PROGRAMMING IN PYTHON I

Editor and Debugger

Michael Widrich
Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

Institute for
Machine Learning

EDITOR AND DEBUGGER

Editor

- Comfort in programming has come a looong way
- You don't have to program in a plain text editor anymore
- Modern editors allow for:
  - Syntax highlighting
  - Auto-completion of variable names and small syntax
  - Automatic check for errors and warnings in your code
  - Automatic reformatting of your code to specific coding standards
- Many editors for Python also include a debugger

```
1  print("Hello World!")
2  a = 5
3  b = 4
4  c = a + b
5  print(c)
```

Syntax highlighting in PyCharm Editor

Debugger

- Unintended errors/behaviors in a program are referred to as bugs
- Searching for and removing these bugs is referred to as debugging
- Debuggers allow you to analyse your program while it is executed
- Modern debuggers allow for:
  - Exploring variables during runtime
  - Executing your code line by line and pausing the program at will
  - Interacting with/Modifying the code during a pause
  - Handling multiple parallel processes correctly

PYCHARM

PyCharm

- We recommend using PyCharm
  - Modern editor and debugger for Python (with support for LaTeX, shell scrips, . . .)
  - Free to use even without student licence
  - Integration of version control tools such as git (relevant for next semester)
  - We will only touch upon a small subset of its functions

[Image: PyCharm Logo, JetBrains]

Task 0: Install the Pycharm editor

- Install Pycharm Community Edition (`https://www.jetbrains.com/pycharm/download/`)
  - Installation is straight-forward
  - Community edition is free and sufficient for this lecture
  - Ubuntu:

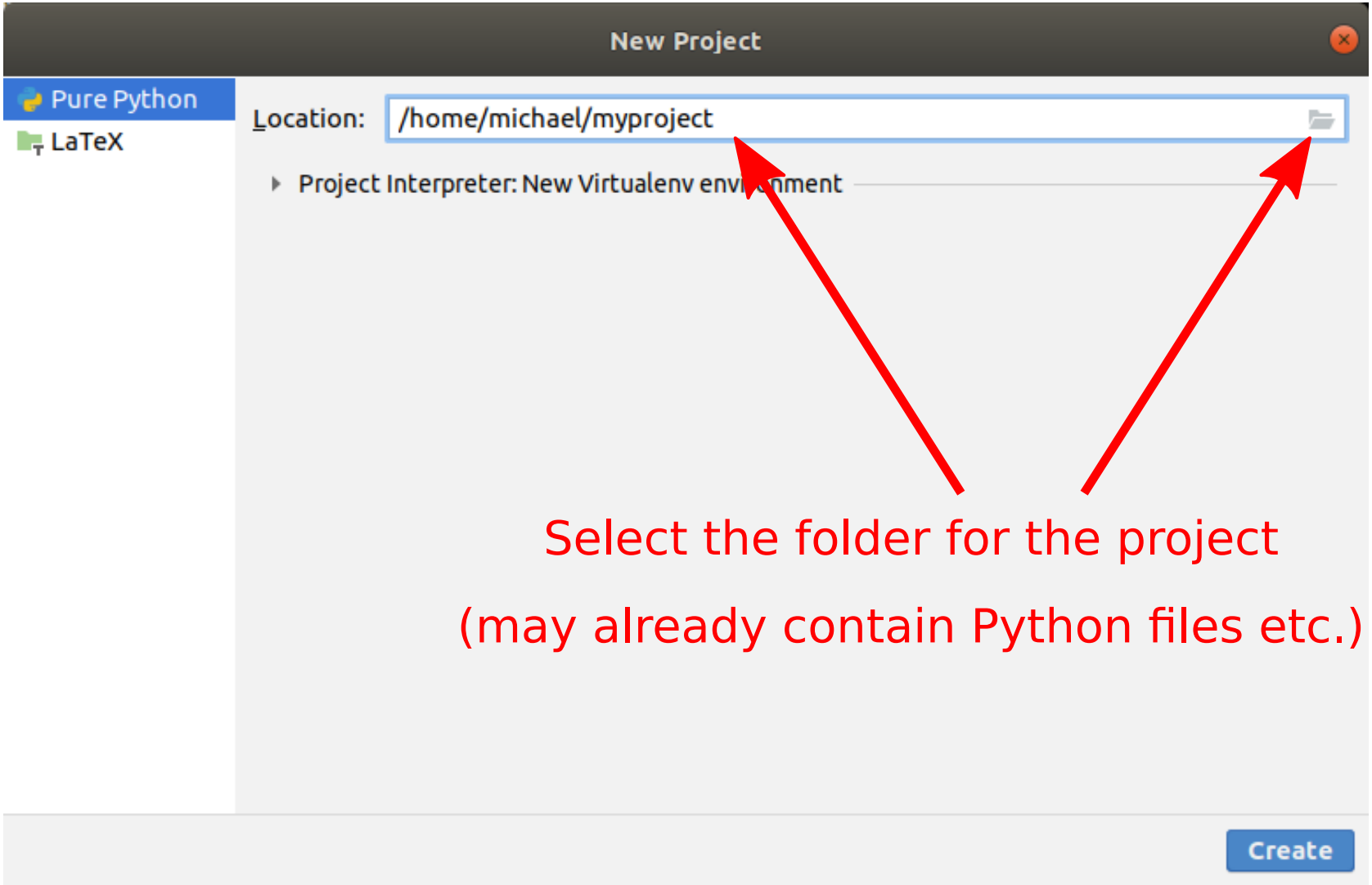
    ```
    sudo snap install pycharm-community --classic
    ```
- The next slides will show you how to use the Editor and Debugger
  - There might be small differences depending on OS and version

PYCHARM – EDITOR

Task 1: Create a new Pycharm project (1)

- We will start by creating a new PyCharm project
  - A project is a folder managed by PyCharm with configurations for Python interpreter, git, etc.
- Follow these steps for the creation of the project (see following slides for help):
  1. Select `File -> New Project...` in the menu or click `Create New Project` at the first start of PyCharm
  2. Select the project folder (does not need to be empty) to create the project in
  3. Select the Python interpreter
  4. Click `Create` to create the project

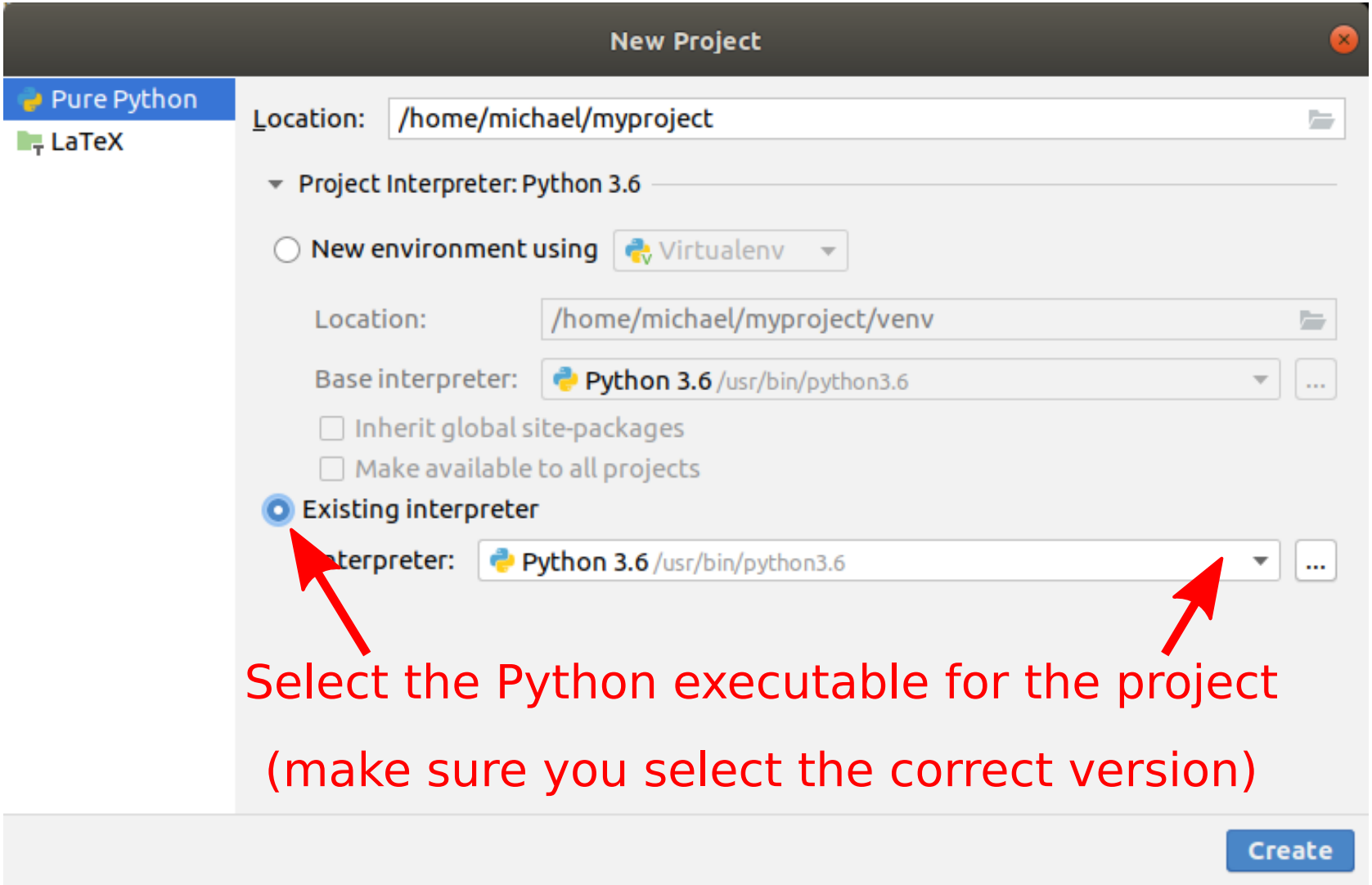
Task 1: Create a new Pycharm project (2)



Select the folder for the project

(may already contain Python files etc.)

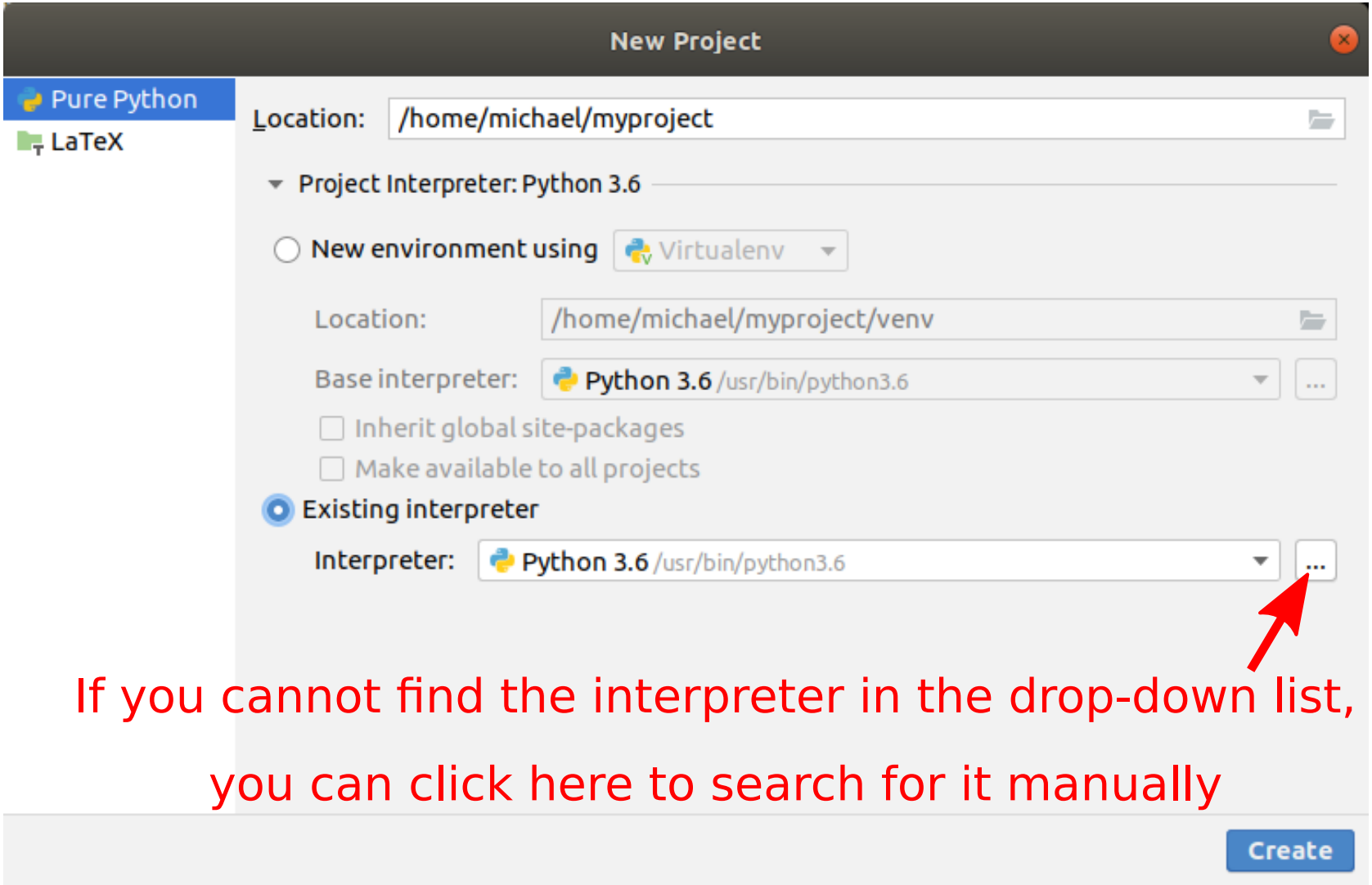
Task 1: Create a new Pycharm project (3)



Select the Python executable for the project

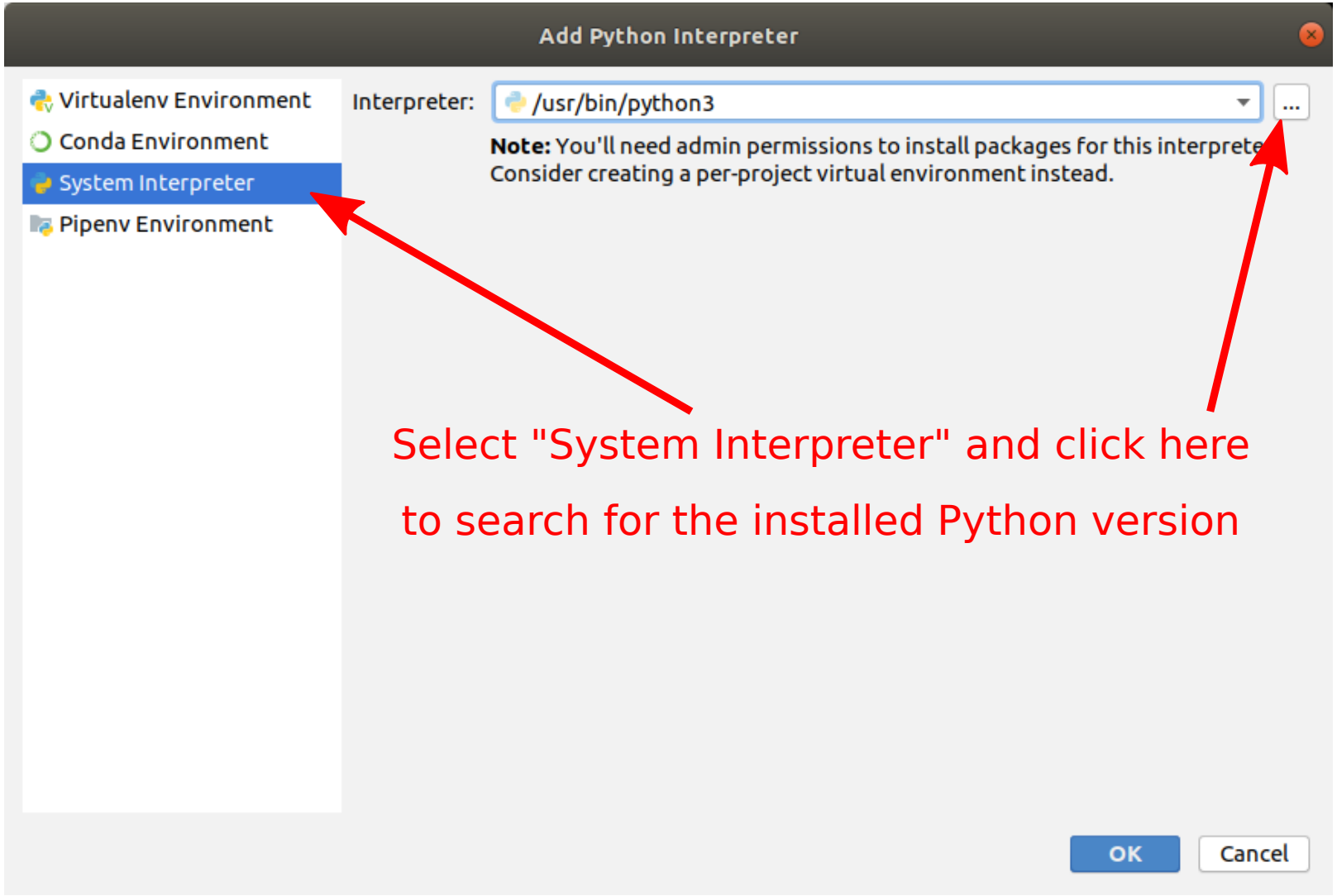
(make sure you select the correct version)

Task 1: Create a new Pycharm project (4)



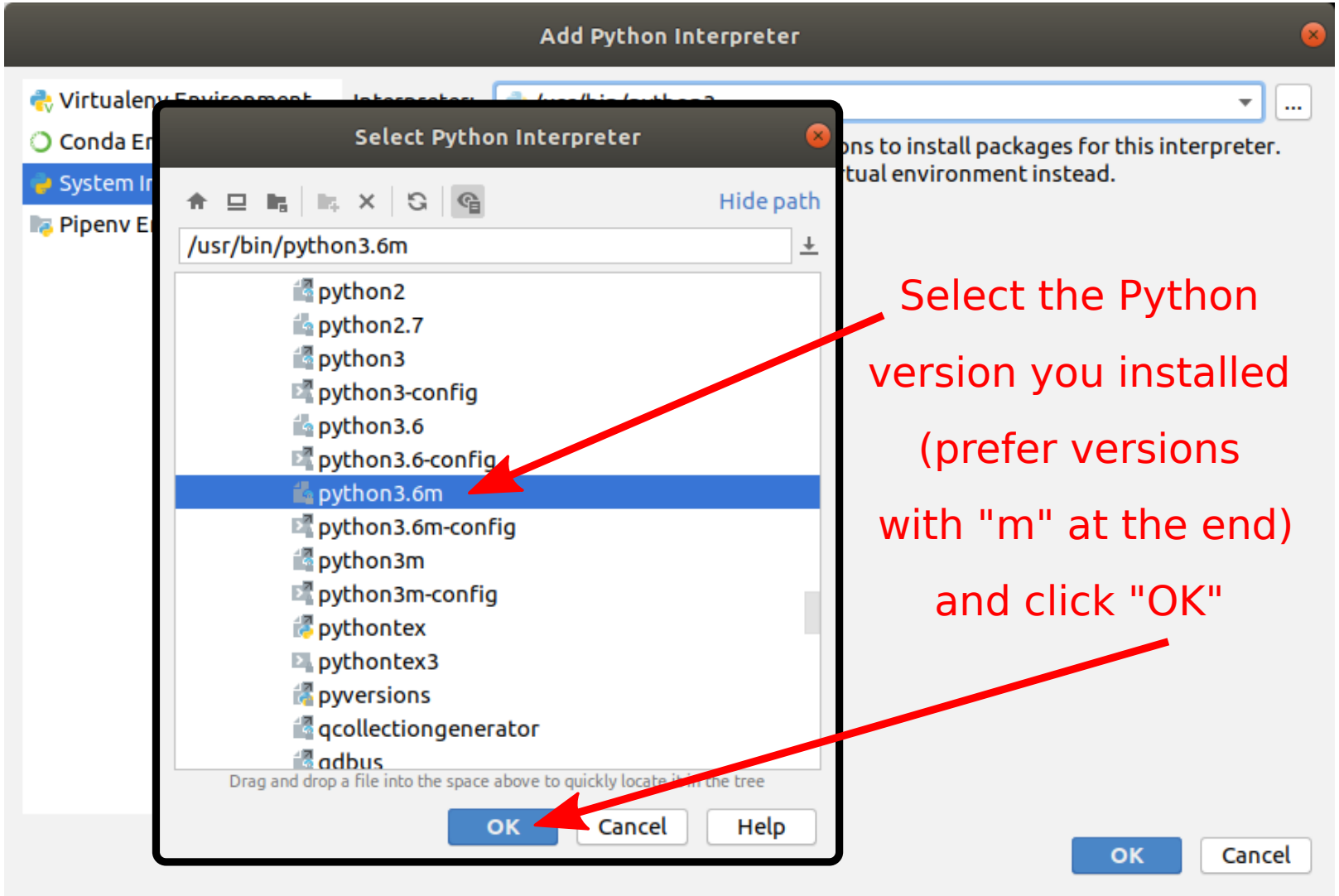
If you cannot find the interpreter in the drop-down list, you can click here to search for it manually

Task 1: Create a new Pycharm project (5)

