





**GeeksHubs**  
academy \_

/ Desarrollamos { talento }

# Entornos de Desarrollo Integrados



# ¿Qué es un Entorno de Desarrollo Integrado?

Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona al desarrollador todas las herramientas necesarias para el desarrollo de software.



IDE



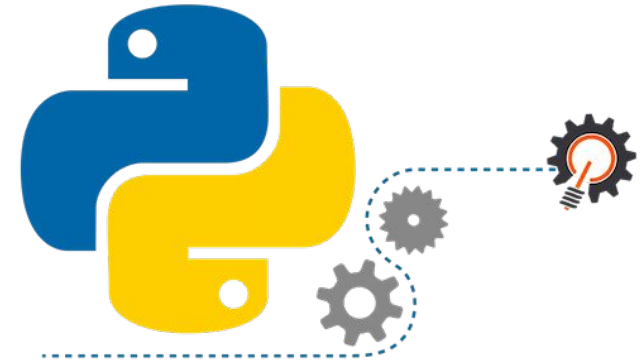
EDITOR



# ¿Qué herramientas incluye un IDE?

Los IDE suelen incluir:

1. Editor de código fuente
2. Herramientas de construcción automáticas
3. Depurador
4. IntelliSense
5. Compilador
6. Intérprete,



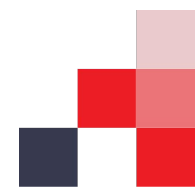
# IDEs para Python



**SPYDER**

The Scientific Python Development Environment

**WING** PYTHON  
IDE  
THE INTELLIGENT DEVELOPMENT ENVIRONMENT FOR PYTHON



# PyCharm IDE





## Community

Para un desarrollo Python puro

 **PyCharm**

IDE de Python  
para desarrolladores  
profesionales

DESCARGAR

Versión Professional completa o versión gratis Community

¿POR QUÉ  
PYCHARM?

Sugerencias

<https://www.jetbrains.com/es-es/pycharm/download>

# Versiones PyCharm

	PyCharm Professional Edition	PyCharm Community Edition
Editor de Python inteligente	✓	✓
Depurador gráfico y ejecutor de pruebas	✓	✓
Navegación y refactorización	✓	✓
Inspecciones de código	✓	✓
Compatibilidad con VCS	✓	✓
Herramientas científicas	✓	
Desarrollo web	✓	
Marcos de trabajo web Python	✓	
Perfilador Python	✓	
Capacidades para desarrollo remoto	✓	
Soporte para bases de datos y SQL	✓	

[Windows](#)[macOS](#)[Linux](#)

## Professional

Para desarrollo de Python tanto científico como de web. Compatible con HTML, JS y SQL.

[Descargar](#)[Prueba gratis](#)

## Community

Para un desarrollo Python puro

[Descargar](#)[Gratis, código abierto](#)

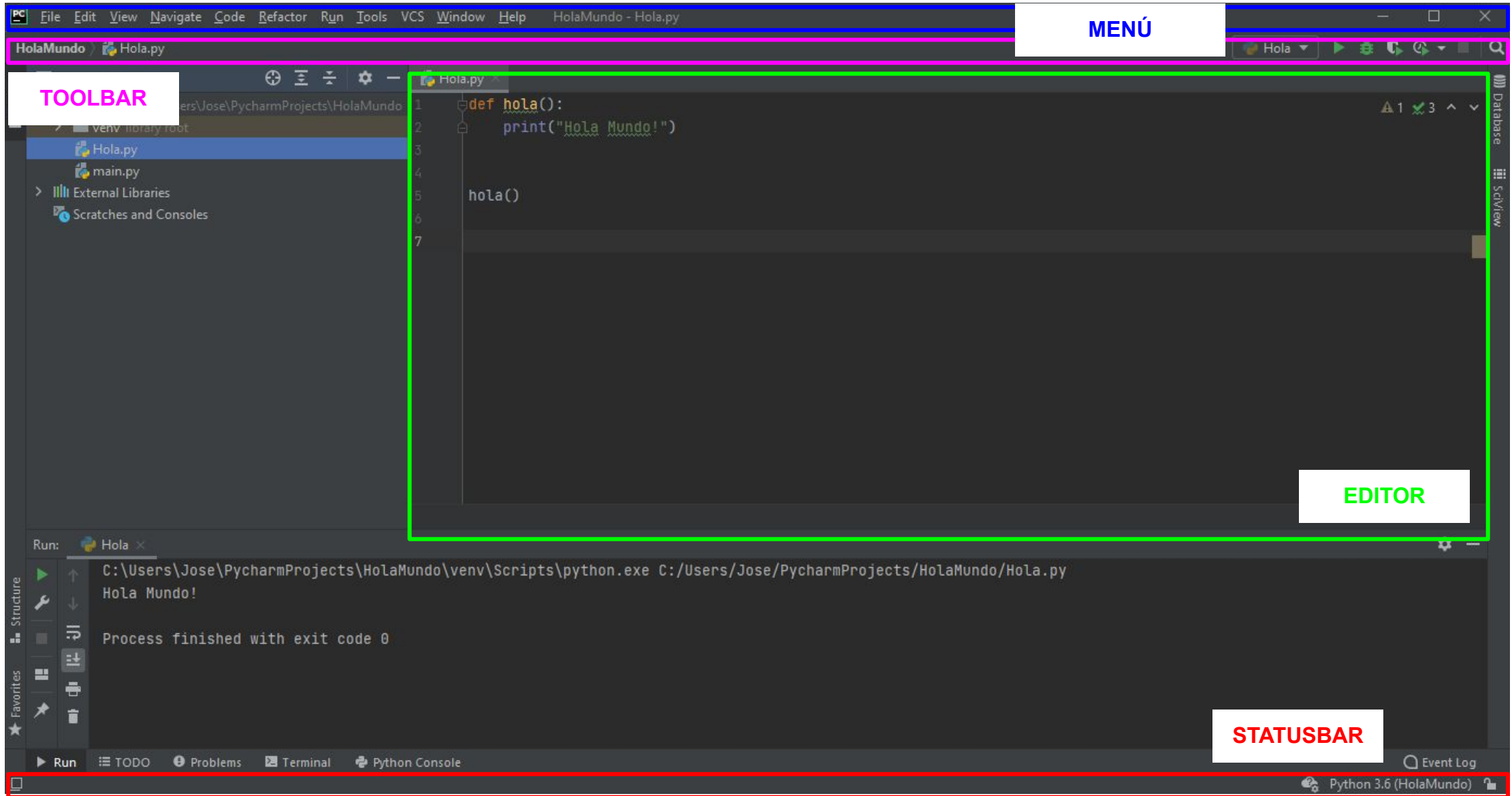


# Componentes PyCharm IDE

En casi todos los IDE, el área de trabajo está dividida en varias partes que ayudan a que trabajar con archivos y código sea más eficiente.

- **Menú:** contiene comandos como abrir archivo, abrir proyecto, crear proyecto o archivo, ejecutar, depurar código, etc.
- **Tool Bar:** es una barra de herramientas con la misma funcionalidad que la barra de menu, pero proporciona acceso rápido a estas funciones.
- **Status Bar:** en la parte inferior hay una barra de estado que muestra mensajes sobre errores, advertencias y el estado de un proyecto.
- **Editor de Código:** se sitúa en la parte central de la pantalla consiste en el editor donde lee y escribe código.



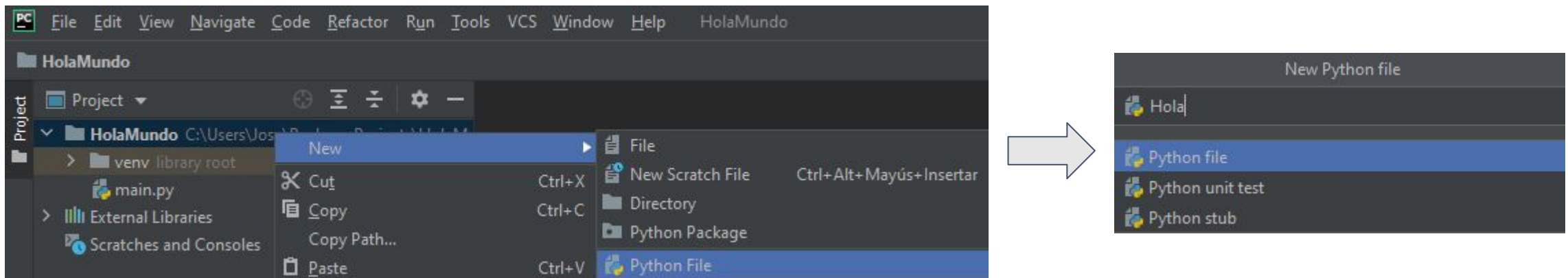


# Mi primer proyecto Python



# Cómo crear scripts en PyCharm

Creamos nuestro primer proyecto python, vamos a **File->New project**. Para crear un script de Python, haga clic con el botón derecho en el nodo del nombre del proyecto y elija en el menú emergente **New -> Python File**.



