PROGRAMMING IN PYTHON I

Editor and Debugger

Andreas Schörgenhumer
Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

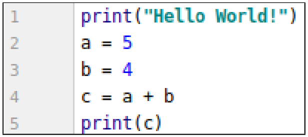
Institute for
Machine Learning

Copyright Statement

EDITOR AND DEBUGGER

Editor

- Comfort in programming has come a long 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**



```
print("Hello World!")
a = 5
b = 4
c = a + b
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 analyze your program while it is executed
- Modern debuggers allow for:
  - Exploring variables during run time
  - 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

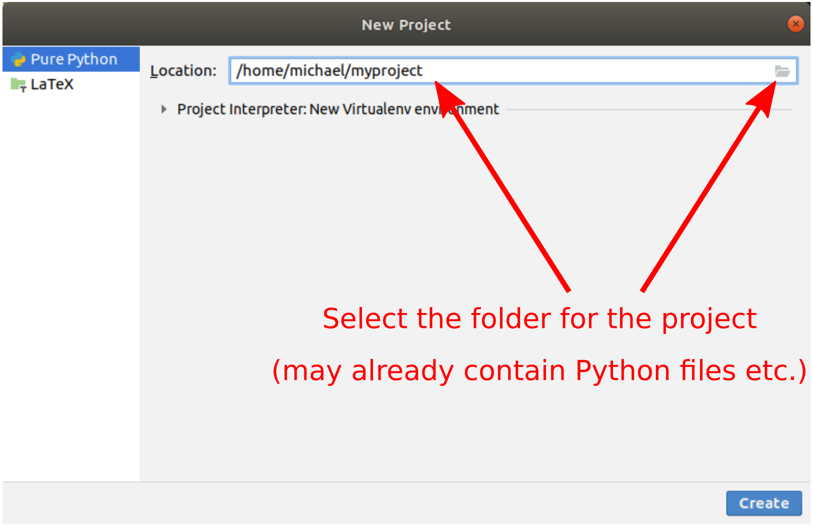
- Install **Pycharm Community Edition** (`https://www.jetbrains.com/pycharm/download/`)
  - Installation is straight-forward
  - Community edition is free and sufficient for this lecture
  - Ubuntu:
    ```
    sudo snap install pycharm-community --classic
    ```
- The next slides will show you how 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)



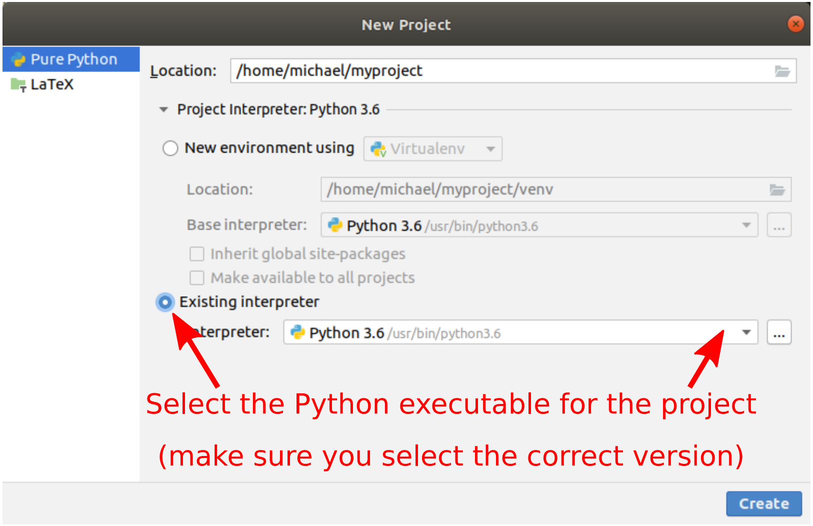
Location: /home/michael/myproject

Project Interpreter: New Virtualenv environment

Select the folder for the project

(may already contain Python files etc.)

Task 1: Create a New PyCharm Project (3)



New Project

Pure Python
LaTeX

Location: /home/michael/myproject

Project Interpreter: Python 3.6

New environment using Virtualenv

Location: /home/michael/myproject/venv

Base interpreter: Python 3.6 /usr/bin/python3.6

Inherit global site-packages

Make available to all projects

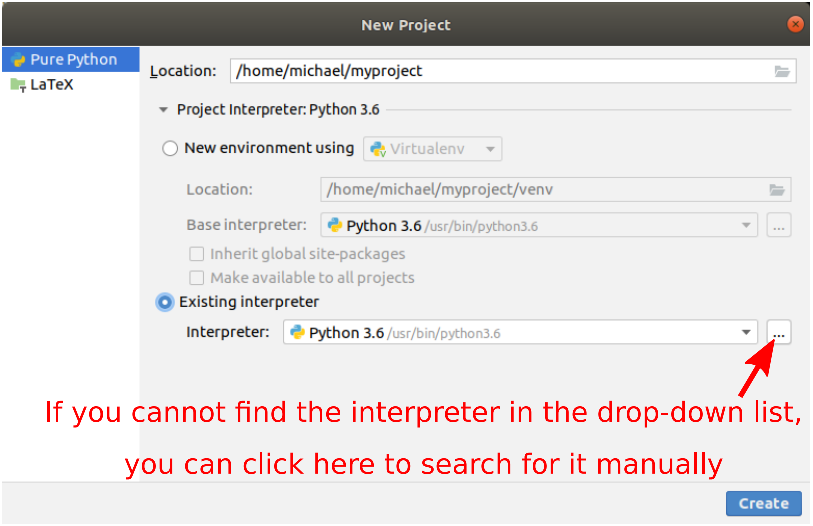
[✓ Existing interpreter]

Interpreter: Python 3.6 /usr/bin/python3.6

Select the Python executable for the project

(make sure you select the correct version)

Task 1: Create a New PyCharm Project (4)



Location: /home/michael/myproject

Project Interpreter: Python 3.6

( ) New environment using Virtualenv

Location: /home/michael/myproject/venv

Base interpreter: Python 3.6 /usr/bin/python3.6

☐ Inherit global site-packages

☐ Make available to all projects

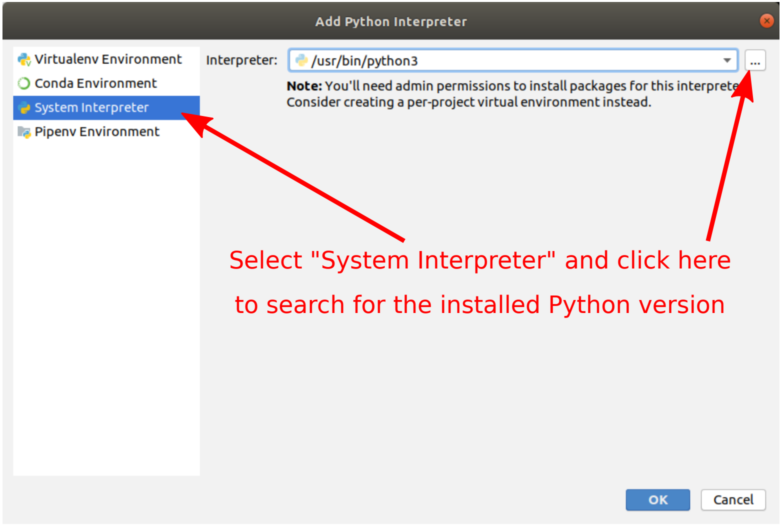
(●) Existing interpreter

Interpreter: Python 3.6 /usr/bin/python3.6

If you cannot find the interpreter in the drop-down list,

you can click here to search for it manually

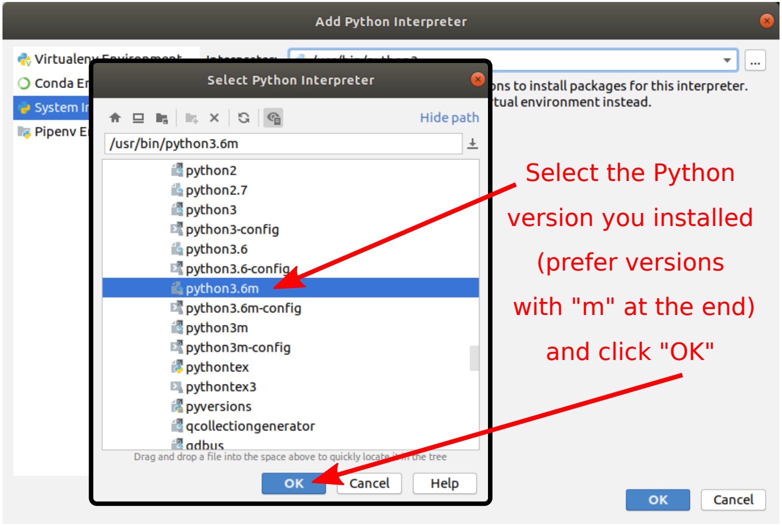
Task 1: Create a New PyCharm Project (5)



Note: You'll need admin permissions to install packages for this interprete
Consider creating a per-project virtual environment instead.

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

Task 1: Create a New PyCharm Project (6)


Add Python Interpreter

Virtualenv Environment
Conda Environment
System Interpreter
Pipenv Environment

Interpreter: /usr/bin/python3

...ons to install packages for this interpreter.
...tual environment instead.

Select Python Interpreter

Hide path

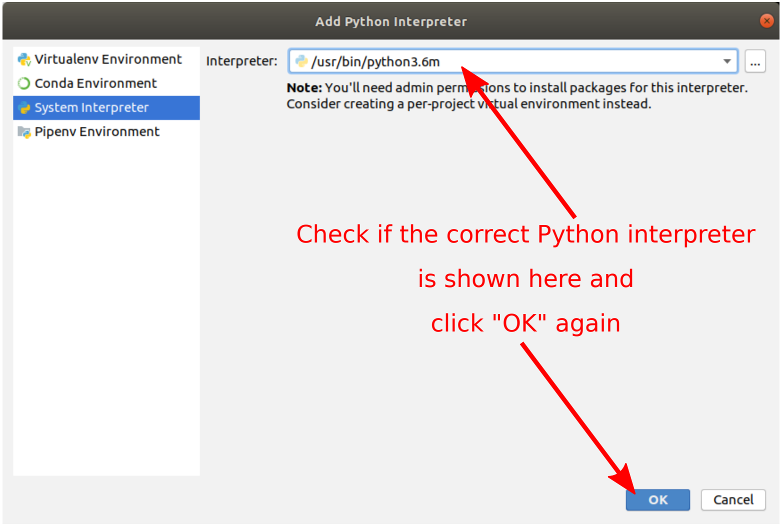
/usr/bin/python3.6m

