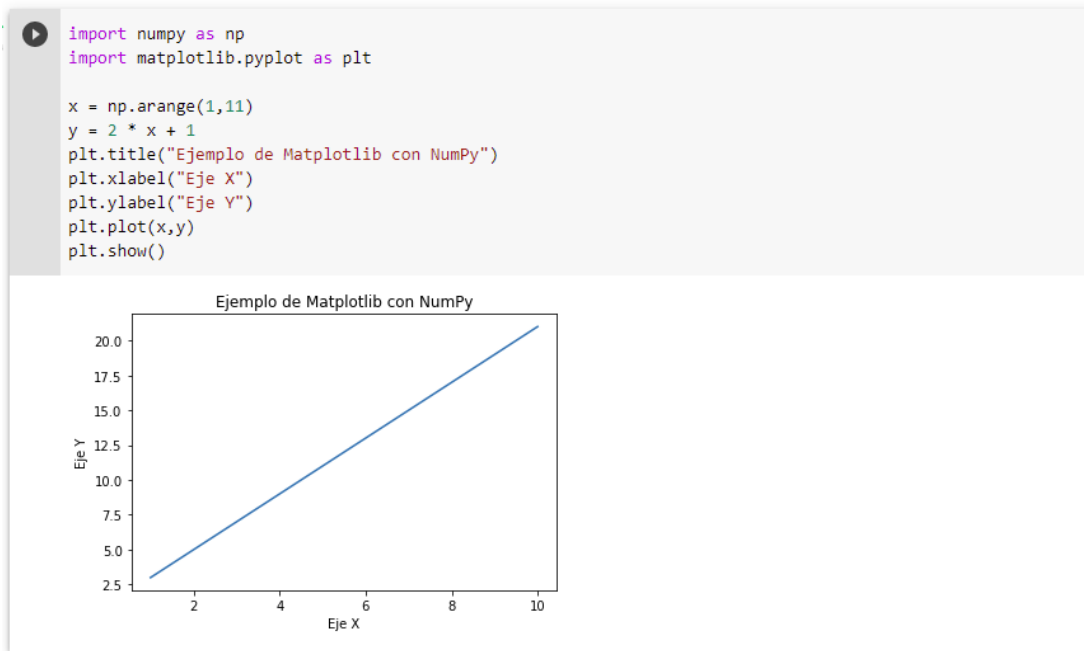
También podemos aprovechar los bloques de código para ejecutar instrucciones en el sistema operativo de la máquina remota donde se está ejecutando el código, anteponiendo un  delante:

```
!python -V
```

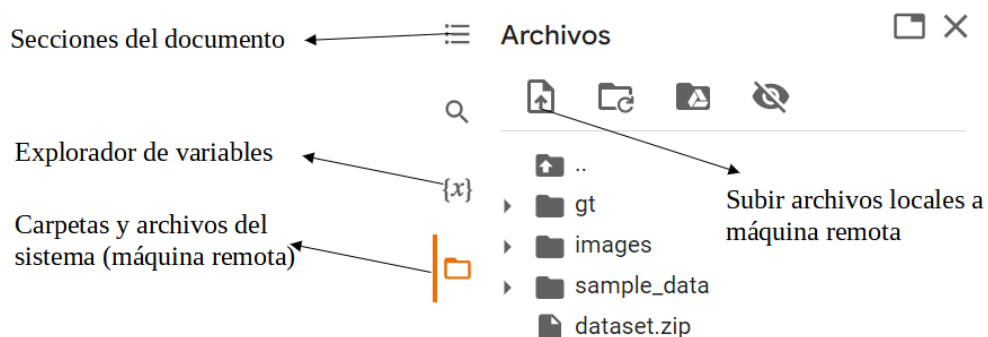

Finalmente, desde el menú *Archivo* anterior también podemos guardar los cambios, tanto en Drive (opción por defecto) como en GitHub si lo preferimos.

Como decíamos, una de las ventajas que ofrece Colab es que nos permite trabajar directamente en la nube con ciertas librerías ya instaladas y que pueden resultar muy útiles para el tratamiento de datos, tales como *NumPy* para tratamiento de arrays o *Matplotlib* para generación de gráficas, en el caso de Python. De este modo, se puede tener en un solo documento la explicación, los datos con los que trabajar y los resultados que se obtienen.

Aquí vemos un ejemplo de cómo trabajar con las librerías NumPy y Matplotlib, incorporadas por defecto en Colab



En la parte izquierda tenemos unos iconos de utilidad, para poder, por ejemplo, ver el árbol de secciones y subsecciones del documento actual, o también los archivos y carpetas que se han generado en la máquina virtual remota (y donde podemos también añadir archivos propios, o descargar archivos generados en dicha máquina).



