

```

import threading, socket, time
class sock(threading.Thread):
    def __init__(self):
        self.sck=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        threading.Thread.__init__(self)
        self.flag=1
    def connect(self, addr, port, func):
        try:
            self.sck.connect((addr, port))
            self.handle=self.sck
            self.todo=2
            self.func=func
            self.start()
        except:
            print "Error: could not connect"
    def listen(self, host, port, func):
        try:
            self.sck.bind((host, port))
            self.sck.listen(5)
            self.todo=1
            self.func=func
            self.start()
        except:
            print "Error: Could not bind"
    def run(self):
        while self.flag:
            if self.todo==1:
                x, ho=self.sck.accept()
                self.todo=2
                self.handle=x
            else:
                dat=self.handle.recv(4096)
                self.data=dat
                self.func()
    def send(self, data):
        self.handle.send(data)
    def close(self):
        self.flag=0
        self.sck.close()

```

DAM/DAW

PROGRAMACIÓN

01.3

Introducción a Python

Indice

- [Características](#)
- [Notas históricas](#)
- [Versiones](#)
- [Instalación](#)
- [La consola de Python](#)
- [IDLE](#)
- [IDE's avanzados](#)

```
import threading, time
class sock(threading.Thread):
    def __init__(self):
        self.sock=socket(socket.AF_INET, socket.SOCK_STREAM)
        threading.Thread.__init__(self)
        self.flag=1
    def connect(self, addr, port, func):
        try:
            self.sock.connect((addr, port))
            self.handle=self.sock
            self.todo=2
            self.func=func
            self.start()
        except:
            print "Error: Could not connect"
    def listen(self, host, port, func):
        try:
            self.sock.bind((host, port))
            self.sock.listen(2)
            self.todo=1
            self.func=func
            self.start()
        except:
            print "Error: Could not bind"
    def run(self):
        while self.flag:
            if self.todo==1:
                x, ho=self.sock.accept()
                self.todo=2
                self.client=ho
                self.handle=x
            else:
                dat=self.handle.recv(4096)
                self.data=dat
                self.func()
            if not dat:
                self.send(self.data)
                self.handle.send(data)
            self.close(self)
            self.flag=0
            self.sock.close()
```

Características (I)

Python es un lenguaje de programación potente y fácil de aprender. En líneas generales, podemos considerar este lenguaje como:

- **de alto nivel**, con un fuerte grado de abstracción de las particularidades del hardware, facilitando el empleo del lenguaje natural y, por tanto, su aprendizaje.
- **de propósito general**, diseñado para la escritura de software en cualquier ámbito de aplicación.
- **interpretado**, de forma que sus instrucciones se ejecutan secuencialmente por el intérprete de Python sin necesidad de una compilación previa.

Características (y II)

- Python soporta diferentes **paradigmas de programación** como la programación estructurada o la orientada a objetos
- Es **multiplataforma**. Existen intérpretes de Python para Linux, Windows y Mac OS X
- Es de **tipado dinámico** y dispone de un sistema de **gestión automática de la memoria**.
- Un objetivo de diseño en su desarrollo fue la **extensibilidad**. Dispone de una amplia biblioteca de módulos que nos permiten el desarrollo de aplicaciones destinadas a todo tipo de fines que podemos ampliar con módulos propios o de terceros desarrollados en Python, C ó C++.

Notas históricas (I)

- El desarrollo de Python se inició a finales de los ochenta como un proyecto personal del programador holandés **Guido van Rossum** que, en ese momento, se encontraba trabajando en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica) en Holanda.
- El nombre del lenguaje proviene de su afición por la serie Monty Python's Flying Circus de la BBC.
- Tras una publicación interna previa en el CWI, van Rossum publicó en 1991 el código de la versión 0.9.0 en el grupo de noticias **alt.sources** con objeto de abrir el desarrollo de python a la comunidad de programadores.
- El proyecto alcanzaría la versión 1.0 en enero de 1994.

Notas históricas (y II)