```
python2
python2.7
python3
python3-config
python3.6
python3.6-config
python3.6m
python3.6m-config
python3m
python3m-config
pythontex
pythontex3
pyversions
qcollectiongenerator
qdbus
```

Drag and drop a file into the space above to quickly locate it in the tree

Select the Python version you installed (prefer versions with "m" at the end) and click "OK"

Task 1: Create a New PyCharm Project (7)

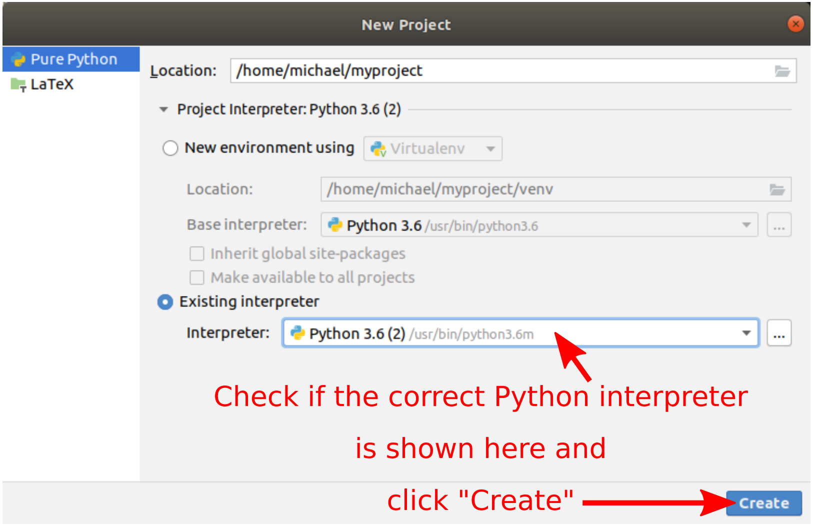


Interpreter: /usr/bin/python3.6m

Note: You'll need admin permissions to install packages for this interpreter. Consider creating a per-project virtual environment instead.

Check if the correct Python interpreter is shown here and click "OK" again

Task 1: Create a New PyCharm Project (8)



Location: /home/michael/myproject

Project Interpreter: Python 3.6 (2)

New environment using Virtualenv

Location: /home/michael/myproject/venv

Base interpreter: Python 3.6 /usr/bin/python3.6

Inherit global site-packages

Make available to all projects

Existing interpreter

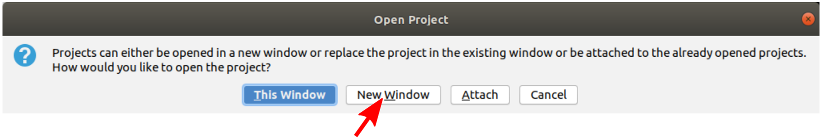
Interpreter: Python 3.6 (2) /usr/bin/python3.6m

Check if the correct Python interpreter

is shown here and

click "Create"

Task 1: Create a New PyCharm Project (9)


Open Project

Projects can either be opened in a new window or replace the project in the existing window or be attached to the already opened projects.
How would you like to open the project?

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

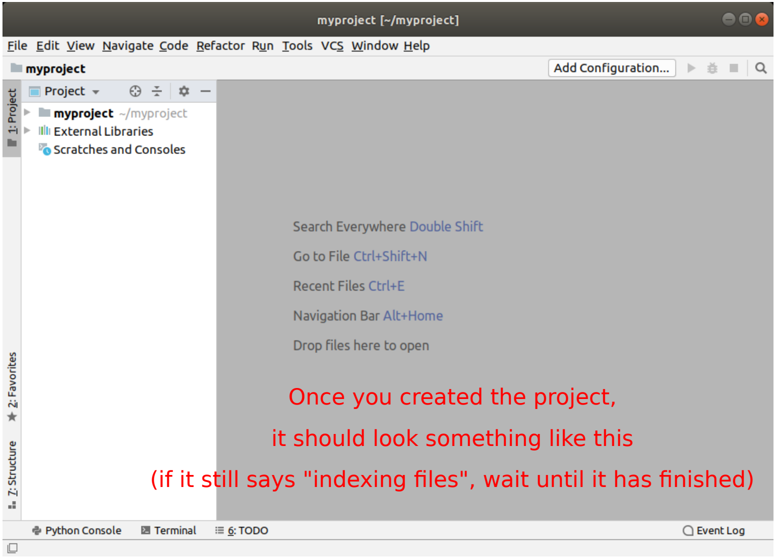
(might take a few seconds)

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 `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)

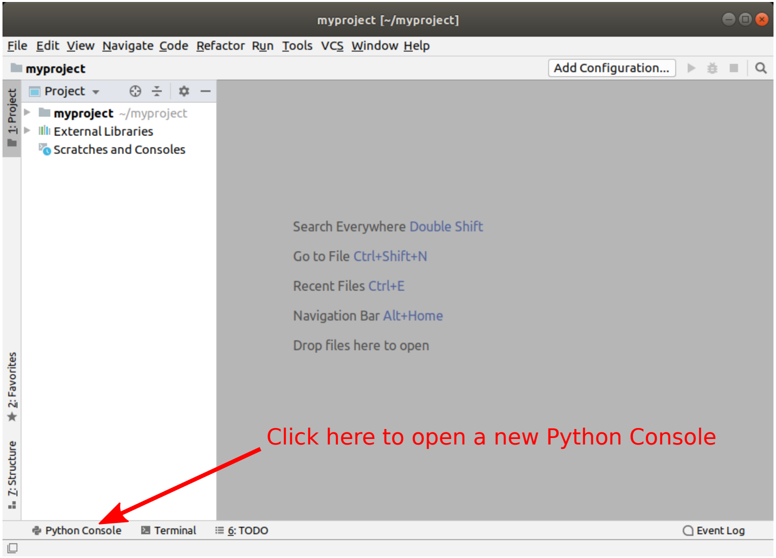


Once you created the project,

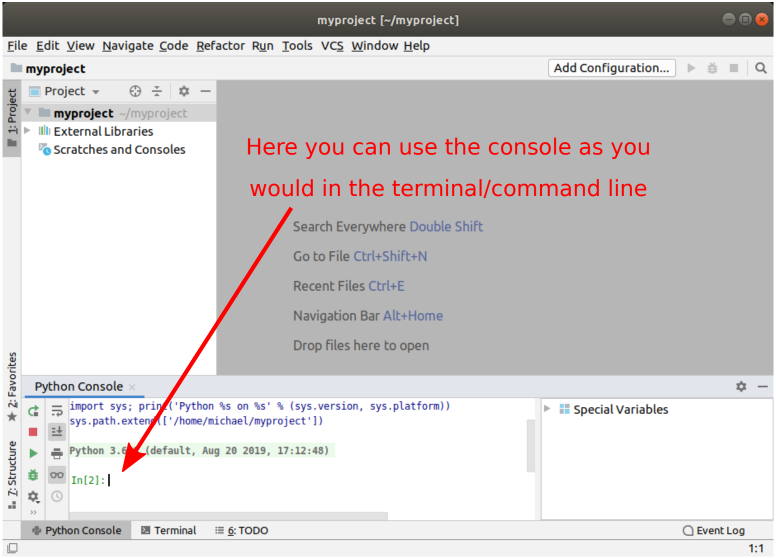
it should look something like this

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

Task 2: Use the Project Python Console (3)

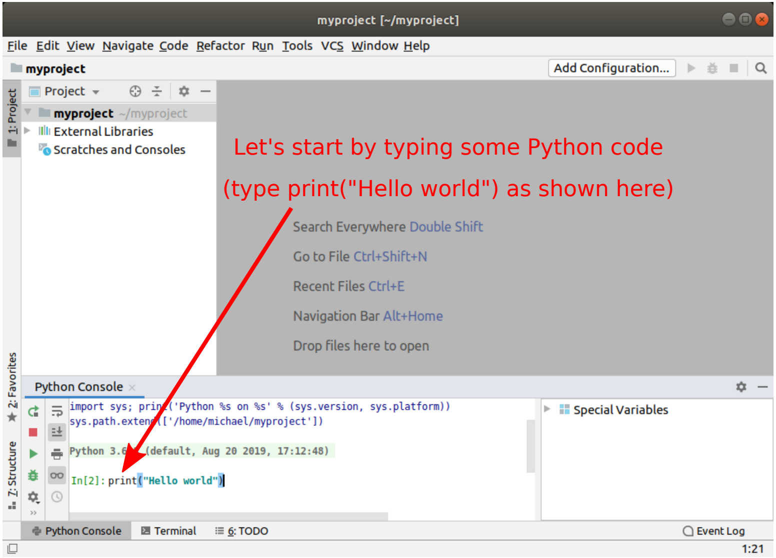


Task 2: Use the Project Python Console (4)



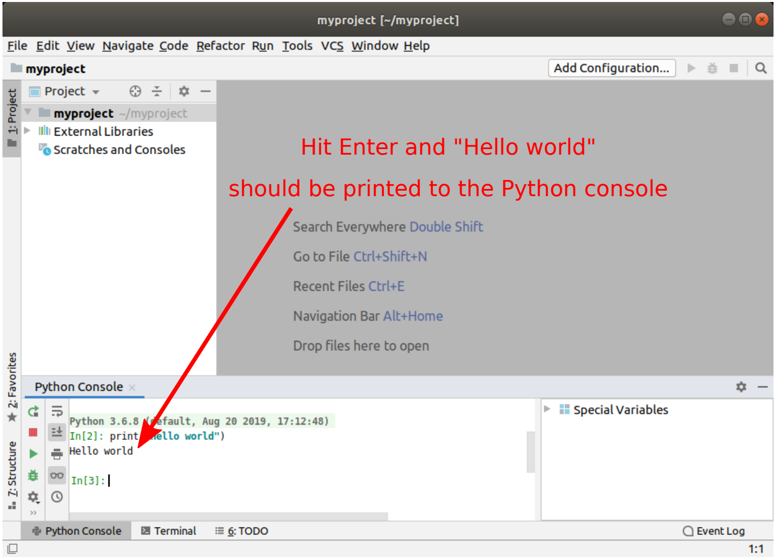
```
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/home/michael/myproject'])