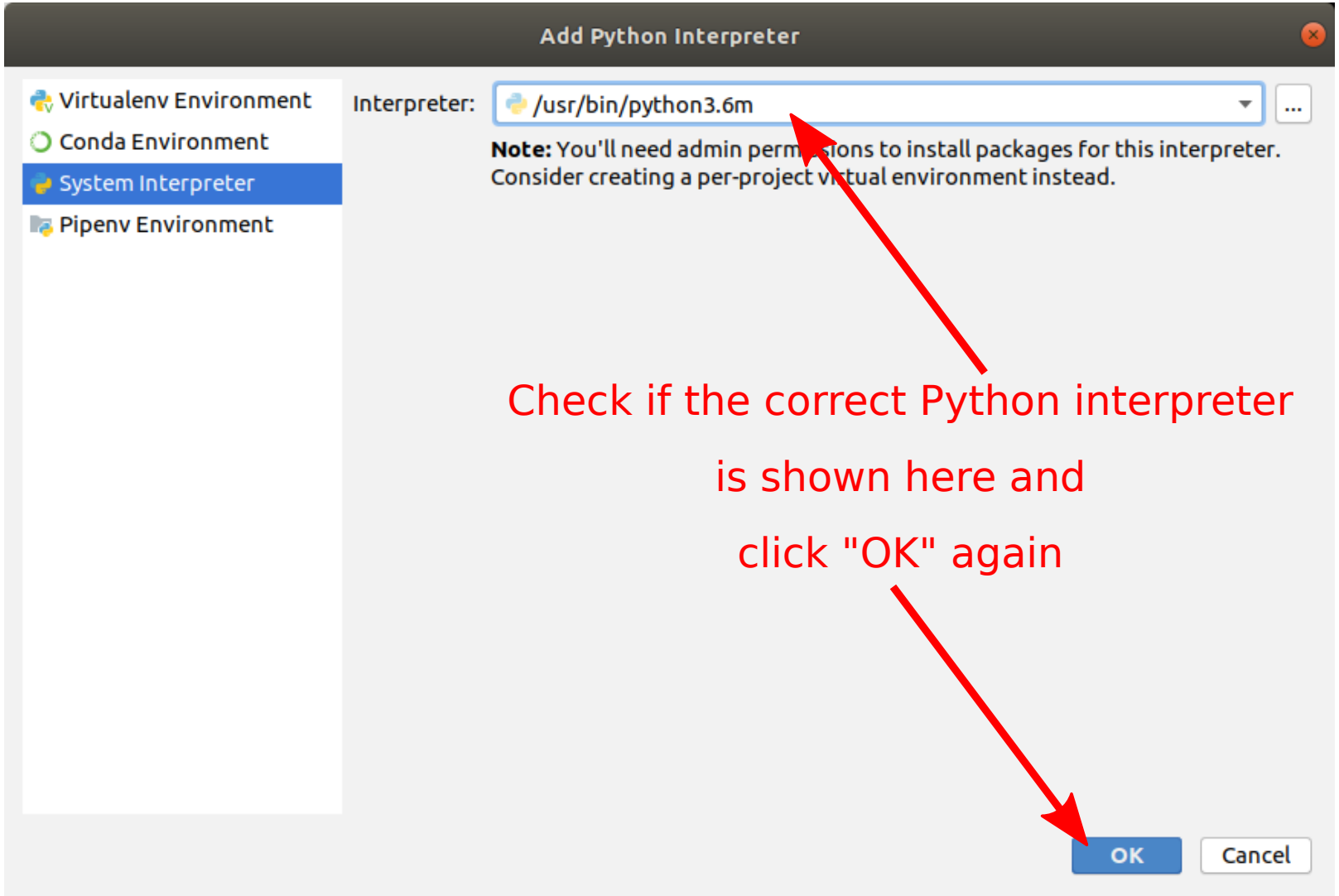


Select "System Interpreter" and click here to search for the installed Python version

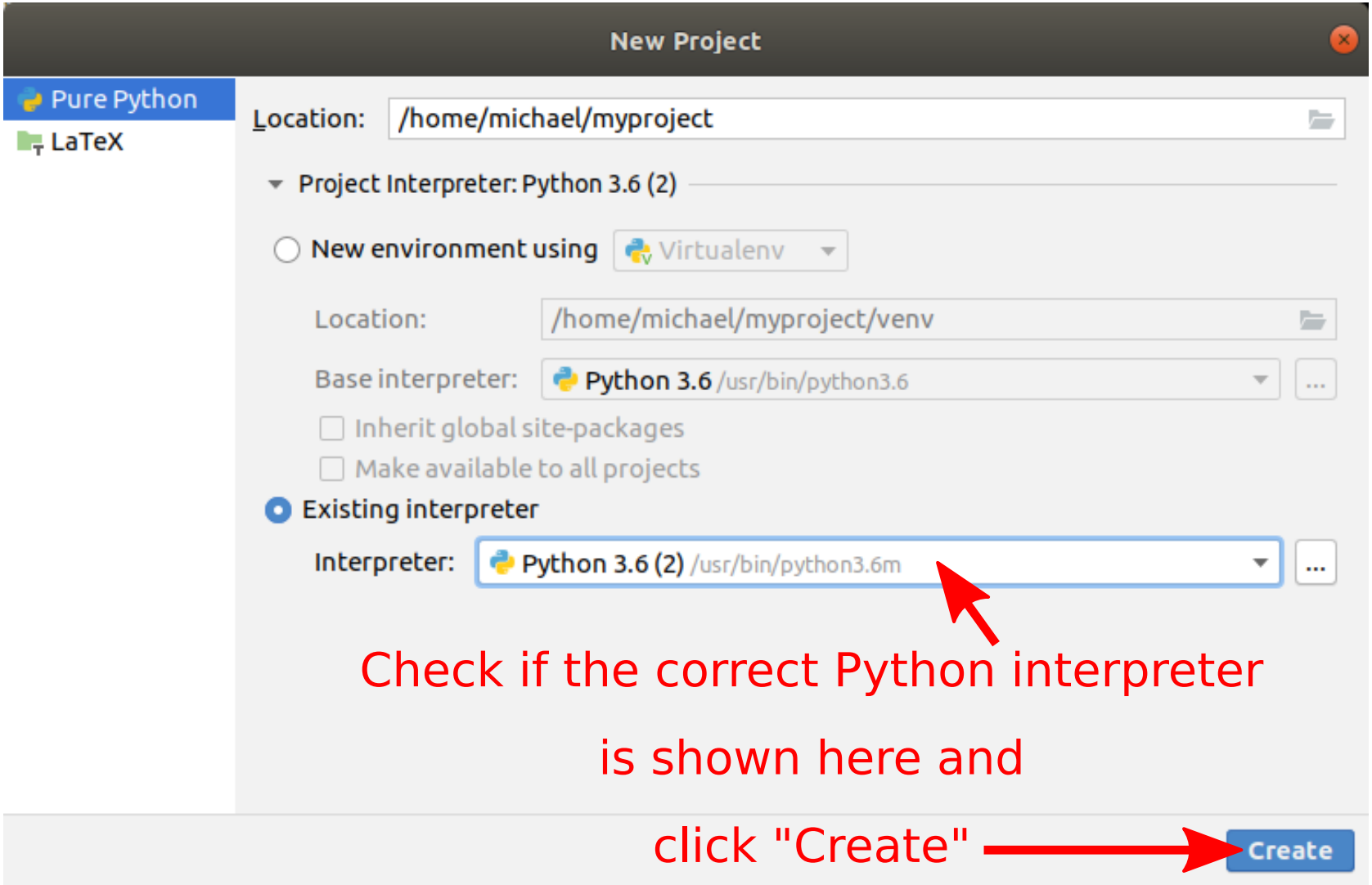
Task 1: Create a new Pycharm project (6)



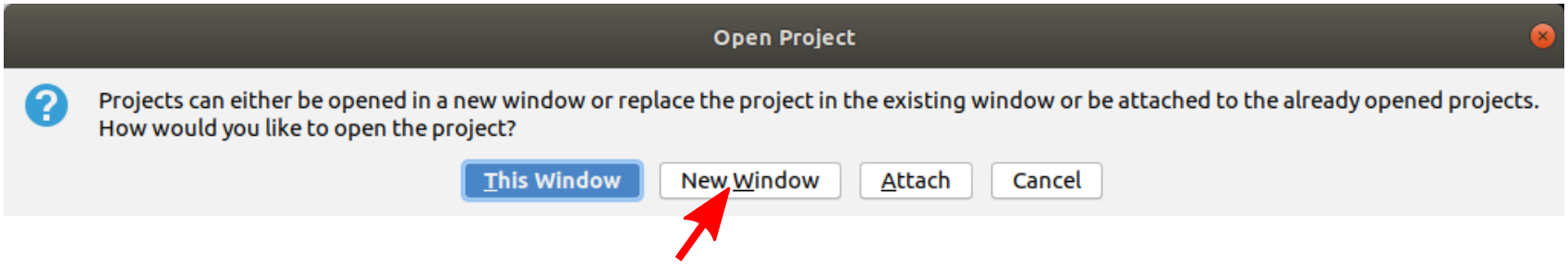
Task 1: Create a new Pycharm project (7)



Task 1: Create a new Pycharm project (8)



Task 1: Create a new Pycharm project (9)



Click "New Window" and wait for the project to be created

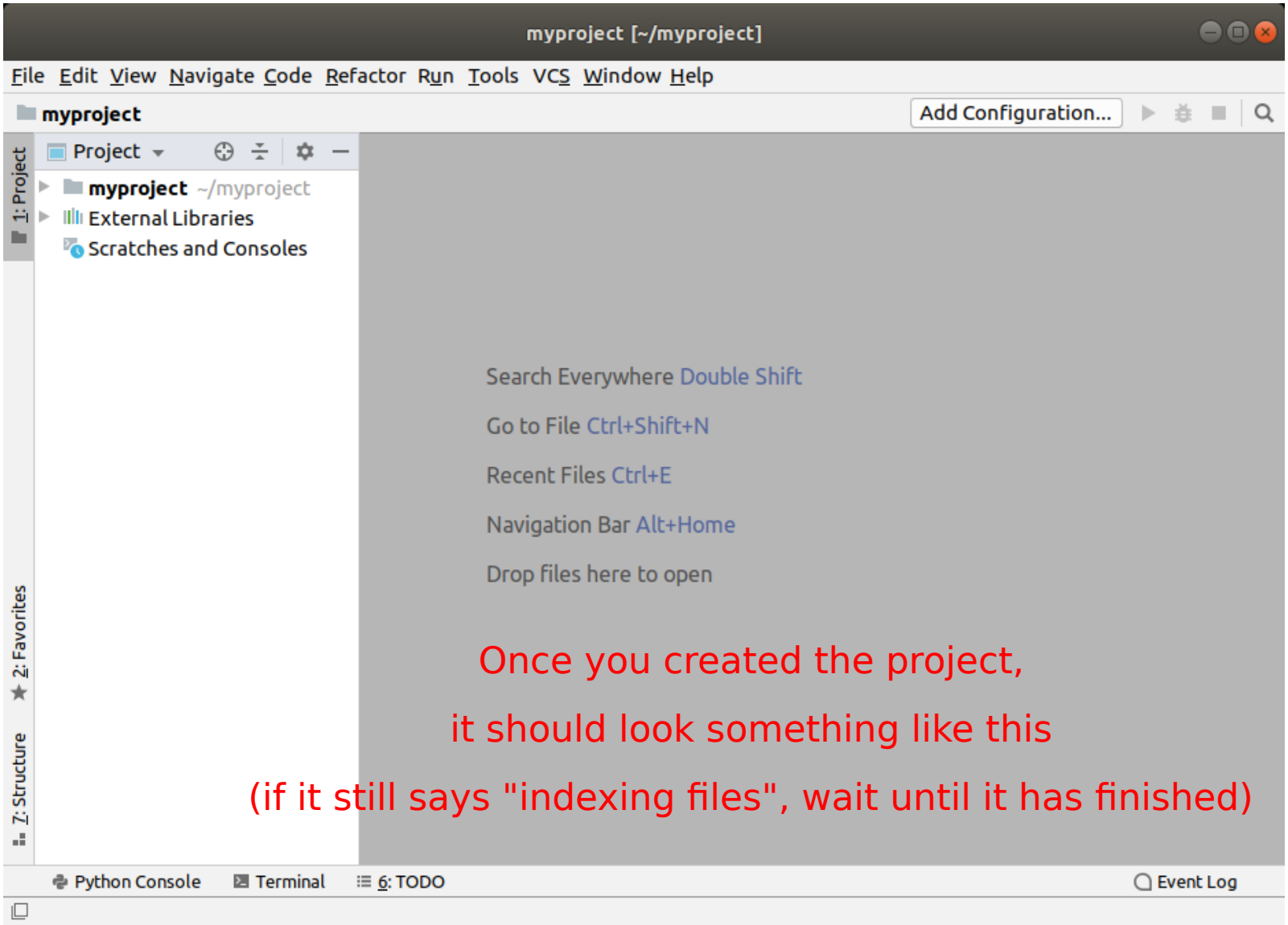
(might take a few seconds)

PYCHARM – PYTHON CONSOLE

Task 2: Use the project Python console (1)

- We will now use a Python console in the PyCharm project in these steps (see following slides for help):
  1. Open a PyCharm project or create a new one
  2. Click on `Python Console` at the lower left corner of the PyCharm window
  3. Type `print("Hello world")` into the console, press Enter, and check the output
  4. Type `print(a=5)` into the console, press Enter, and check the variable explorer on the right side
  5. Close the console by closing the `Python Console` tab

Task 2: Use the project Python console (2)

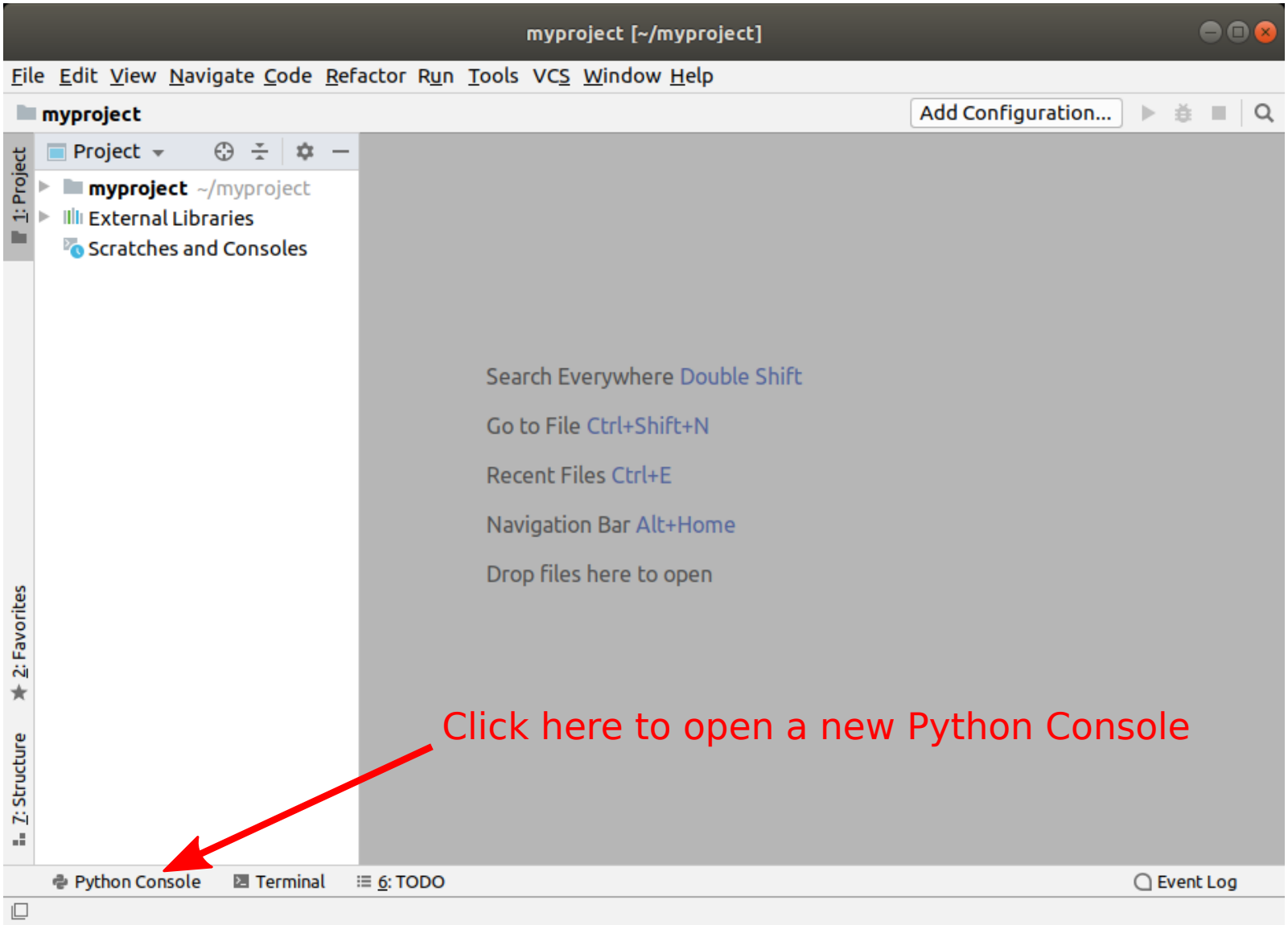


Once you created the project,

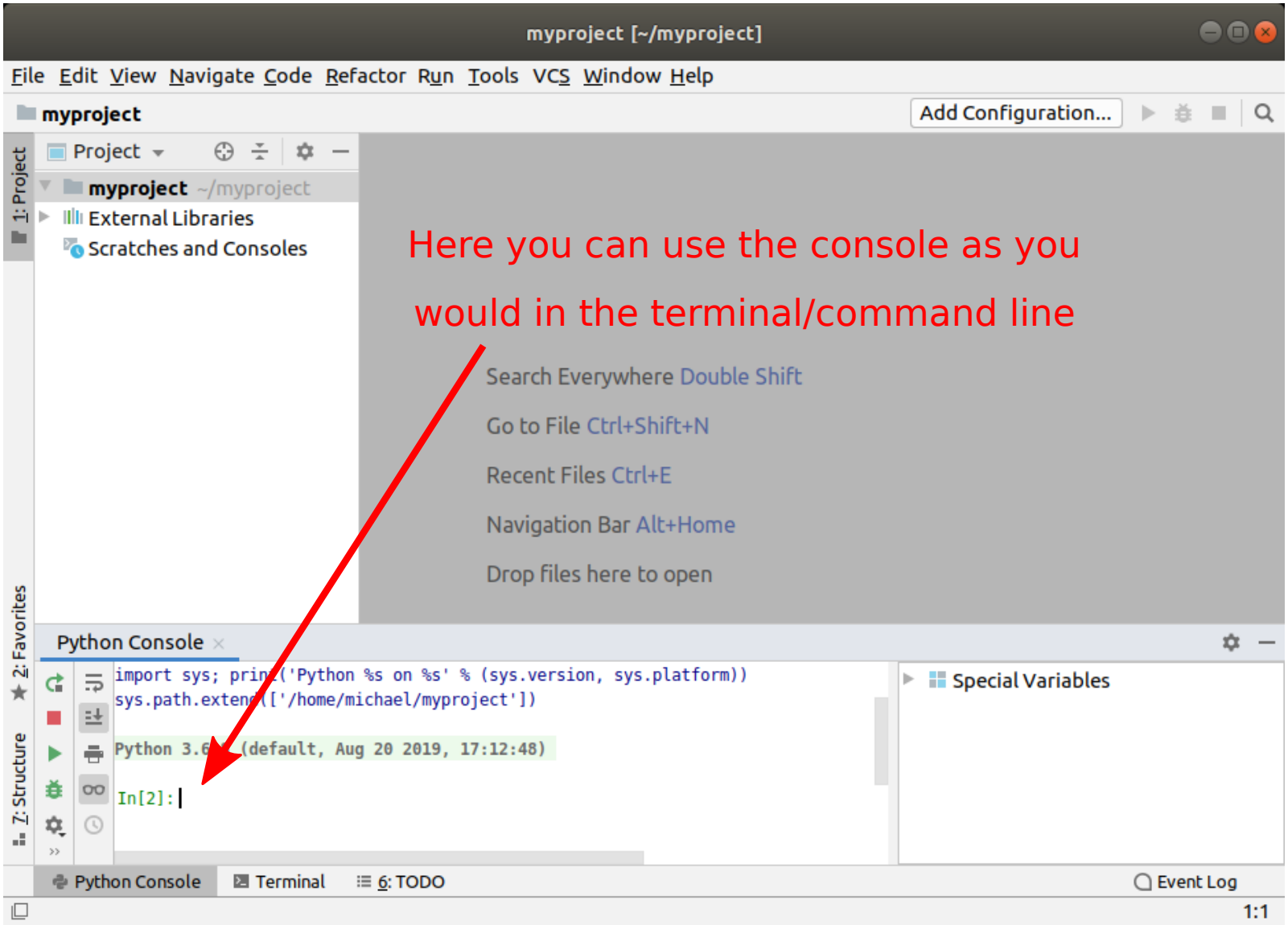
it should look something like this

(if it still says "indexing files", wait until it has finished)

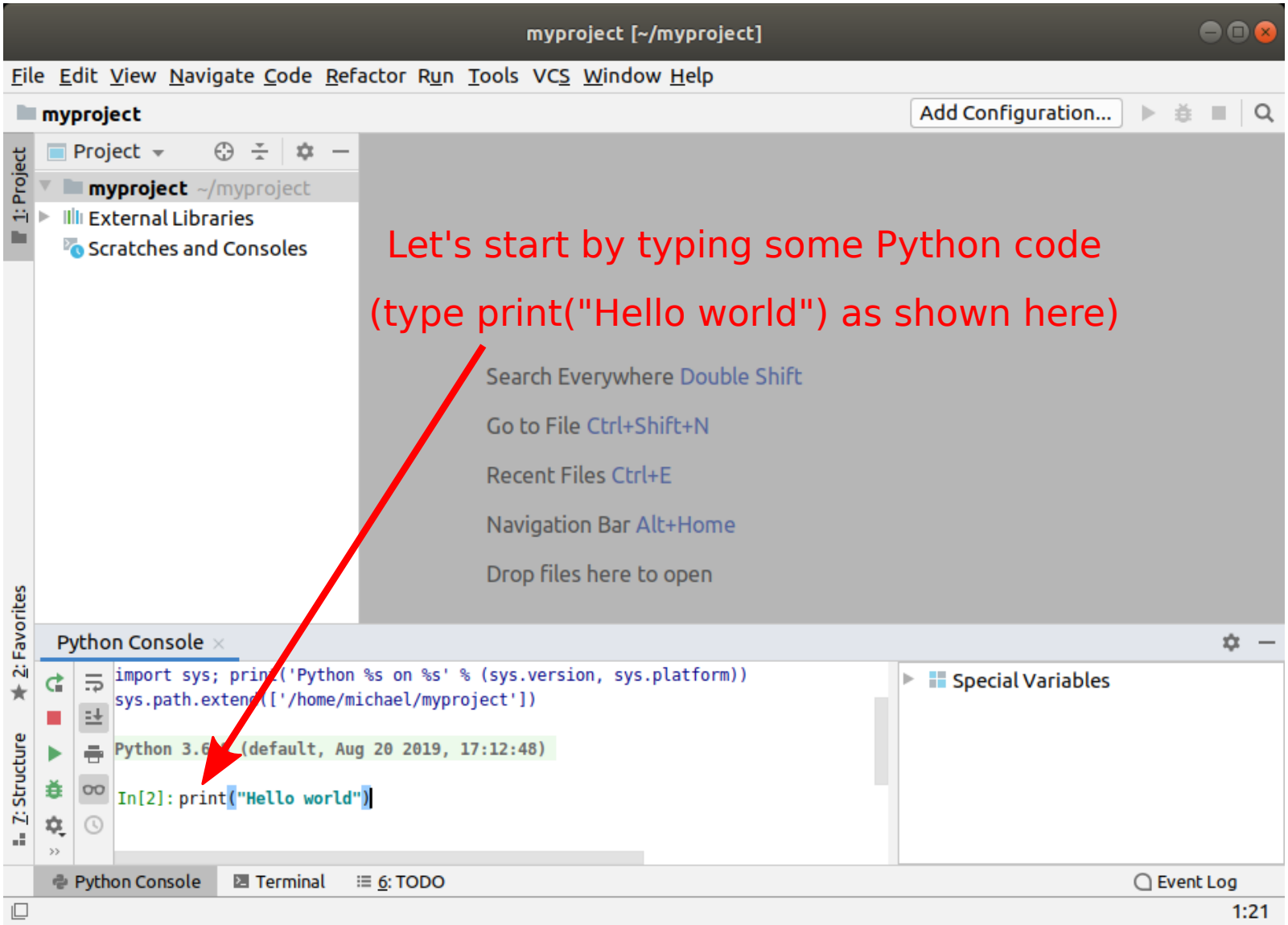
Task 2: Use the project Python console (3)