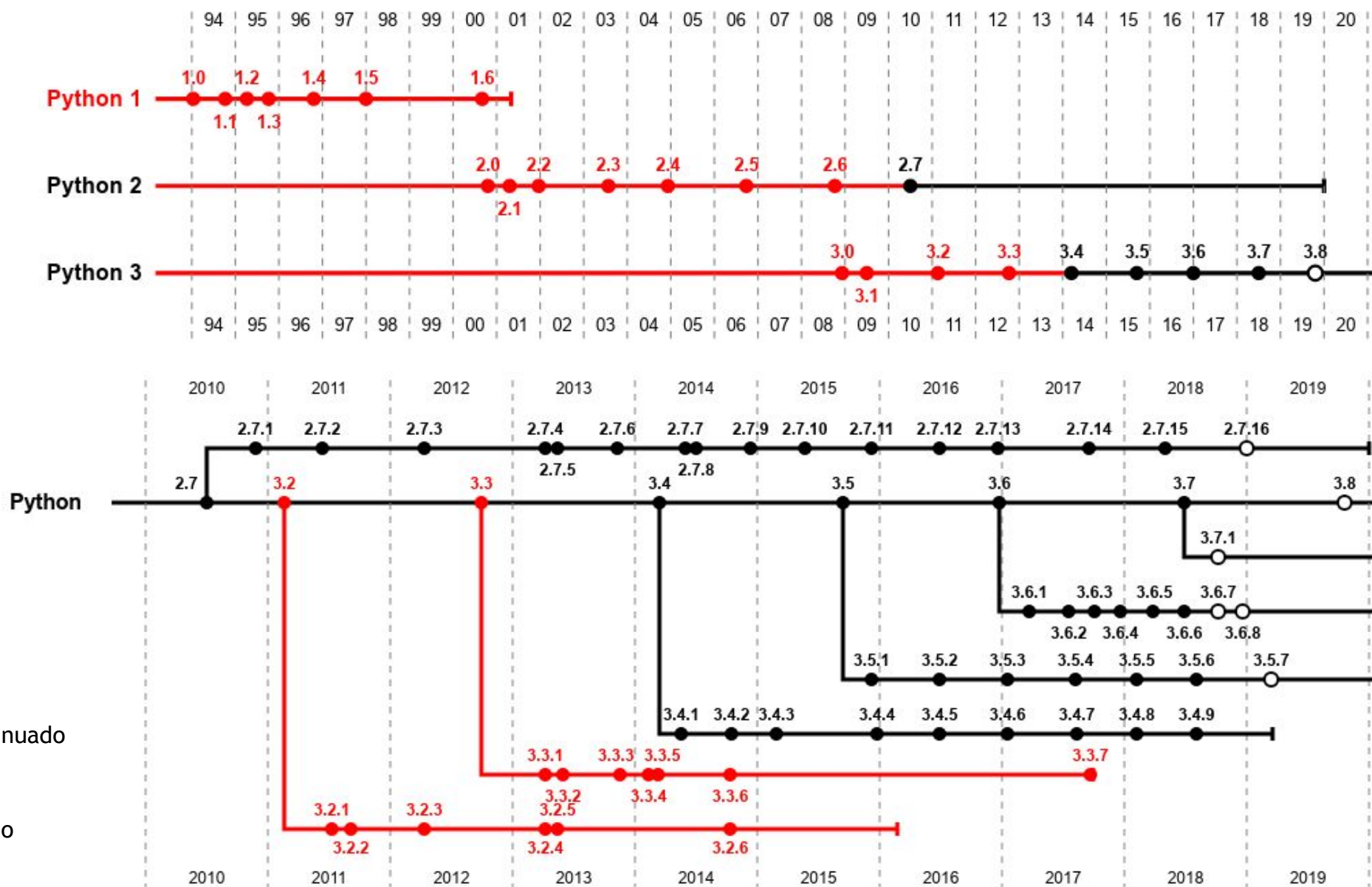
- En la actualidad, el desarrollo y difusión de Python son tareas de la **Python Software Foundation**. Sin embargo, van Rossum continúa ejerciendo un rol central decidiendo la dirección que debe tomar el lenguaje. En la comunidad de Python se le conoce como “Benevolente Dictador Vitalicio”.
- Python es **software libre** licenciado bajo la *Python Software Foundation License*. Esta es una licencia al estilo BSD, más permisiva que la GPL, ya que no requiere que las modificaciones realizadas al código fuente, ni los trabajos derivados, deban ser distribuidos como código abierto.
- Los usuarios de Python se refieren a menudo a la **Filosofía Python** que es bastante análoga a la filosofía de Unix. El código que sigue los principios de Python de **legibilidad, simpleza, transparencia y practicidad** se dice que es *"pythonico"*

Versiones (I)

Las versiones de Python se identifican por tres números **X.Y.Z**:

- **X** corresponde a las grandes versiones de Python (1, 2 y 3), siendo incompatibles entre sí. Actualmente se mantienen en desarrollo dos versiones de Python, la **2** y la **3**
- **Y** corresponde a versiones importantes en las que se introducen novedades en el lenguaje pero manteniendo la compatibilidad (salvo excepciones). Suelen publicarse cada año y medio y se mantienen durante cinco años, excepto la versión **2.7** (última que se liberó de la versión 2) con soporte durante 10 años (**finalizó** el 1 de enero de 2020)
- **Z** se corresponde con revisiones menores que son liberadas durante el periodo de mantenimiento. Corrigen errores y fallos de seguridad.
- Las últimas versiones publicadas de las ramas 2 y 3 son la **2.7.18** (abril de 2020) y la **3.9.0** (octubre de 2020)

Versiones (II)



- Discontinuado
- Activo
- Planeado

Versiónes (y III)

Version ↕	Latest micro version ↕	Release date ↕	End of full support ↕	End of security fixes ↕
0.9	0.9.9 ^[2]	1991-02-20 ^[2]	1993-07-29 ^{[a][2]}	
1.0	1.0.4 ^[2]	1994-01-26 ^[2]	1994-02-15 ^{[a][2]}	
1.1	1.1.1 ^[2]	1994-10-11 ^[2]	1994-11-10 ^{[a][2]}	
1.2		1995-04-13 ^[2]	Unsupported	
1.3		1995-10-13 ^[2]	Unsupported	
1.4		1996-10-25 ^[2]	Unsupported	
1.5	1.5.2 ^[39]	1998-01-03 ^[2]	1999-04-13 ^{[a][2]}	
1.6	1.6.1 ^[39]	2000-09-05 ^[40]	2000-09 ^{[a][39]}	
2.0	2.0.1 ^[41]	2000-10-16 ^[42]	2001-06-22 ^{[a][41]}	
2.1	2.1.3 ^[41]	2001-04-15 ^[43]	2002-04-09 ^{[a][41]}	
2.2	2.2.3 ^[41]	2001-12-21 ^[44]	2003-05-30 ^{[a][41]}	
2.3	2.3.7 ^[41]	2003-06-29 ^[45]	2008-03-11 ^{[a][41]}	
2.4	2.4.6 ^[41]	2004-11-30 ^[46]	2008-12-19 ^{[a][41]}	
2.5	2.5.6 ^[41]	2006-09-19 ^[47]	2011-05-26 ^{[a][41]}	
2.6	2.6.9 ^[26]	2008-10-01 ^[26]	2010-08-24 ^{[b][26]}	2013-10-29 ^[26]
2.7	2.7.18 ^[31]	2010-07-03 ^[31]	2020-01-01 ^{[c][31]}	
3.0	3.0.1 ^[41]	2008-12-03 ^[26]	2009-02-13 ^[48]	
3.1	3.1.5 ^[49]	2009-06-27 ^[49]	2011-06-12 ^[50]	2012-06 ^[49]
3.2	3.2.6 ^[51]	2011-02-20 ^[51]	2013-05-13 ^{[b][51]}	2016-02-20 ^[51]
3.3	3.3.7 ^[52]	2012-09-29 ^[52]	2014-03-08 ^{[b][52]}	2017-09-29 ^[52]
3.4	3.4.10 ^[53]	2014-03-16 ^[53]	2017-08-09 ^[54]	2019-03-18 ^{[a][53]}
3.5	3.5.10 ^[55]	2015-09-13 ^[55]	2017-08-08 ^[56]	2020-09-30 ^[55]
3.6	3.6.12 ^[57]	2016-12-23 ^[57]	2018-12-24 ^{[b][57]}	2021-12 ^[57]
3.7	3.7.9 ^[58]	2018-06-27 ^[58]	2020-06-27 ^{[b][58]}	2023-06 ^[58]
3.8	3.8.6 ^[59]	2019-10-14 ^[59]	2021-04 ^[59]	2024-10 ^[59]
3.9	3.9.0 ^[60]	2020-10-05 ^[60]	2022-05 ^[61]	2025-10 ^{[60][61]}
3.10		2021-10-25 ^[62]	2023-05 ^[62]	2026-10 ^[62]
Legend: ■ Old version ■ Older version, still maintained ■ Latest version ■ Latest preview version ■ Future release				
<i>Italic is the latest micro version of currently supported versions as of 2020-10-03.</i>				