Python 3.6.9 (default, Aug 20 2019, 17:12:48)

In[2]:
```

Task 2: Use the Project Python Console (5)



```
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/home/michael/myproject'])

Python 3.6.9 (default, Aug 20 2019, 17:12:48)

In[2]: print("Hello world")
```

Task 2: Use the Project Python Console (6)

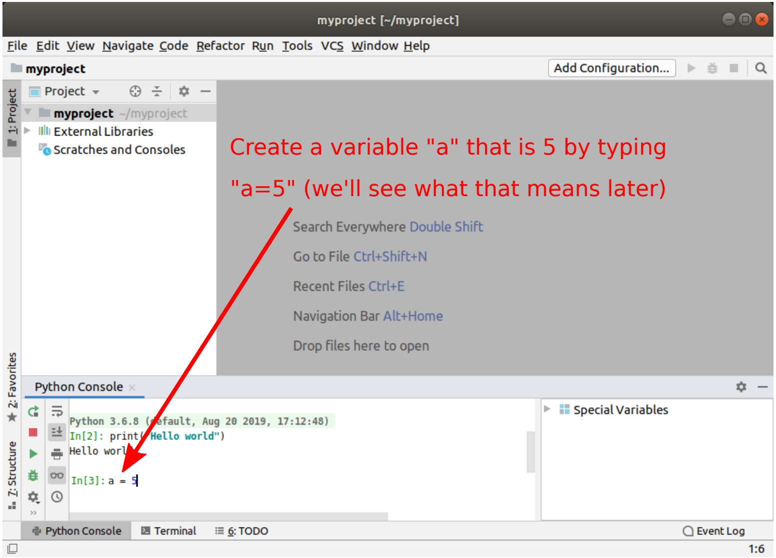


Hit Enter and "Hello world"

should be printed to the Python console

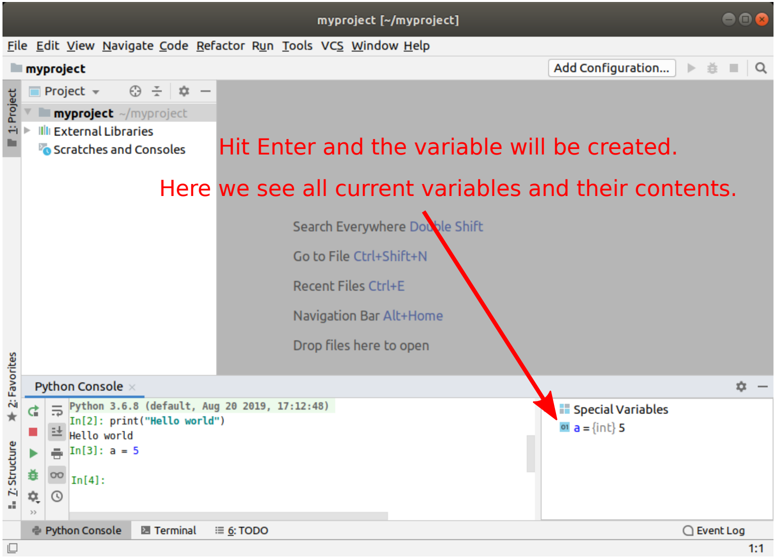
```
Python 3.6.8 (default, Aug 20 2019, 17:12:48)
In[2]: print("Hello world")
Hello world

In[3]:
```

Task 2: Use the Project Python Console (7)



```
Python 3.6.8 (default, Aug 20 2019, 17:12:48)
In[2]: print("Hello world")
Hello world

In[3]: a = 5
```

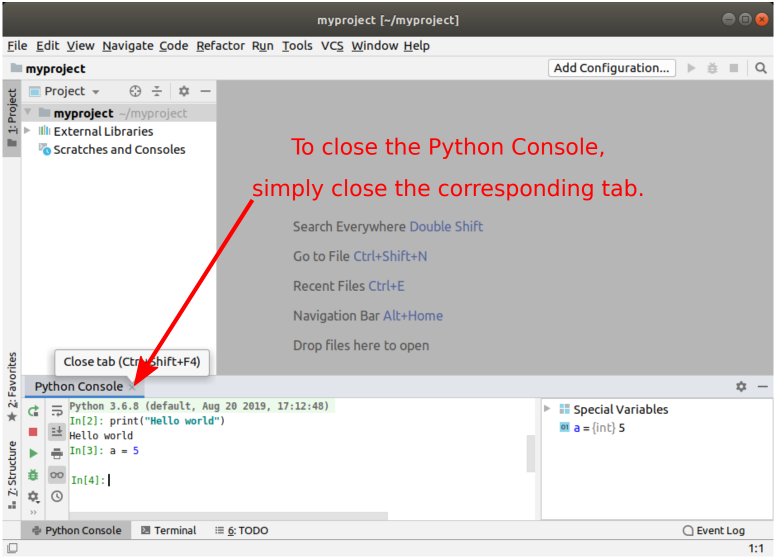
Task 2: Use the Project Python Console (8)



```
Python 3.6.8 (default, Aug 20 2019, 17:12:48)
In[2]: print("Hello world")
Hello world
In[3]: a = 5

In[4]:
```

Special Variables
a = {int} 5

Task 2: Use the Project Python Console (9)



To close the Python Console, simply close the corresponding tab.

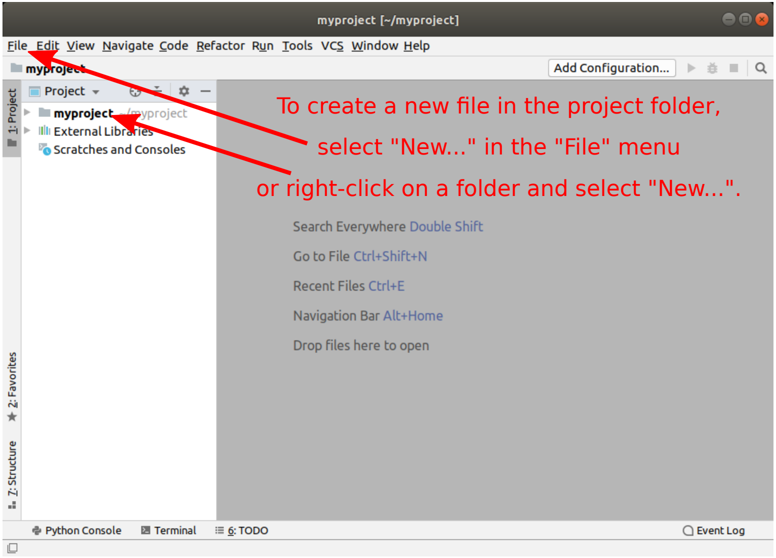
```
Python 3.6.8 (default, Aug 20 2019, 17:12:48)
In[2]: print("Hello world")
Hello world
In[3]: a = 5

