

## Aprende a programar Python



*Logo Python*

**Versión 0.99.2**



*Licencia CC by SA by @javacasm*

José Antonio Vacas Martínez

<https://elCacharreo.com>

Octubre 2022

## Primer programa: ¡Hola Python!

Vamos a empezar usando la consola REPL (a veces diré consola, otras REPL). Su propio nombre indica REPL (Read-Eval-Print Loop): es un bucle que lee nuestro código, lo evalúa y muestra el resultado continuamente. Cuando trabajamos así estamos usando el intérprete de Python en modo interactivo.

Nos va a permitir probar y ejecutar nuestro código de un modo interactivo. Yo lo uso para probar fragmentos de código que luego voy a introducir como parte de un programa.

Cuando nos referimos a un programa python, estamos hablando de un fichero, normalmente con extensión **.py**, donde hemos incluido órdenes para que se ejecuten de manera consecutiva. Cuando Python ejecuta uno de estos ficheros no funciona en modo interactivo, sino que ejecuta todas las órdenes del fichero hasta terminarlas todas.

### Usando la consola

Todo lo que vamos a hacer en la consola de Thonny podríamos hacerlo en una consola de python cualquiera. La ventaja de Thonny es que ya nos da todo integrado y facilita enormemente el arrancar. En los sistemas que tienen Python integrado, como por ejemplo Linux, si escribimos en cualquier shell “python3”, accedemos a la REPL de Python.

Entramos en la consola de Thonny (recuerda comprobar que tenemos al menos una versión 3.9, si no es así entra ejecutando python3)

y ponemos tras el prompt “>>>”

```
print("¡Hola Python!")
```

Al pulsar la tecla Enter veremos que aparece en pantalla el texto

```
¡Hola Python!
```

A screenshot of a terminal window titled 'Consola' and 'Excepción'. The prompt is 'Python 3.8.0 (/usr/bin/python3.8)'. The user enters '>>> print(";Hola Python!")' and the output is ';Hola Python!'. The prompt '>>>' is followed by a vertical bar '|'.

```
Python 3.8.0 (/usr/bin/python3.8)
>>> print(";Hola Python!")
;Hola Python!
>>> |
```

*"Hello Python" en la consola*

En la imagen vemos que nos resalta la sintaxis, indicando con colores distintos cada parte del código.

También vemos en la imagen que aparece la versión del intérprete de Python que estamos usando.

También podemos hacer que se impriman números pero en este caso no son necesarias las comillas.

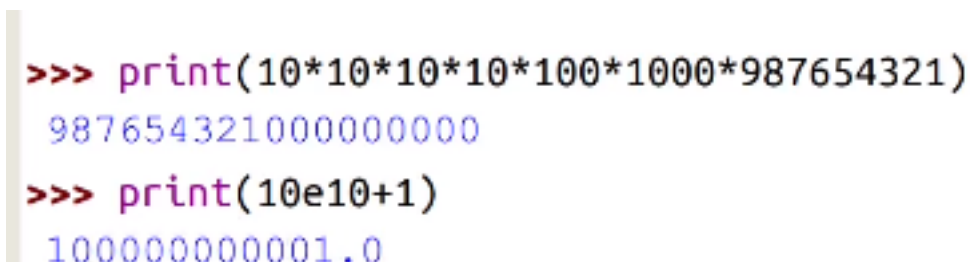
```
print(314)
print(3.14)
```

Podemos usar números enteros o decimales, usando el punto decimal "." como separador.

Incluso podemos poner operaciones ...

```
print(314 + 50)
```

Podemos hacer cálculos con muchas cifras y Python se porta realmente bien, incluso podemos usar notación científica:

A screenshot of a terminal window showing two Python commands and their outputs. The first command is '>>> print(10\*10\*10\*10\*100\*1000\*987654321)' and the output is '987654321000000000'. The second command is '>>> print(10e10+1)' and the output is '100000000001.0'.

```
>>> print(10*10*10*10*100*1000*987654321)
987654321000000000
>>> print(10e10+1)
100000000001.0
```

*Cálculos matemáticos grandes*

¿Qué ocurre si ponemos?

```
print("314 + 50")
```

La diferencia está en que al poner las comillas estamos diciéndole que muestre ese contenido literalmente, si no ponemos las comillas intenta evaluar la expresión.



## Ejercicio Propuesto

1. Modifica este código para que imprima otro texto diferente

## Uso de la Consola o REPL

Ya hemos trabajado un poco con la consola, y hemos visto que podemos ejecutar pequeños programas.

Vamos a resumir ahora algunas posibilidades que tiene:

- Podemos recuperar las últimas instrucciones usando las teclas “Flecha Arriba”  y “Flecha abajo”  del cursor del teclado
- Ctrl + L borra todo el contenido de la consola, pero seguimos pudiendo recuperar los comandos anteriores. Nos muestra el prompt de Python “>>>”
- Ctrl + D Reinicia la consola, reiniciando el intérprete. Nos mostrará la versión del intérprete que estamos usando.
- Ctrl + C Detiene el código que se esté ejecutando en ese momento.
- Podemos acceder a la documentación incluida en el intérprete de python con la función **help()**. Al ejecutar esta función entramos directamente en modo de ayuda interactivo y nos dará la documentación sobre las funciones cuyo nombre hemos indicado. Saldremos de la ayuda interactiva con “quit”
- También podemos usar la ayuda directamente dándole el nombre del que queremos obtenerla usando **help(función)**, como por ejemplo

```
help(print)
```

## Errores de sintaxis

Vamos a ver con un poco más de detalle: Hemos usado la función (más adelante veremos qué es una función, pero me gusta ir adelantando términos para que se vayan fijando) **print()** con comillas simples ‘.’ el texto que queremos que aparezca. También podemos usar comillas dobles “.”. ¿Por qué esta variedad?, creo que por motivos históricos pero ahora nos permite hacer cosas como estas:

```
print('Hola "Python"')  
print("Hola 'Python'")
```

Siempre tenemos que tener cuidado de que las comillas funcionan como los paréntesis en matemáticas: El último en abrir el primero que tengo que cerrar. Por tanto no podemos alternar los tipos.

¿Y qué ocurre si no hacemos? Pues que se genera un error de sintaxis, también si nos olvidamos de poner alguna, o no cerramos los paréntesis....