https://en.wikipedia.org/wiki/History_of_Python#Table_of_versions

Instalación (I)

Linux

- En general, en las distribuciones Linux nos encontraremos con que Python ya se encuentra instalado por defecto en el sistema, bien en su versión 2, en la 3 ó en ambas. Podemos comprobarlo mediante la ejecución de los siguientes comandos (las versiones pueden variar):
- Versión 2:

```
bowman@hal:~$ python -V  
Python 2.7.18
```

- Versión 3:

```
bowman@hal:~$ python3 -V  
Python 3.8.5
```

Instalación (II)

- En caso contrario, los paquetes necesarios para la instalación se encontrarán en el repositorio de la distribución correspondiente y podrán ser instalados con las herramientas disponibles en cada distribución para ello. Por ejemplo, en Debian (como *root*):
- Versión 2:

```
bowman@hal:~# apt-get update  
bowman@hal:~# apt-get install python idle
```

- Versión 3:

```
bowman@hal:~# apt-get update  
bowman@hal:~# apt-get install python3 idle3
```

- Notar que, en ambos casos, además de la instalación del intérprete del lenguaje (y las librerías), se ha instalado el entorno de desarrollo IDLE

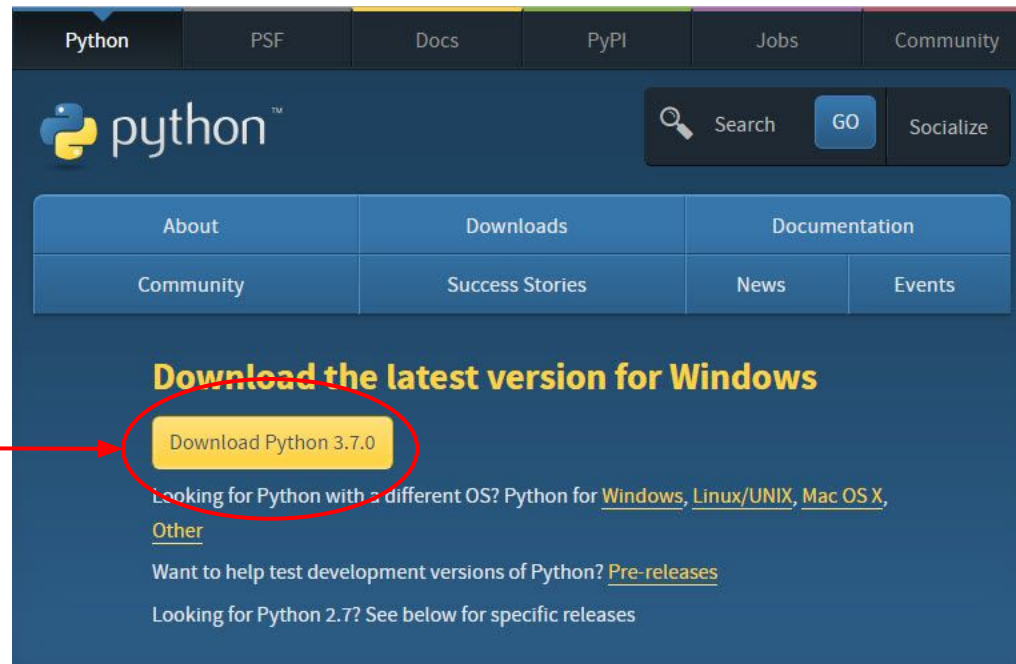
Instalación (III)

Windows

- Las siguientes imágenes muestran el proceso de instalación en Windows 7 de Python 3.7.0. El instalador incluye el intérprete y las herramientas IDLE y PIP. El espacio total requerido es de ~100MB