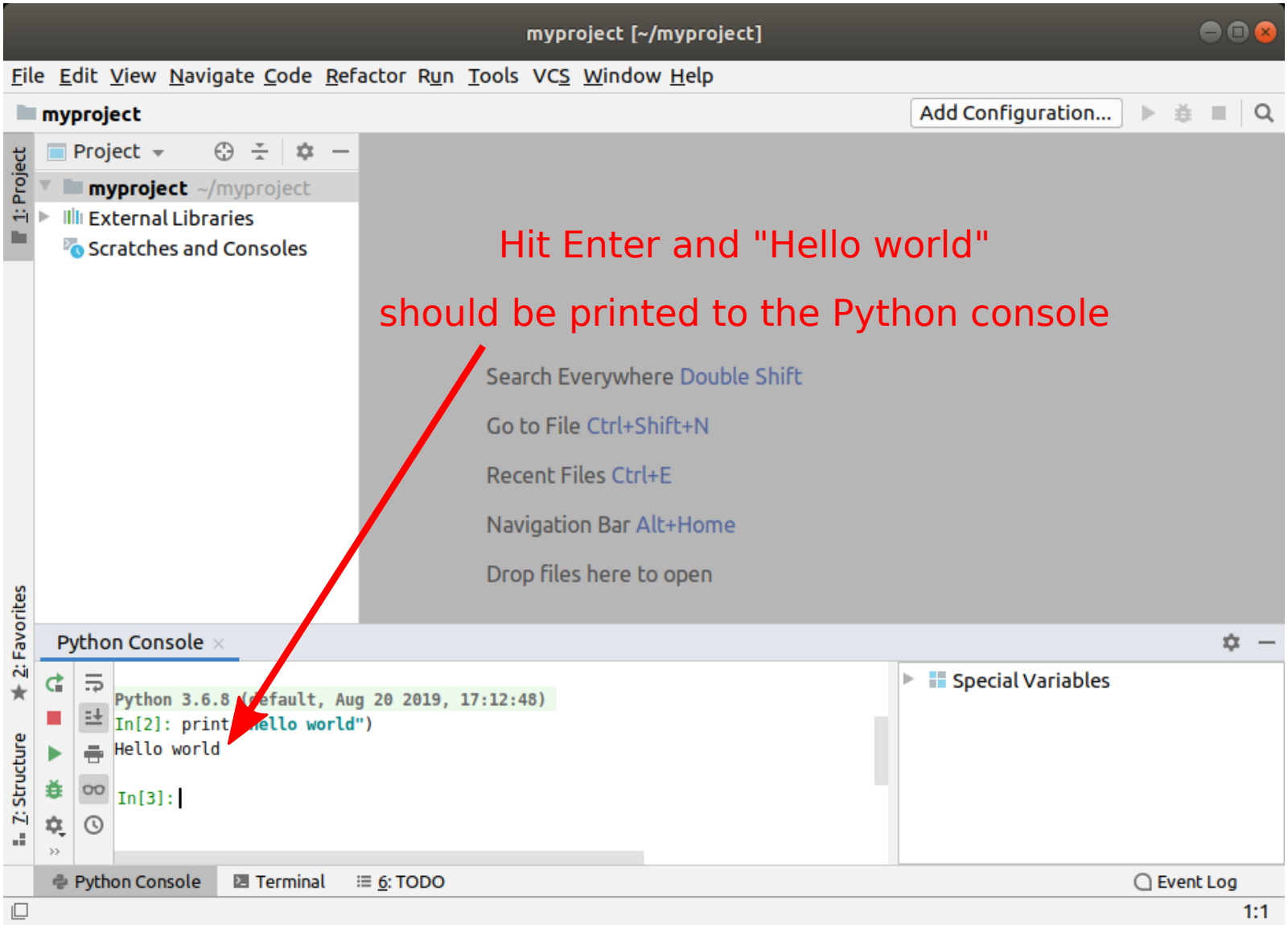
Task 2: Use the project Python console (4)



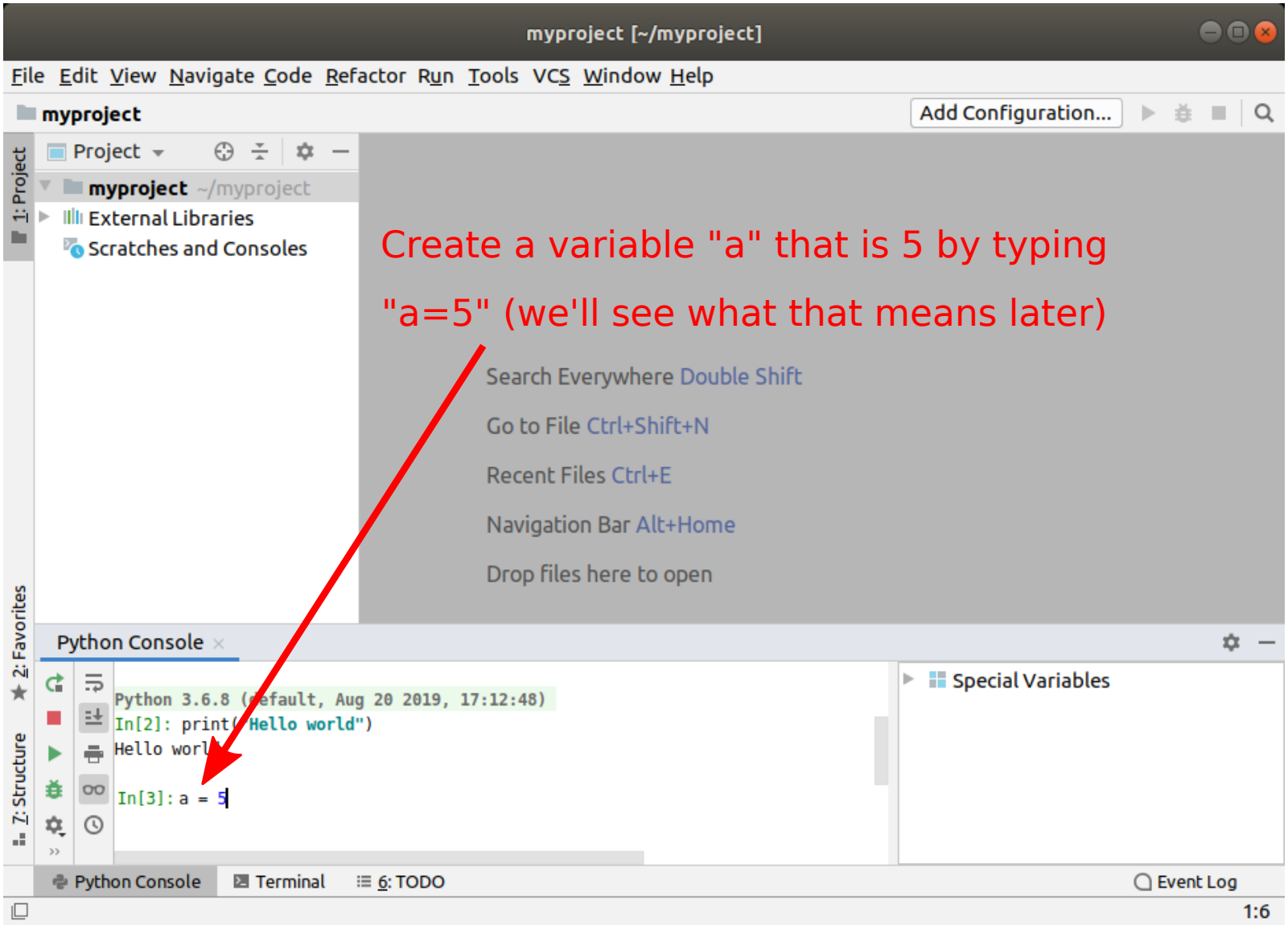
Task 2: Use the project Python console (5)



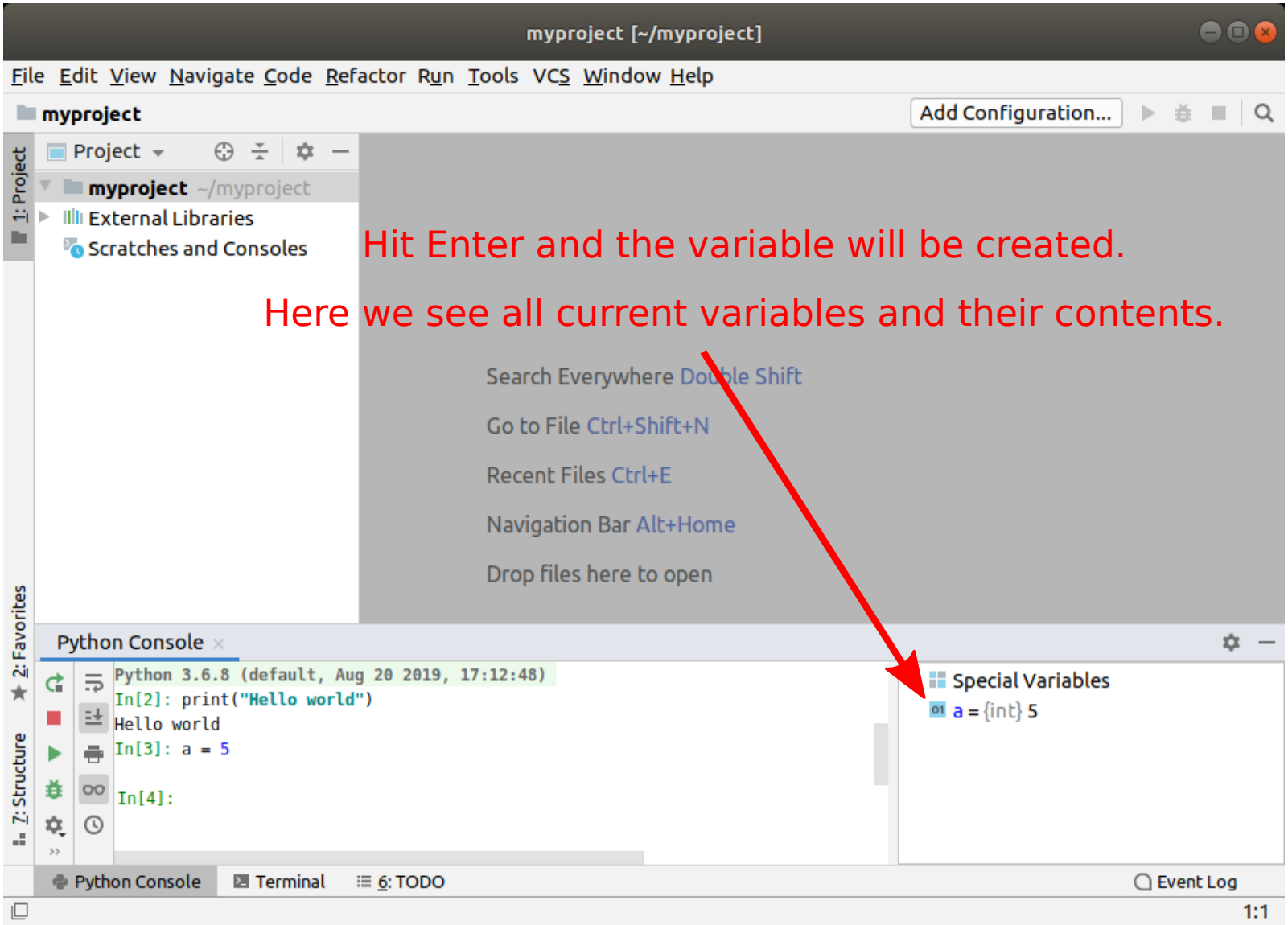
Task 2: Use the project Python console (6)



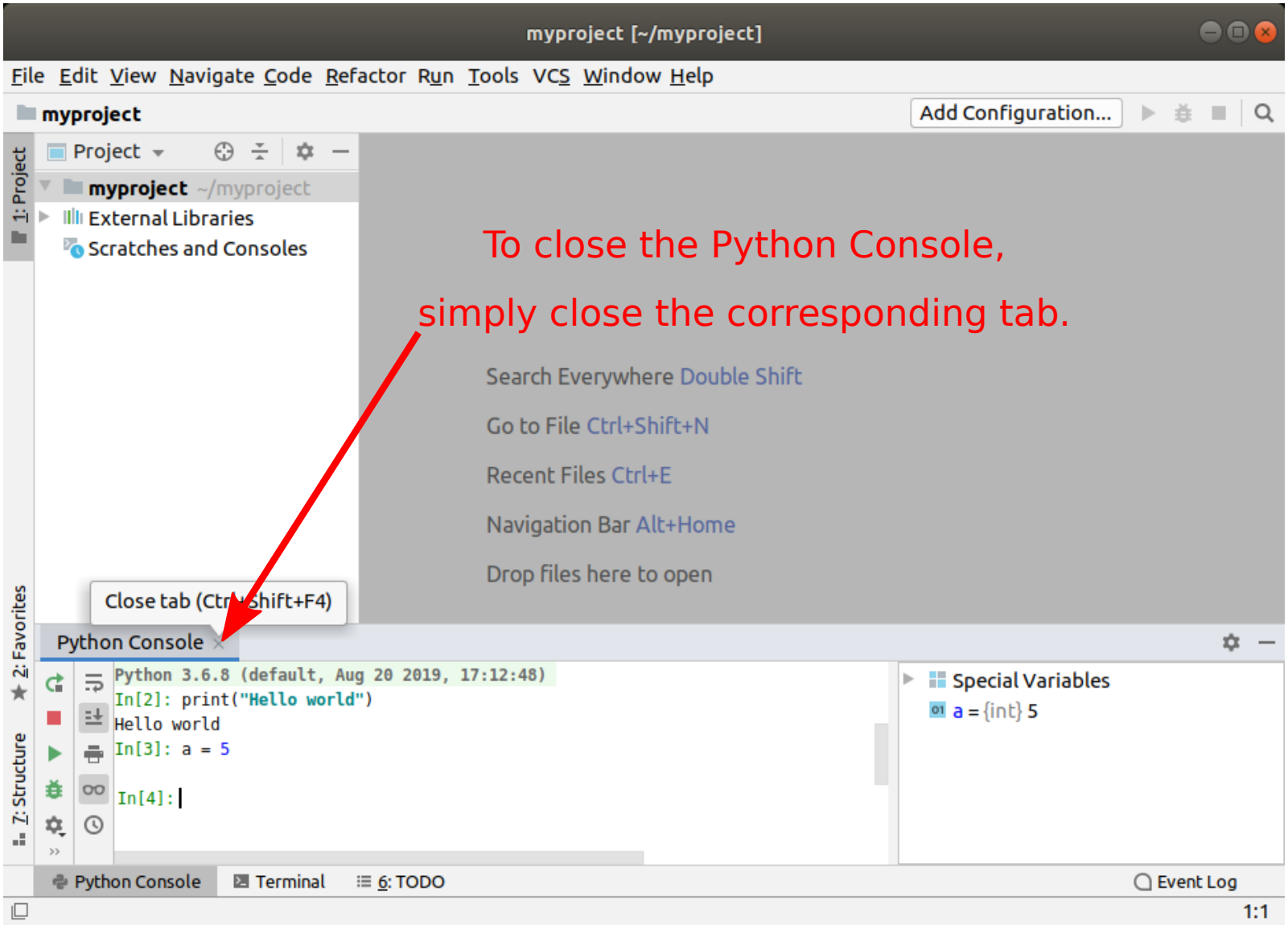
Task 2: Use the project Python console (7)



Task 2: Use the project Python console (8)



Task 2: Use the project Python console (9)

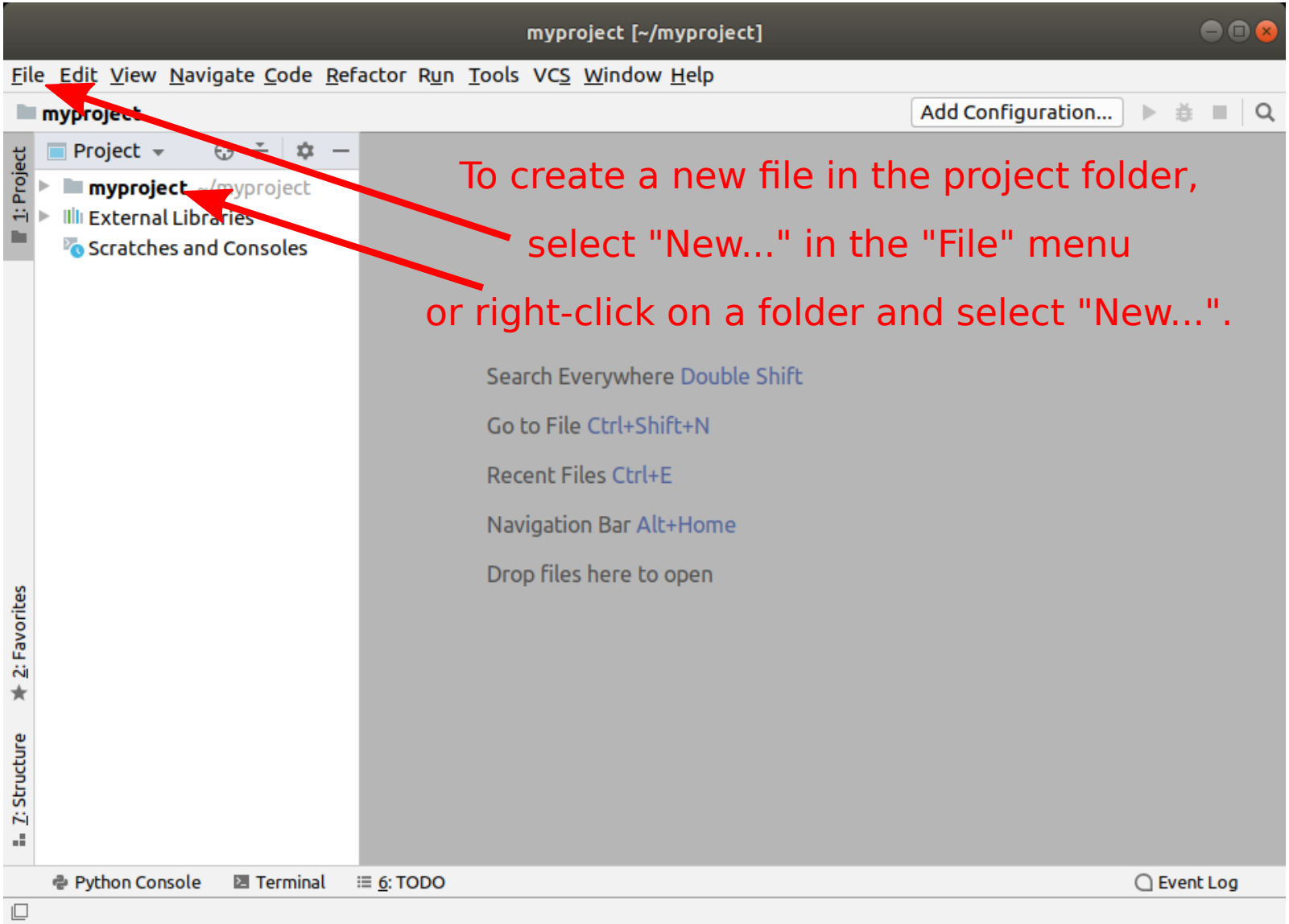


PYCHARM – RUNNING A PYTHON PROGRAM

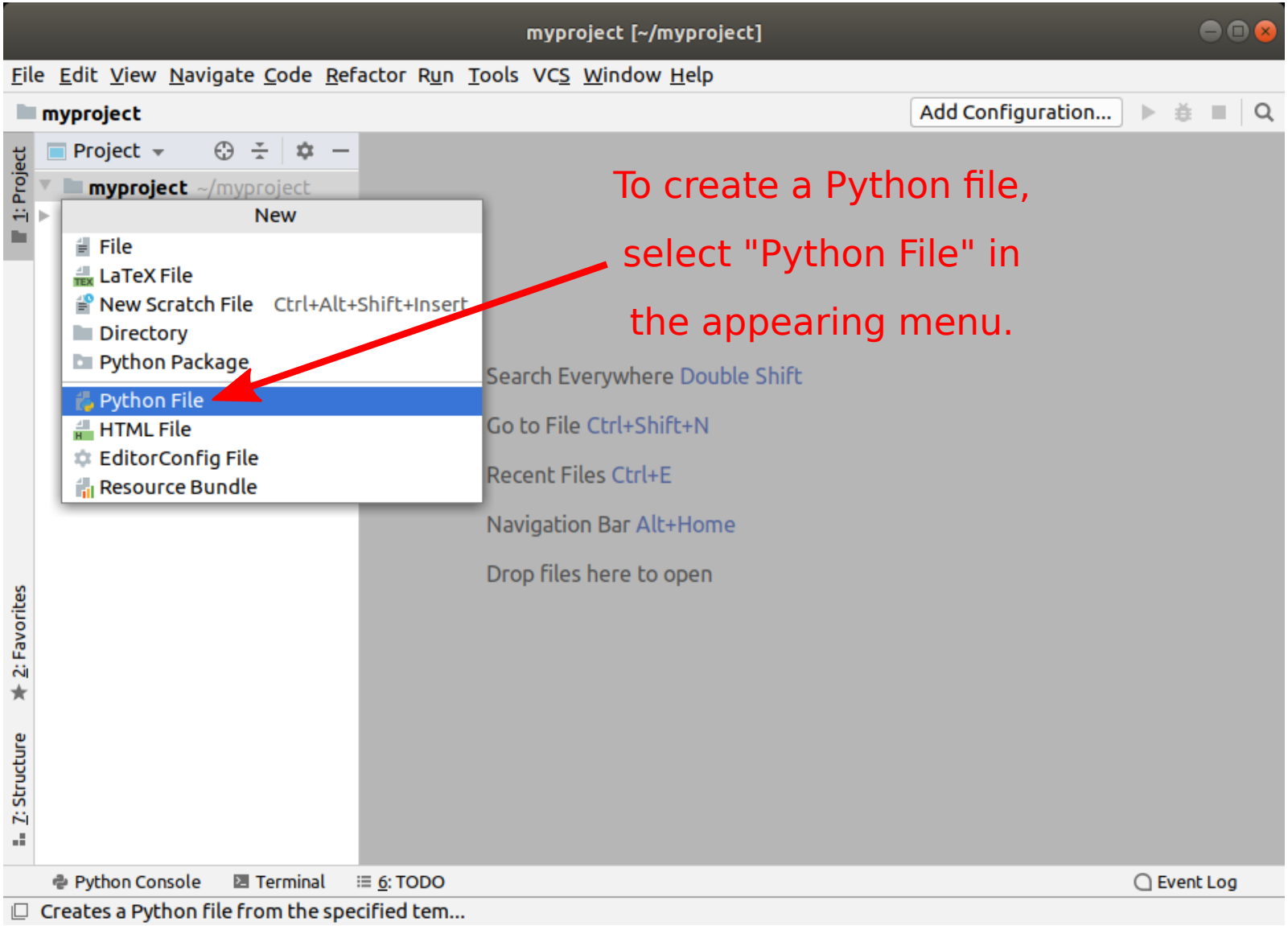
Task 3: Running a Python program (1)

- We will now execute (=run) a Python program in PyCharm in these steps (see following slides for help):
  1. Create a new Python file `test.py` with content `print("Hello World!")`
  2. Create a run configuration for this file
  3. Run the file by clicking on the "Run" button (green triangle)
  4. Check the `Console` tab output (bottom of the screen); it should write "Hello World!"