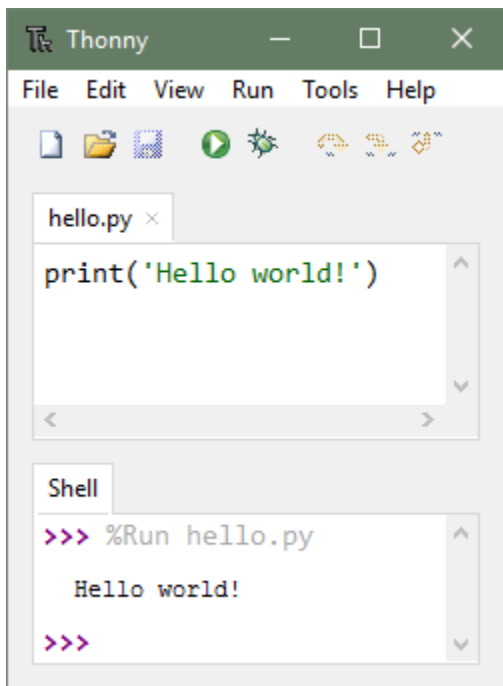
Afortunadamente Thonny nos ayuda resaltando nuestro código y si nos olvidamos de cerrar una cadena, nos lo indica marcando toda la línea en blanco con un color distinto. También ocurre algo similar si nos olvidamos de cerrar los paréntesis.

También se producen errores cuando ponemos alguna operación matemática incompleta.

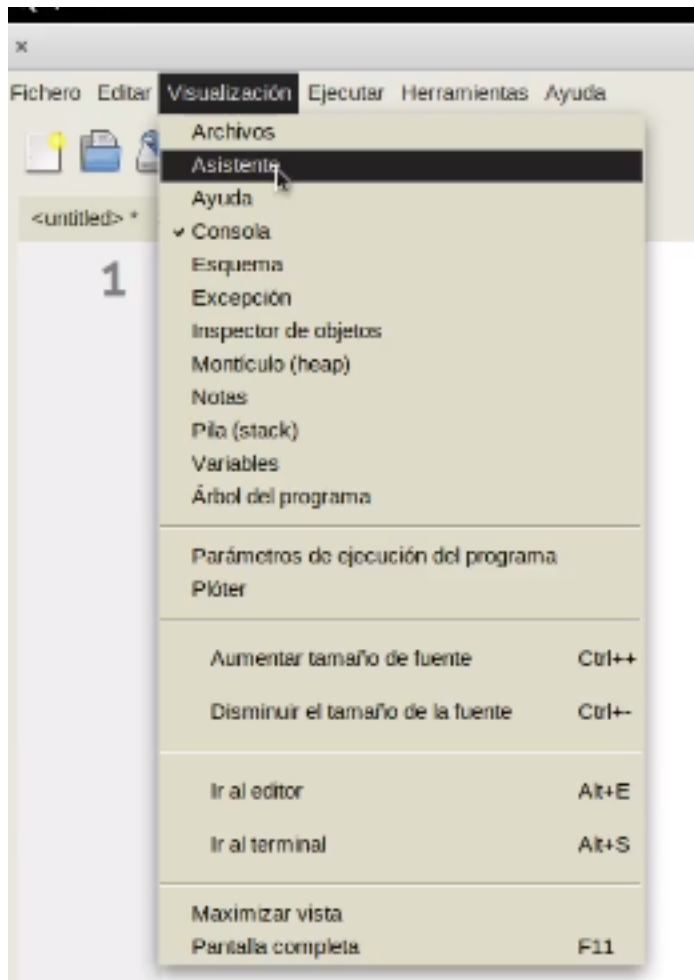
## Uso del IDE Thonny

Vamos a repasar ahora algunas de las características de Thonny. Dejaremos las más avanzadas para cuando las necesitamos.

- La pantalla principal está dividida en la parte central para el editor de código y otras pestañas alrededor. En la parte de abajo está la consola:

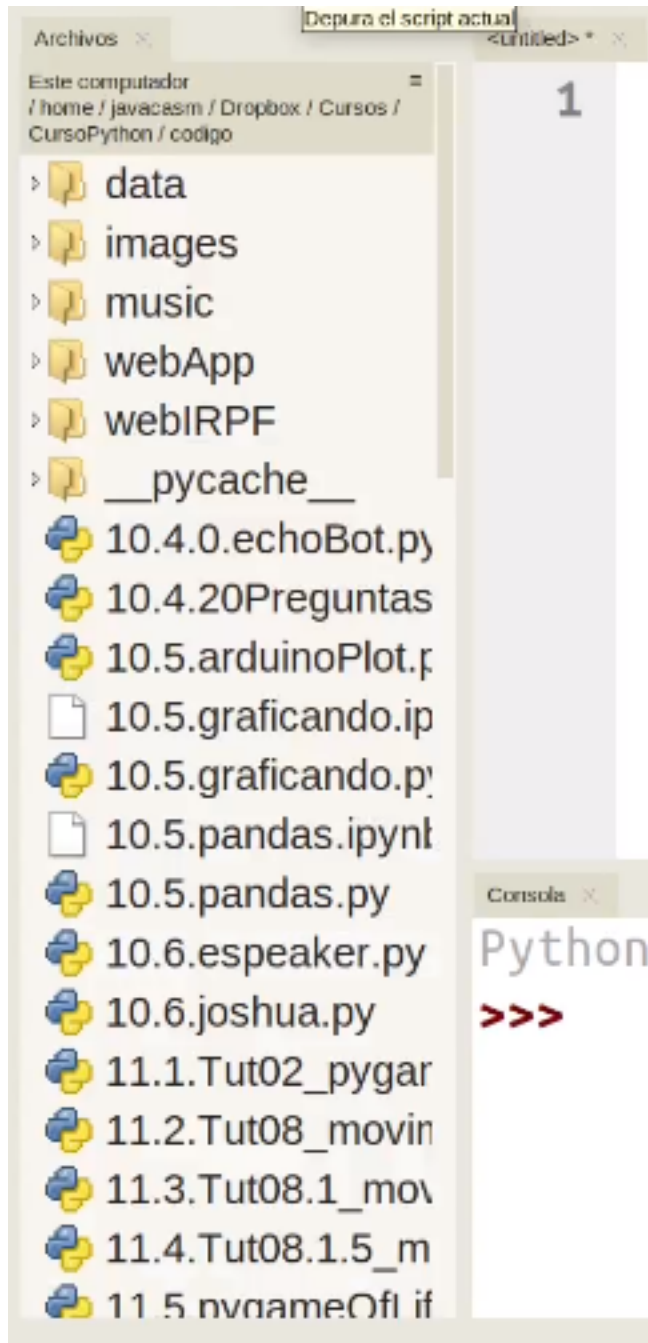


- Podemos abrir estas pestañas desde el menú **Visualización**



Aquí están las ventanas que más adelante nos permitirá ver los valores de las variables o la memoria.