1) Descargar Python de la web oficial: <https://www.python.org/downloads/>

Pulsar para descargar
el instalador
python-3.7.0.exe

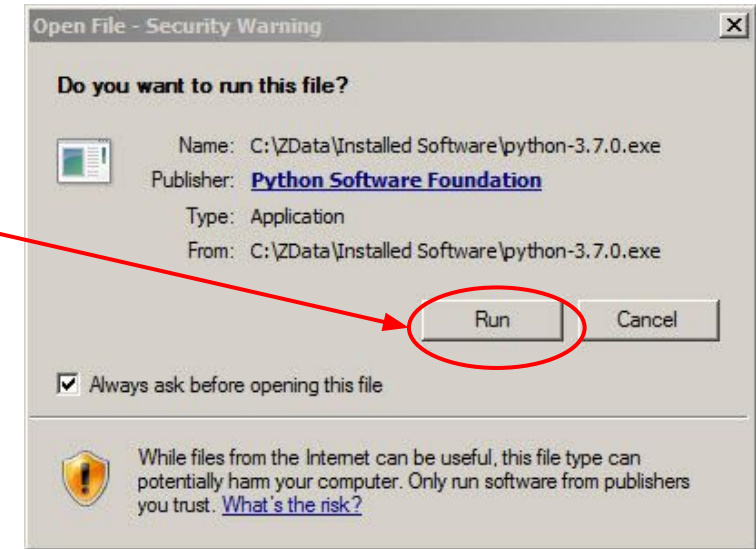
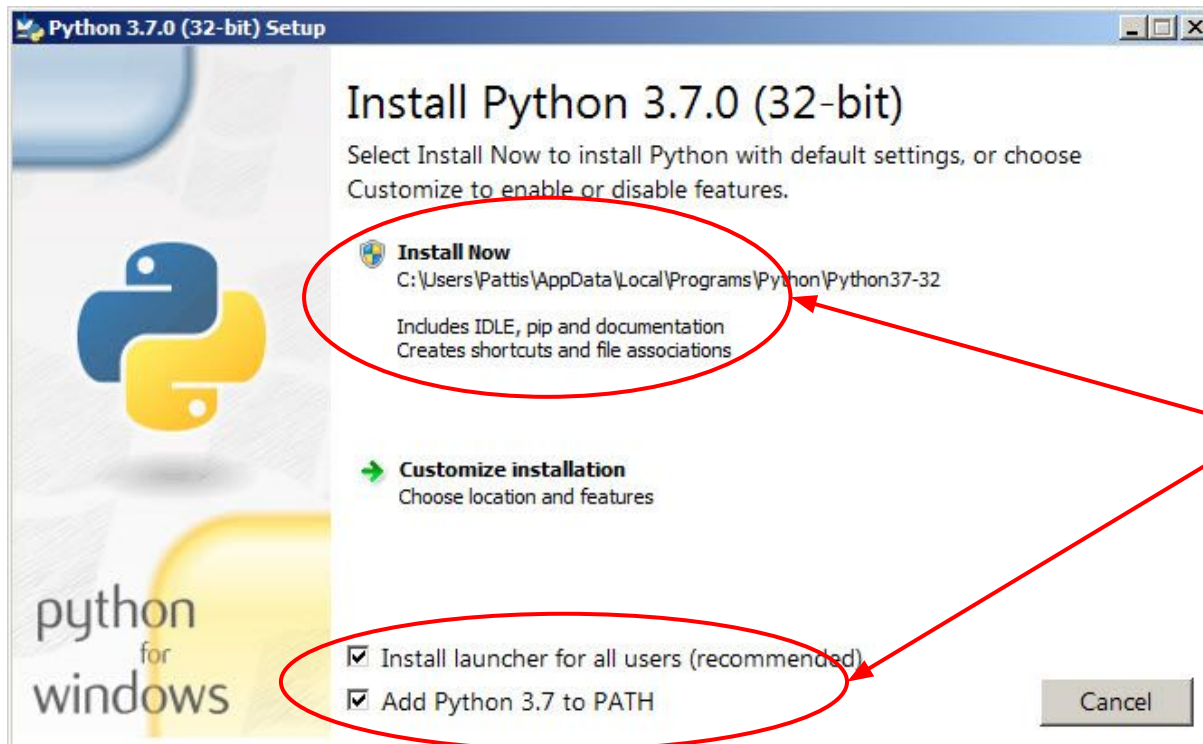


01.3 - Introducción a Python

Instalación (IV)

2) Ejecutar el fichero **python-3.7-0.exe** del instalador y aceptamos en el aviso de seguridad

3) Se abrirá la ventana de inicio de la instalación:

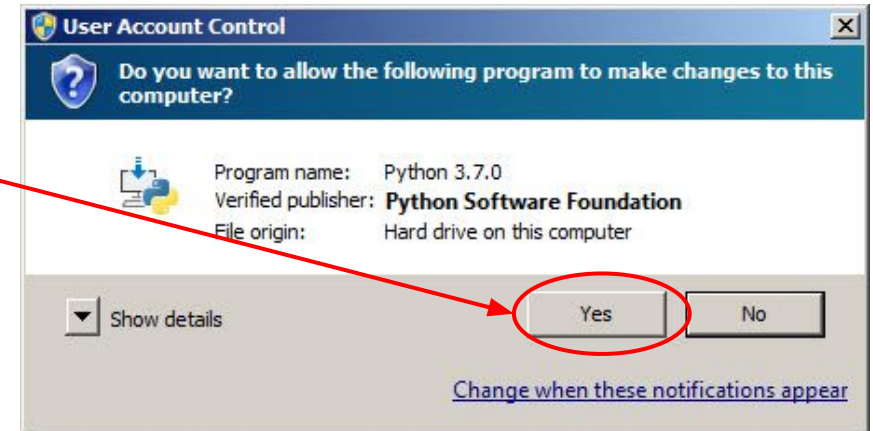


- Verificaremos la carpeta de instalación y que las casillas de “*Instalación para todos los usuarios*” y “*Añadir Python 3.7 al PATH*” estén marcadas.
- Pulsaremos sobre el mensaje “*Install Now*” para continuar