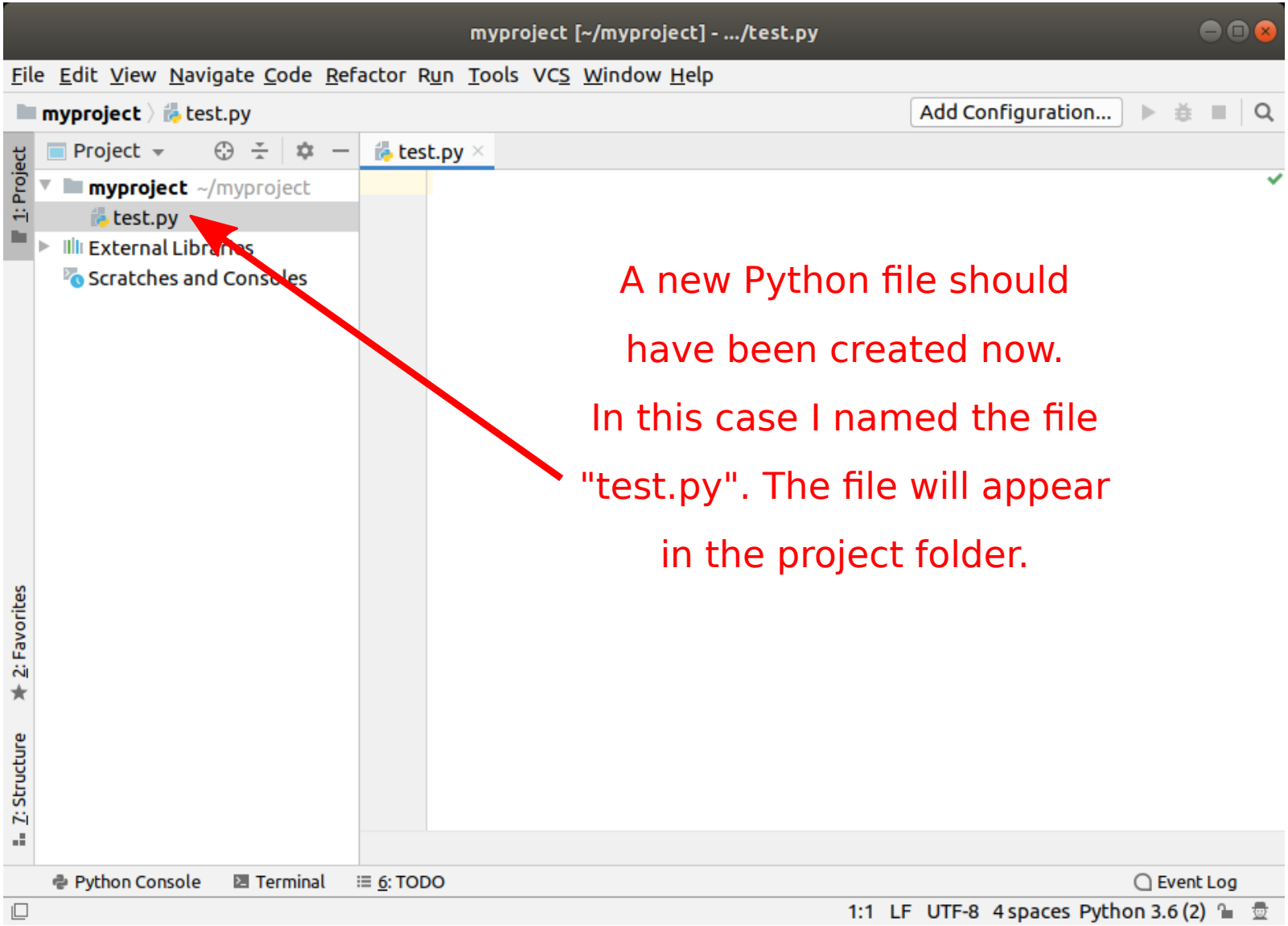
Task 3: Running a Python program (2)



Task 3: Running a Python program (3)

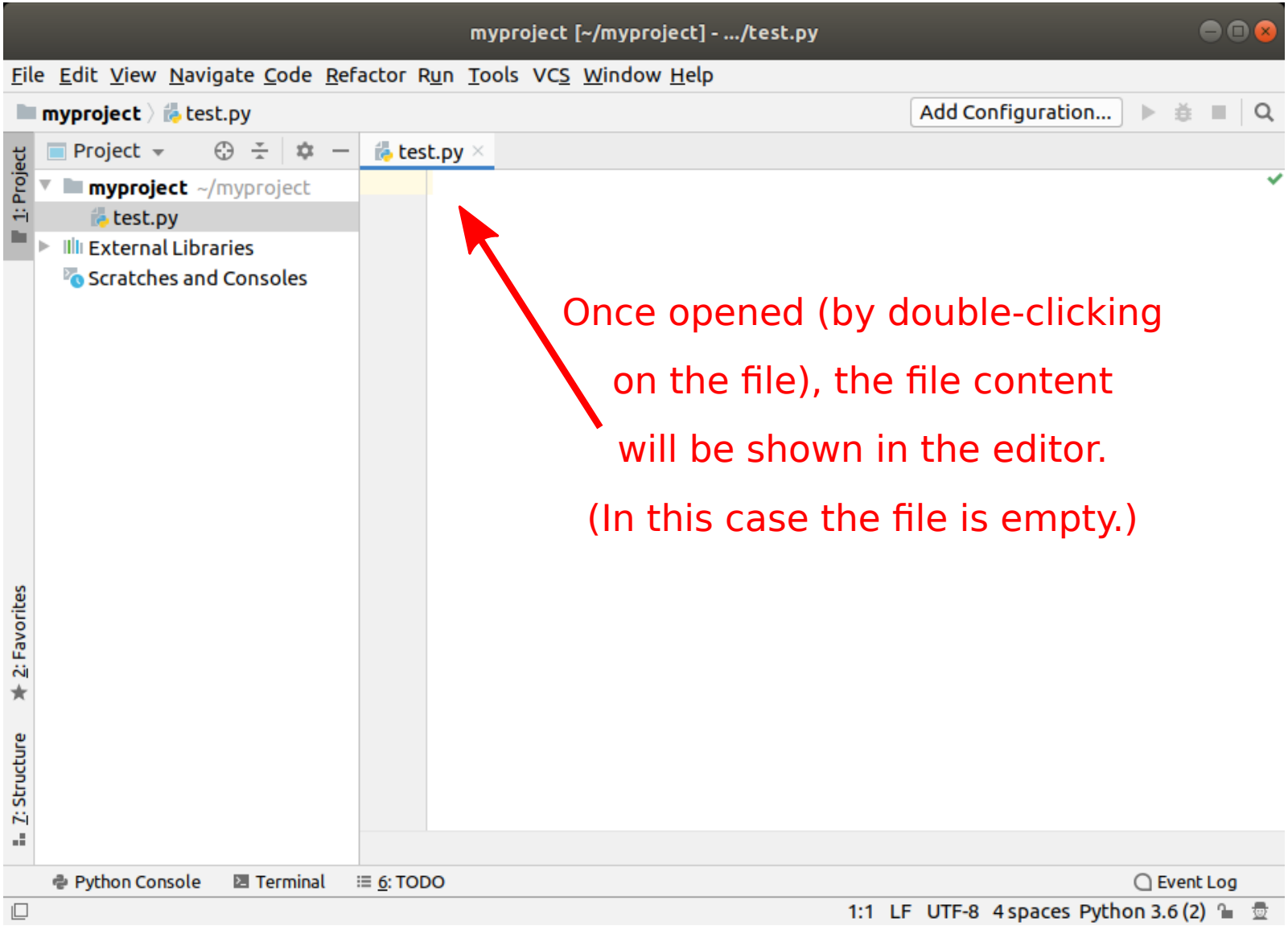


Task 3: Running a Python program (4)



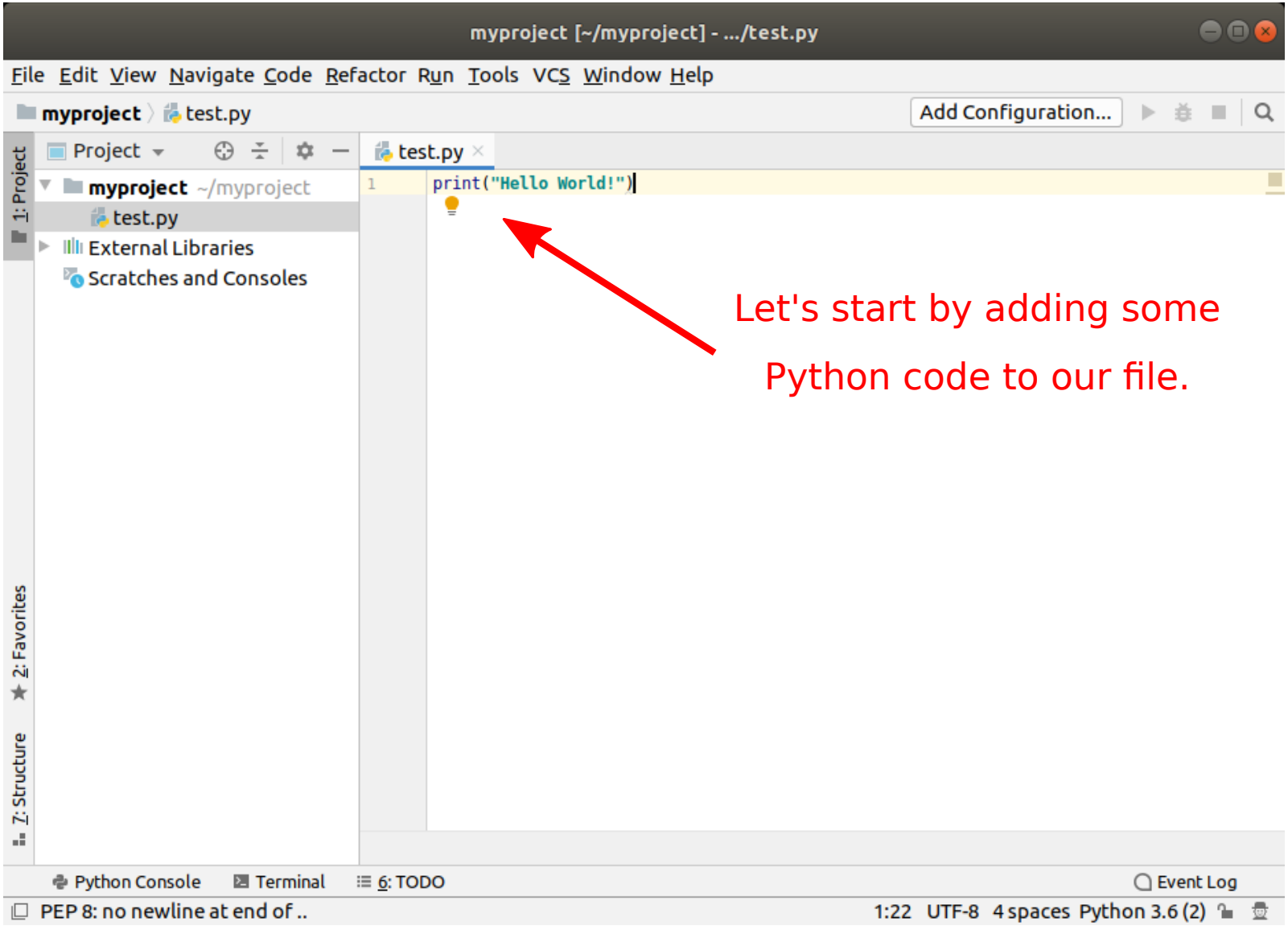
A new Python file should have been created now. In this case I named the file "test.py". The file will appear in the project folder.

Task 3: Running a Python program (5)

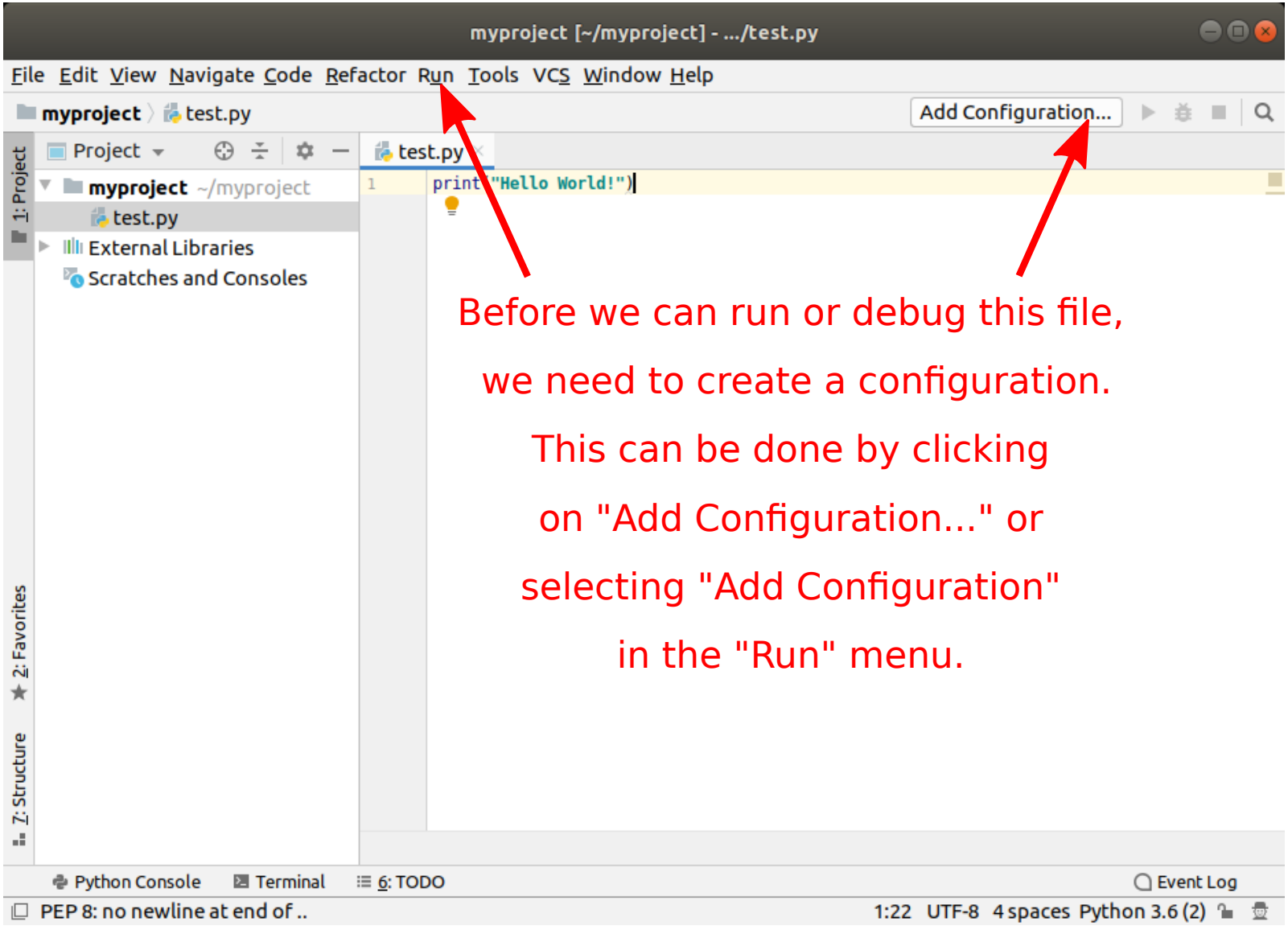


Once opened (by double-clicking on the file), the file content will be shown in the editor. (In this case the file is empty.)

Task 3: Running a Python program (6)

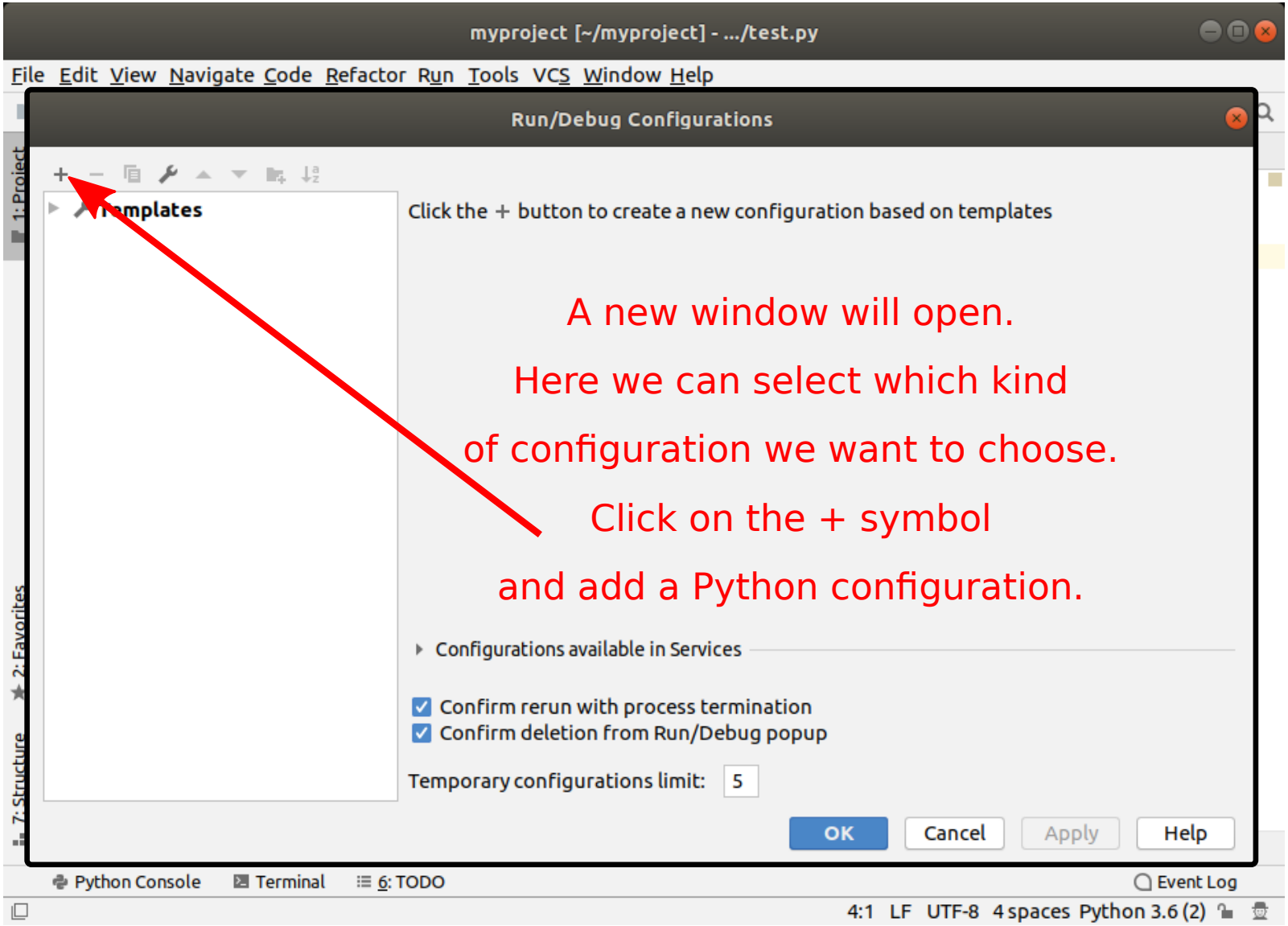


Task 3: Running a Python program (7)

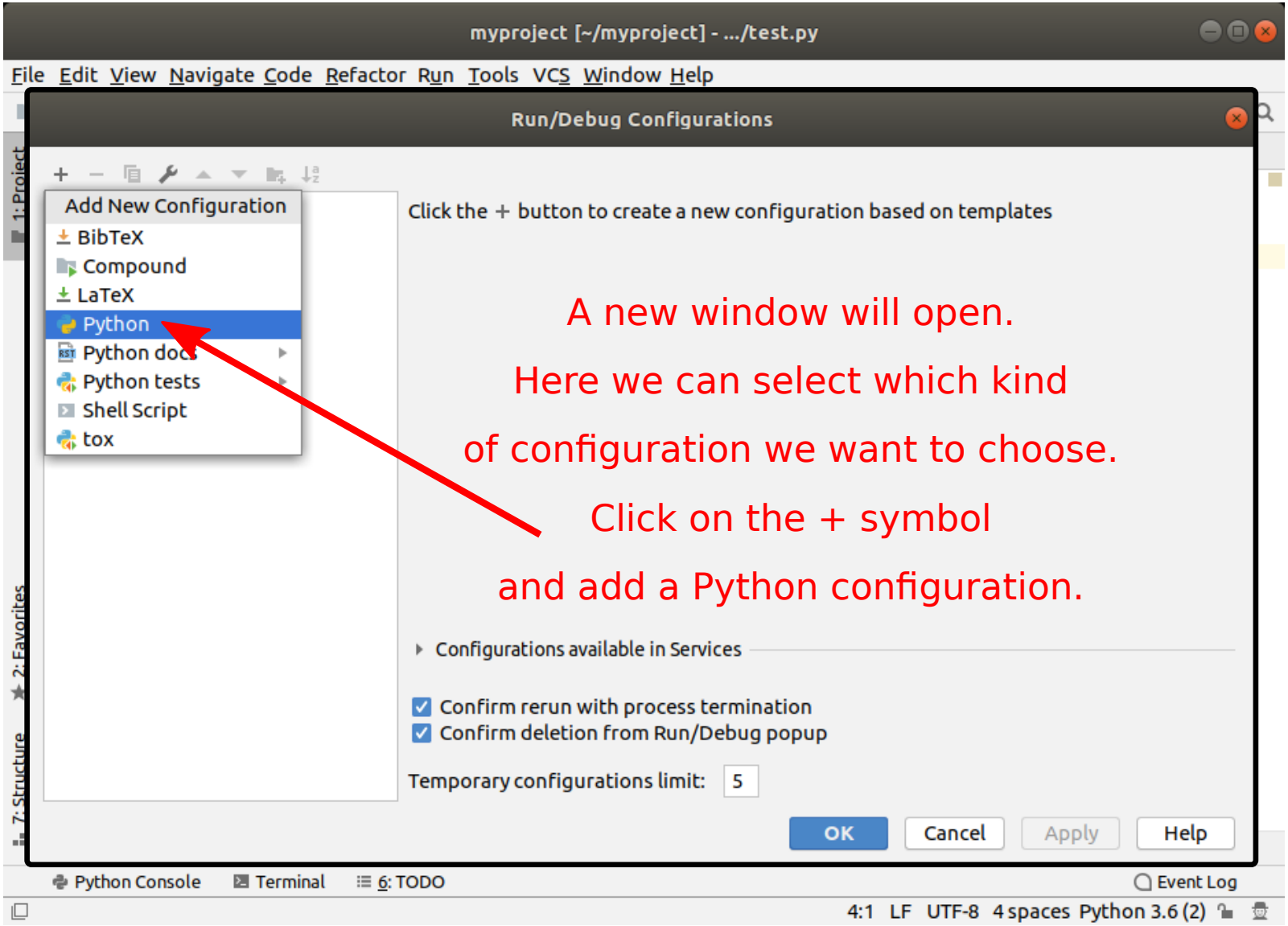


Before we can run or debug this file, we need to create a configuration. This can be done by clicking on "Add Configuration..." or selecting "Add Configuration" in the "Run" menu.