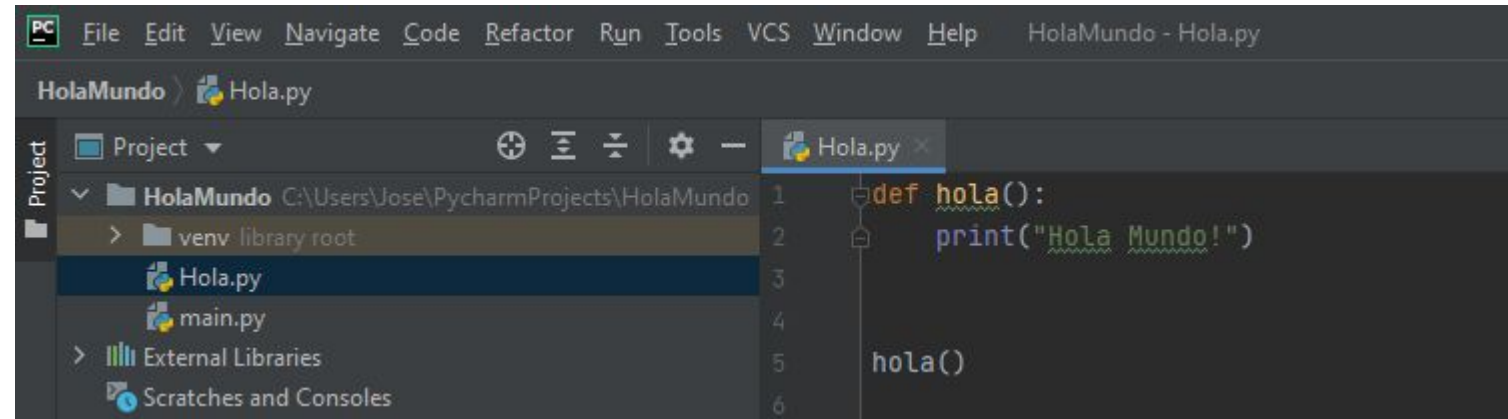
Aparece la ventana New Python File. Añadimos el nombre del archivo y le damos a **OK**.



# Cómo editar scripts en PyCharm

Una vez creado nuestro primer script abrimos el editor de PyCharm y añadimos el siguiente código:

```
def hola():  
    print("Hola Mundo!")  
  
hola()
```

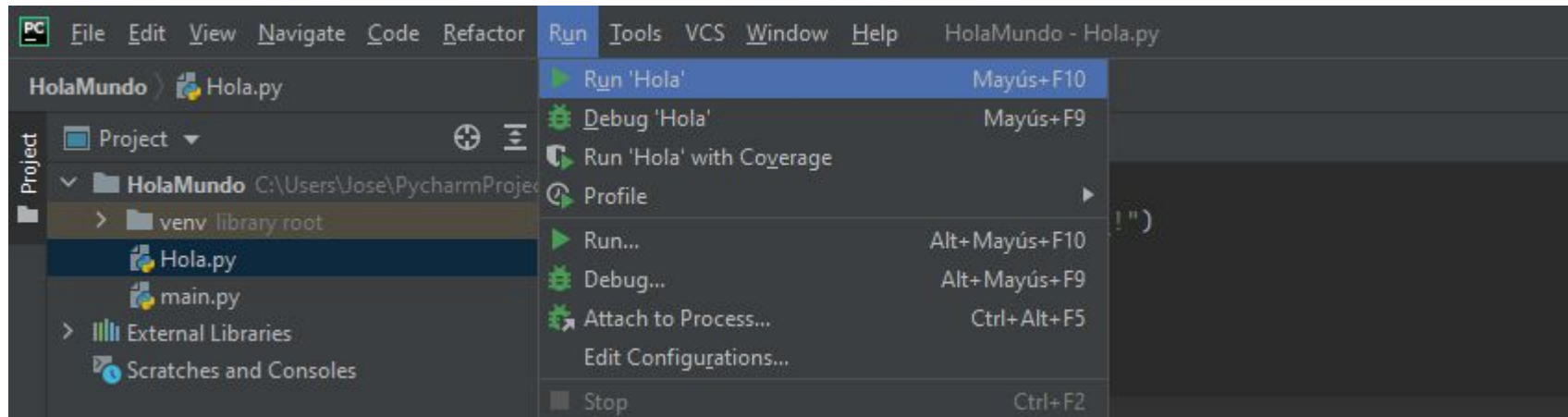


Guarda el archivo haciendo clic en la pestaña de título, luego presiona **Ctrl + S** o haciendo clic en **File -> Save All**.



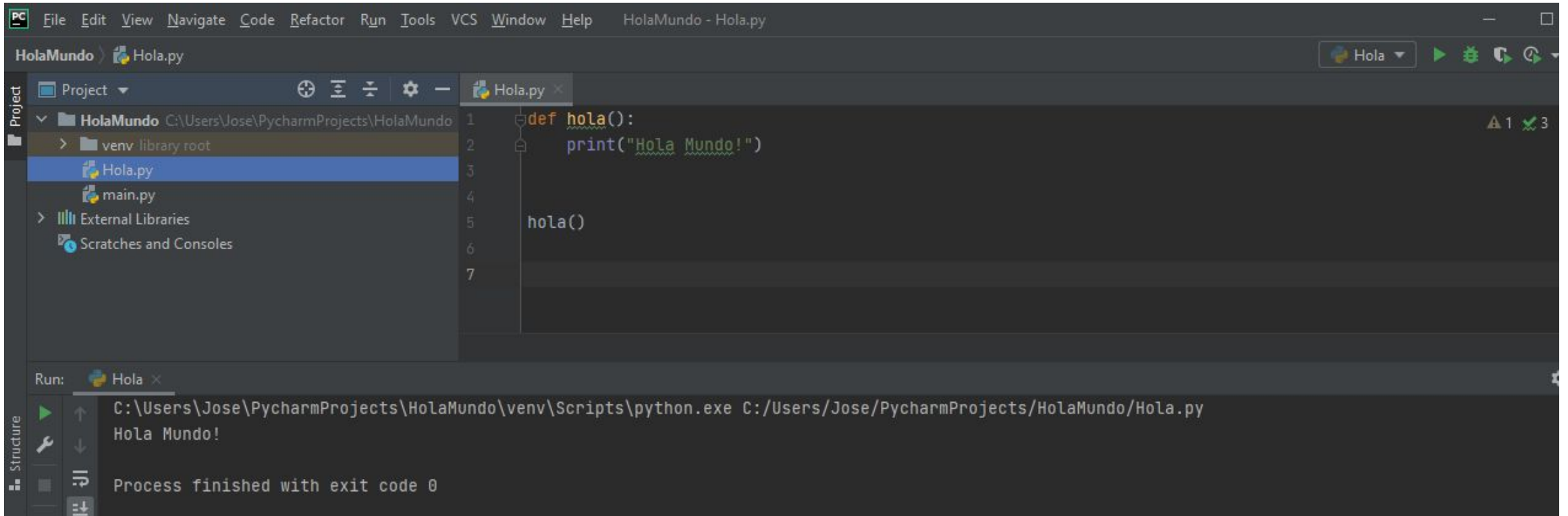
# Cómo ejecutar scripts en PyCharm

Podemos ejecutar un proyecto o un único archivo seleccionando el proyecto / nodo de archivo y eligiendo en el menú **Run -> Run (el botón verde Ejecutar)** o usando una combinación de teclado **Alt + Mayús + F10**.



# Cómo ejecutar scripts en PyCharm

En la parte inferior del editor se mostrará la ventana Run con el resultado.