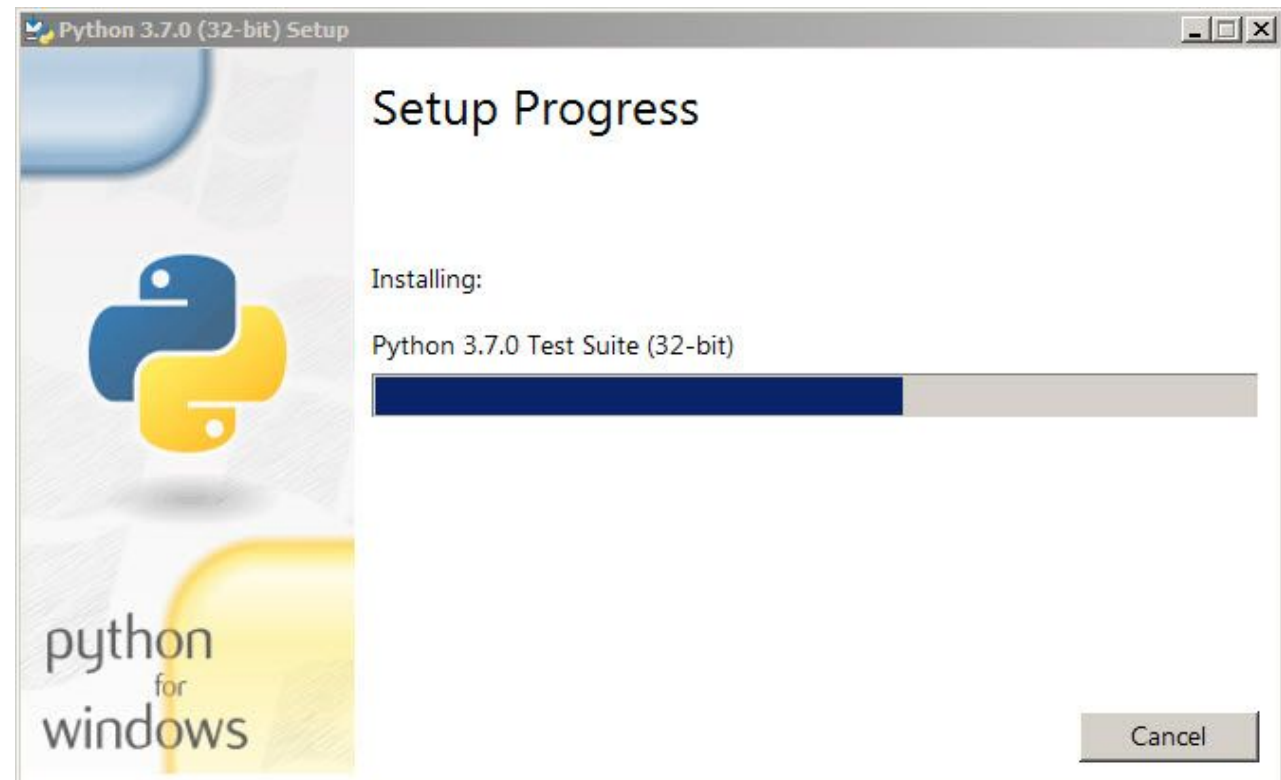
01.3 - Introducción a Python

Instalación (V)

4) Se abrirá una ventana solicitando permisos para que la nueva aplicación haga cambios en el sistema. Aceptaremos y ...

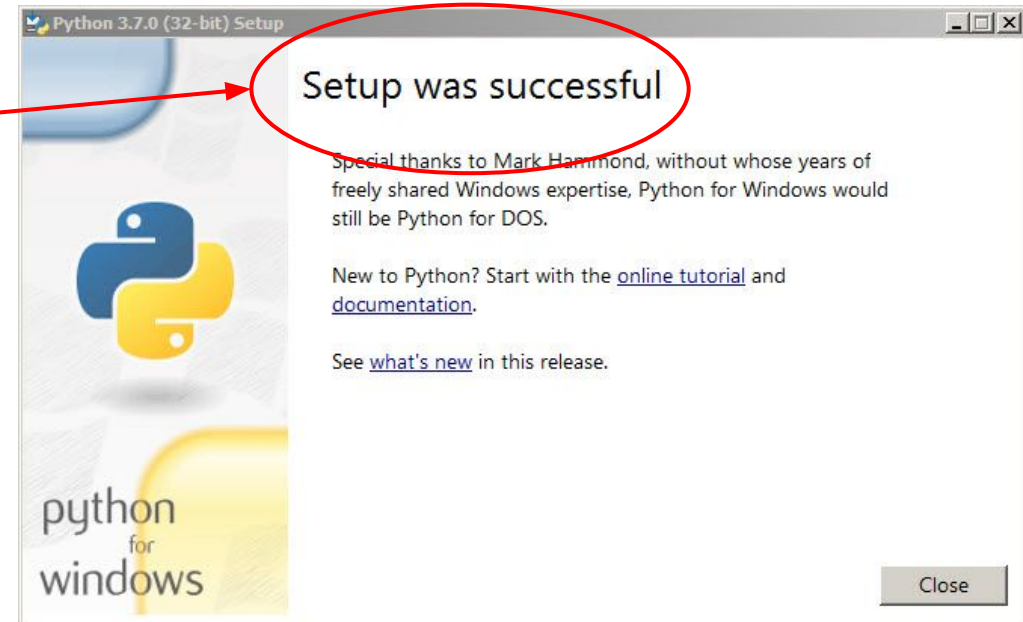


5) Se mostrará la ventana de Progreso de Instalación



Instalación (y VI)

6) Por último, se mostrará un mensaje indicado la finalización correcta de la instalación.



Verificación de la instalación

- En el menú de programas se habrá creado una entrada nueva llamada Python. Pulsaremos sobre el icono **Python 3.7 (32-bit)**, que lanzará una consola del intérprete.
- Para verificar que el **PATH** se ha actualizado correctamente, abriremos una consola de Windows y ejecutaremos el comando **python**, que debería lanzar la misma consola



La consola de Python (I)

- Al lanzar el intérprete mediante el comando correspondiente, si no le pasamos como argumento el nombre de un archivo fuente para que lo ejecute, nos mostrará la **consola interactiva** del propio intérprete.

```
bowman@hal:~$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- En la primera línea de cabecera se indica la versión de Python que está ejecutando (**3.8.5**). Tras el **prompt** (**>>>**), podremos escribir comandos de Python que se ejecutarán al pulsar **Intro**. Si el comando genera algún tipo de resultado, este se mostrará a continuación. Una vez finalizada la ejecución del comando, se mostrará nuevamente el prompt a la espera de una nueva orden.

La consola de Python (II)

- En la consola de Python podremos escribir todo tipo de expresiones algebraicas que serán ejecutadas por el intérprete:

```
>>> 3 + 2
5
>>> 3/2
```

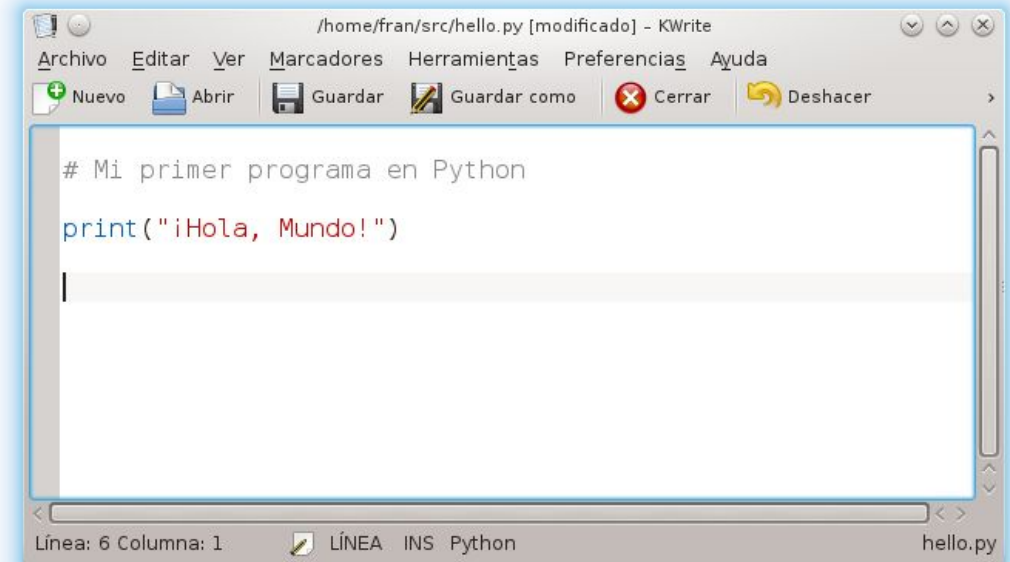
- Podemos ejecutar también instrucciones y funciones de Python:

```
>>> print("Hola, Mundo Python!")
¡Hola, Mundo!
>>>
```

En este ejemplo se hace uso de una función **print** de Python para mostrar el mensaje que se pasa como argumento. El texto del mensaje puede ir encerrado con **entrecorillado** simple o doble.

La consola de Python (III)

- Para crear nuestros programas en Python, sólo necesitaremos un simple editor de textos. A modo de ejemplo, vamos a crear un simple “Hola Mundo”.
- Abrimos un editor y escribimos el código de la imagen. La primera línea comienza con **#** que le indica la intérprete que el texto que va a continuación es un **comentario** para documentar el programa y no debe ser ejecutado. La siguiente línea es una llamada a la función **print** que imprimirá el mensaje pasado como argumento



The screenshot shows a KWrite text editor window titled "/home/fran/src/hello.py [modificado] - KWrite". The menu bar includes Archivo, Editar, Ver, Marcadores, Herramientas, Preferencias, and Ayuda. The toolbar has buttons for Nuevo, Abrir, Guardar, Guardar como, Cerrar, and Deshacer. The text area contains the following code:

```
# Mi primer programa en Python
print("¡Hola, Mundo!")
|
```

The status bar at the bottom indicates "Línea: 6 Columna: 1" and "LÍNEA INS Python". The file name "hello.py" is shown in the bottom right corner.

- El archivo lo guardaremos con la extensión **.py**, para indicar que contiene código fuente Python

La consola de Python (y IV)

- Para ejecutar el código anterior desde la consola, sólo tenemos que lanzar el intérprete indicándole el nombre (y la ruta) del archivo fuente

```
bowman@hal:~$ python3 src/hello.py
¡Hola, Mundo!
```

- En Linux, podemos convertir estos archivos en *scripts ejecutables*. Para ello, añadimos la siguiente línea (*hashbang*) al **comienzo** del archivo:

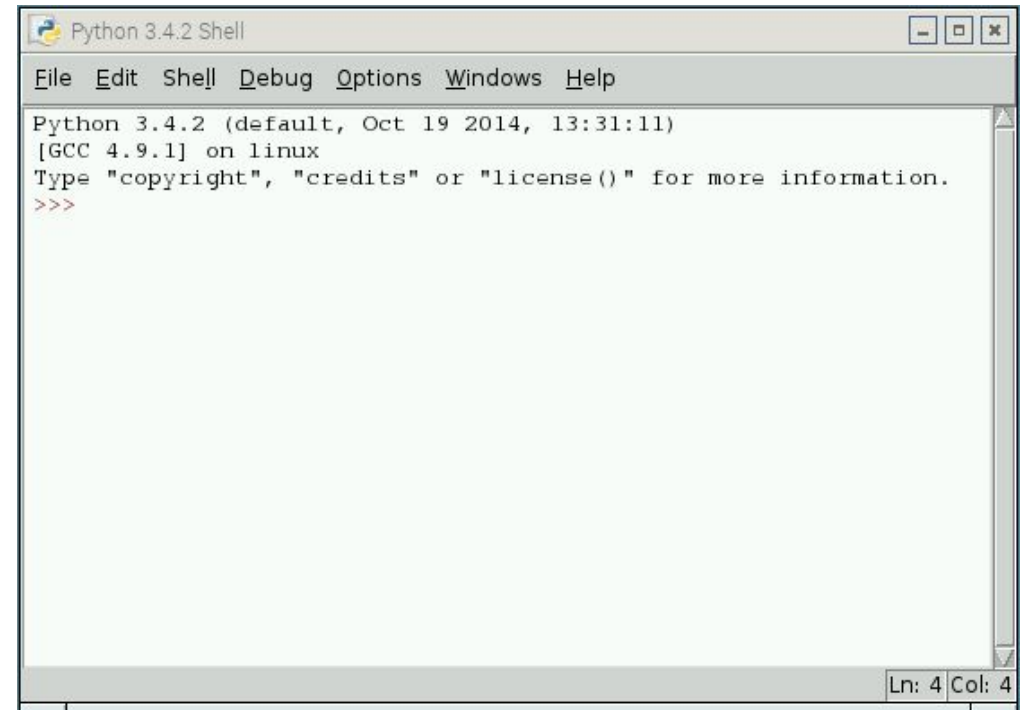
```
#!/usr/bin/env python3
```

- Este *hashbang* le indica al shell que ejecute el intérprete (python3) para procesar el contenido del mismo. Ahora podemos darle permisos de ejecución y lanzarlo como cualquier otro programa