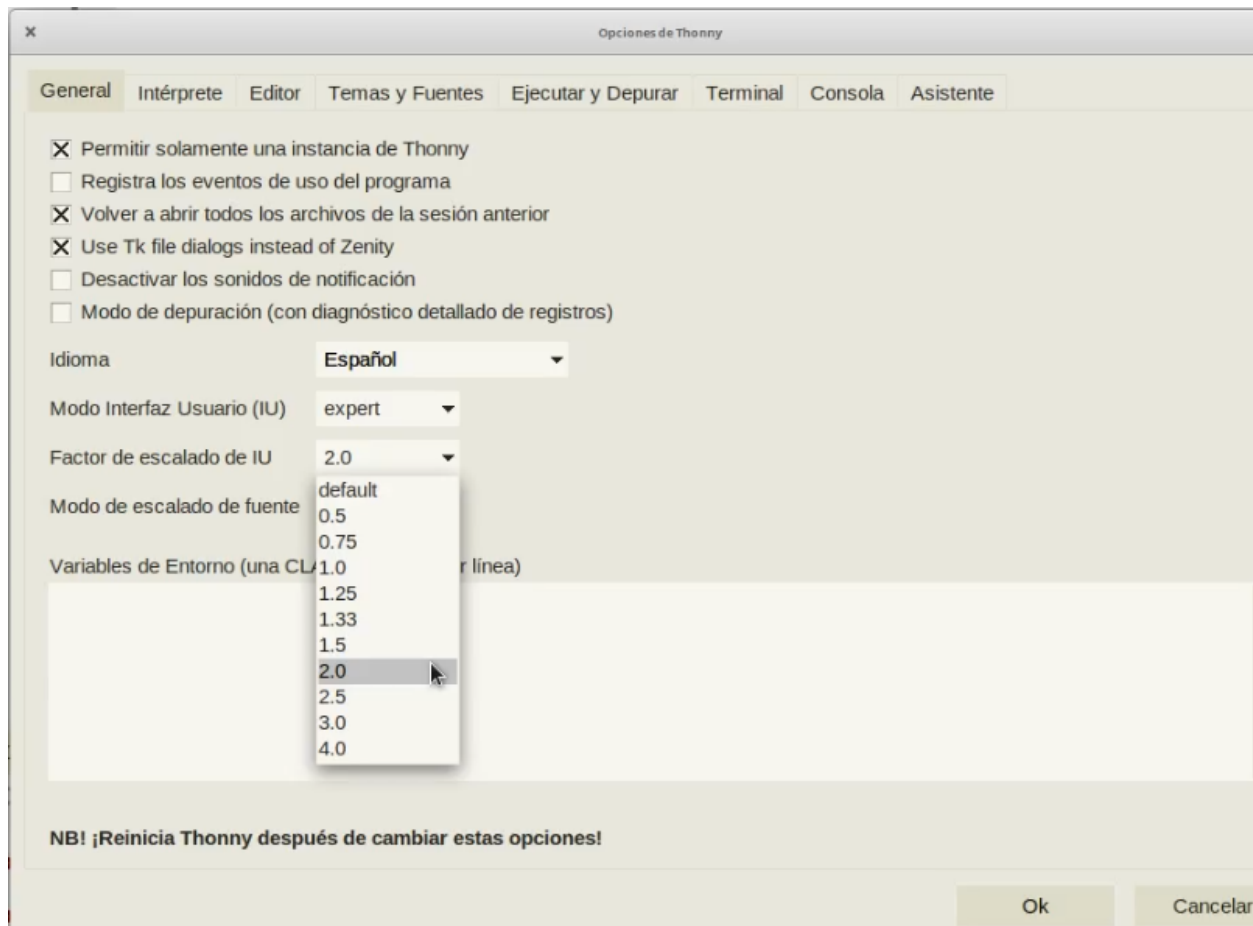
- Hay una pestaña que sirve como navegador de archivos, desde donde podemos navegar por carpetas, crearlas o borrar ficheros.



### *Pestaña navegador de Archivos*

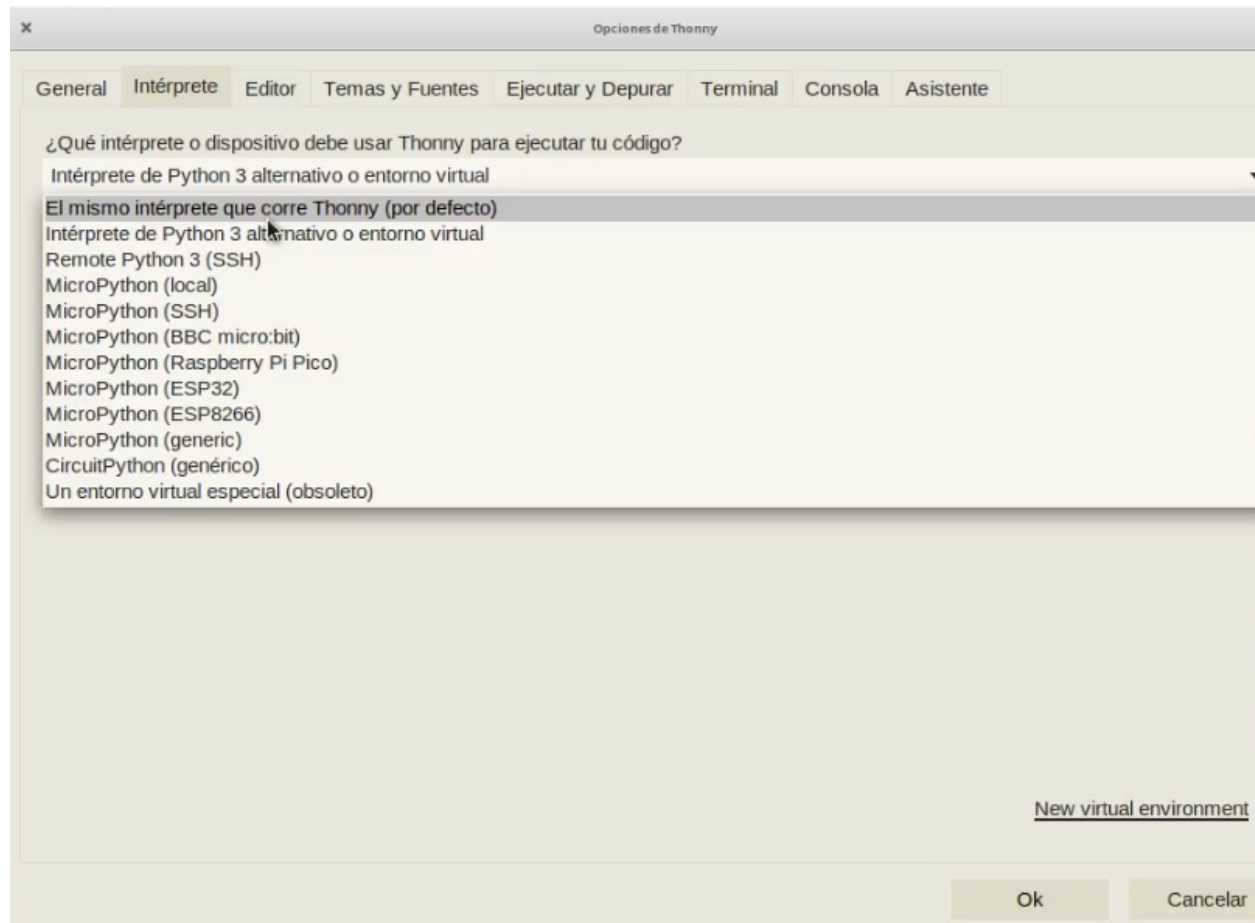
- Ctrl + "+" y Ctrl + "-" aumenta o disminuyen el tamaño del tipo de letra
- Desde la barra de herramientas podemos **Abrir**, **Guardar** y **Nuevo** los ficheros de código. Junto a ellos están los iconos de **Ejecutar** y **Depurar**, con los que trabajaremos en breve.