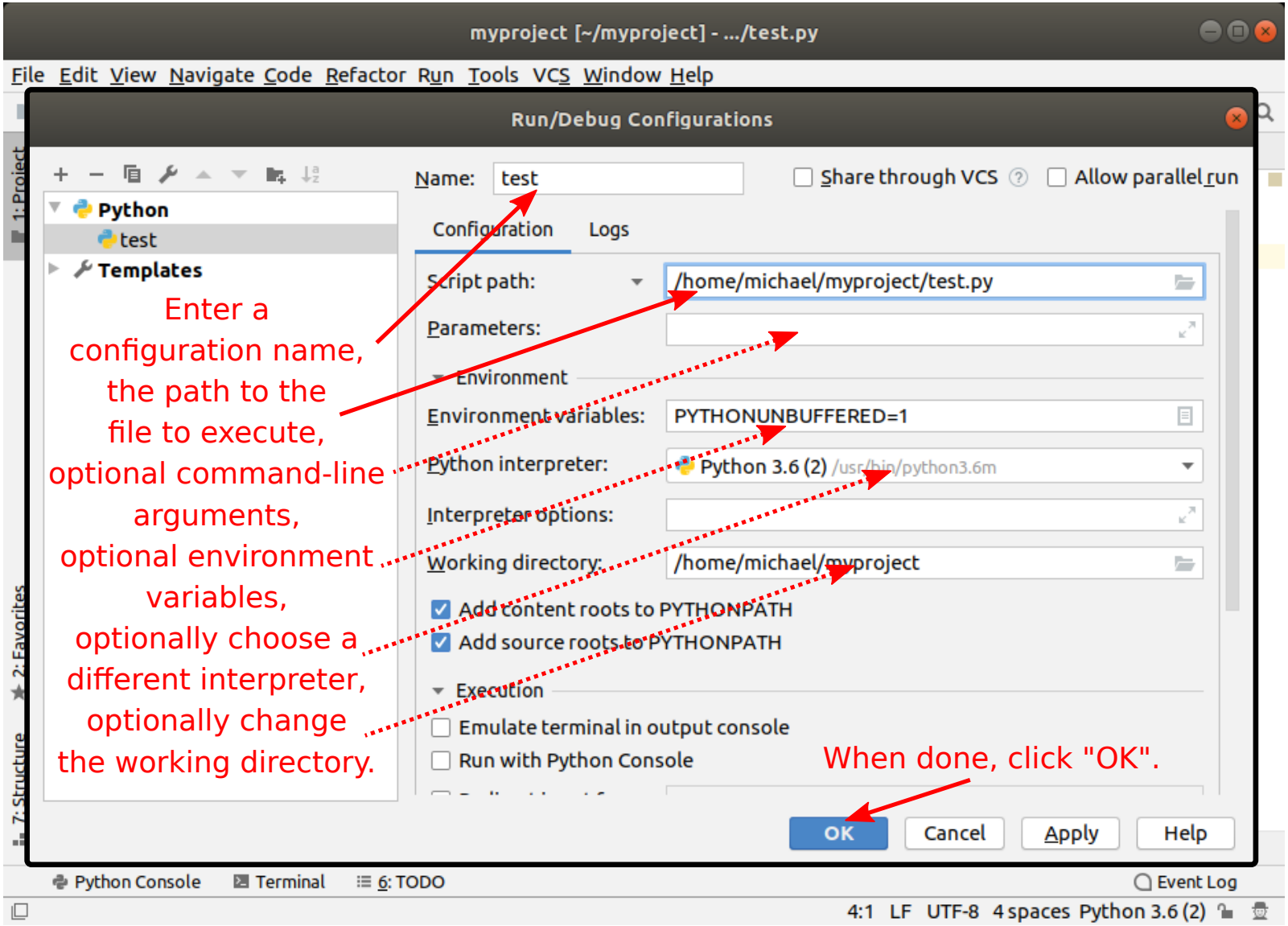
Task 3: Running a Python program (8)



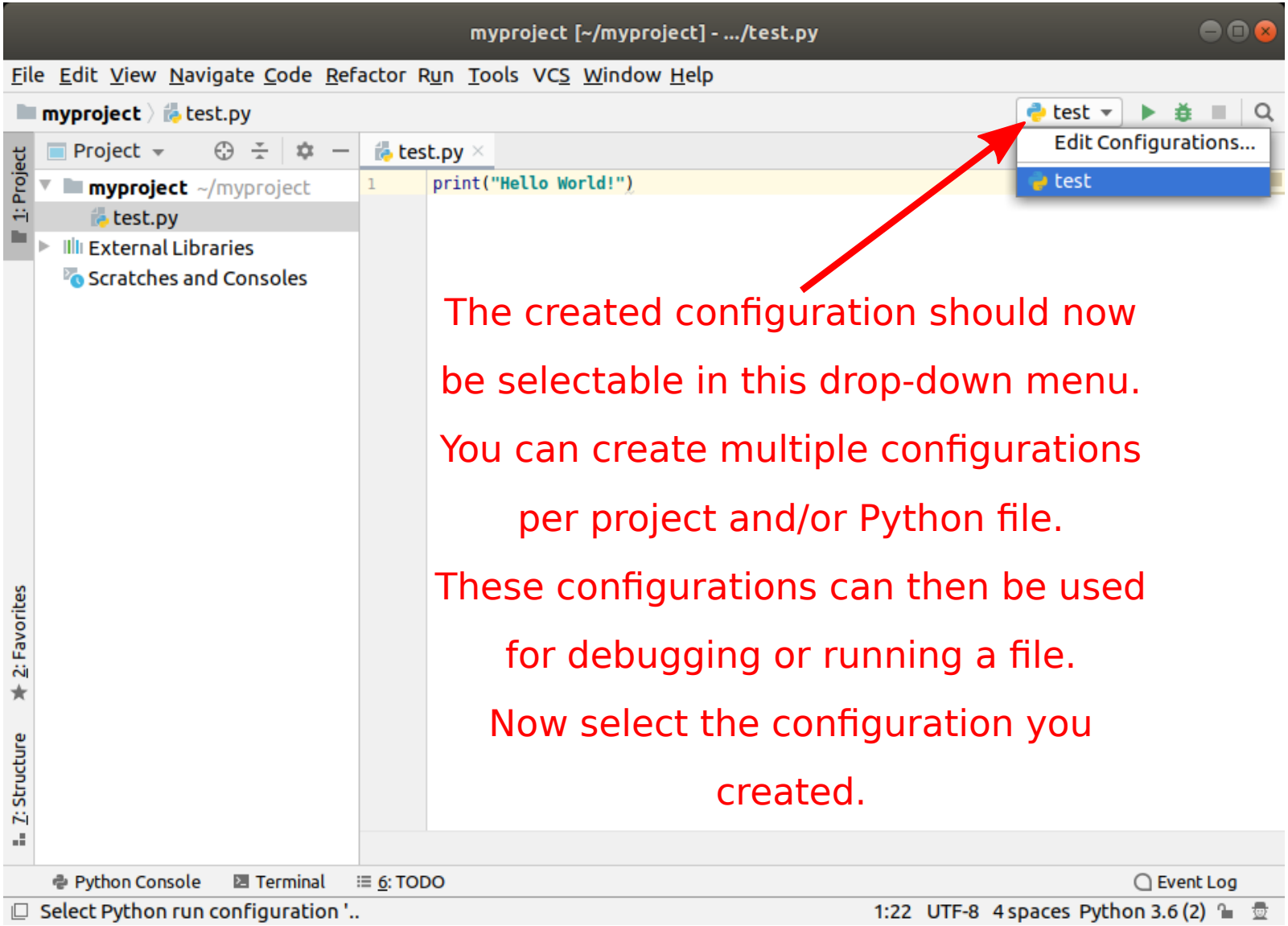
Task 3: Running a Python program (9)



Task 3: Running a Python program (10)

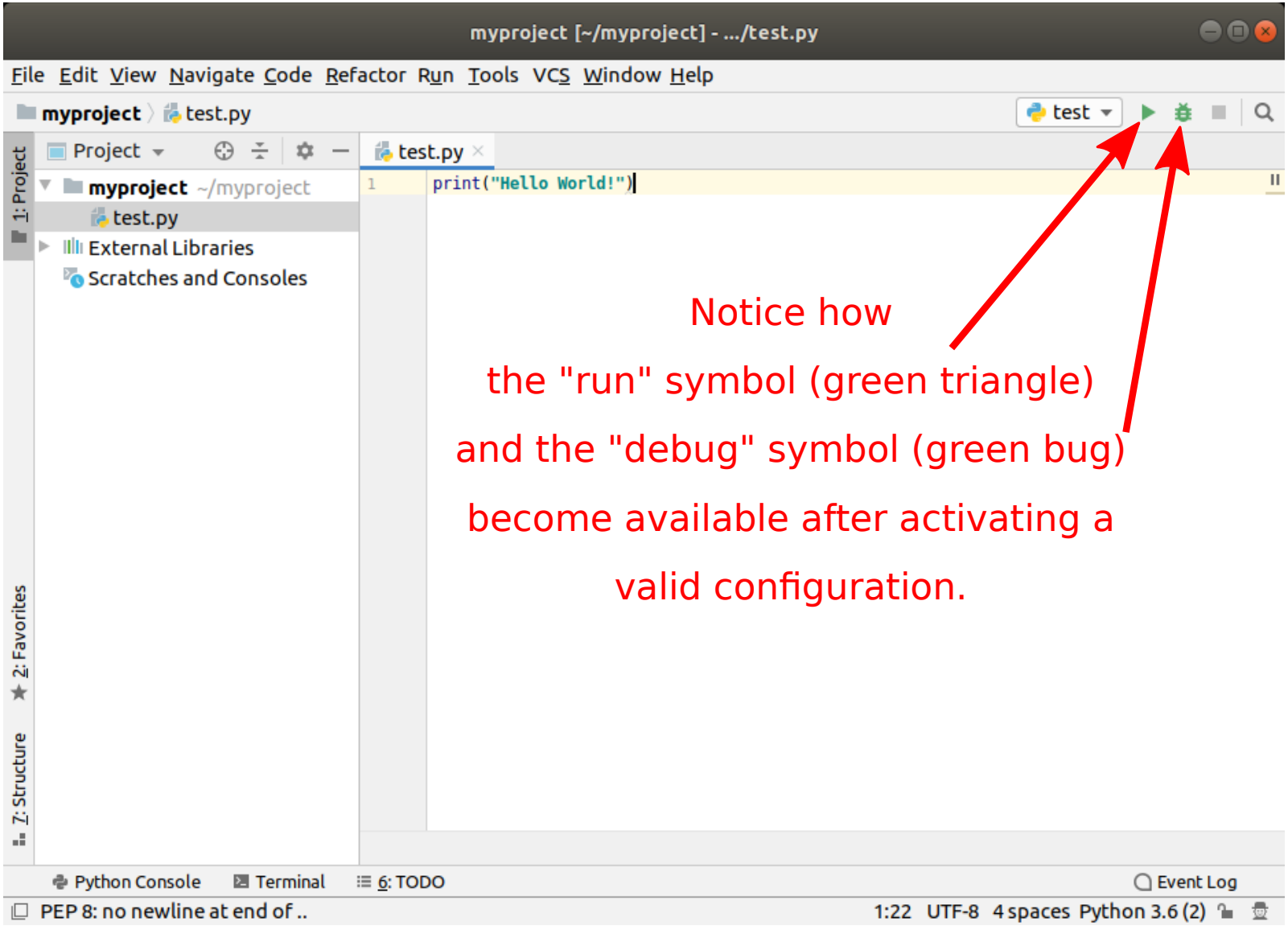


Task 3: Running a Python program (11)



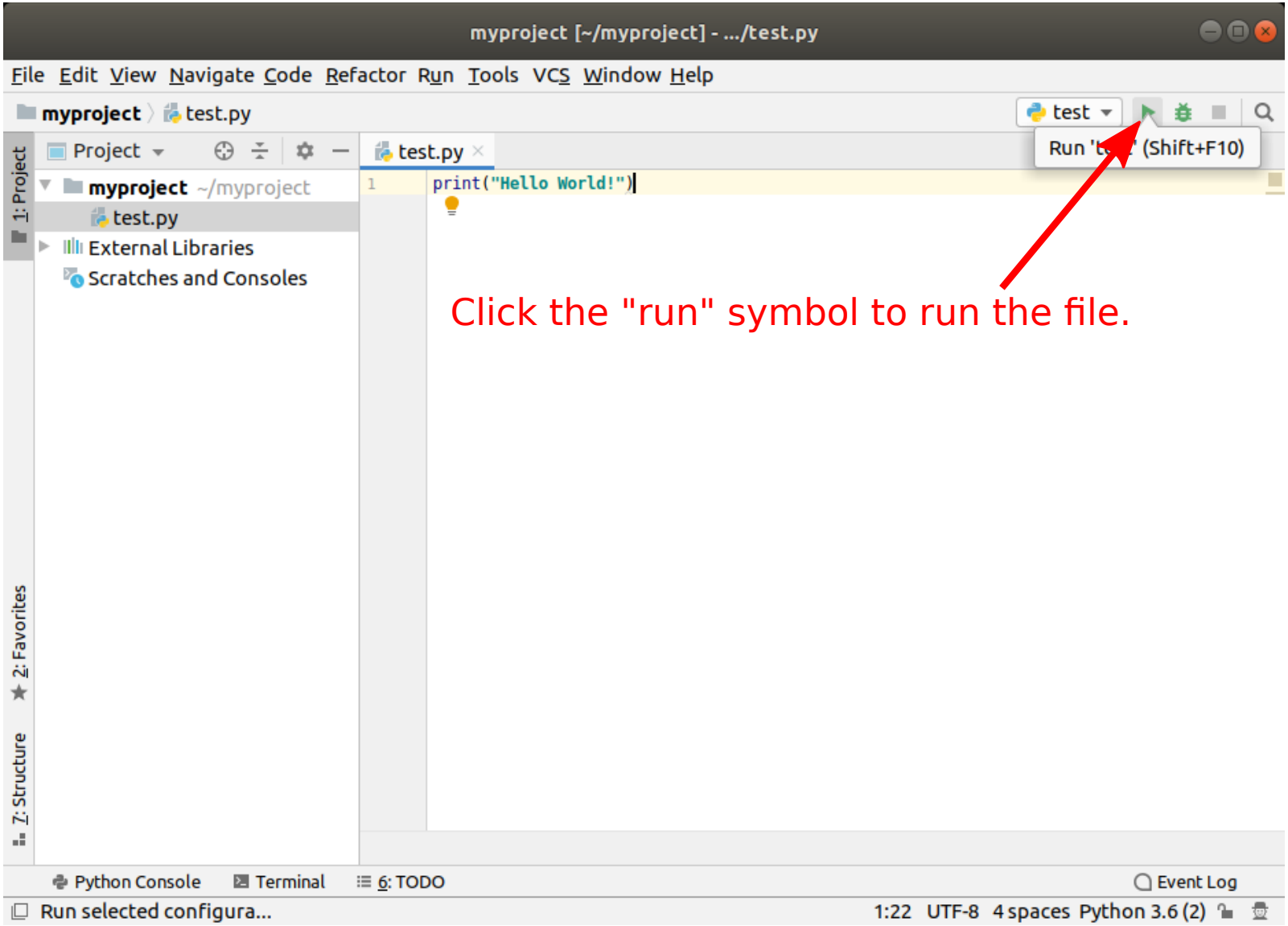
The created configuration should now be selectable in this drop-down menu. You can create multiple configurations per project and/or Python file. These configurations can then be used for debugging or running a file. Now select the configuration you created.

Task 3: Running a Python program (12)

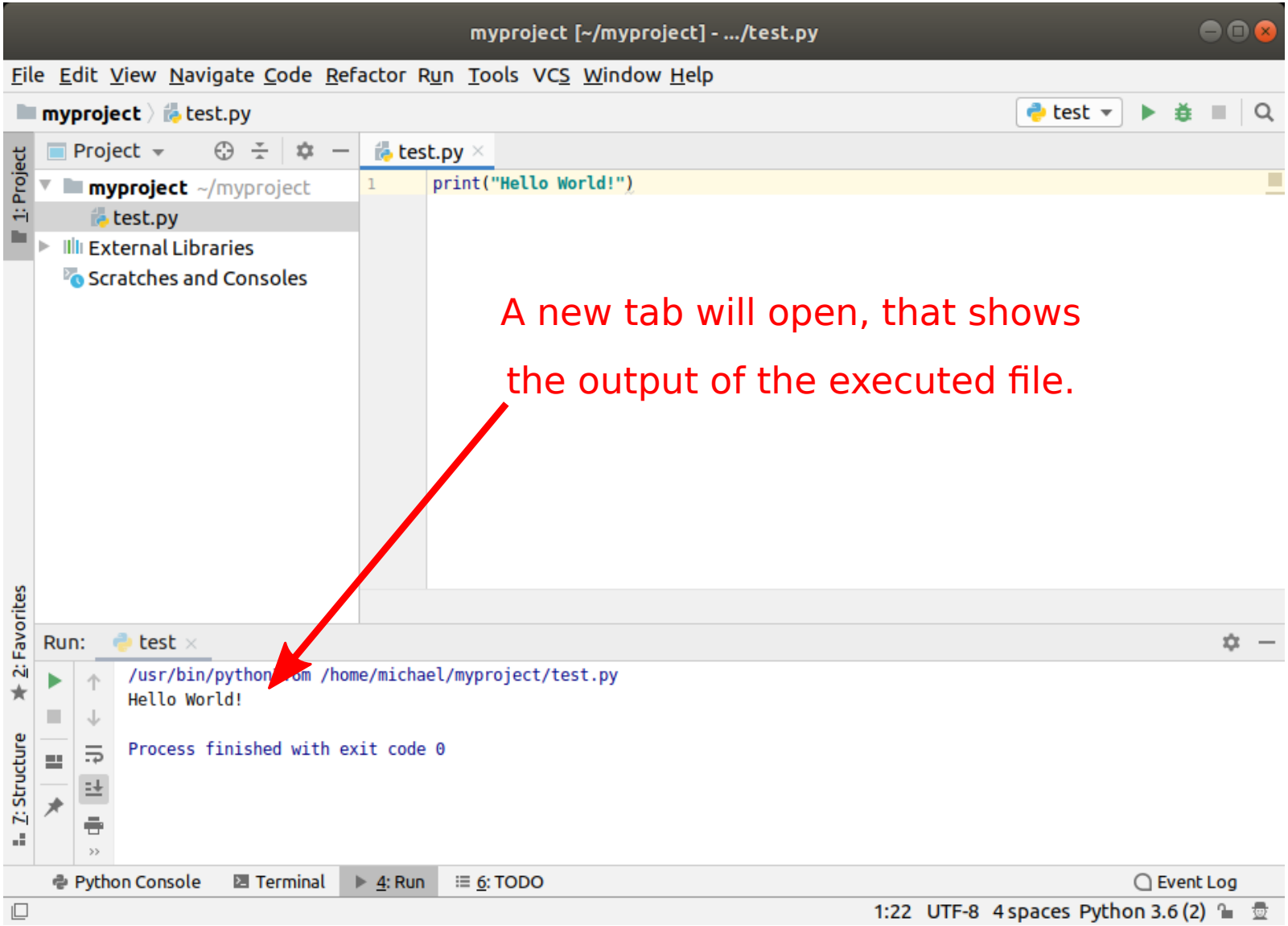


Notice how the "run" symbol (green triangle) and the "debug" symbol (green bug) become available after activating a valid configuration.