In[4]:
```

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)



To create a new file in the project folder, select "New..." in the "File" menu or right-click on a folder and select "New...".

Search Everywhere Double Shift

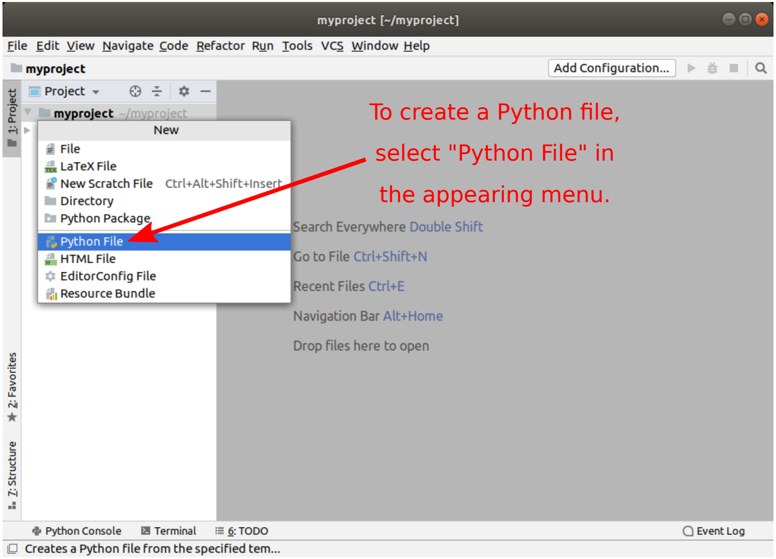
Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

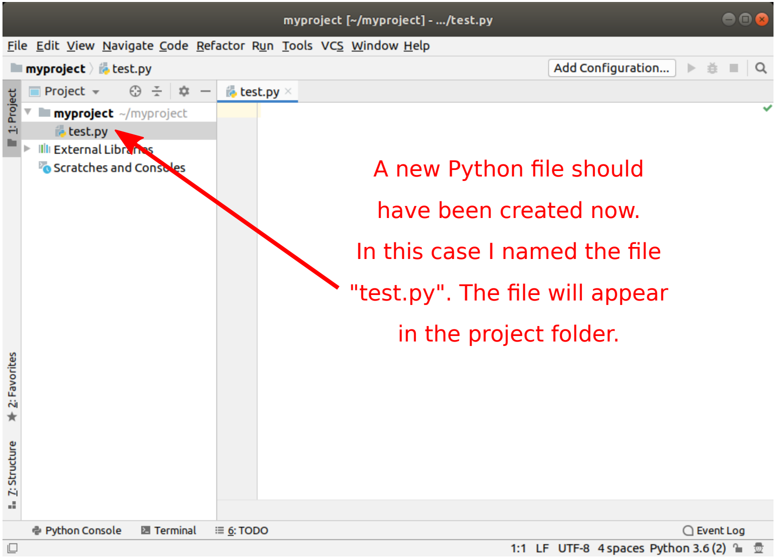
Task 3: Running a Python Program (3)



To create a Python file, select "Python File" in the appearing menu.

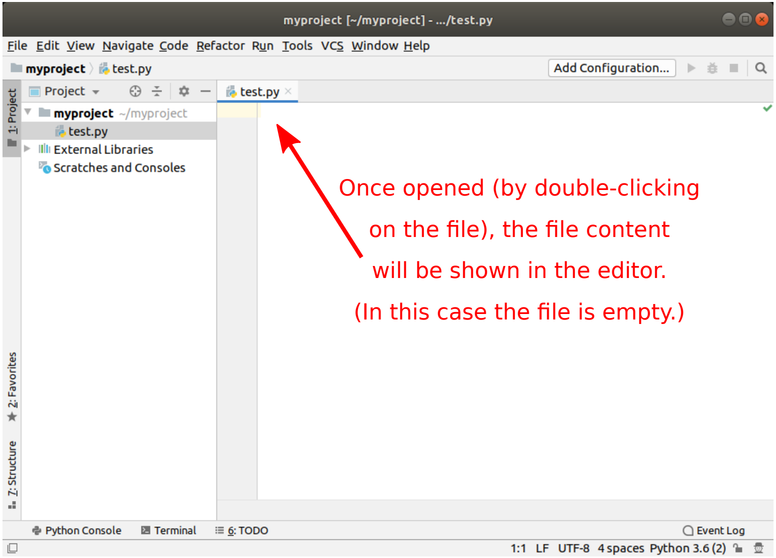
Creates a Python file from the specified tem...

Task 3: Running a Python Program (4)



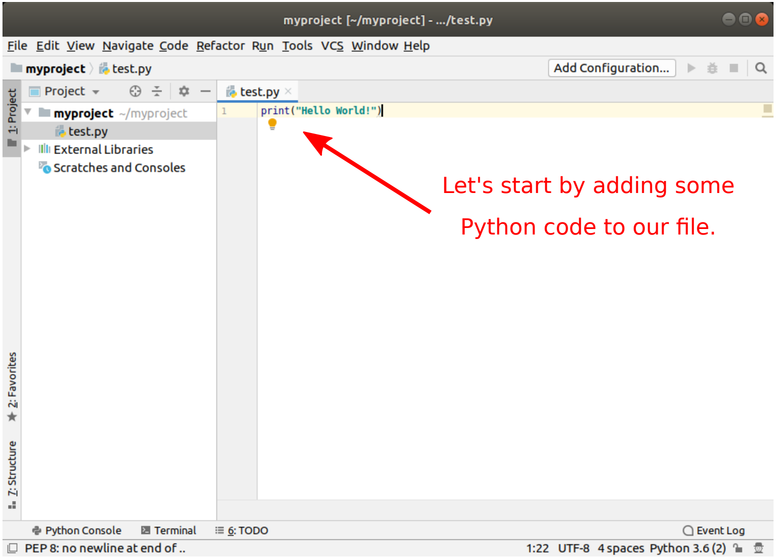
A new Python file should have been created now. In this case I named the file "test.py". The file will appear in the project folder.

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)



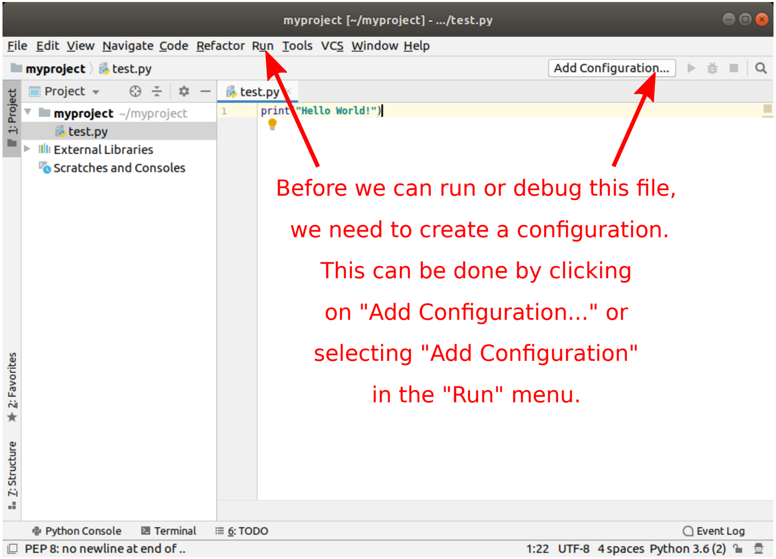
```
print("Hello World!")
```

PEP 8: no newline at end of ..

1:22 UTF-8 4 spaces Python 3.6 (2)

Let's start by adding some Python code to our file.

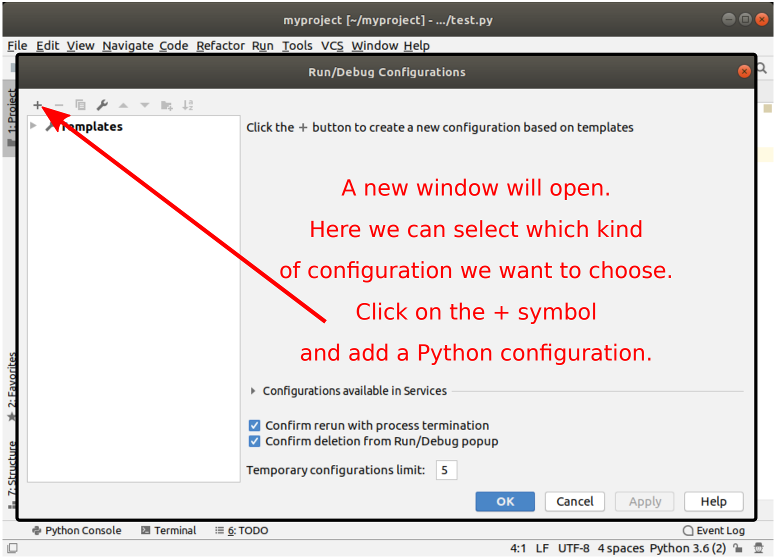
Task 3: Running a Python Program (7)