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The project name is 'HolaMundo'. The left sidebar shows the Project view with the following structure:

- HolaMundo
  - venv library root
  - Hola.py (selected)
  - main.py
- External Libraries
- Scratches and Consoles

The main editor window displays the code in 'Hola.py':

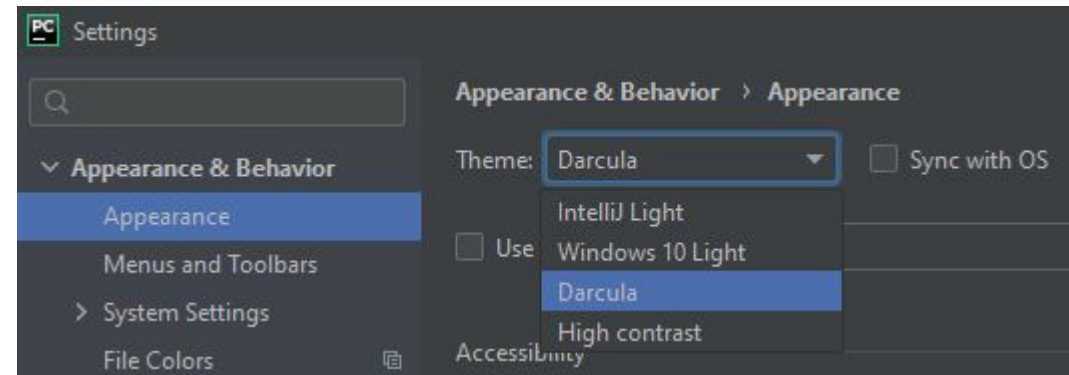
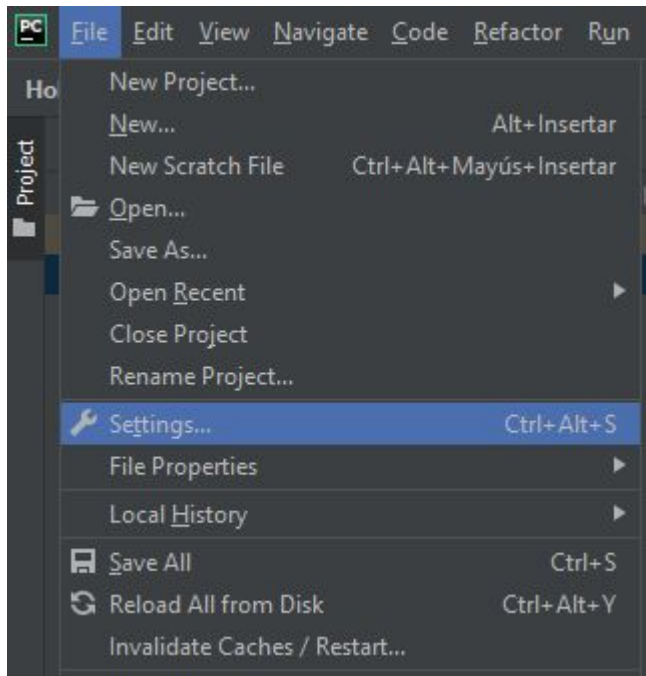
```
1 def hola():  
2     print("Hola Mundo!")  
3  
4  
5 hola()  
6  
7
```

The bottom panel shows the Run window with the following output:

```
Run: Hola x  
C:\Users\Jose\PycharmProjects\HolaMundo\venv\Scripts\python.exe C:/Users/Jose/PycharmProjects/HolaMundo/Hola.py  
Hola Mundo!  
Process finished with exit code 0
```

# Personalizar el tema y la combinación de colores de PyCharm

PyCharm IDE permite personalizar muchas opciones. Entre otras la personalización del tema del IDE. Para acceder a los temas, haz clic en **File -> Settings**



También puede cambiar los colores de fuente. Encontrarás todas las opciones disponibles en la ventana settings.





## Code Highlighting

El editor resalta símbolos y palabras clave en el código. Esta función ayuda a los desarrolladores a encontrar rápidamente palabras en el script, ayuda a detectar errores y mejora la legibilidad del código.

Al hacer clic en el símbolo o la palabra, se resaltará todas las apariciones de esta palabra o símbolo.

```
1 a = 2
2 b = 'John'
3 print(b, ' has ', a, ' cars.')
```

```
ts\firstProject 1 a = 2
2 b = 'John'
3 print(b, ' has ', a, ' cars.')
```

Unresolved reference 'prin' [more...](#) (Ctrl+F1)

PyCharm también le ayuda con los errores. Si comete un error en el código, esta palabra queda subrayada en rojo. Al pasar el mouse sobre esta palabra, se muestra la información sobre herramientas con detalles.

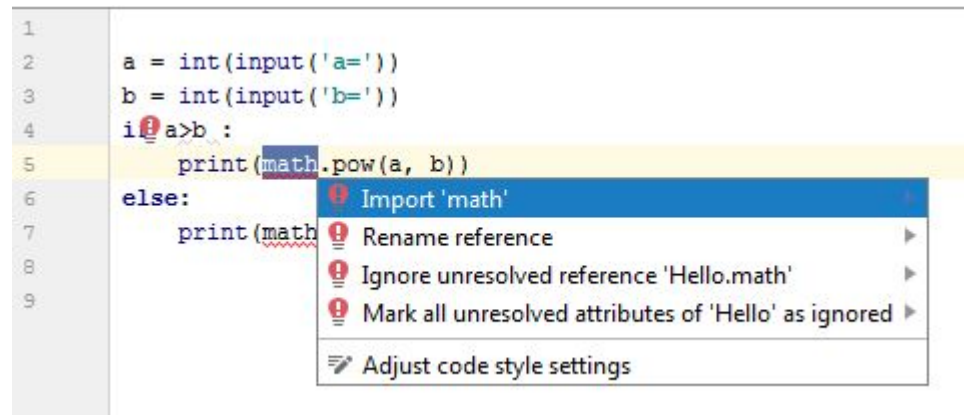


# Code Completion

Mientras escribes código, no siempre se recuerda todas las palabras clave en Python. Si se comienza a escribir una palabra, el IDE brinda sugerencias en una lista emergente, de la cual se puede elegir la palabra adecuada. Puedes invocar la finalización del código con el comando **Ctrl + Espacio**.

Por ejemplo, si escribes código sin importar un paquete obligatorio, verás esto subrayado en rojo. Para solucionar esto debes seleccionar la palabra y presionar **Alt + Enter** o hacer click en la bombilla roja.

```
1
2 a = int(input('a='))
3 b = int(input('b='))
4 if a>b :
5     print(math.pow(a, b))
6 else:
7     print(math
8
9
```

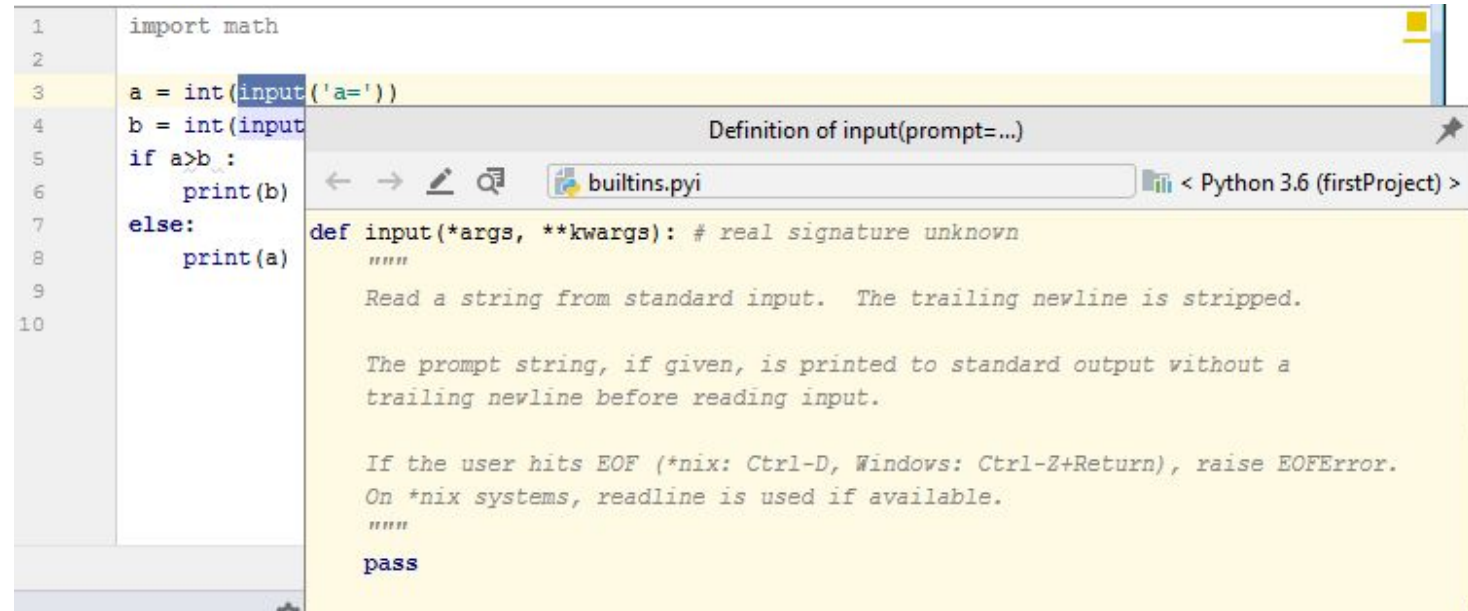


# Input Definition

La siguiente característica útil de PyCharm es la capacidad de ver la definición de una función.