4.1.3. Limitaciones y otras consideraciones

En cuanto a sus **limitaciones**, en la versión gratuita sólo se permite el uso de una GPU (para procesamiento paralelo, útil en el trabajo con redes neuronales), una RAM limitada y una actividad continuada de hasta 12 horas por sesión. Para activar la GPU deberemos hacerlo desde el menú *Entorno de ejecución - Cambiar tipo de entorno de ejecución*.

4.2. Jupyter

Jupyter es un conjunto de herramientas que ofrece una funcionalidad similar a Colab (gestionar cuadernos interactivos que intercalan texto y código Python), pero que debe instalarse de forma local en el sistema. De hecho, Colab es básicamente un servidor Jupyter instalado en Google.

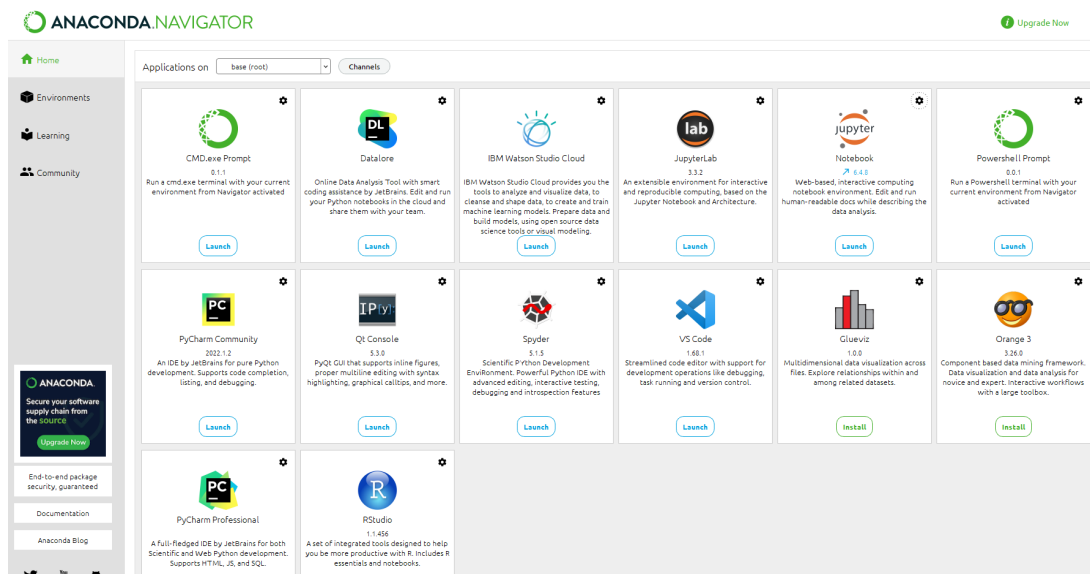
Aquí tenemos el enlace a la web oficial de Jupyter. Como podemos ver, se distinguen dos aplicaciones:

- *Jupyter Notebook*, que es algo más básico para editar archivos Jupyter (*ipynb*), y definir bloques de texto, código, etc.
- *JupyterLab*, que integra una serie de herramientas adicionales para trabajo más avanzado en *machine learning* o *data science*.

Como decimos, el principal inconveniente de estas herramientas es que se instalan en local, con lo que, por un lado, perdemos ese elemento colaborativo online que ofrece *Colab*, y por otro, tenemos que instalar manualmente en el sistema las librerías que necesitemos para nuestros trabajos (*NumPy*, *Matplotlib*, etc).

4.3. Anaconda

Anaconda es un pack de software que integra diferentes herramientas orientadas al trabajo con Python y otros lenguajes (como R), en ámbitos como *data science* o *machine learning*. Entre otras cosas, podemos instalar el propio entorno Python (o R), junto con algunos editores predefinidos (como PyCharm), así como el propio *Jupyter Notebook* y también algunas librerías típicas de *data science* o *machine learning* tales como *NumPy* o *TensorFlow*. Podemos obtener más información [aquí](#).



Como principal ventaja sobre el entorno *Jupyter* anterior, podemos decir que Anaconda ya incorpora Jupyter, y también incorpora las principales librerías para trabajo con *data science* o *machine learning*, con lo que nos facilita bastante el comenzar a trabajar en estos entornos.

NOTA: Anaconda incorpora su propia versión de Python y/o lenguaje R, que no interfiere con la que podamos tener instalada en el sistema previamente.