Task 3: Running a Python program (13)



Task 3: Running a Python program (14)

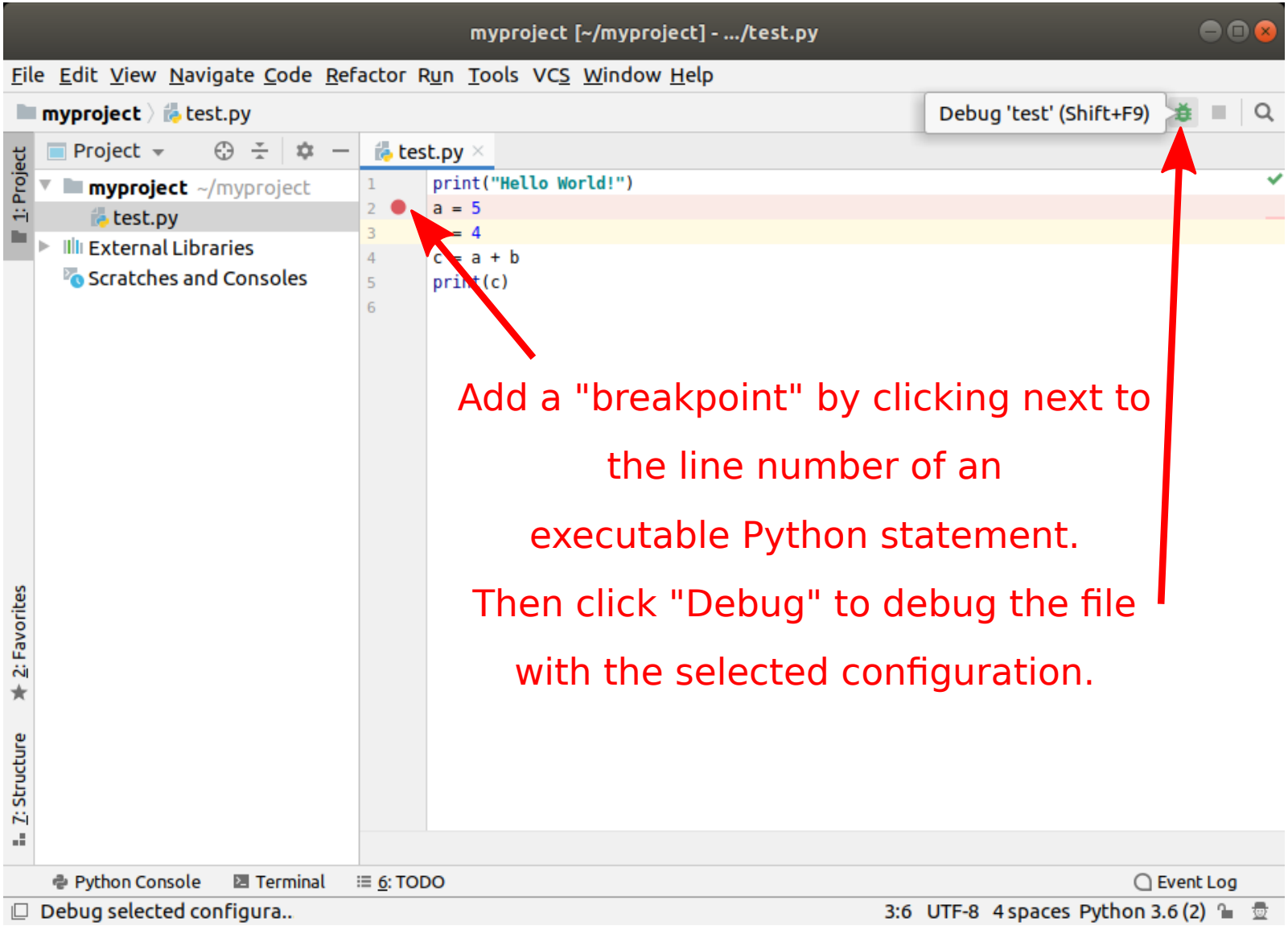


PYCHARM – DEBUGGING A PYTHON PROGRAM

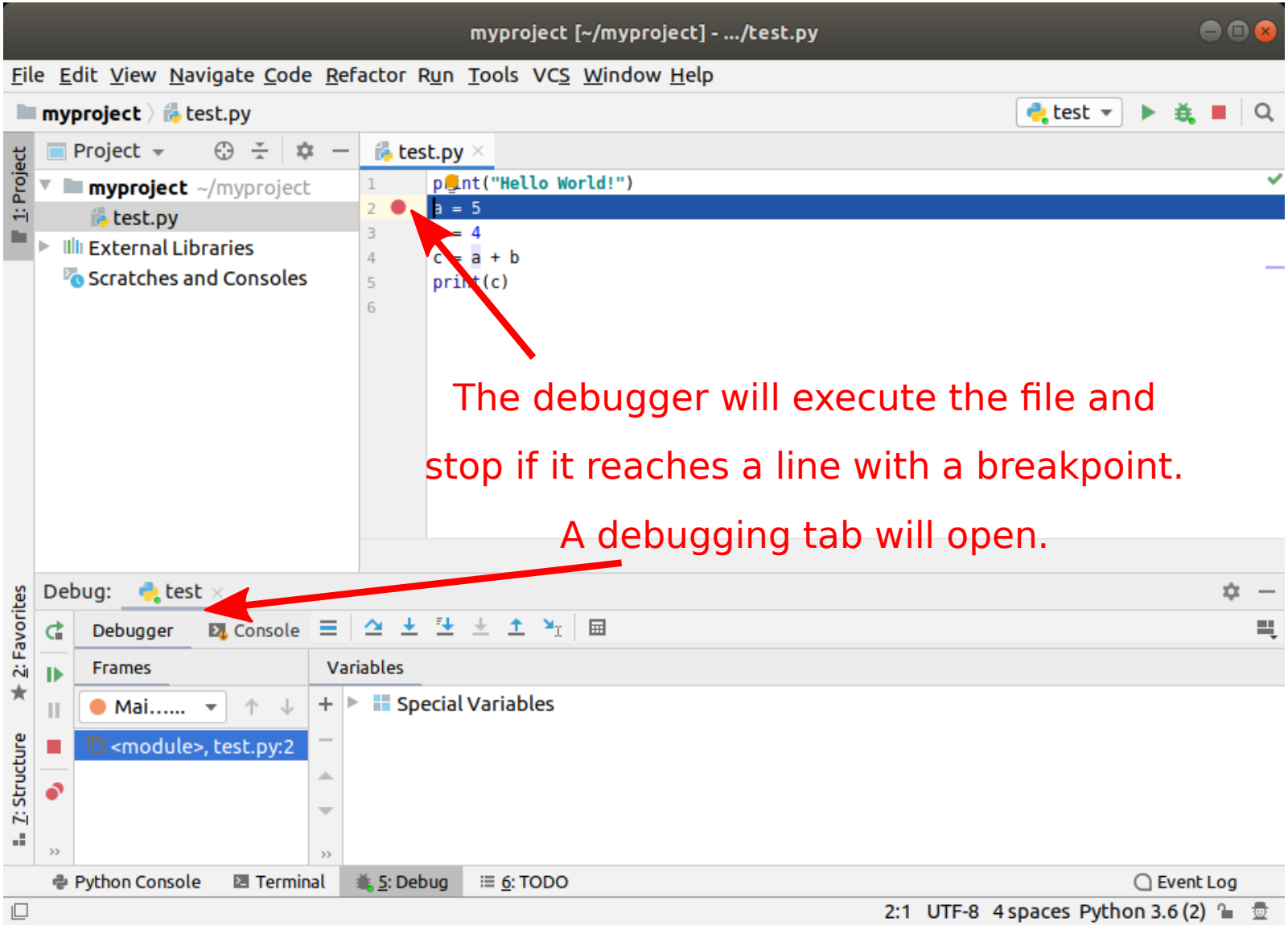
Task 4: Debugging a Python program (1)

- We will now debug a Python program in PyCharm in these steps (see following slides for help):
  1. Select a valid run configuration
  2. Left-click next to the line number in the editor to create a breakpoint
  3. Click the `Debug` symbol (small green bug)
  4. The program should be executed until the breakpoint or the end of the program is reached
  5. Use the `Debugger` tab to inspect or change variables
  6. Use the `Console` tab to inspect the program output
  7. Enter Python code by clicking the `Show Python Prompt` symbol
  8. Execute a single line of the program by clicking on `Step Over` or continue execution via `Resume Program`

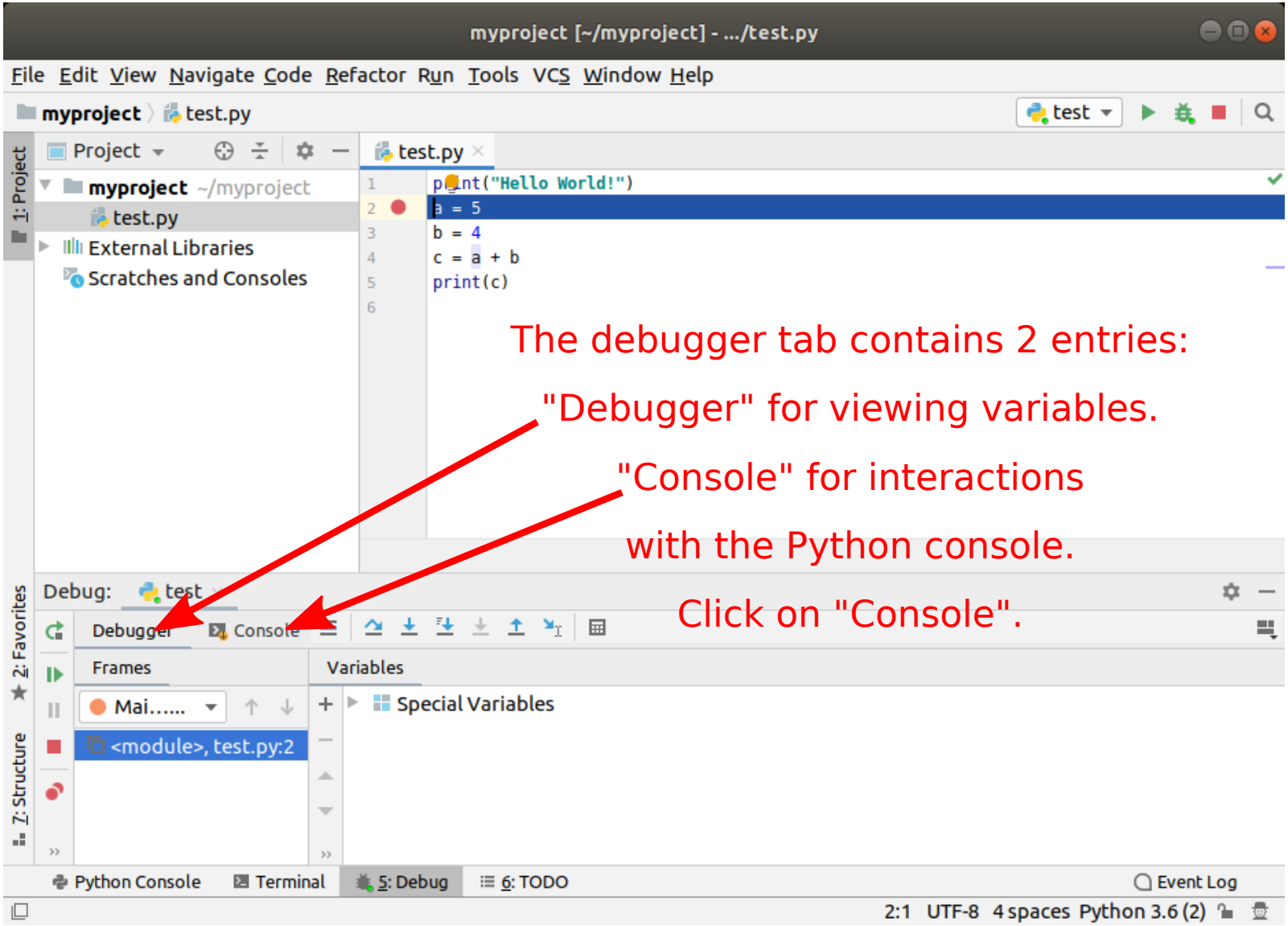
Task 4: Debugging a Python program (3)



Task 4: Debugging a Python program (4)



Task 4: Debugging a Python program (5)



The debugger tab contains 2 entries:

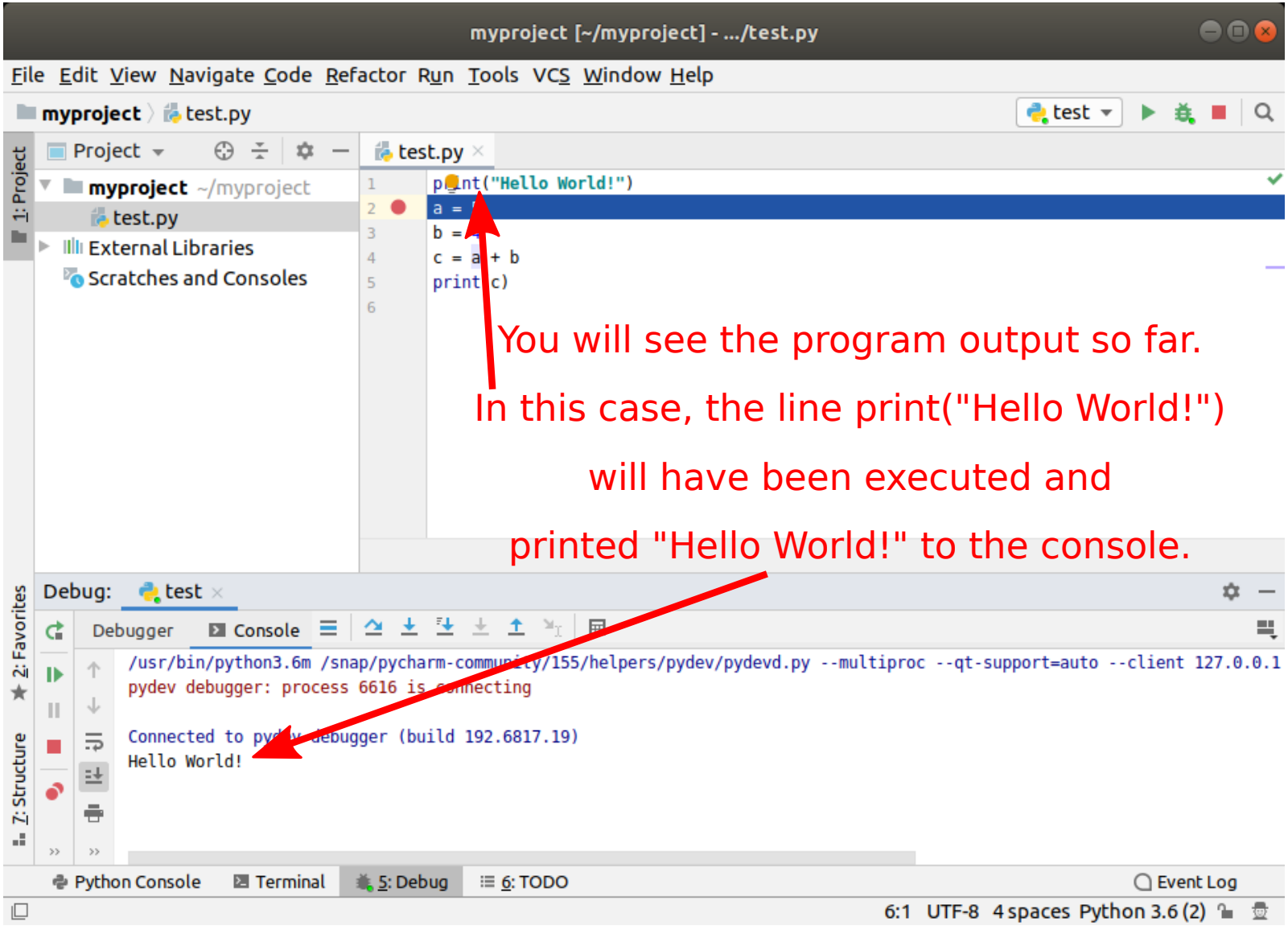
"Debugger" for viewing variables.

"Console" for interactions

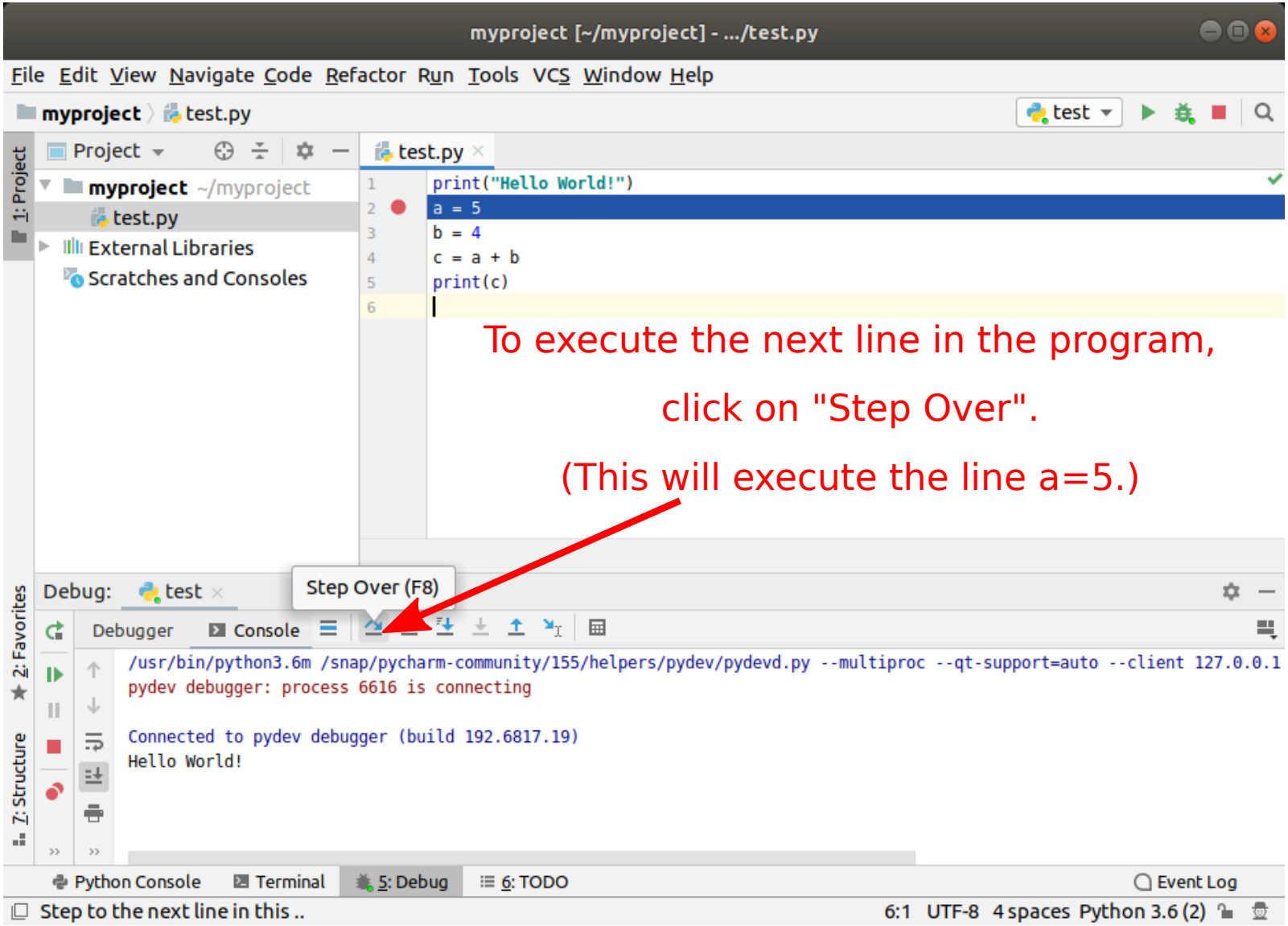
with the Python console.

Click on "Console".

Task 4: Debugging a Python program (6)



Task 4: Debugging a Python program (7)



Task 4: Debugging a Python program (8)

myproject [~/myproject] - .../test.py

File Edit View Navigate Code Refactor Run Tools VCS Window Help

myproject 〉 test.py

```
1  print("Hello World!")
2  a = 5   a: 5
3  b = 4
4  c = a + b
5  print(c)
6
```

Notice how the debugger now stopped at the next line b=4.