Si, por ejemplo, escribes `input`, pero olvidaste cuántos y qué parámetros forman parte de esta función, puedes consultar la documentación sobre esta función.

Simplemente has de seleccionar el nombre de la función y elegir el menú **View -> Quick Definition**.



The screenshot shows a PyCharm editor window with a Python script on the left and a 'Definition of input(prompt=...)' window on the right. The script contains the following code:

```
1 import math
2
3 a = int(input('a='))
4 b = int(input('b='))
5 if a>b:
6     print(b)
7 else:
8     print(a)
9
10
```

The 'Definition of input(prompt=...)' window displays the function signature and documentation for the `input` function from the `builtins.pyi` file in the Python 3.6 (firstProject) environment. The documentation includes the following text:

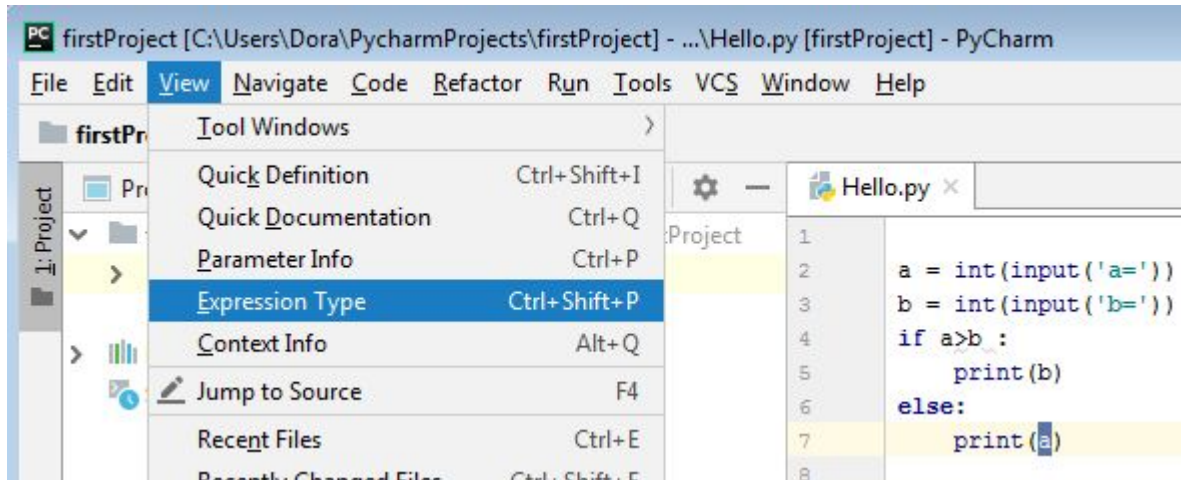
```
def input(*args, **kwargs): # real signature unknown
    """
    Read a string from standard input. The trailing newline is stripped.

    The prompt string, if given, is printed to standard output without a
    trailing newline before reading input.

    If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError.
    On *nix systems, readline is used if available.
    """
    pass
```

# Expression Type

Python no requiere declaración de tipos de datos para variables. Sin embargo, durante la escritura de muchas líneas de código, a veces es necesario averiguar qué tipo de datos se asigna a una variable. En PyCharm es fácil. Selecciona la variable y elija del menú **View -> Expression type**. También puede utilizar el método abreviado de teclado **Ctrl + Alt + P**. En una ventana emergente, verá el tipo de datos.



```
1  
2 a = int(input('a='))  
3 b = int(input('b='))  
4 if a>b :  
5     print(int)  
6 else:  
7     print(a)
```



# Code Folding

Code	Refactor	Run	Tools	VCS	Window	Help
Override Methods...						Ctrl+O
Implement Methods...						Ctrl+I
Generate...						Alt+Insert
Surround With...						Ctrl+Alt+T
Unwrap/Remove...						Ctrl+Shift+Delete
Completion						>
<b>Folding</b>						
Insert Live Template...						Ctrl+J
Surround with Live Template...						Ctrl+Alt+J
Comment with Line Comment						Ctrl+/
Comment with Block Comment						Ctrl+Shift+/
Reformat Code						Ctrl+Alt+L
Show Reformat File Dialog						Ctrl+Alt+Shift+L
Auto-Indent Lines						Ctrl+Alt+I
Optimize Imports						Ctrl+Alt+O
Rearrange Code						
Move Statement Down						Ctrl+Shift+Down
Move Statement Up						Ctrl+Shift+Up
						Expand Ctrl+NumPad +
						Collapse Ctrl+NumPad -
						Expand Recursively Ctrl+Alt+NumPad +
						Collapse Recursively Ctrl+Alt+NumPad -
						Expand All Ctrl+Shift+NumPad +
						Collapse All Ctrl+Shift+NumPad -
						Expand to level >
						Expand all to level >
						Expand doc comments
						Collapse doc comments
						<b>Fold Selection / Remove region Ctrl+.</b>
						Fold Code Block Ctrl+Shift+.

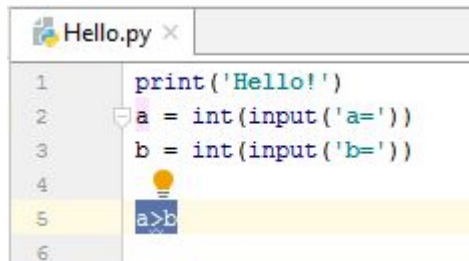
La siguiente característica útil es ocultar partes del código. Si su código es largo, es muy útil ocultar secciones de código para mejorar la legibilidad. Puede seleccionar un bloque de código para ocultar y elegir en el menú **Code -> Folding -> Fold Selection**.

El código seleccionado se plegará. Si hace clic en el símbolo más ("+"), expande el bloque de código.

# Code Constructs

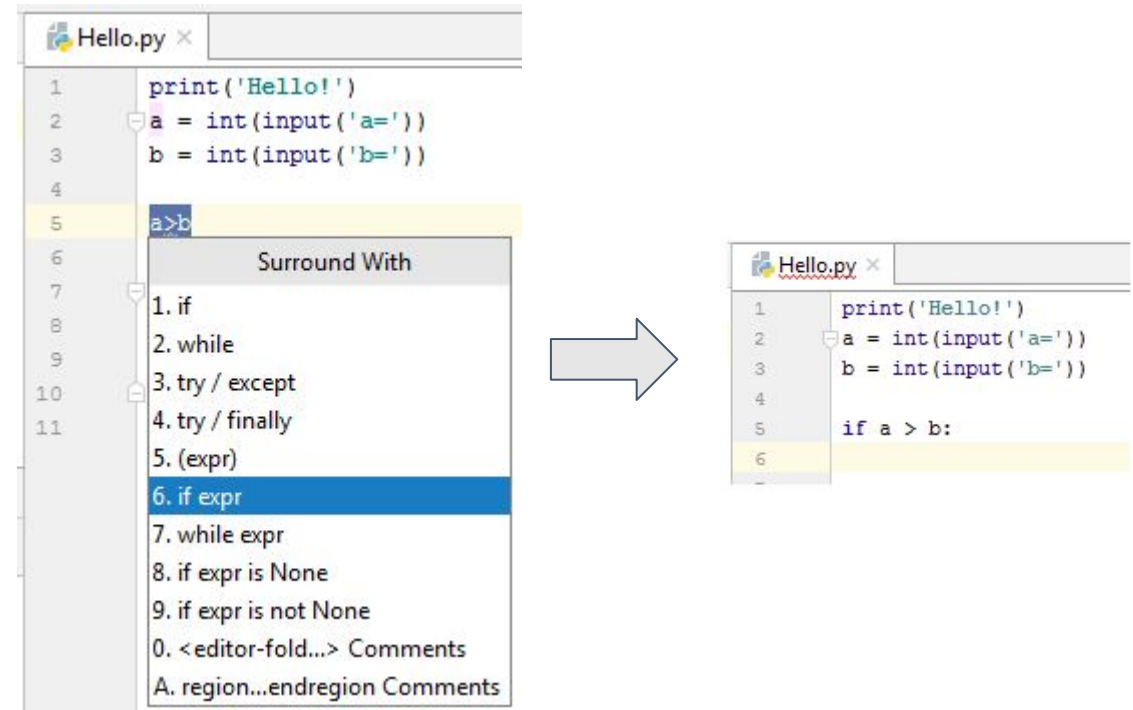
PyCharm proporciona la adición de fragmentos de código con construcciones basadas en el lenguaje del código fuente.