```
bowman@hal:~$ cd src
bowman@hal:~/src$ chmod u+x hello.py
bowman@hal:~/src$ ./hello.py
¡Hola, Mundo!
```

IDLE (I)

- Las distribuciones de Python incluyen un IDE simple, denominado IDLE (o IDLE3), para facilitar la escritura, ejecución y depurado de programas en Python
- Podemos lanzar IDLE desde el menú de Python o desde la consola con el comando *idle* (*idle3* para Python 3)
- Al iniciarse nos muestra la típica consola Python, con la diferencia de que incluye resaltado de sintaxis
- Los atajos **Alt+p** y **Alt+n** nos permiten recuperar comandos que introdujimos en la consola con anterioridad.



```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

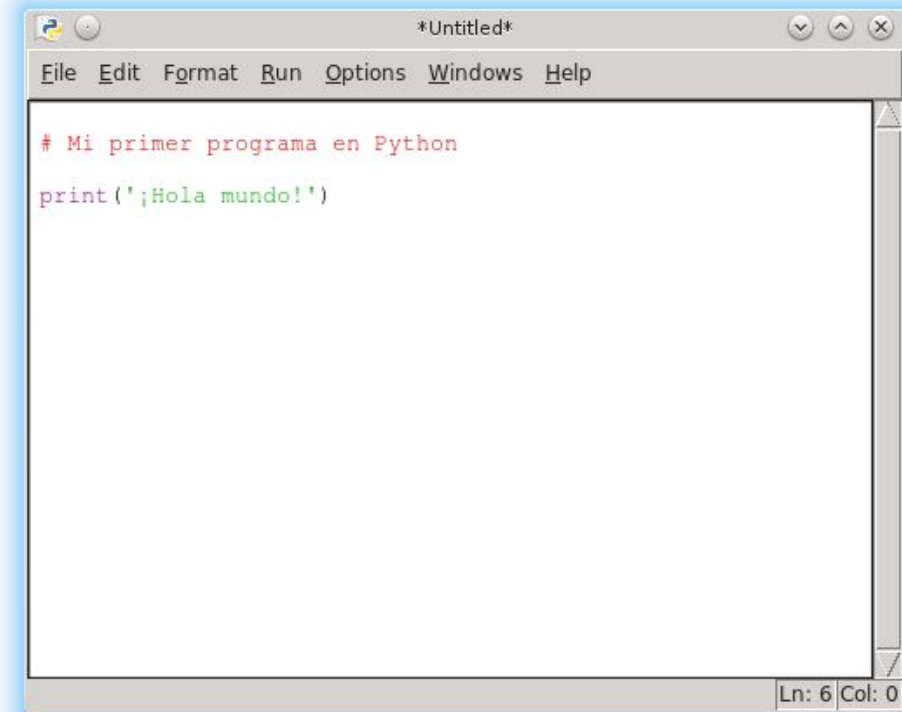
IDLE (II)

- IDLE utiliza **resaltado de sintaxis** para identificar y diferenciar diferentes elementos del lenguaje o errores:
 - Las palabras reservadas de Python (las que forman parte del lenguaje) se muestran en color **naranja**
 - Las cadenas de texto se muestran en **verde**
 - Los resultados de las órdenes se escriben en **azul**
 - Los mensajes de error se muestran en **rojo**
 - Las funciones se muestran en **púrpura**
- El siguiente ejemplo muestra un error al tratar de ejecutar una función (print) sin encerrar sus argumentos (en este caso, el mensaje a mostrar) entre paréntesis:

```
>>> print ";Hola, Mundo!"  
SyntaxError: invalid syntax  
>>>
```

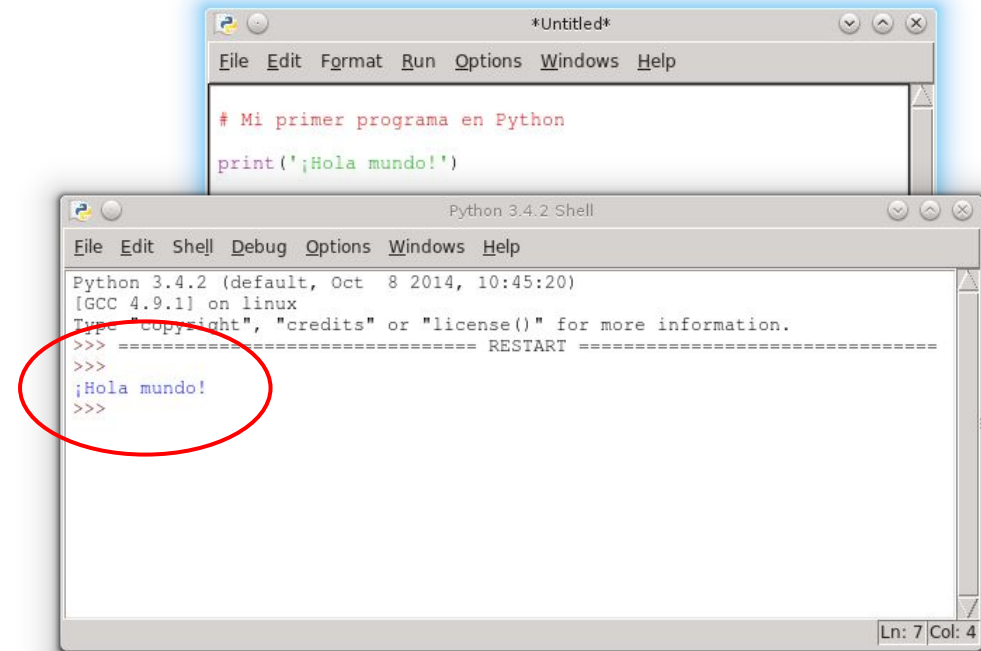
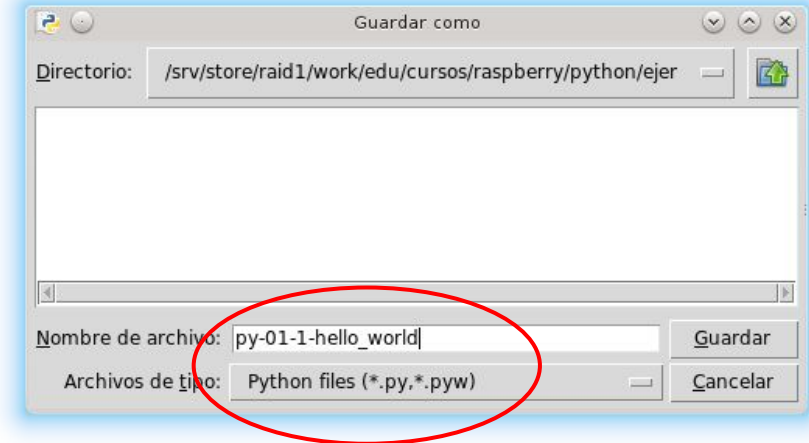
IDLE (III)