```
print("Hello World!")
```

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)


myproject [~/myproject] - .../test.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Run/Debug Configurations
Templates
1: Project
2: Favorites
7: Structure
OK
Cancel
Apply
Help
Python Console
Terminal
6: TODO
Event Log
4:1 LF UTF-8 4 spaces Python 3.6 (2)

Click the + button to create a new configuration based on templates

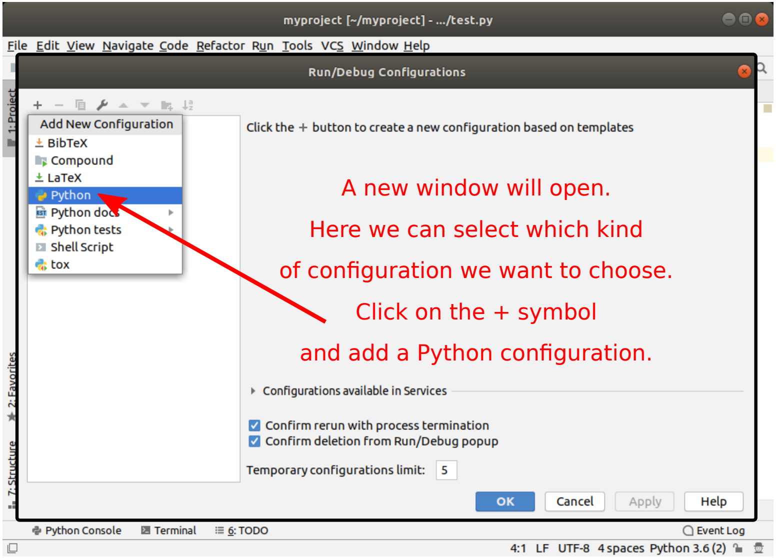
A new window will open.
Here we can select which kind
of configuration we want to choose.
Click on the + symbol
and add a Python configuration.

Configurations available in Services

- [x] Confirm rerun with process termination
- [x] Confirm deletion from Run/Debug popup

Temporary configurations limit: 5

Task 3: Running a Python Program (9)



Click the + button to create a new configuration based on templates

A new window will open.

Here we can select which kind

of configuration we want to choose.

Click on the + symbol

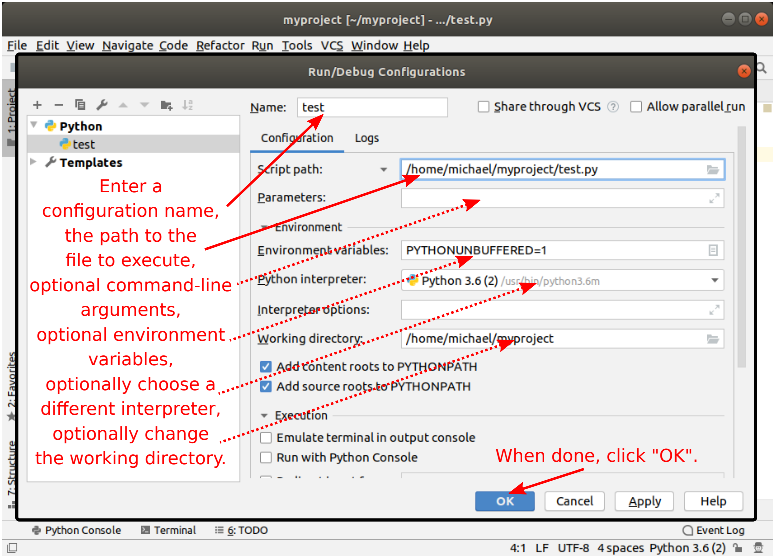
and add a Python configuration.

Configurations available in Services

- [x] Confirm rerun with process termination
- [x] Confirm deletion from Run/Debug popup

Temporary configurations limit: 5

Task 3: Running a Python Program (10)



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

Run/Debug Configurations

Name: test

Share through VCS

Allow parallel run

Script path: /home/michael/myproject/test.py

Parameters:

Environment

Environment variables: PYTHONUNBUFFERED=1

Python interpreter: Python 3.6 (2) /usr/bin/python3.6m

Interpreter options:

Working directory: /home/michael/myproject

[✓] Add content roots to PYTHONPATH

[✓] Add source roots to PYTHONPATH

Execution

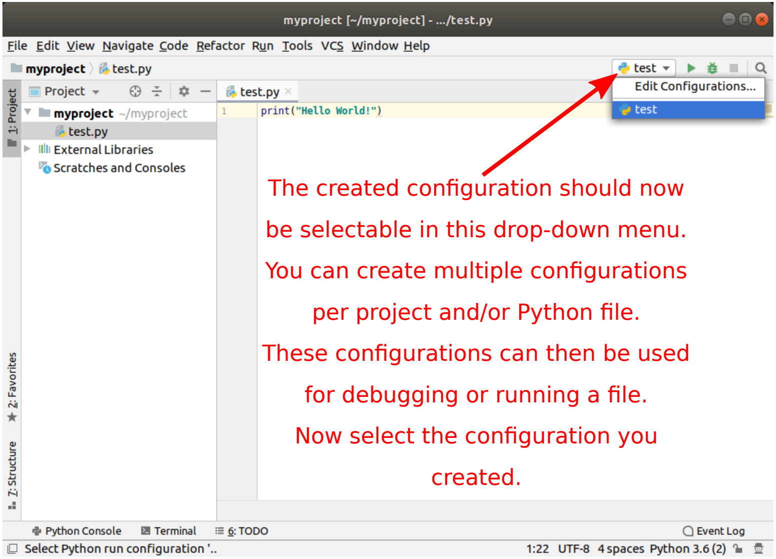
[ ] Emulate terminal in output console

[ ] Run with Python Console

Enter a configuration name, the path to the file to execute, optional command-line arguments, optional environment variables, optionally choose a different interpreter, optionally change the working directory.

When done, click "OK".

Task 3: Running a Python Program (11)

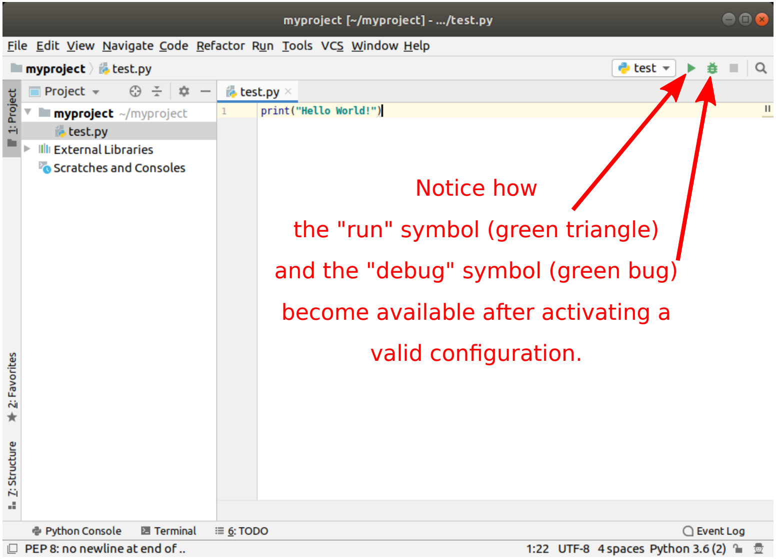


```
print("Hello World!")
```

Select Python run configuration '..

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)