Por ejemplo, puedes escribir solo la condición de una instrucción `if` y seleccionarla.



```
1 print('Hello!')
2 a = int(input('a='))
3 b = int(input('b='))
4
5 a>b
6
```

Luego puedes elegir en el menú **Code -> Surround With** o presionar **Ctrl + Alt + T** y aparece la lista de declaraciones.



```
Hello.py x
1 print('Hello!')
2 a = int(input('a='))
3 b = int(input('b='))
4
5 a>b
6
```

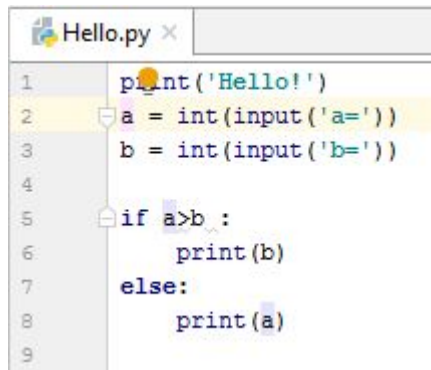
Surround With

- 1. if
- 2. while
- 3. try / except
- 4. try / finally
- 5. (expr)
- 6. if expr
- 7. while expr
- 8. if expr is None
- 9. if expr is not None
- 0. <editor-fold...> Comments
- A. region...endregion Comments

```
Hello.py x
1 print('Hello!')
2 a = int(input('a='))
3 b = int(input('b='))
4
5 if a > b:
6
```



# Code Inspector

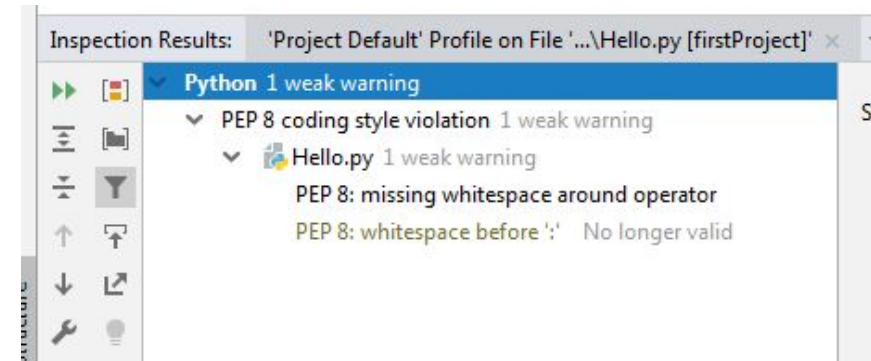


```
1 print('Hello!')
2 a = int(input('a='))
3 b = int(input('b='))
4
5 if a>b:
6     print(b)
7 else:
8     print(a)
9
```

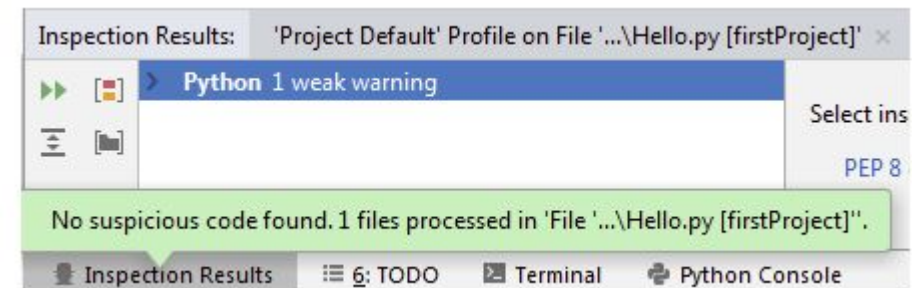
PyCharm permite el análisis de código mediante inspección. Detecta errores de compilación, código no utilizado, pérdidas de memoria y otros problemas.

De forma predeterminada, analiza todos los archivos abiertos y resalta todos los problemas detectados.

En la parte inferior del IDE, verá errores y advertencias. En nuestro script faltan espacios alrededor de los operadores y espacios en blanco antes del carácter ':':



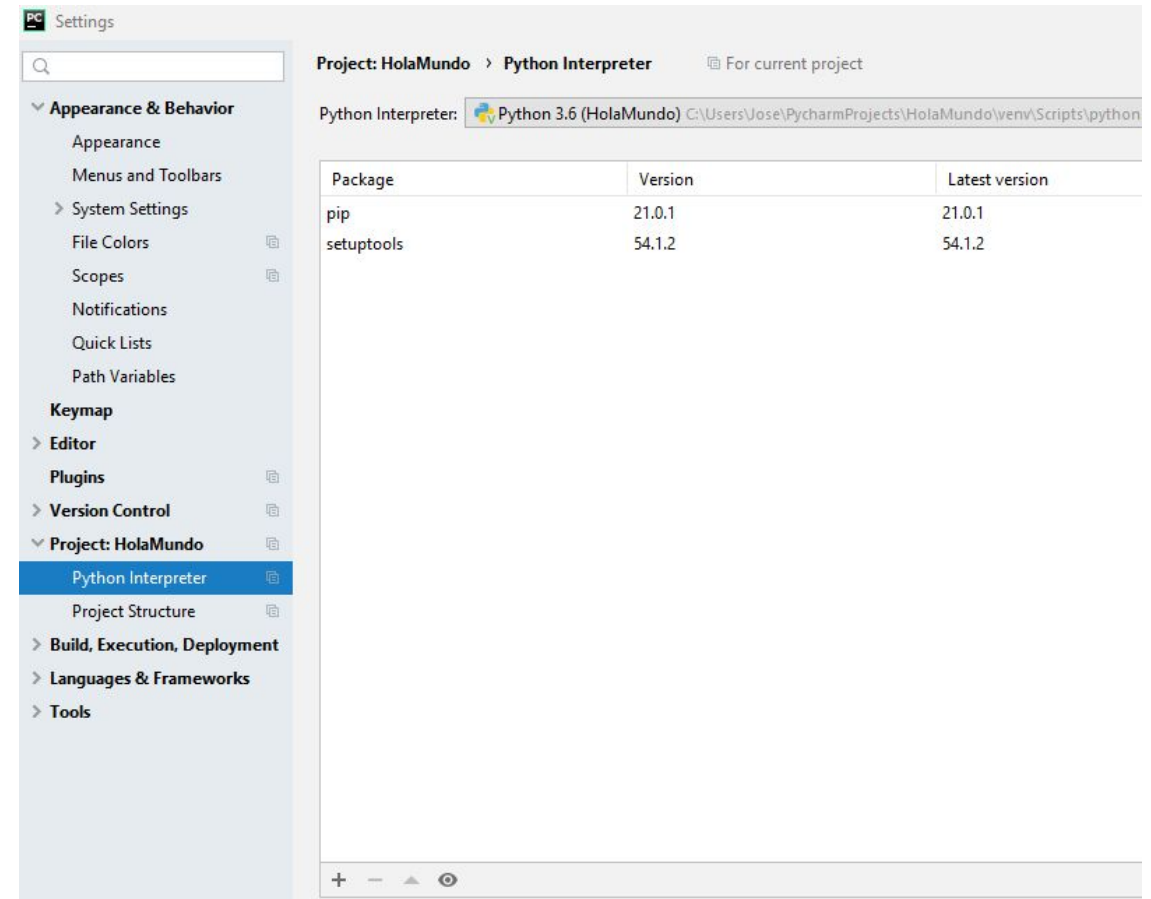
Si no hay errores o advertencias, verá un mensaje al respecto.



# Instalar paquetes en PyCharm IDE

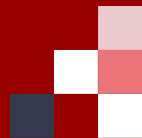
PyCharm permite instalar paquetes y módulos. Este es un proceso muy sencillo. Primero elije del menú **File -> Settings -> Project: project name -> Project Interpreter**

Aquí se muestra una lista de paquetes instalados. Para instalar un nuevo paquete, haga clic en el icono más, que se encuentra junto a los nombres de las columnas. Se muestra la ventana Paquetes disponibles. Busque el paquete y haga clic en el botón Instalar paquete en la parte inferior de esta ventana.