- En el menú **Edición** tenemos lo relacionado con **Copiar/Pegar** y algunas herramientas para el código que ya iremos viendo.
- En el menú **Herramientas** encontramos, las opciones para instalar paquetes Python y para instalar Complementos de Thonny
- También está el menú **Herramientas**, donde encontramos las **Opciones** que a su vez incluyen varias pestañas. Veamos algunas:
  - En la pestaña General podemos seleccionar el idioma y además yo le tengo puesto el **Factor de escalado UI** a 2.0 para que el interfaz se vea más grande.



- En la pestaña **Intérprete** podemos seleccionar la versión de Python que usaremos. Normalmente se usa la “Incluída con Thonny” pero pudiera ocurrir que queramos usar otra

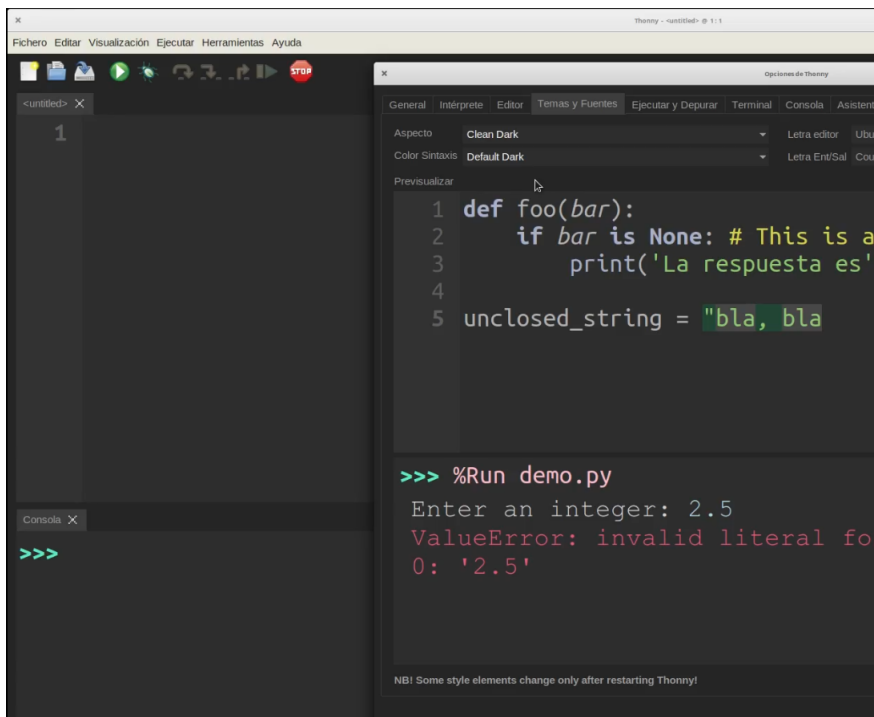
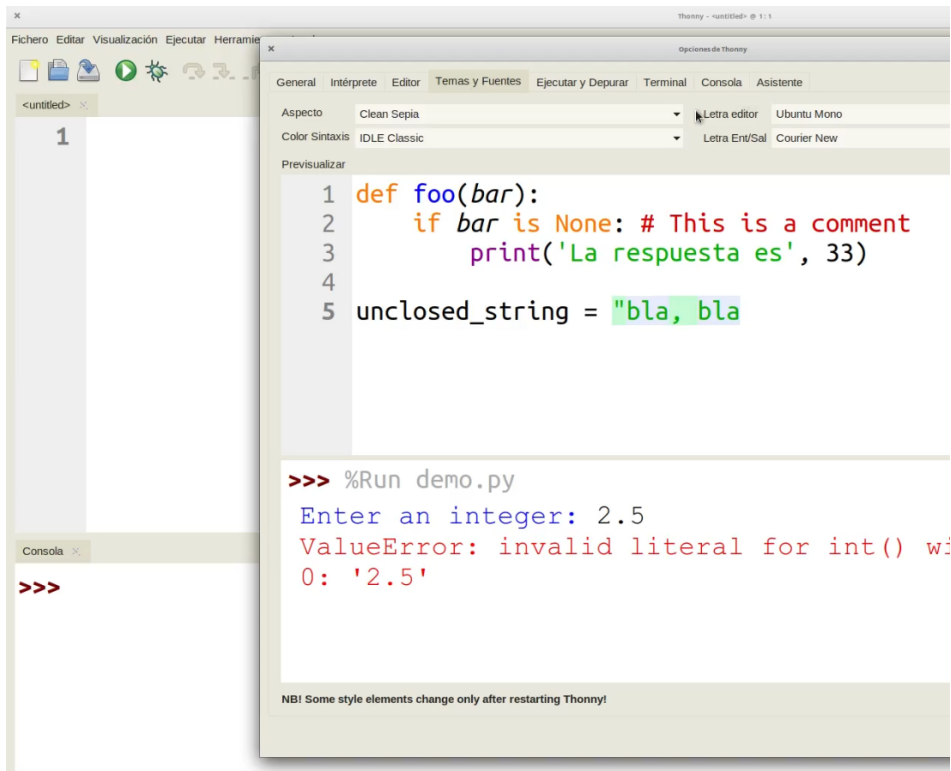