Debug: test

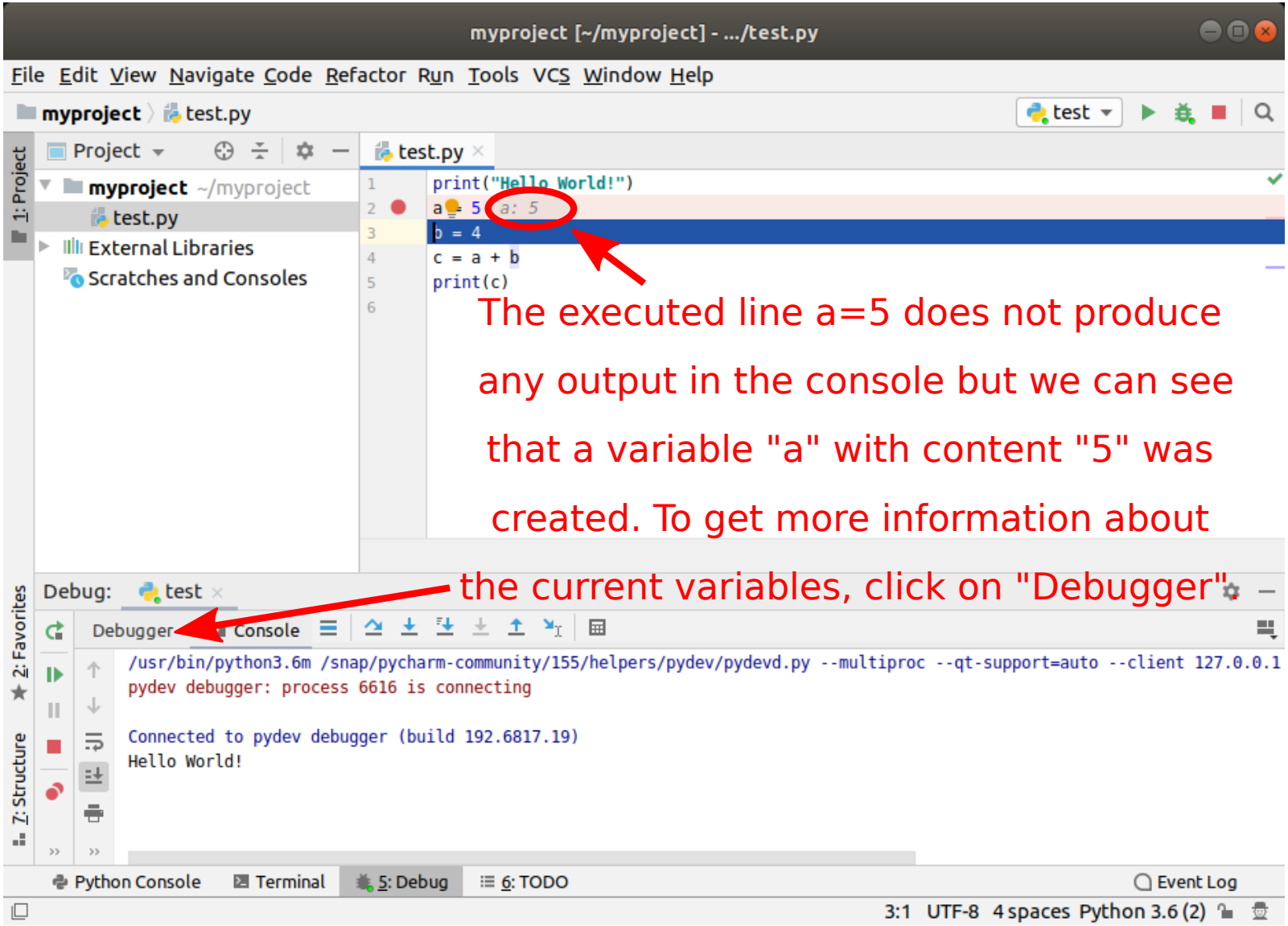
Debugger | Console

```
/usr/bin/python3.6m /snap/pycharm-community/155/helpers/pydev/pydevd.py --multiproc --qt-support=auto --client 127.0.0.1
pydev debugger: process 6616 is connecting

Connected to pydev debugger (build 192.6817.19)
Hello World!
```

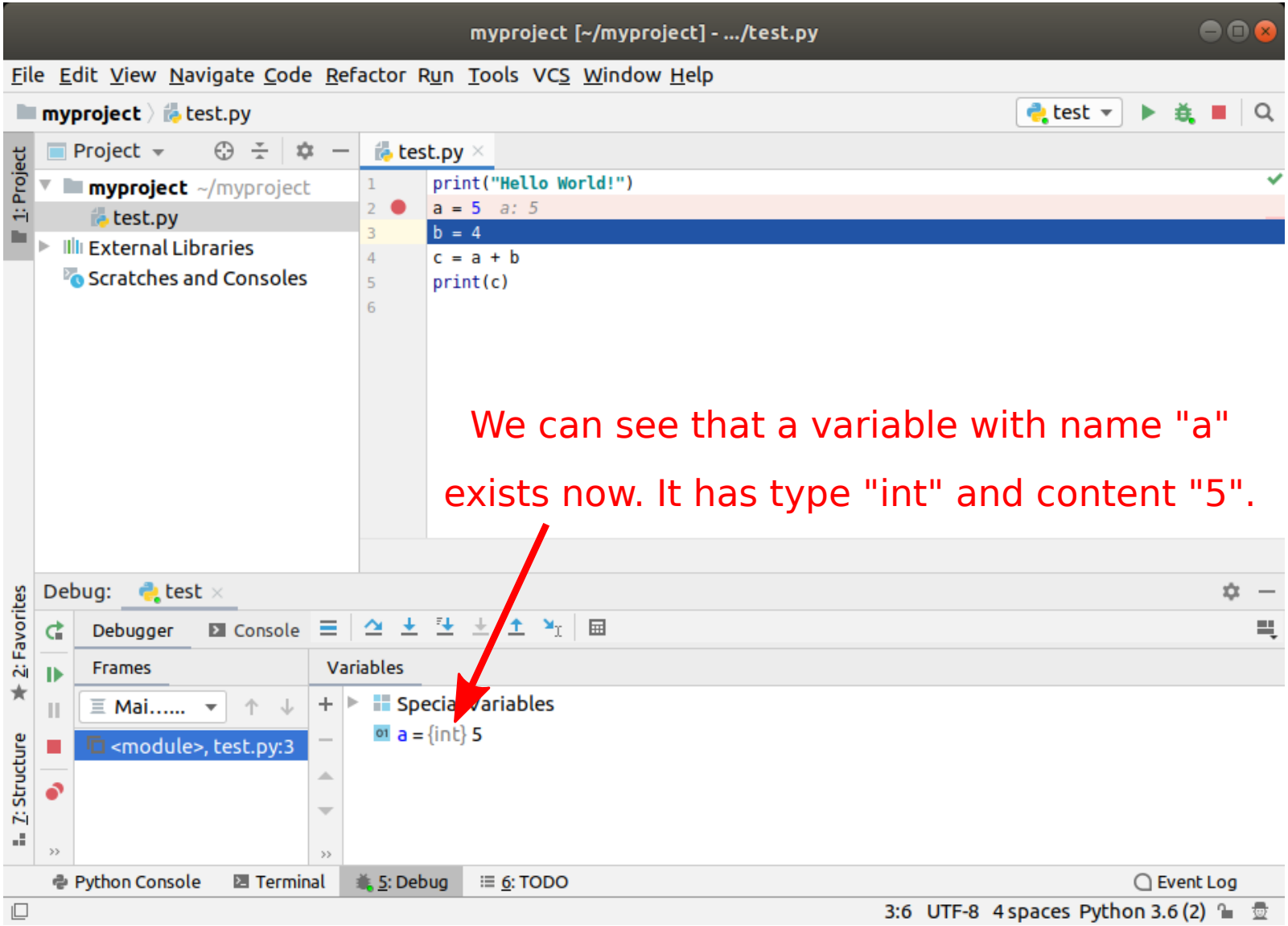
Python Console | Terminal | 5: Debug | 6: TODO | Event Log

3:1 UTF-8 4 spaces Python 3.6 (2)

Task 4: Debugging a Python program (9)

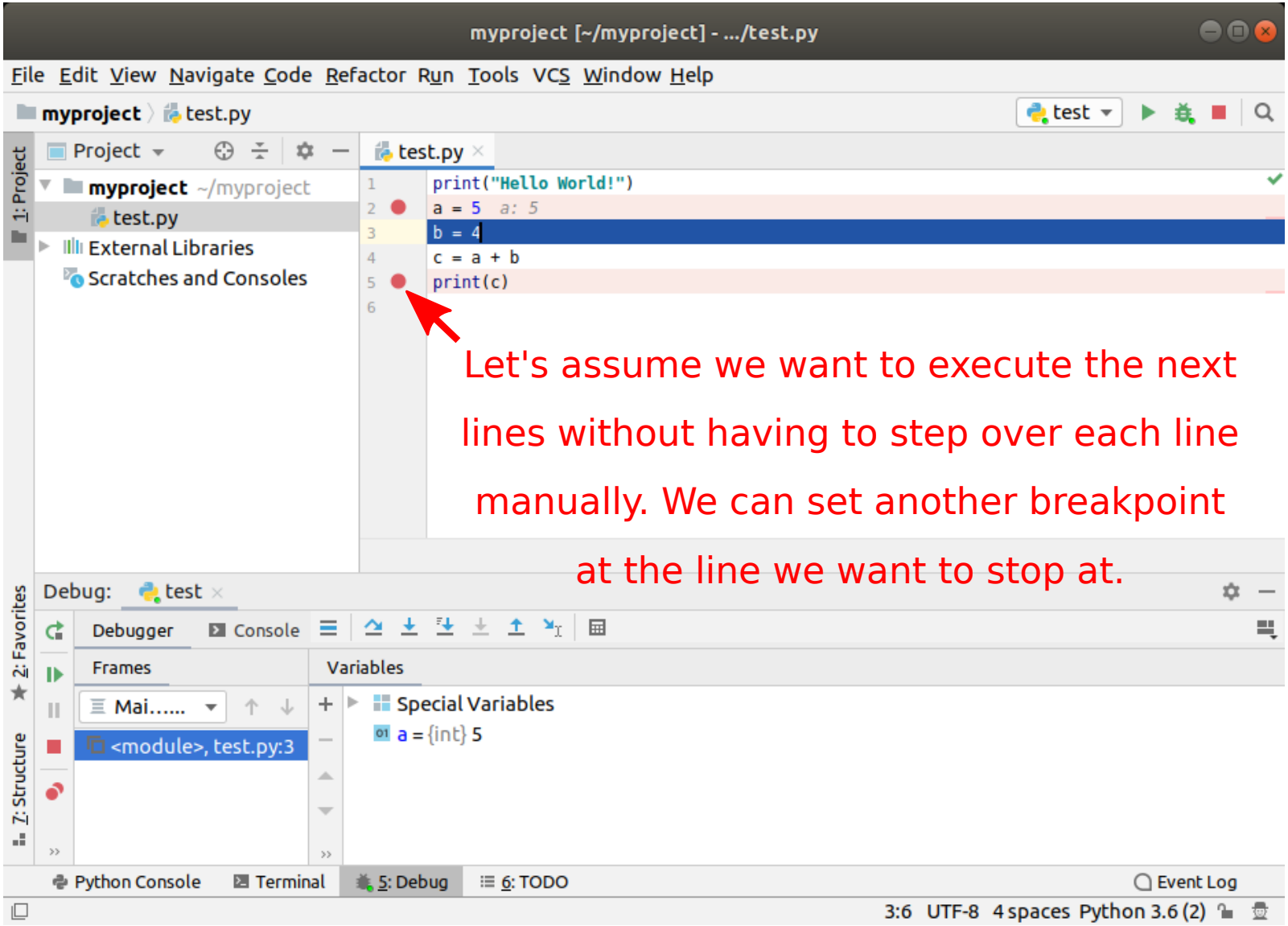


Task 4: Debugging a Python program (10)

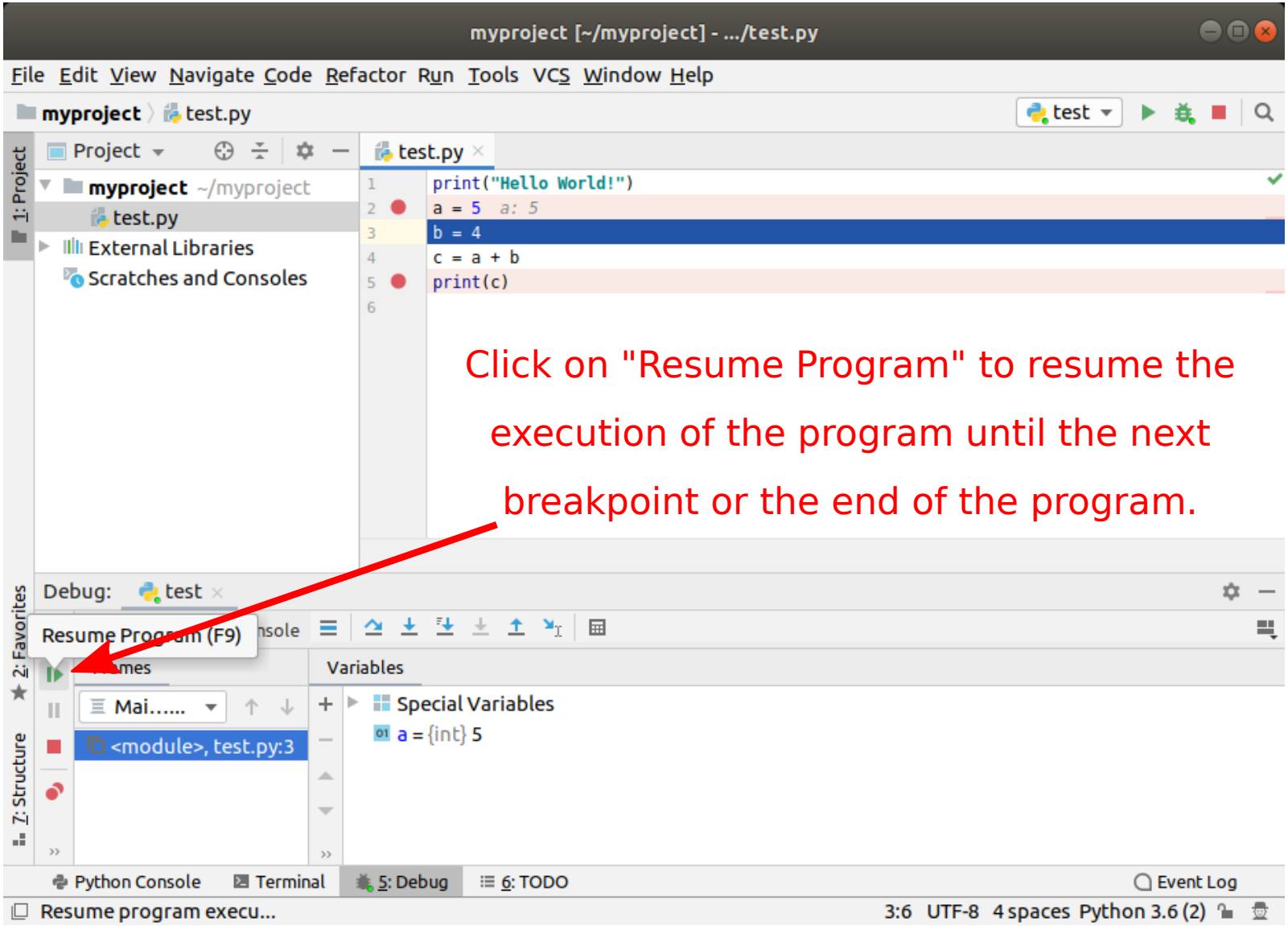


We can see that a variable with name "a" exists now. It has type "int" and content "5".

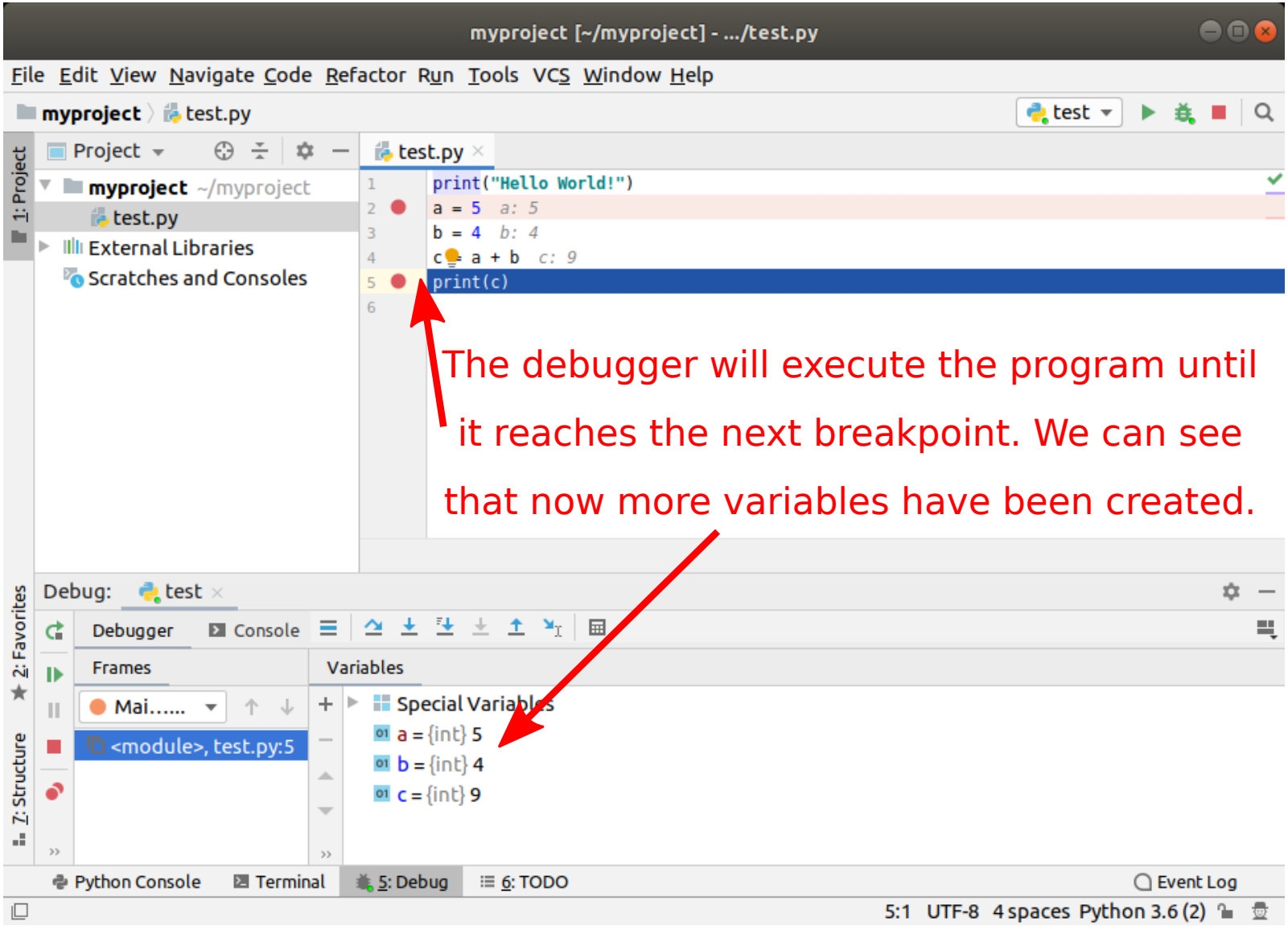
Task 4: Debugging a Python program (11)



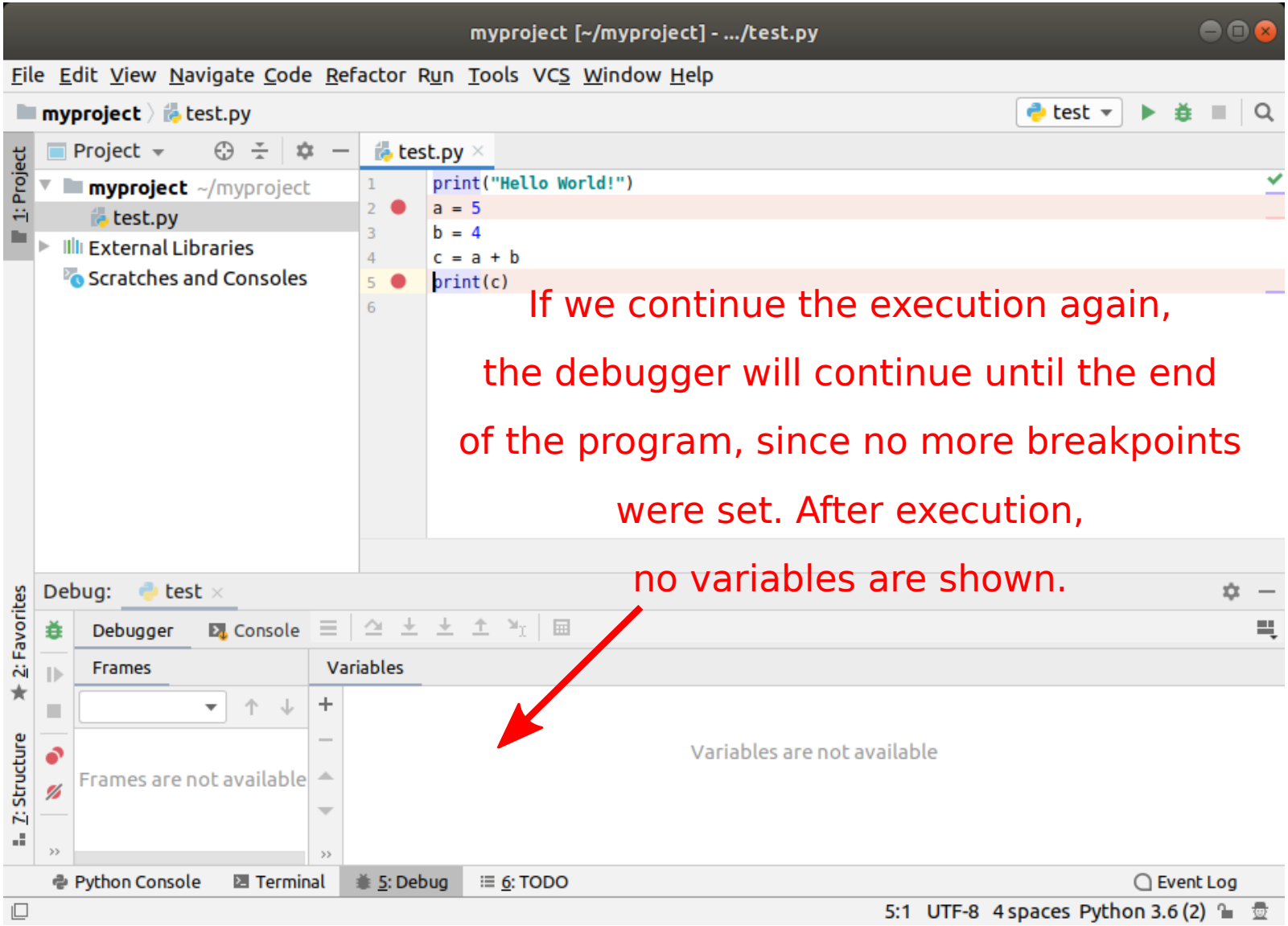
Task 4: Debugging a Python program (12)



Task 4: Debugging a Python program (13)

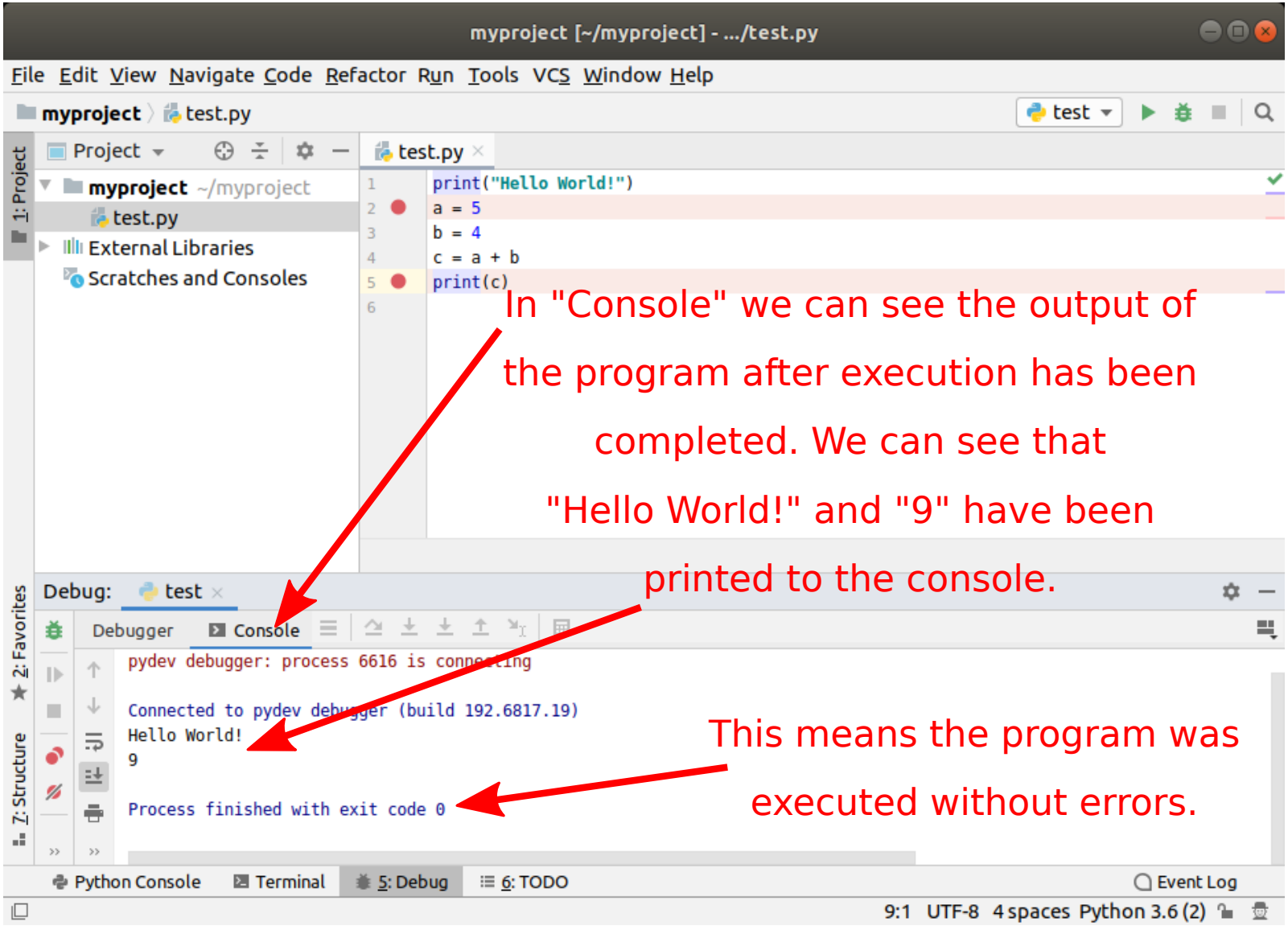


Task 4: Debugging a Python program (14)



If we continue the execution again, the debugger will continue until the end of the program, since no more breakpoints were set. After execution, no variables are shown.