# Kata I



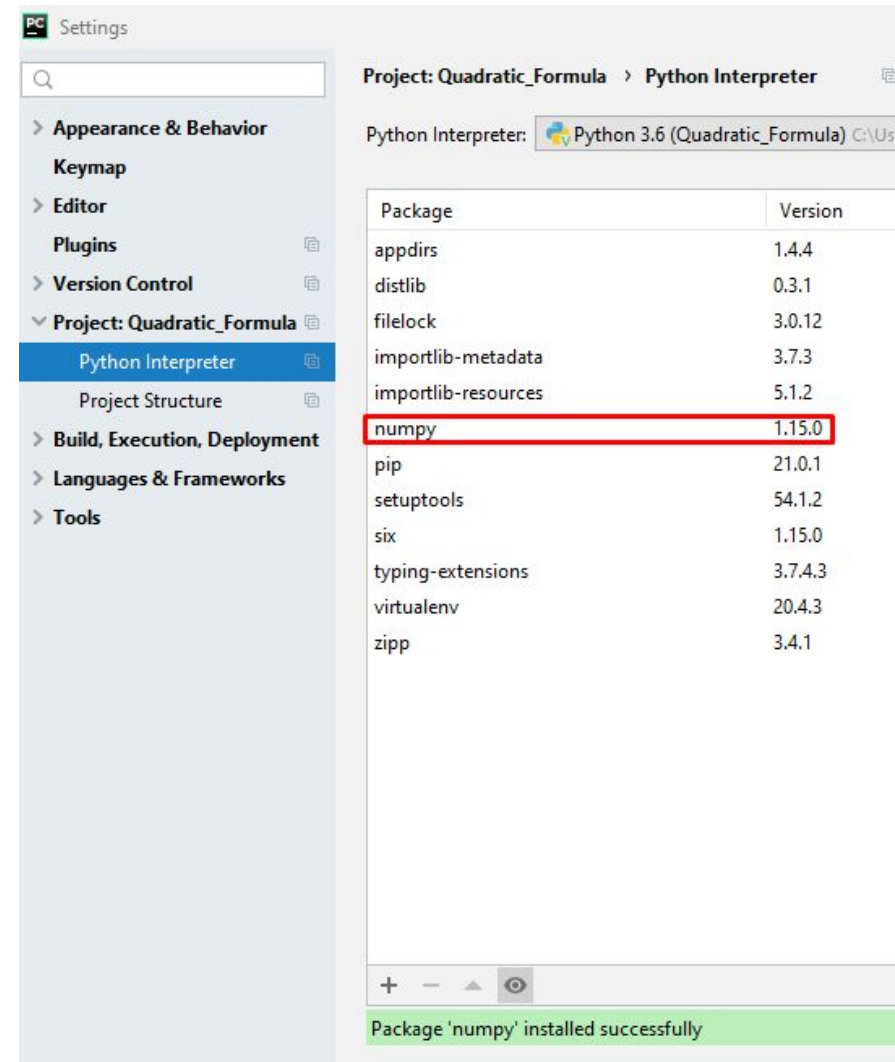
# KATA I:

NumPy es una biblioteca de Python que se utiliza para trabajar con matrices y cuenta con una gran colección de funciones matemáticas de alto nivel para operar con ellas.

Puedes encontrar más información sobre esta herramienta en el siguiente enlace:

<https://numpy.org/>

Sobre el proyecto Quadratic\_Formula vamos a instalar la librería de Numpy 1.15 configurado para trabajar con el intérprete de Python 3.6.



# PyCharm Debugger



# Debugger Python Code

Vamos a crear nuestro primer proyecto de Python.

El siguiente script calcula el valor de X en función de los valores de A, B y C por medio de la siguiente función cuadrática:

$$ax^2 + bx + c = 0$$

Aparentemente está todo OK. para asegurarnos vamos a utilizar las herramientas de debugging que vienen incluidas en PyCharm.

```
import math

class Solver:

    def demo(self, a, b, c):
        d = b ** 2 - 4 * a * c
        if d > 0:
            disc = math.sqrt(d)
            root1 = (-b + disc) / (2 * a)
            root2 = (-b - disc) / (2 * a)
            return root1, root2
        elif d == 0:
            return -b / (2 * a)
        else:
            return "This equation has no roots"

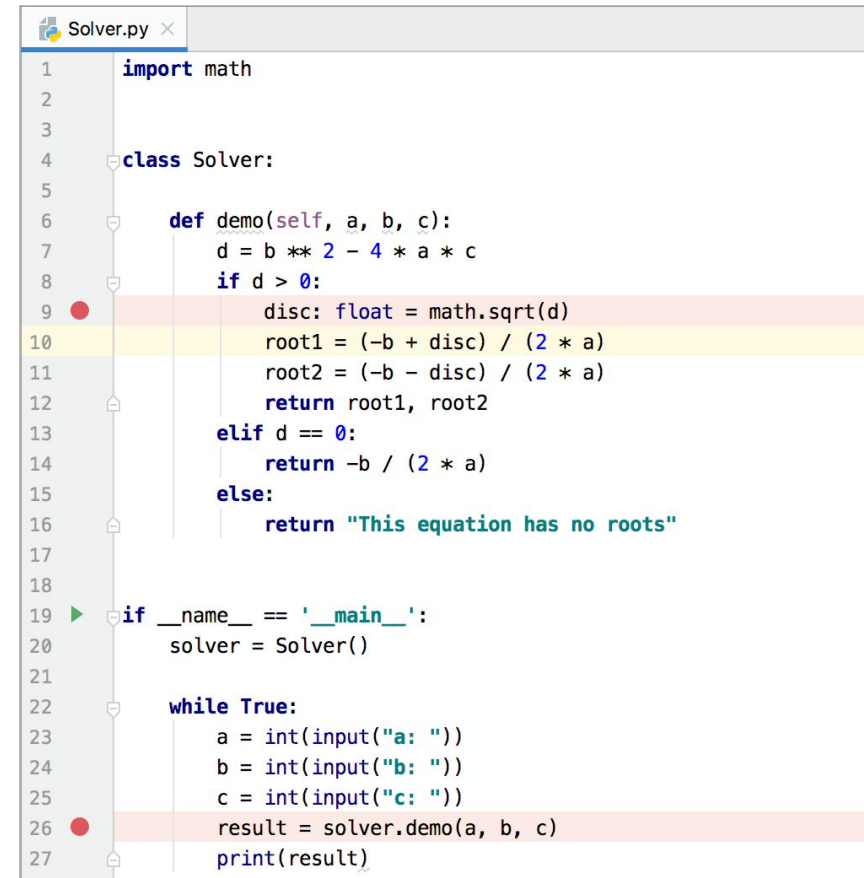
if __name__ == '__main__':
    solver = Solver()

    while True:
        a = int(input("a: "))
        b = int(input("b: "))
        c = int(input("c: "))
        result = solver.demo(a, b, c)
        print(result)
```

# Breakpoints

Para colocar puntos de interrupción, simplemente haz click en la franja gris junto a la línea en la que desea detener la ejecución.

Puedes añadir tantos como sean necesarios, recuerda que podrás inspeccionar las instrucciones anteriores a esta.