Thonny nos permite programar Python para todo tipo de equipos, es decir podemos programar en Python dispositivos diferentes a un PC, como son **micro:bit**, **ESP8266** o **ESP32**, en los que usaremos una versión reducida de Python, llamada Micropython, pero este lo veremos en otro curso.

- En la pestaña **Editor** configuramos cómo queremos que se resalte el código.
- En **Temas y fuentes** seleccionamos el aspecto, tema, tamaño de letra, etc. Cambiando el **Aspecto** y el **Editor** a Dark cambiaremos el aspecto de todo el entorno.

¿Cuál prefieres?



## Usando Ficheros para guardar nuestro código

Desde la consola podemos ejecutar cualquier cosa pero las órdenes que usemos se perderán al salir

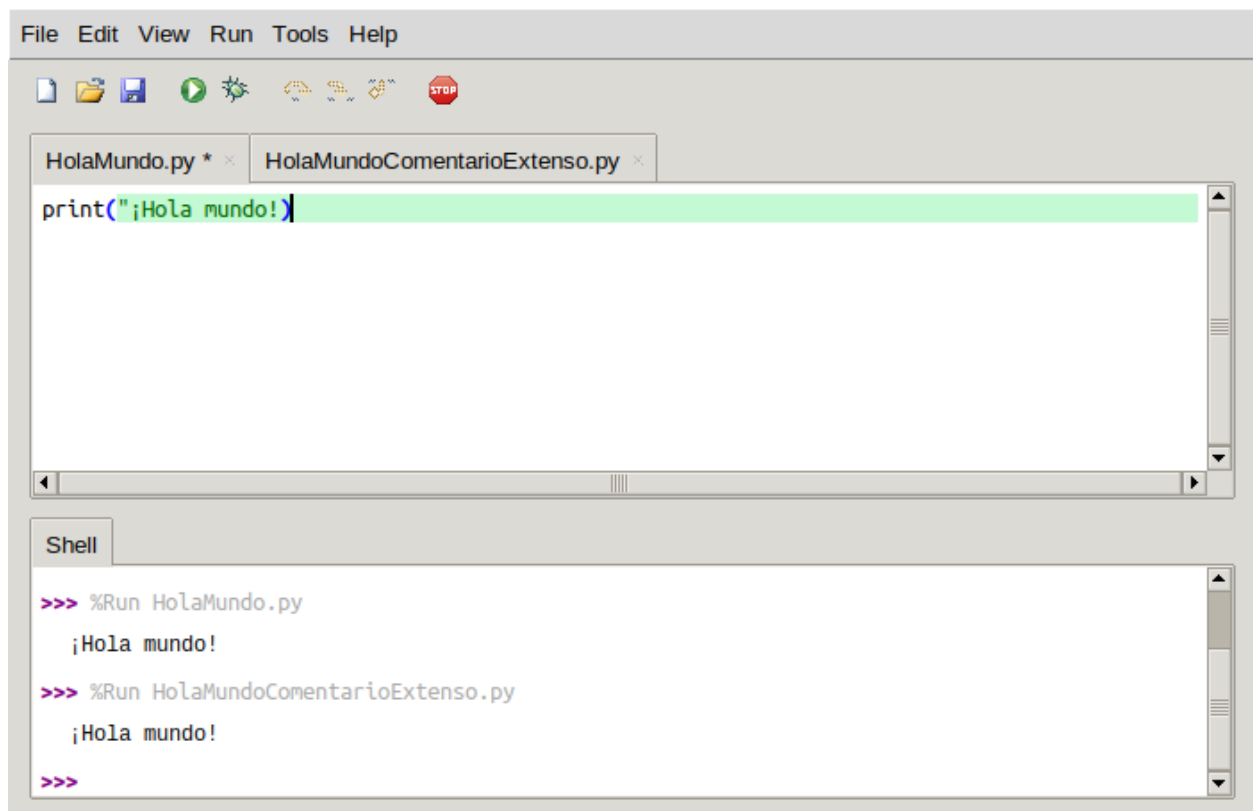
Por ello vamos a ver cómo podemos trabajar con ficheros donde incluiremos todas nuestras órdenes

Desde Thonny podemos crear carpetas y si somos ordenados crearemos una llamada “Hello World” para poner dentro el fichero correspondiente. Creamos un nuevo fichero que guardaremos en esa carpeta como **HolaMundo.py** con el siguiente contenido

```
print("¡Hola mundo!")
```