```
print("Hello World!")
```

Notice how

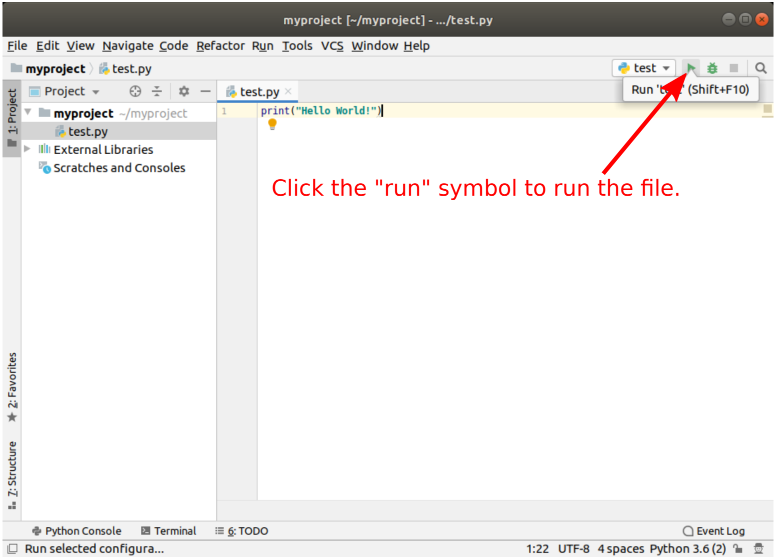
the "run" symbol (green triangle)

and the "debug" symbol (green bug)

become available after activating a

valid configuration.

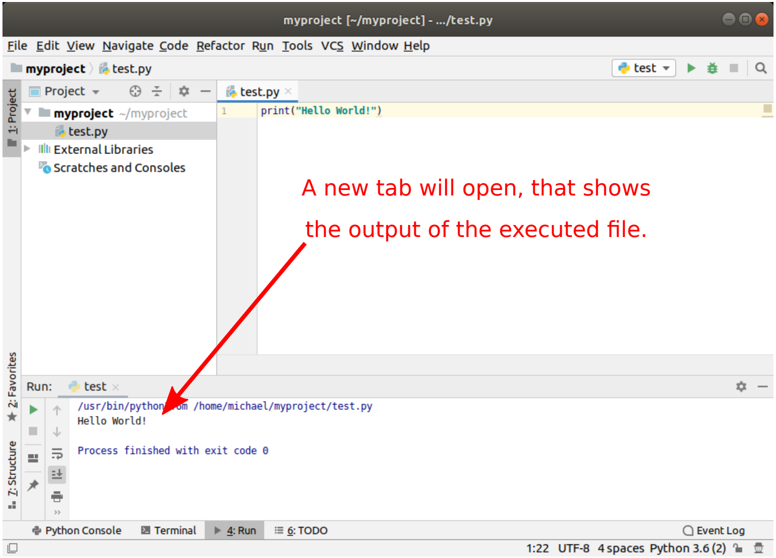
Task 3: Running a Python Program (13)



```
print("Hello World!")
```

Click the "run" symbol to run the file.

Task 3: Running a Python Program (14)


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
test.py
1: Project
2: Favorites
7: Structure
Run: test
Python Console
Terminal
4: Run
6: TODO
Event Log
1:22 UTF-8 4 spaces Python 3.6 (2)
A new tab will open, that shows
the output of the executed file.

```
print("Hello World!")
```

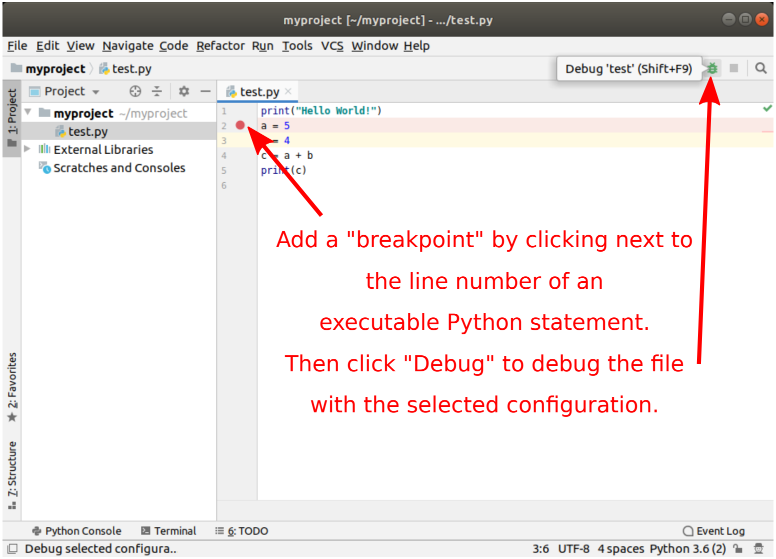
```
/usr/bin/python3.6 /home/michael/myproject/test.py
Hello World!

Process finished with exit code 0
```

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`

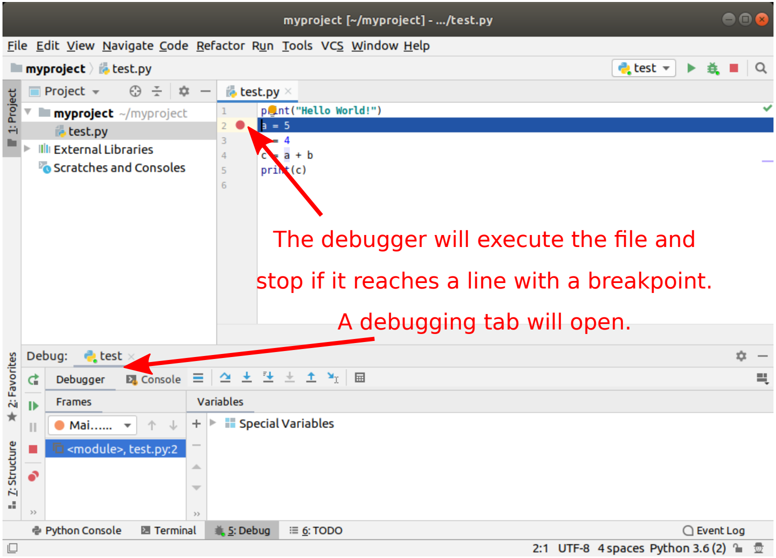
Task 4: Debugging a Python Program (3)



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

Add a "breakpoint" by clicking next to the line number of an executable Python statement. Then click "Debug" to debug the file with the selected configuration.

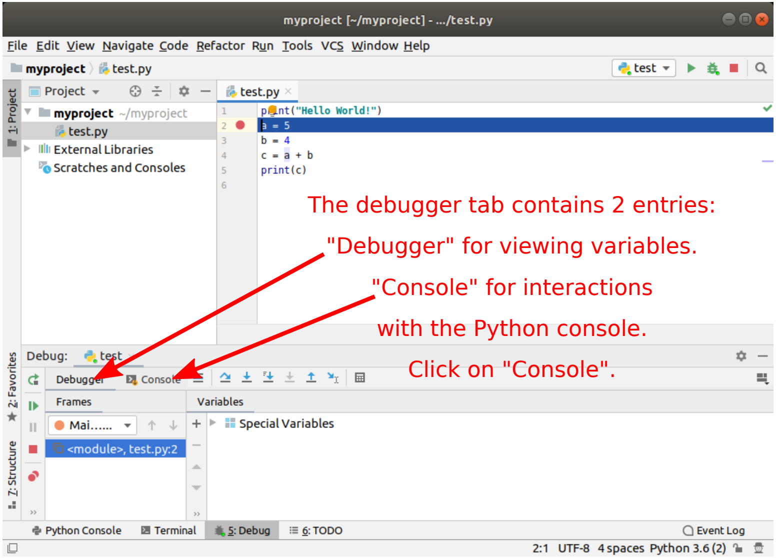
Task 4: Debugging a Python Program (4)



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

The debugger will execute the file and stop if it reaches a line with a breakpoint. A debugging tab will open.

Task 4: Debugging a Python Program (5)



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

The debugger tab contains 2 entries:

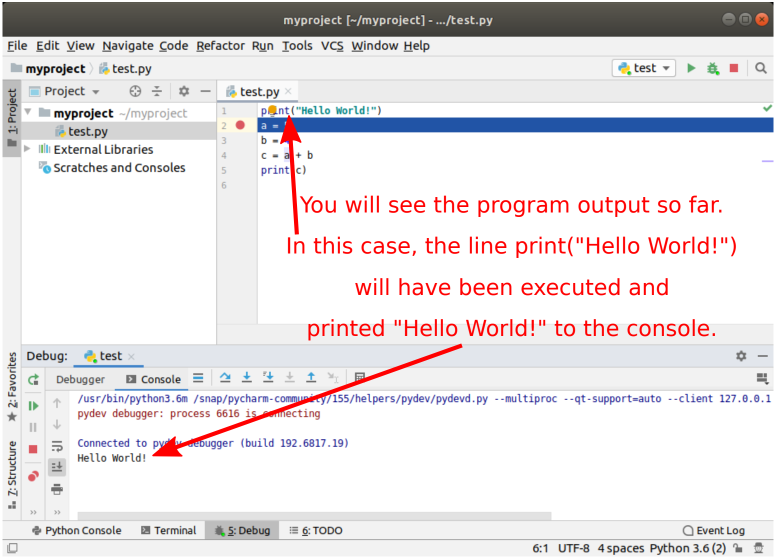
"Debugger" for viewing variables.

"Console" for interactions

with the Python console.

Click on "Console".

Task 4: Debugging a Python Program (6)


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
Debug: test
Debugger Console
Python Console Terminal 5: Debug 6: TODO Event Log
6:1 UTF-8 4 spaces Python 3.6 (2)