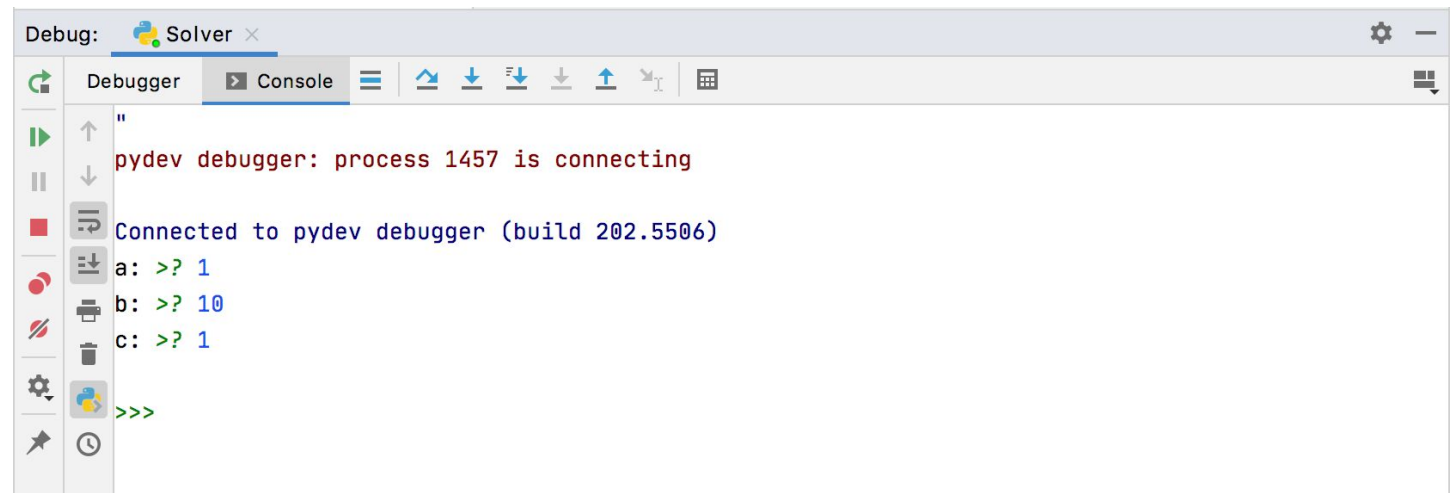
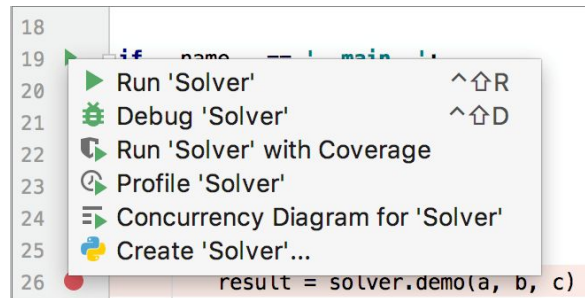


```
1 import math
2
3
4 class Solver:
5
6     def demo(self, a, b, c):
7         d = b ** 2 - 4 * a * c
8         if d > 0:
9             disc = float(math.sqrt(d))
10            root1 = (-b + disc) / (2 * a)
11            root2 = (-b - disc) / (2 * a)
12            return root1, root2
13        elif d == 0:
14            return -b / (2 * a)
15        else:
16            return "This equation has no roots"
17
18
19 if __name__ == '__main__':
20     solver = Solver()
21
22     while True:
23         a = int(input("a: "))
24         b = int(input("b: "))
25         c = int(input("c: "))
26         result = solver.demo(a, b, c)
27         print(result)
```

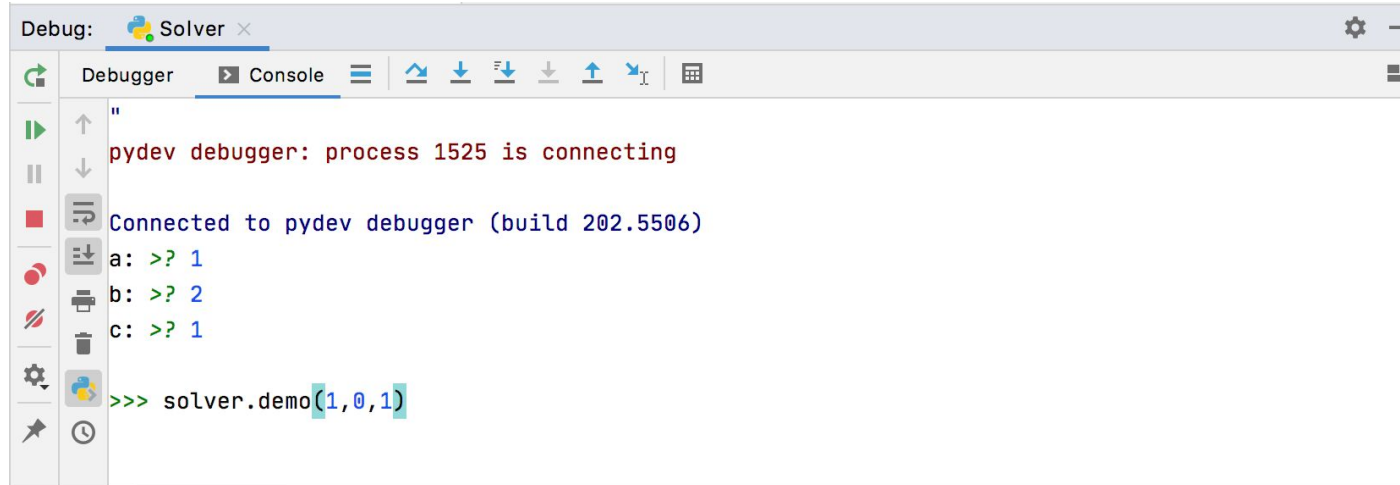
# Iniciando el inspector de código

Una vez que hemos agregado puntos de interrupción, todo está listo para depurar.

PyCharm permite iniciar la sesión del debugger de varias formas. Sobre la línea a evaluar puedes hacer click en el comando **debug 'Solver'** en el menú emergente que se abre. Una vez hecho esto el depurador se inicia, muestra la pestaña Consola de la ventana de la herramienta Depuración y le permite ingresar los valores deseados:

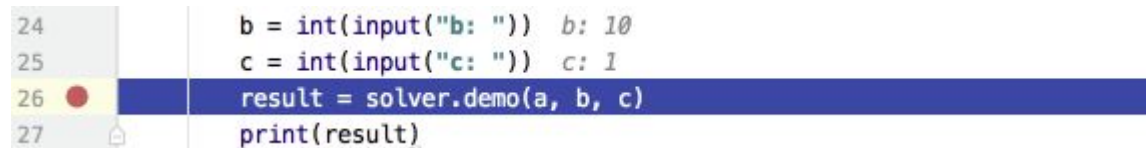


En la Consola de depuración se puede ingresar los comandos de Python:



```
Debug: Solver x
Debugger Console
pydev debugger: process 1525 is connecting
Connected to pydev debugger (build 202.5506)
a: >? 1
b: >? 2
c: >? 1
>>> solver.demo(1,0,1)
```

Seguidamente, el debugger suspende el programa en el primer punto de interrupción. esto significa que la línea con el punto de interrupción aún no se ha ejecutado. La línea se vuelve de color azul:



```
24 b = int(input("b: ")) b: 10
25 c = int(input("c: ")) c: 1
26 result = solver.demo(a, b, c)
27 print(result)
```

# Depuración en Línea

En el editor de código podemos ver un texto gris junto a las líneas de código:

```
19 ▶ if __name__ == '__main__':  
20     solver = Solver() solver: <__main__.Solver object at 0x10d0846d0>  
21
```

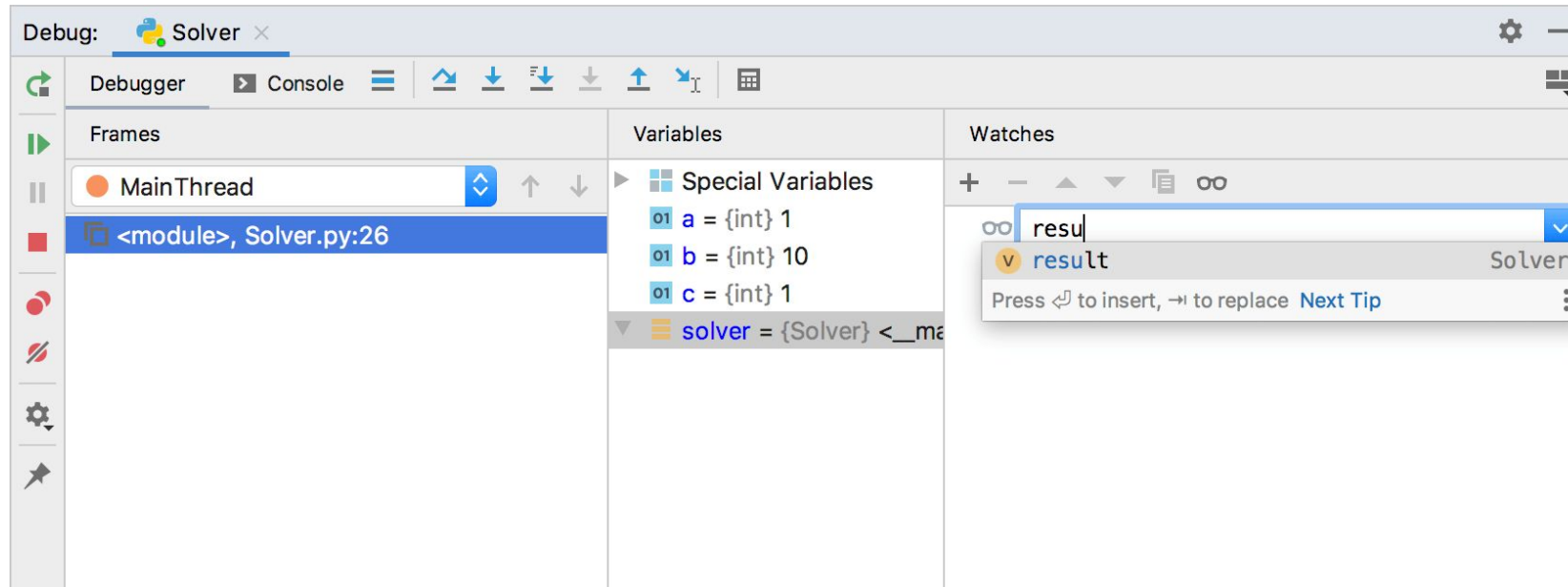
Este es el resultado de la denominada **depuración en línea**. Esta muestra la dirección del objeto **Solver** y los valores de las variables **a**, **b** y **c** que ha ingresado.