- Si bien la consola puede ser útil para realizar pequeñas pruebas, a la hora de programar aplicaciones desearemos que nuestro código se guarde en un archivo que podamos editar posteriormente. Desde el menú *File > New File* (ctrl+n) podremos lanzar una nueva ventana del **editor** de IDLE para crear un nuevo archivo de Python.
- Al escribir nuestro código en la ventana del editor podemos observar que, al igual que pasaba en la consola, se utiliza resaltado de sintaxis



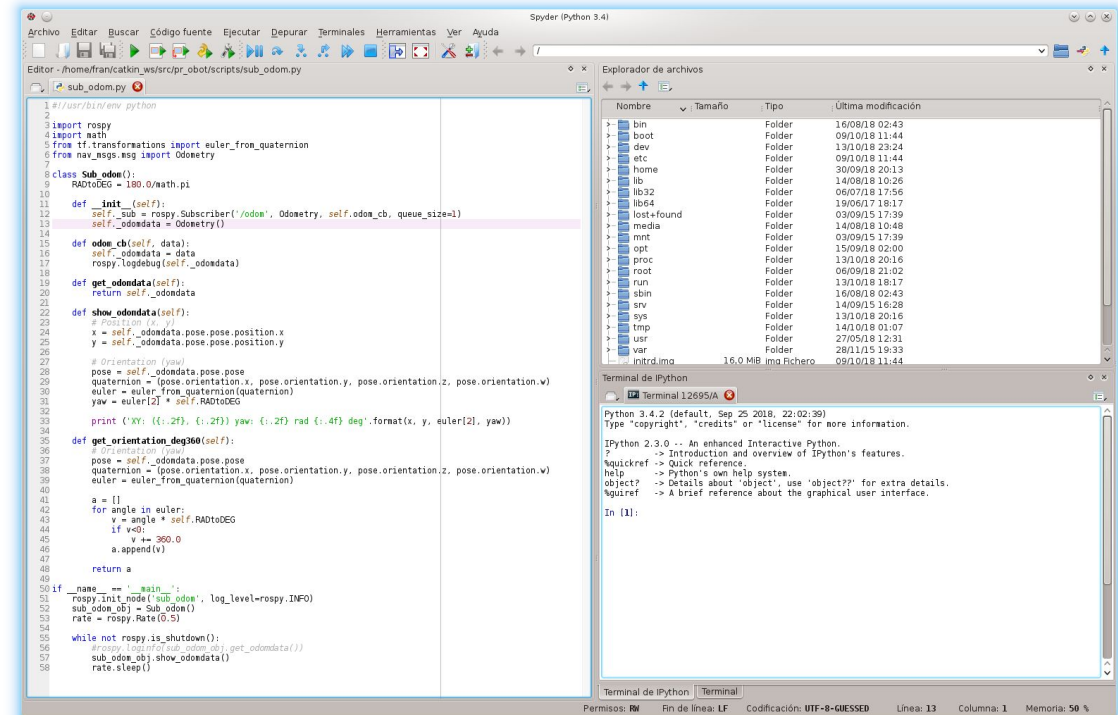
IDLE (y IV)

- Para poder ejecutar nuestro programa, antes deberemos guardarlo. Usaremos la opción *File > Save* (ctrl+S) ó *File > Save As* (ctrl+shift+S) del menú.
- Es importante que el archivo tenga la extensión **.py** para que el editor active el resaltado de sintaxis al abrirlo.
- Para ejecutarlo usaremos la opción *Run > Run Module* (F5)
- En la consola de la ventana principal de IDLE se nos mostrará la salida del programa, en este caso, el mensaje que pasamos a la función print



IDE's avanzados (I)

- A medida que nuestros programas se vuelvan más complejos e incluyen más archivos y librerías, se hace necesario el empleo de un IDE más potente que permita: gestión de proyectos, múltiples pestañas de edición, diseño de interfaces gráficos, depurado avanzado, autocompletado de código,...
- Dos IDE de este tipo son:
 - **Spyder**, IDE libre (MIT license) desarrollado en Python y orientado al campo científico
 - **pyCharm**, IDE escrito en Java, dispone de versiones comercial y libre



IDE's avanzados (y II)

- En el siguiente enlace, se muestra una comparativa de diferentes IDE's:
https://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments
- **Spyder**
Home: <https://www.spyder-ide.org/>
Windows (WinPython): <https://winpython.github.io/>
Linux (Debian/Ubuntu): instalar desde repositorios
- **pyCharm**
Home: <https://www.jetbrains.com/pycharm/>
Descarga (Windows/Linux): <https://www.jetbrains.com/pycharm/download/>
Instalación: <https://www.jetbrains.com/help/pycharm/install-and-set-up-pycharm.html>