Task 4: Debugging a Python program (15)



In "Console" we can see the output of the program after execution has been completed. We can see that "Hello World!" and "9" have been printed to the console.

This means the program was executed without errors.

Task 4: Debugging a Python program (16)

myproject [~/myproject] - .../test.py

File Edit View Navigate Code Refactor Run Tools VCS Window Help

myproject 〉 test.py

```
1  print("Hello World!")
2  a = 5   a: 5
3  b = 4   b: 4
4  c = a + b
5  print(c)
6
```

Let's start the debugger again and set a breakpoint to stop at line 4.

We will now change the program during debugging.

Debug: test

Debugger Console

Frames: Mai..... <module>, test.py:4

Variables: Special Variables, a = {int} 5, b = {int} 4

Python Console Terminal 5: Debug 6: TODO Event Log

4:1 UTF-8 4 spaces Python 3.6 (2)

Task 4: Debugging a Python program (17)

myproject [~/myproject] - .../test.py

File Edit View Navigate Code Refactor Run Tools VCS Window Help

myproject 〉 test.py

test

Project

myproject ~/myproject

test.py

External Libraries

Scratches and Consoles

```
1  print("Hello World!")
2  a = 5   a: 5
3  b = 4   b: 4
4  c = a + b
5  print(c)
6
```

Click here to enter Python code that should be executed. This is like pretending that the entered code is written at the current position of the original code.

Evaluate Expression (Alt+Shift+8)

Debug: test

Debugger Console

Frames

Mai.....

<module>, test.py:4

Variables

Special Variables

a = {int} 5

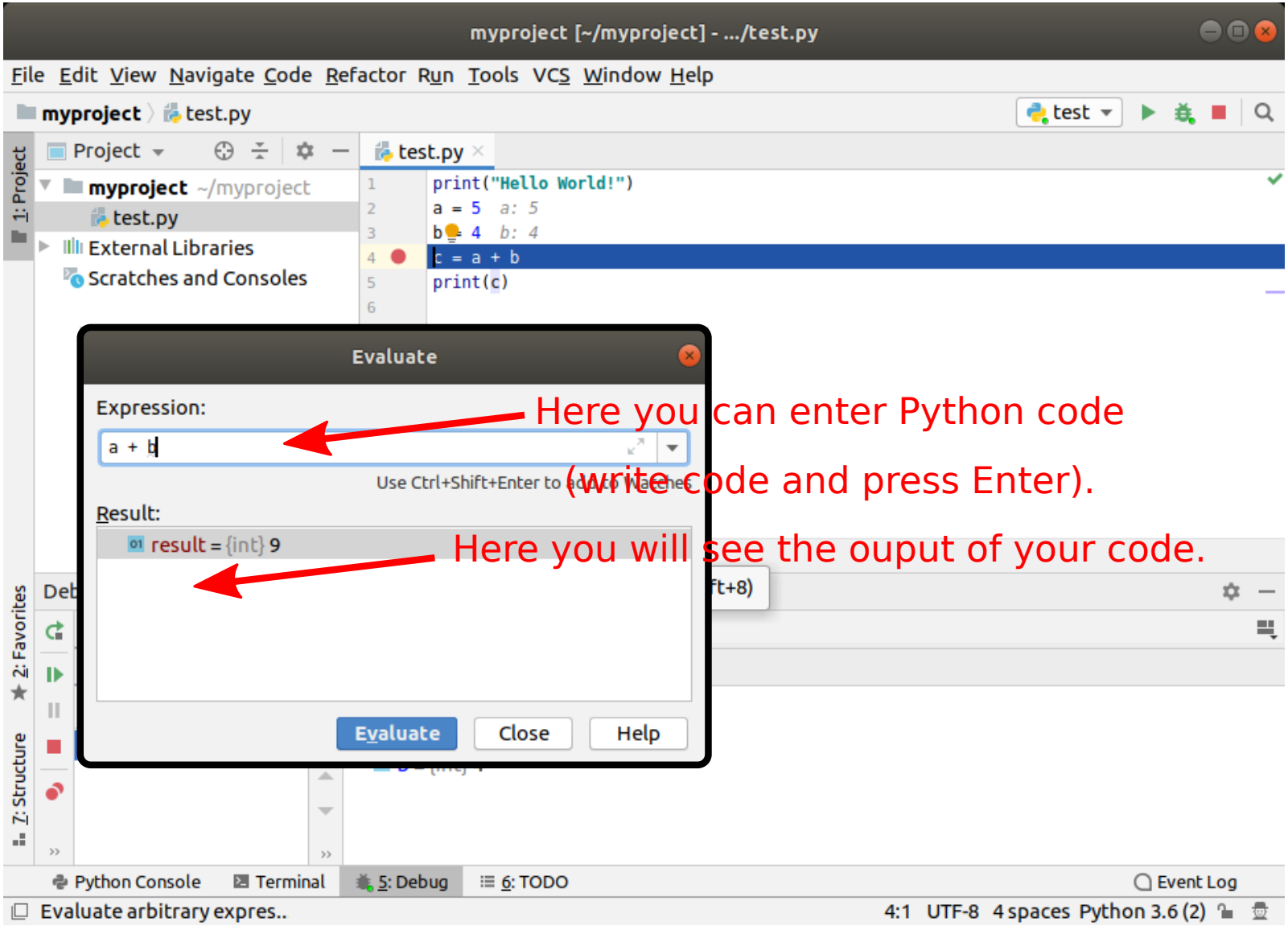
b = {int} 4

Python Console Terminal 5: Debug 6: TODO Event Log

Evaluate arbitrary expres..

4:1 UTF-8 4 spaces Python 3.6 (2)

Task 4: Debugging a Python program (18)



Task 4: Debugging a Python program (19)

myproject [~/myproject] - .../test.py

File Edit View Navigate Code Refactor Run Tools VCS Window Help

myproject 〉 test.py

```
1  print("Hello World!")
2  a = 5   a: 5
3  b = 4   b: 4
4  c = a + b
5  print(c)
6
```

Debug: test

Debugger | Console

```
pydev debugger: process 6695 is connecting

Connected to pydev debugger (build 192.6817.19)
Hello World!
Python 3.6.8 (default, Aug 20 2019, 17:12:48)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.7.0 -- An enhanced Interactive Python. Type '?' for help.
using IPython 7.7.0

Python 3.6.8 (default, Aug 20 2019, 17:12:48)
[GCC 8.3.0] on linux
```

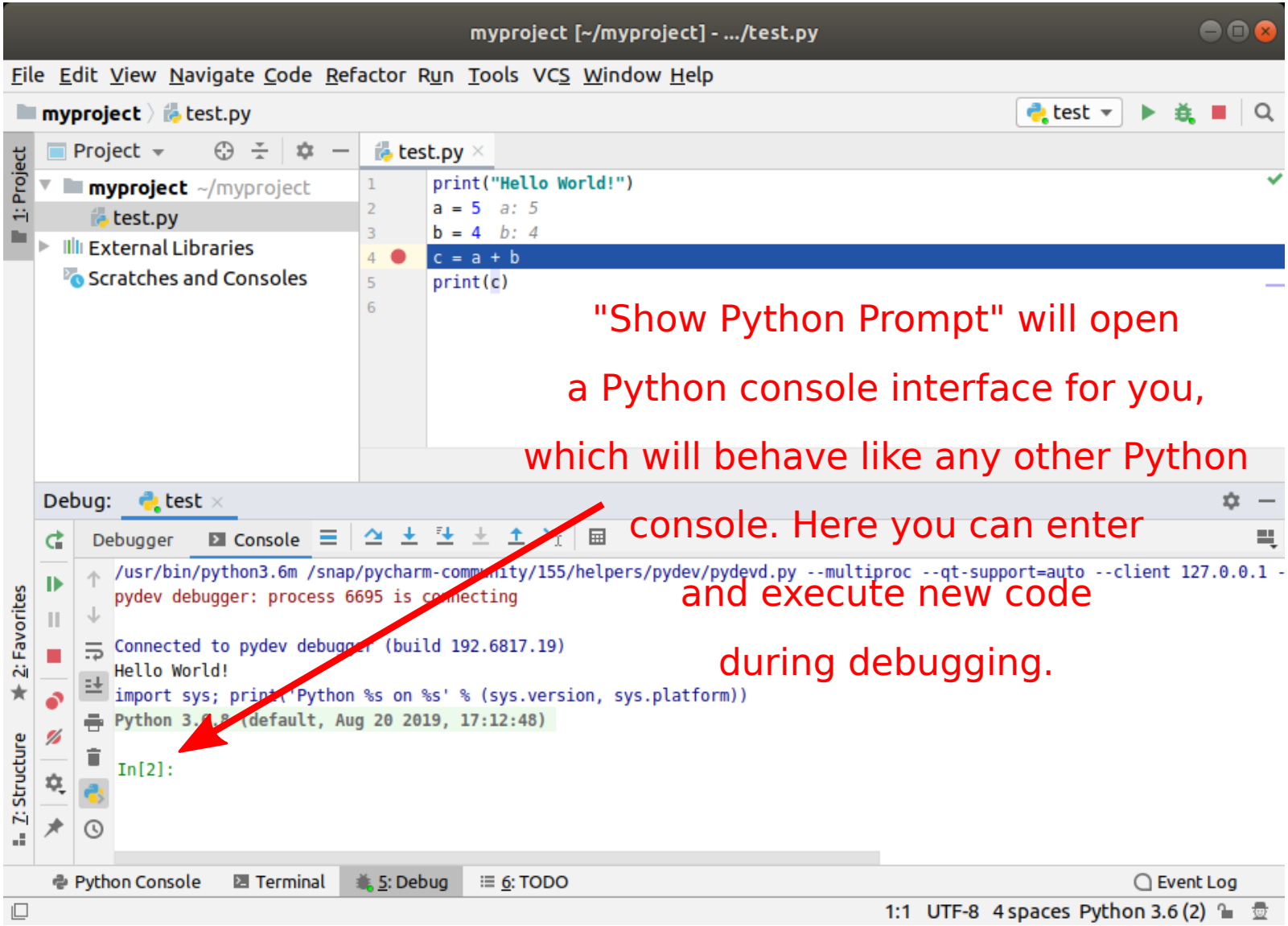
Show Python Prompt

Alternatively, you can enter code in "Console" by clicking on "Show Python Prompt" (might be not directly shown if window is too small)

Python Console | Terminal | 5: Debug | 6: TODO | Event Log

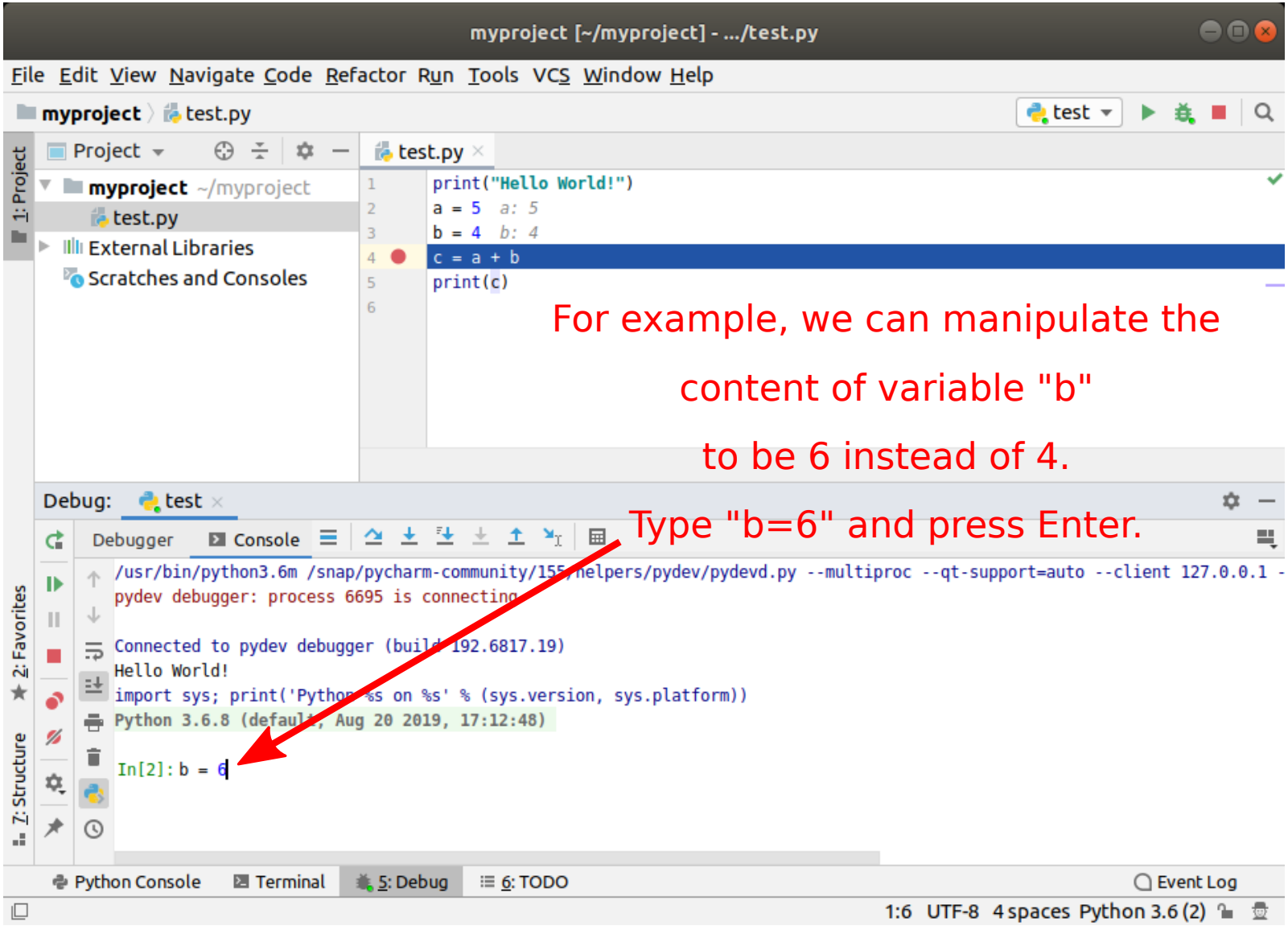
4:1 UTF-8 4 spaces Python 3.6 (2)

Task 4: Debugging a Python program (20)



"Show Python Prompt" will open a Python console interface for you, which will behave like any other Python console. Here you can enter and execute new code during debugging.

Task 4: Debugging a Python program (21)



Task 4: Debugging a Python program (22)

myproject [~/myproject] - .../test.py

File Edit View Navigate Code Refactor Run Tools VCS Window Help

myproject 〉 test.py

```
1  print("Hello World!")
2  a = 5   a: 5
3  b = 4   b: 6
4  c = a + b
5  print(c)
6
```

You should now see that the content of variable "b" changed to 6.

Debug: test

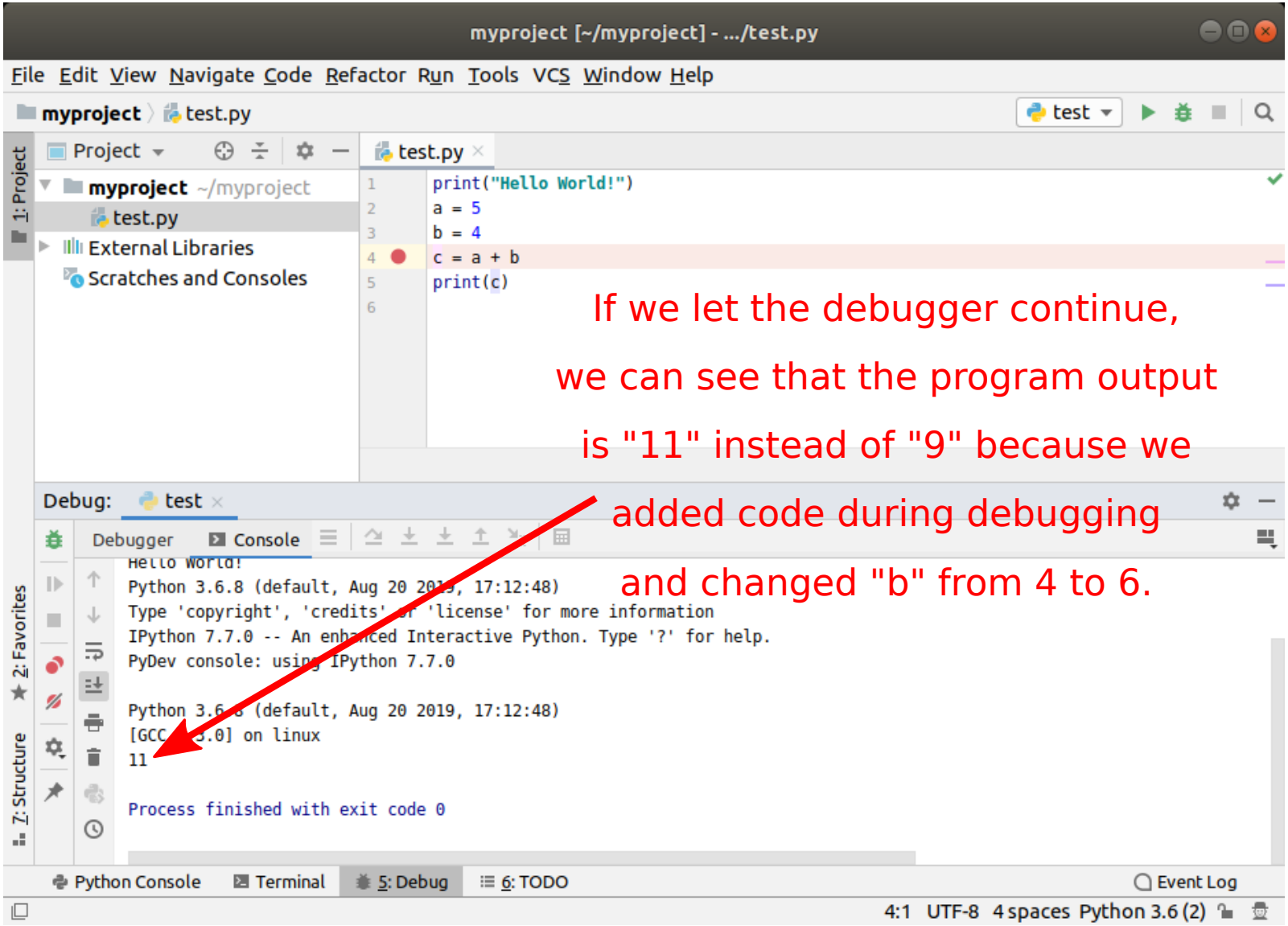
Debugger | Console

```
/usr/bin/python3.6m /snap/pycharm-community/155/helpers/pydev/pydevd.py --multiproc --qt-support=auto --client 127.0.0.1 -
pydev debugger: process 6695 is connecting

Connected to pydev debugger (build 192.6817.19)
Hello World!
import sys; print('Python %s on %s' % (sys.version, sys.platform))
Python 3.6.8 (default, Aug 20 2019, 17:12:48)
In[2]: b = 6

In[3]:
```

Python Console | Terminal | 5: Debug | 6: TODO | Event Log

1:1 UTF-8 4 spaces Python 3.6 (2)

Task 4: Debugging a Python program (23)



If we let the debugger continue, we can see that the program output is "11" instead of "9" because we added code during debugging and changed "b" from 4 to 6.

Now you are set up and ready to code!

OUTLOOK

Outlook

- Next time we will finally dive into Python code and learn about
  - Python syntax, file structure, and variables
  - Basic datatypes and conversions
  - Basic Python operators