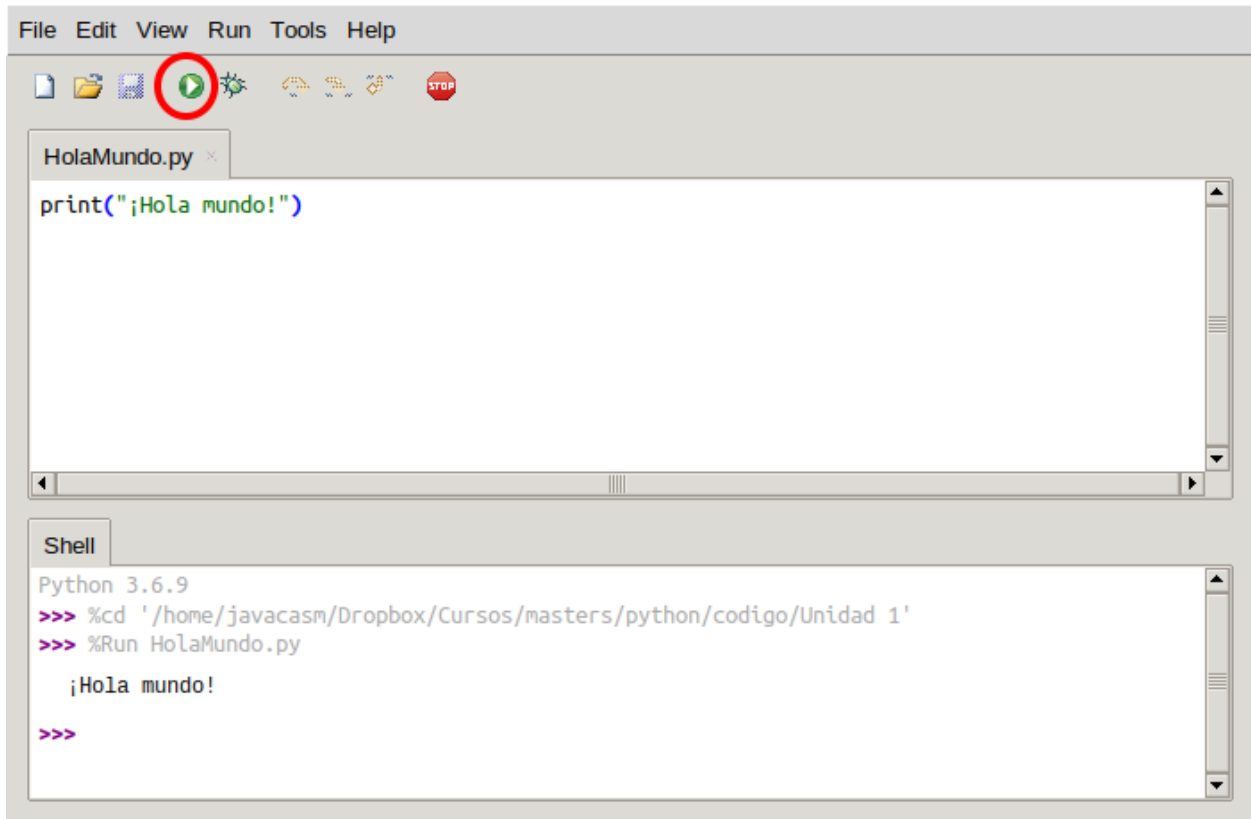
Se suele utilizar la extensión **.py** para indicar que el fichero contiene código python

Vemos que el editor resalta la sintaxis indicándonos las distintas partes. Esto nos puede ayudar si hemos cometido un error. Por ejemplo el editor Thonny nos resalta la línea si nos hemos olvidado de cerrar unas comillas



*Error de sintaxis en Thonny*

Desde el editor Thonny también podemos ejecutar nuestro programa pulsando el botón **Ejecutar**



### *“Hola Mundo” en Thonny*

Veremos en la parte de la consola el resultado y después el prompt esperando un nuevo comando

Los programas en Python siempre se ejecutan desde la primera línea hasta la última y al llegar a esta terminan, no ocurre como en otros entornos (Arduino por ejemplo) donde el código se repite automáticamente. Si queremos que algo se repita se lo tendremos que decir explícitamente.

## Comentarios

Ahora vamos a modificar el programa añadiendo un **comentario**. Un **comentario** es una indicación que se incluye en el código pero que sólo sirve para el programador, es decir no tendrá ningún efecto en el resultado del programa pero da información a quien lee el código

Si nuestro comentario sólo ocupa una línea (o parte de ella) sólo tenemos que añadir el carácter **#** y a partir de éste carácter el intérprete ignorará lo que hayamos escrito

```
# Nuestro primer programa en python que muestra el mensaje ¡Hola mundo!  
print("¡Hola mundo!")
```

Si queremos añadir un comentario que ocupe varias líneas podemos encerrar entre triples comillas dobles `"""` todo el texto

```
"""  
Nuestro primer programa en python que muestra el mensaje ¡Hola mundo!  
Escrito por @javacasm  
03/03/2021  
Licencia CC by sa  
"""  
print("¡Hola mundo!")
```

Si ejecutamos cualquier de las 3 versiones veremos que el resultado es exactamente el mismo

## Cuidados al escribir código

Las líneas no pueden empezar con un espacio o tabulador porque es lo que usa Python para agrupar líneas, y ya veremos las normas que se tienen que cumplir

Detrás del signo de comentario de línea `#` da igual lo que pongamos, porque se ignora todo el resto de la línea

## Depuración

Durante el proceso de creación de nuestro código, cuando estamos trabajando con nuestro programa nos encontramos con la necesidad de depurar el código. Es decir, de encontrar los posibles errores y revisar de una forma más o menos exhaustiva todos aquellos pasos que se van dando.

Para ello es muy cómodo el uso de las funciones de depuración que nos facilitan el ejecutar nuestro programa paso a paso, es decir línea a línea e ir viendo en cada uno de estos pasos los valores que van tomando las diferentes variables y cómo van funcionando las distintas partes de nuestro programa.



### Iconos depuración

Veamos las distintas opciones de depuración y los iconos para usarlas



### Depuración Thonny Play

Cuando ejecutamos normalmente nuestro programa, pulsando sobre el icono del “Play”, estas funciones de depuración no se pueden utilizar.



### Depuración Thonny Debug

Para usarlas tenemos que ejecutar nuestro código en **Modo Depuración**, pulsando sobre icono del “bichito verde” (por si quieres saber el [origen del término bug](#))



### Depuración Thonny Saltando

Cuando estamos ejecutando un programa en modo depuración podemos hacer que nuestro código se vaya ejecutando línea línea usando la opción **saltando**.

Veremos que la línea actual se resalta

```
1  """
2  Primer programa
3  Que he hecho con Thonny
4  """
5  print("Hola Python") # Primera linea print("hola")
6  print("Hola Mundo")  # segunda linea
```