Recuerda que la depuración en línea, al igual que muchas de las utilidades de Pycharm, se puede desactivar.





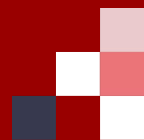
# Watches



PyCharm le permite ver una variable. Simplemente haga clic en el botón Ver en la barra de herramientas de la pestaña Variables y escriba el nombre de la variable que desea ver. Tenga en cuenta que la finalización de código está disponible.



# Reto I



# Reto I:

Sobre el siguiente fragmento de código que aparece a continuación vamos a crear un proyecto y a evaluar, por medio de las herramientas de *debuggin*, los valores de las variables A, B y C al ejecutar el método `result`.

Añade tantos breakpoints como consideres necesarios.

```
import math

class Solver:

    def demo(self, a, b, c):
        d = b ** 2 - 8 * a * c
        if d > 0:
            disc = math.sqrt(d)
            root1 = (-b + disc) / (2 * a)
            root2 = (-b - disc) / (2 * a)
            return root1, root2
        elif d == 0:
            return -b / (2 * a)
        else:
            return "This equation has no roots"

if __name__ == '__main__':
    solver = Solver()

    while True:
        a = int(input("a: "))
        b = int(input("b: "))
        c = int(input("c: "))
        result = solver.demo(a, b, c)
        print(result)
```

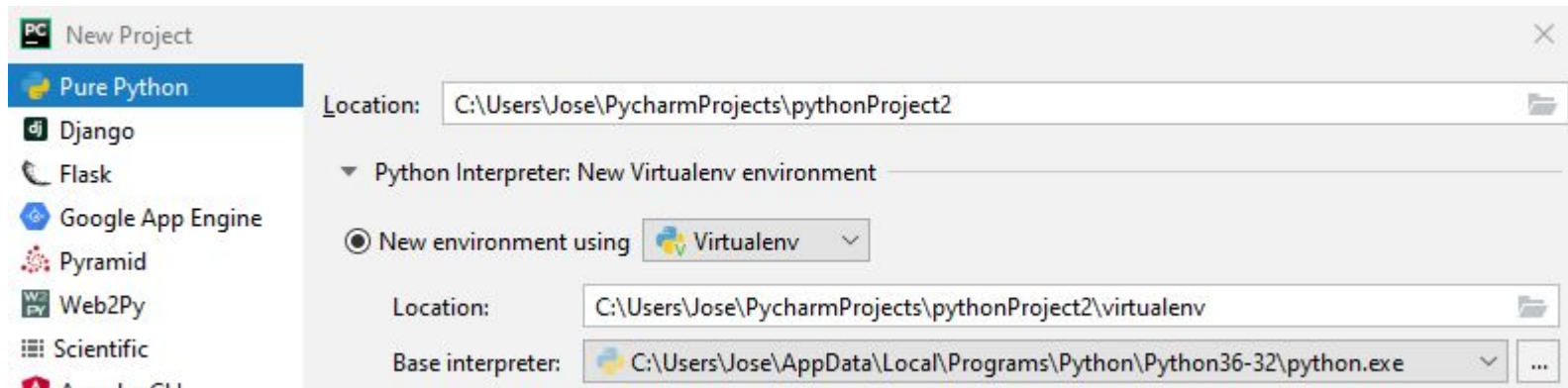


# Virtual Env



## ¿Qué es VirtualEnv?

Virtualenv es una herramienta utilizada para crear un entorno Python aislado. Este entorno tiene sus propios directorios de instalación que no comparten bibliotecas con otros entornos virtualenv o las bibliotecas instaladas globalmente en el servidor.



Es la manera más fácil recomendada para configurar un entorno personalizado de Python.



# Diferencia entre virtualenv y venv



**venv** -> es un paquete que viene con Python 3.

**virtualenv** -> es una biblioteca que ofrece más funcionalidades que venv.

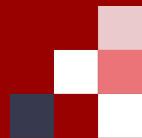
Características que venv no ofrece en comparación con virtualenv:

1. Es más lento (al no tener el app-data seed method).
2. No es tan ampliable.
3. No puede crear entornos virtuales para versiones de Python instaladas arbitrariamente.
4. No se puede actualizar a través de pip.
5. Su API no es tan completa.

Aunque puedes crear un entorno virtual usando venv con Python3, se recomienda que instales y use virtualenv en su lugar.



# Reto II

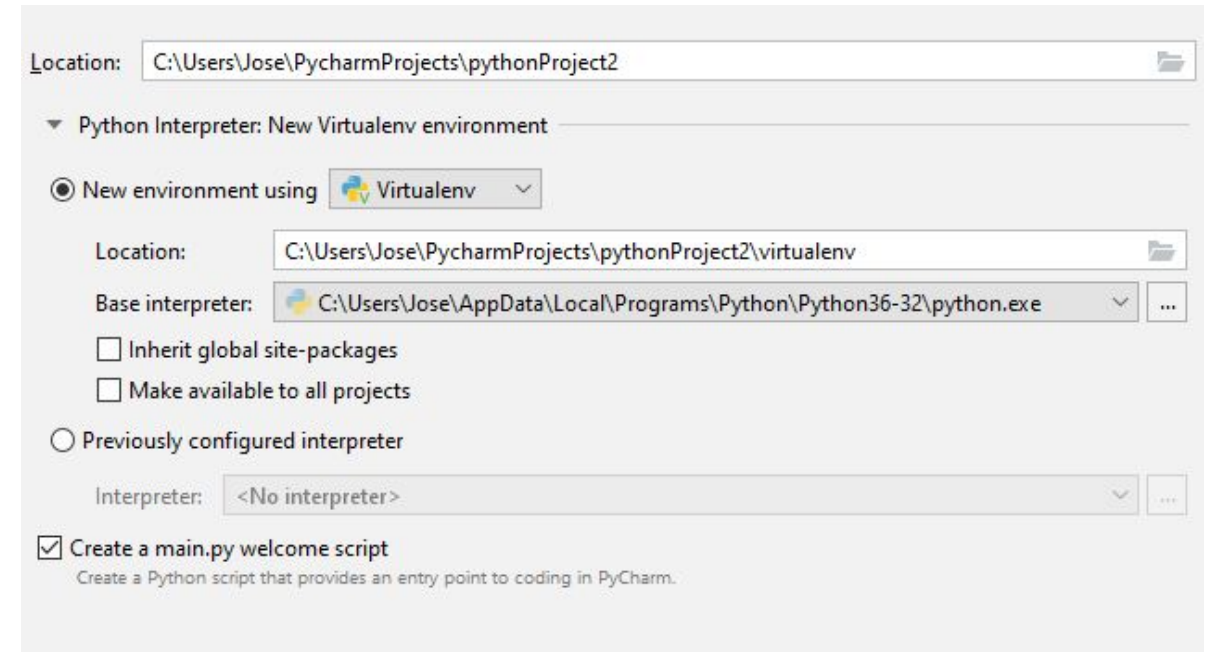


## Reto II: Configuración del virtualenv

Vamos a crear un proyecto de Python 3.6 ,titulado `Reto_venv`, sobre el que vamos a configurar su propio virtualenv.

Utilizamos para ello el asistente de PyCharm.

Una vez creado comprobaremos que se ha creado de forma correcta y aparece vinculado a nuestro proyecto.





**!Gracias por  
vuestra atención!**



# Jose Marín

[www.geekshubsacademy.com](http://www.geekshubsacademy.com)

@geekshubs