```
print("Hello World!")
a = 
b = 
c = a + b
print(c)
```

You will see the program output so far.

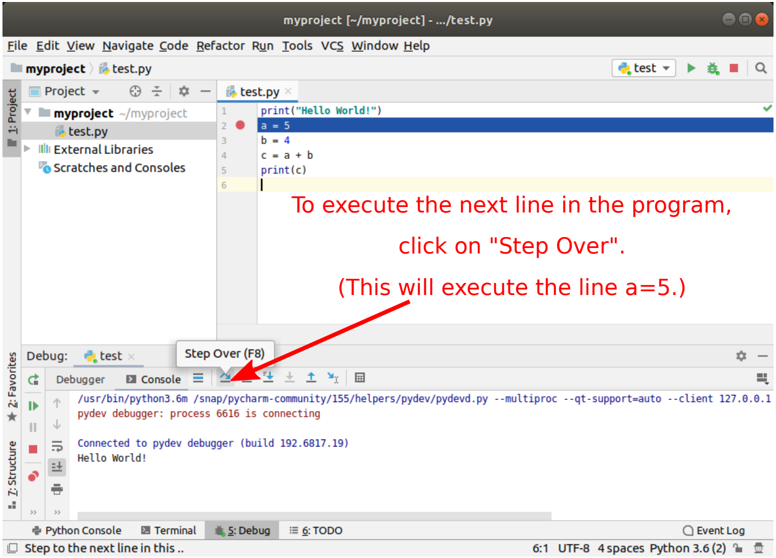
In this case, the line print("Hello World!")

will have been executed and

printed "Hello World!" to the 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!
```

Task 4: Debugging a Python Program (7)



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

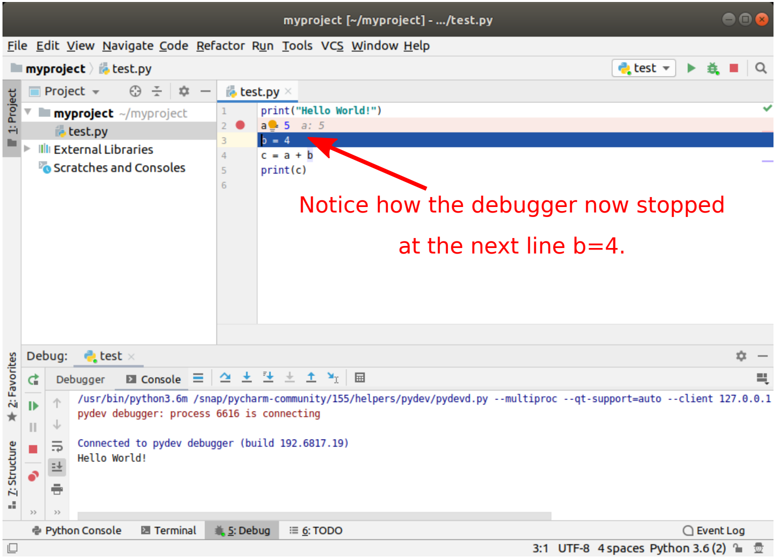
To execute the next line in the program,

click on "Step Over".

(This will execute the line a=5.)

```
/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!
```

Task 4: Debugging a Python Program (8)



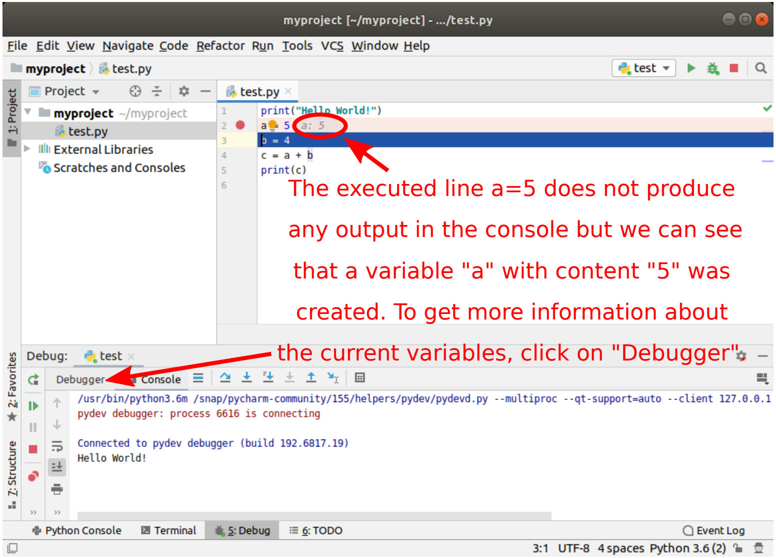
```
print("Hello World!")
a = 5   a: 5
b = 4
c = a + b
print(c)
```

Notice how the debugger now stopped

at the next line b=4.

```
/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!
```

Task 4: Debugging a Python Program (9)

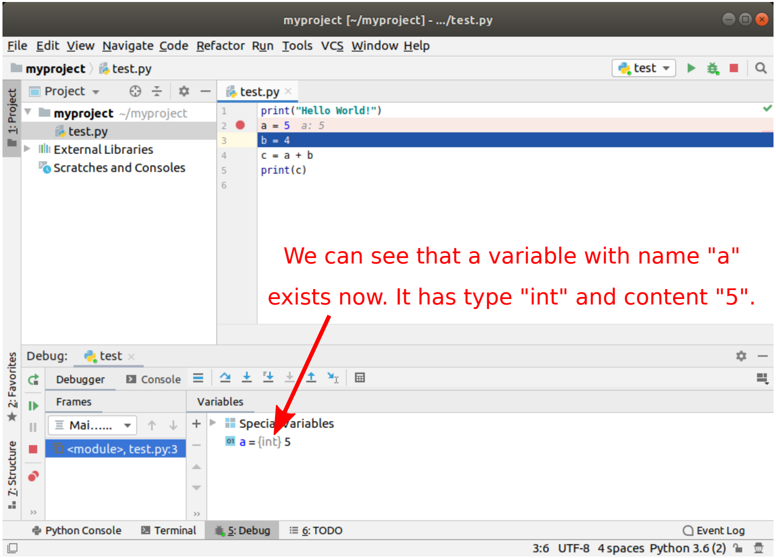


```
print("Hello World!")
a = 5   a: 5
b = 4
c = a + b
print(c)
```

The executed line a=5 does not produce any output in the console but we can see that a variable "a" with content "5" was created. To get more information about the current variables, click on "Debugger".

```
/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!
```

Task 4: Debugging a Python Program (10)

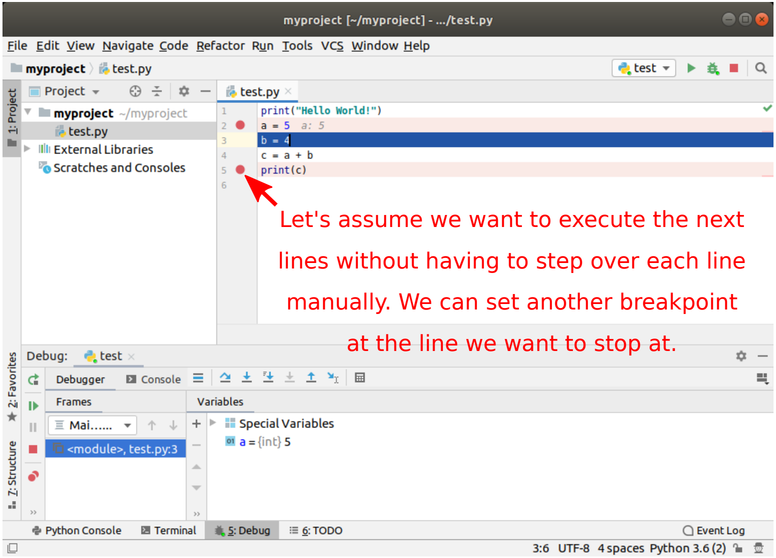


```
print("Hello World!")
a = 5   a: 5
b = 4
c = a + b
print(c)
```

a = {int} 5

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)

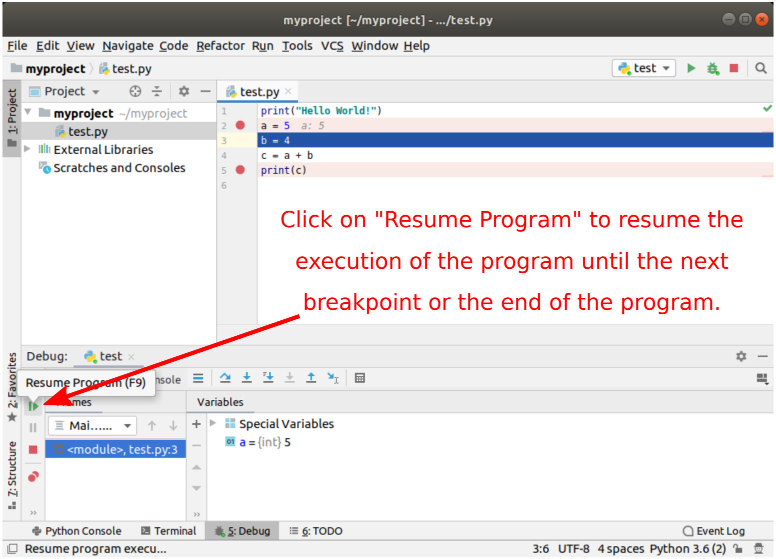


```
print("Hello World!")
a = 5   a: 5
b = 4
c = a + b
print(c)
```

a = {int} 5

Let's assume we want to execute the next lines without having to step over each line manually. We can set another breakpoint at the line we want to stop at.