### Depuración Línea

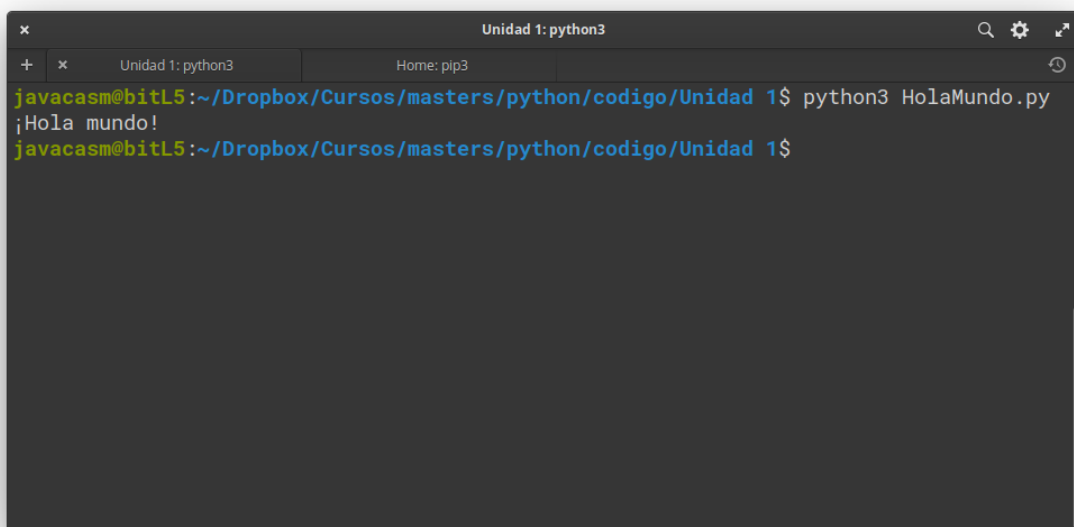
## Ejecución desde línea de comandos

Si tenemos instalado el intérprete de comandos independiente o nuestro sistema lo incluye, podemos

```
python3 HolaMundo.py
```

(En algunos sistemas operativos el nombre de los ficheros es sensible a mayúsculas/minúsculas por lo que tendrás que ejecutarlo tal y como lo creaste)

Y veremos el resultado

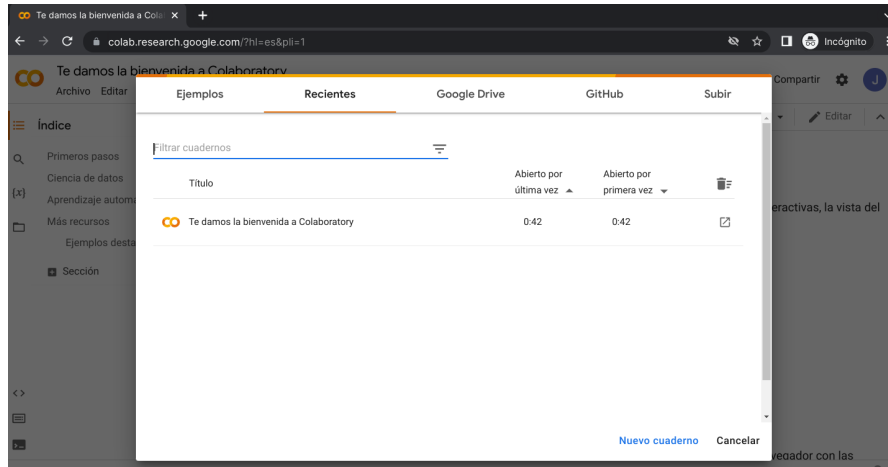


```
Unidad 1: python3
javacasm@bitL5:~/Dropbox/Cursos/masters/python/codigo/Unidad 1$ python3 HolaMundo.py
¡Hola mundo!
javacasm@bitL5:~/Dropbox/Cursos/masters/python/codigo/Unidad 1$
```

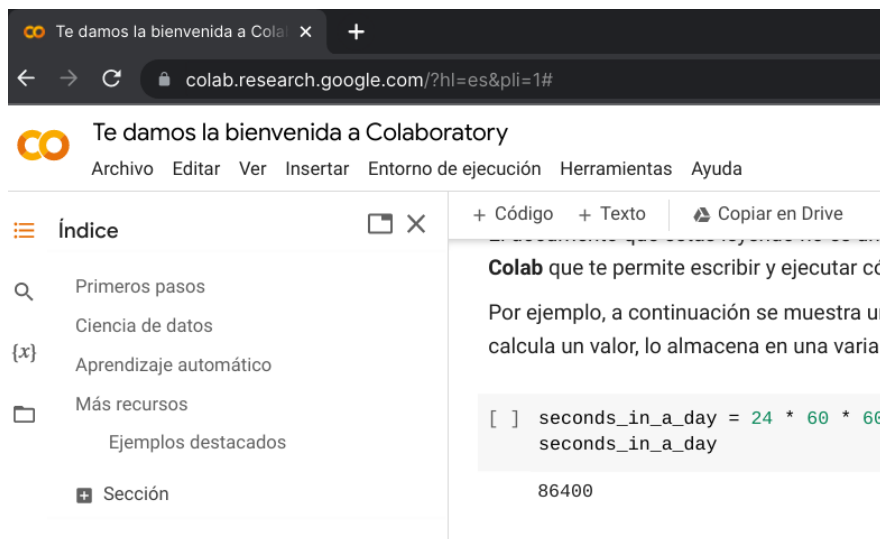
## Empezando a usar Google Colab

Para empezar con Google Colab necesitamos una cuenta de gmail. Desgraciadamente a día de hoy los correos corporativos de Google Suite de EducaAnd no los incluyen entre sus aplicaciones.

Para empezar entraremos en la [página de Colab](#) e iniciamos sesión con nuestro usuario de correo de Gmail. Una vez dentro nos encontramos con un documento de bienvenida:



En este documento tenemos un tutorial, con vídeos, para aprender a usar Google Colab.



## Primer programa: ¡Hola Python!

Es una costumbre de todo programador novato, que el primer programa ha de mostrar en pantalla un saludo.

Para crear nuestro primer documento Colab:

- Pulsamos en el botón “Nuevo Cuaderno” o desde el menú “Archivo” -> “Nuevo Cuaderno” y veremos un nuevo documento que recuerda bastante a los de Google Suite.
- Pulsamos sobre el nombre del documento “Untitled0” y lo renombramos por “Hola Mundo”



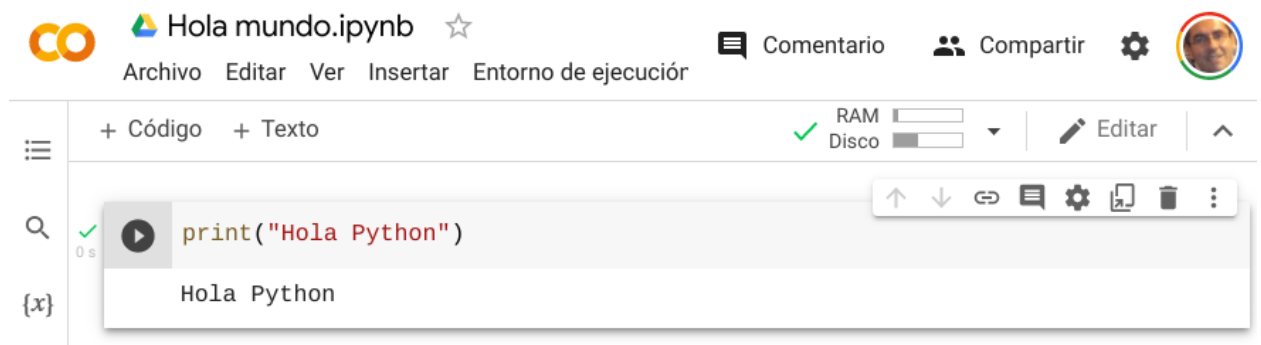
- Escribimos nuestra primera línea de código:

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

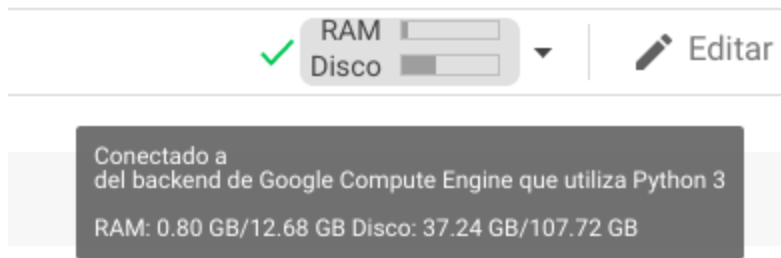
que significa: Muestra en pantalla el texto entre comillas (podemos usar comillas simples `'...'` o comillas dobles `"..."` pero en cada texto del mismo tipo)



- Ahora pulsamos el círculo con el signo Play y nuestro código se ejecutará, mostrando el texto que había entre comillas. También podemos ejecutar el código pulsando Ctrl+Enter.



Cuando ejecutamos por primera vez vemos que se muestra un mensaje de “Conectando con el servidor” y tras la conexión aparecen los recursos que estamos consumiendo, en forma de memoria RAM y de almacenamiento en Disco:



Si dejamos el ratón encima nos muestra los límites que tenemos con nuestro usuario actual. Si necesitáramos más podemos contratarlo.

Si queremos cambiar el texto podemos tocar sobre el código y modificar la parte entre comillas.

Podemos añadir otra línea de código pulsando sobre “+ Código” y añadir código para mostrar distintos textos.

También podemos añadir textos que se mostrarán en nuestro documento. Podemos conseguir formatear el texto usando los botones del bloque de texto o un sencillo formato llamado Markdown.

Mezclando texto y código podemos conseguir generar documentos interactivos de manera muy sencilla.

En la imagen vemos que en nuestro código nos resalta la sintaxis, indicando con colores distintos cada parte del código.

También podemos hacer que se impriman números, en este caso no son necesarias las comillas y usaremos

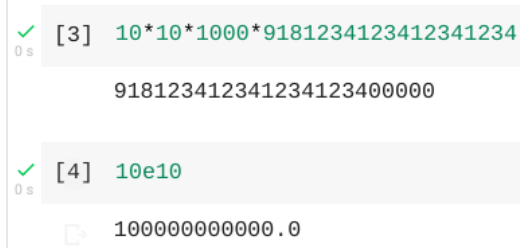
```
print(314)
print(3.14)
```

Podemos usar números enteros o decimales, usando el punto decimal “.” como separador.

Incluso podemos poner operaciones usando los tradicionales operadores aritméticos (usando el asterisco “\*” para indicar multiplicación)

```
print(314 + 50)
```

Podemos hacer cálculos con muchas cifras y Python se porta realmente bien, incluso podemos usar notación científica (10e6 para 1000000):



```
[3] 10*10*1000*9181234123412341234
918123412341234123400000

[4] 10e10
100000000000.0
```

¿Qué ocurre si ponemos?

```
print("314 + 50")
```

La diferencia está en que al poner las comillas estamos diciéndole que muestre ese contenido literalmente, si no ponemos las comillas intenta evaluar la expresión.

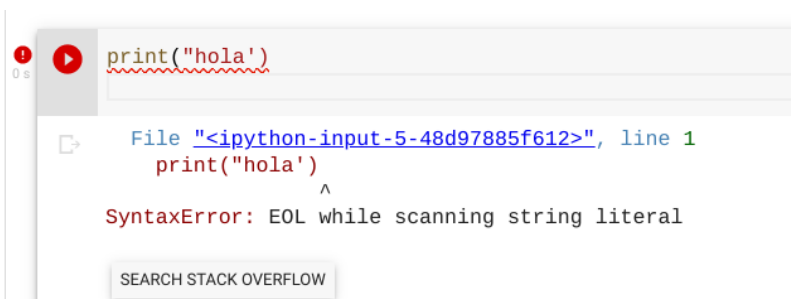
## Errores de sintaxis

Vamos a ver con un poco más de detalle: Hemos usado la función (más adelante veremos qué es una función, pero me gusta ir adelantando términos para que se vayan fijando) **print()** con comillas simples `'..'` el texto que queremos que aparezca. También podemos usar comillas dobles `".."`. ¿Por qué esta variedad?, creo que por motivos históricos pero ahora nos permite hacer cosas como estas:

```
print('Hola "Python"')
print("Hola 'Python'")
```

Siempre tenemos que tener cuidado de que las comillas funcionan como los paréntesis en matemáticas: El último en abrir el primero que tengo que cerrar. Por tanto no podemos alternar los tipos.

¿Y qué ocurre si no hacemos? Pues que se genera un error de sintaxis, también si nos olvidamos de poner alguna, o no cerramos los paréntesis....



```
print('hola')
```

File "<ipython-input-5-48d97885f612>", line 1  
print('hola')  
^  
SyntaxError: EOL while scanning string literal

SEARCH STACK OVERFLOW

Afortunadamente Colab nos ayuda resaltando nuestro código y si nos olvidamos de cerrar una cadena, nos lo indica marcando toda la línea en rojo. También se producen errores cuando ponemos alguna operación matemática incompleta.

## Uso del editor de Colab

- Ya hemos visto que podemos editar cualquier línea escrita tanto de código como de texto o de su formato
- Podemos pulsar entre las bloques ya escritos y añadir contenidos de cualquier tipo
- Igual que con cualquier documento de Google Suite, podemos copiarlo o compartirlo con otros usuarios, o ver el historial de cambios.
- También podemos descargar desde “Archivo” -> “Descargar”