Task 4: Debugging a Python Program (12)

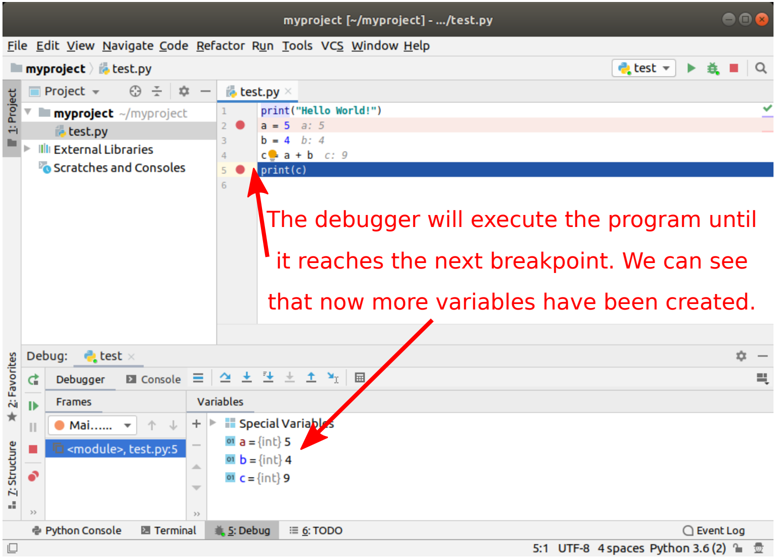


```
print("Hello World!")
a = 5   a: 5
b = 4
c = a + b
print(c)
```

a = {int} 5

Click on "Resume Program" to resume the execution of the program until the next breakpoint or the end of the program.

Task 4: Debugging a Python Program (13)



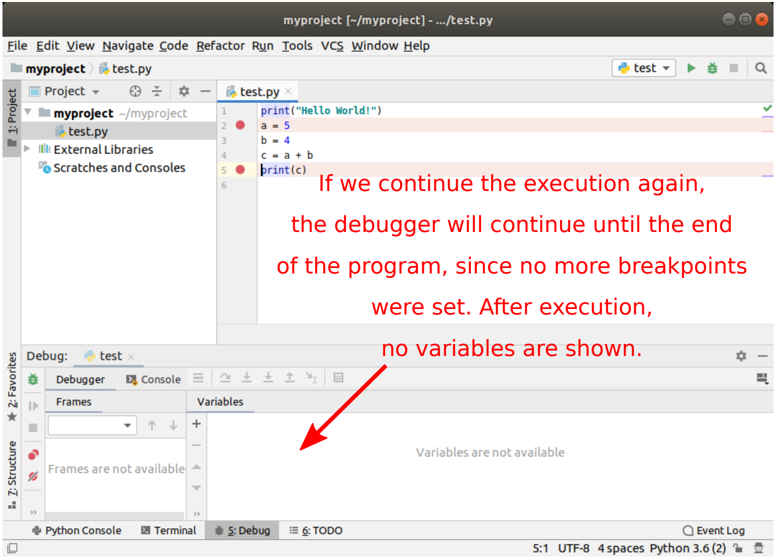
myproject [~/myproject] - .../test.py

```
print("Hello World!")
a = 5   a: 5
b = 4   b: 4
c = a + b   c: 9
print(c)
```

The debugger will execute the program until it reaches the next breakpoint. We can see that now more variables have been created.

```
Special Variables
a = {int} 5
b = {int} 4
c = {int} 9
```

Task 4: Debugging a Python Program (14)

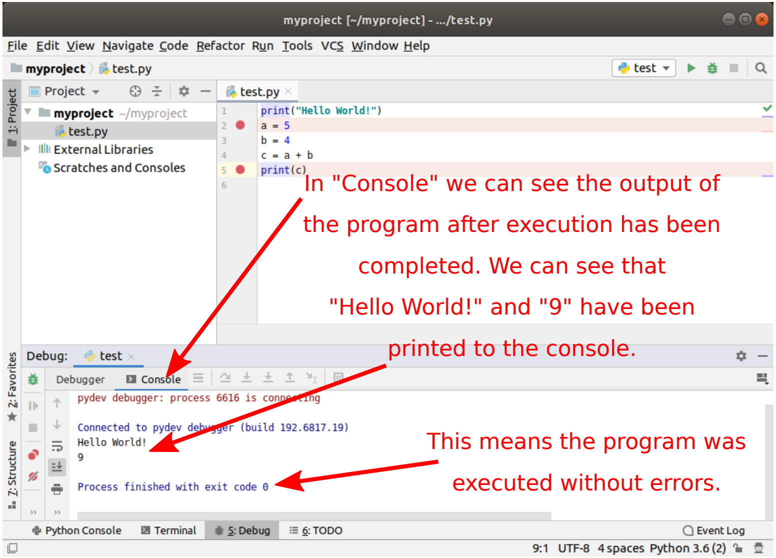


myproject [~/myproject] - .../test.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
myproject test.py
test
Project
myproject ~/myproject
test.py
External Libraries
Scratches and Consoles
Debug: test
Debugger Console
Frames
Variables
Frames are not available
Variables are not available
Python Console Terminal 5: Debug 6: TODO Event Log
5:1 UTF-8 4 spaces Python 3.6 (2)

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

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)



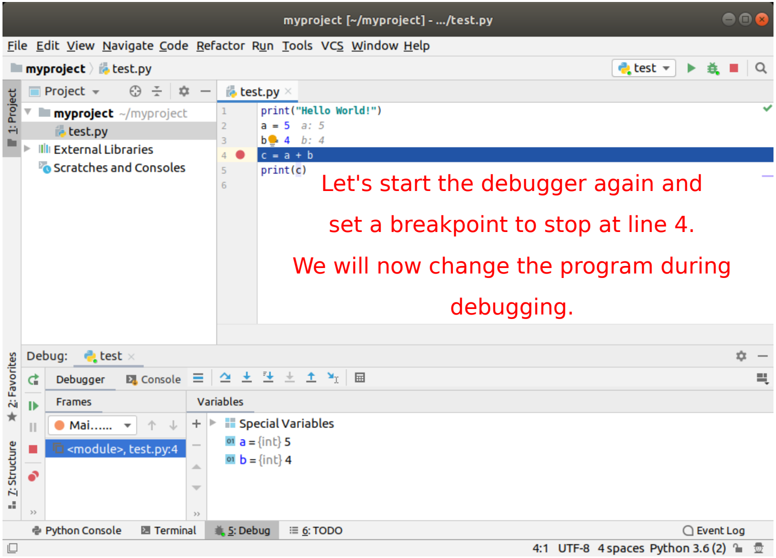
```
print("Hello World!")
a = 5
b = 4
c = a + b
print(c)
```

```
pydev debugger: process 6616 is connecting

Connected to pydev debugger (build 192.6817.19)
Hello World!
9

Process finished with exit code 0
```

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)

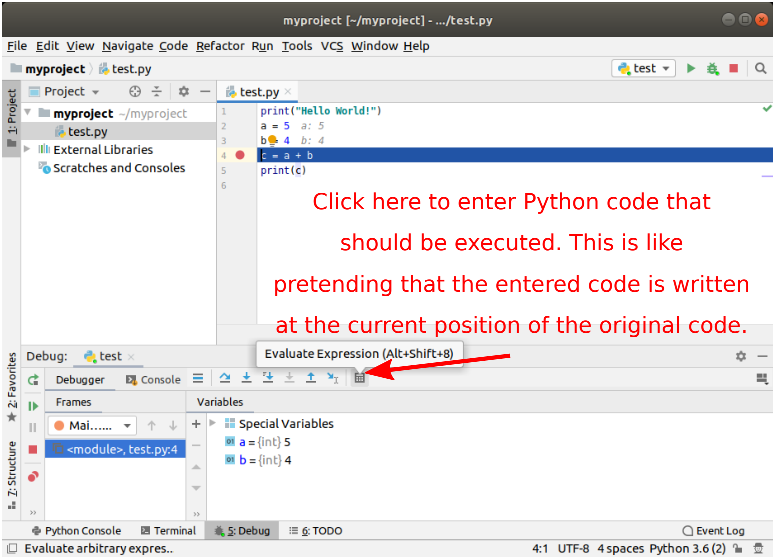


```
print("Hello World!")
a = 5   a: 5
b = 4   b: 4
c = a + b
print(c)
```

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

Let's start the debugger again and set a breakpoint to stop at line 4. We will now change the program during debugging.

Task 4: Debugging a Python Program (17)



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

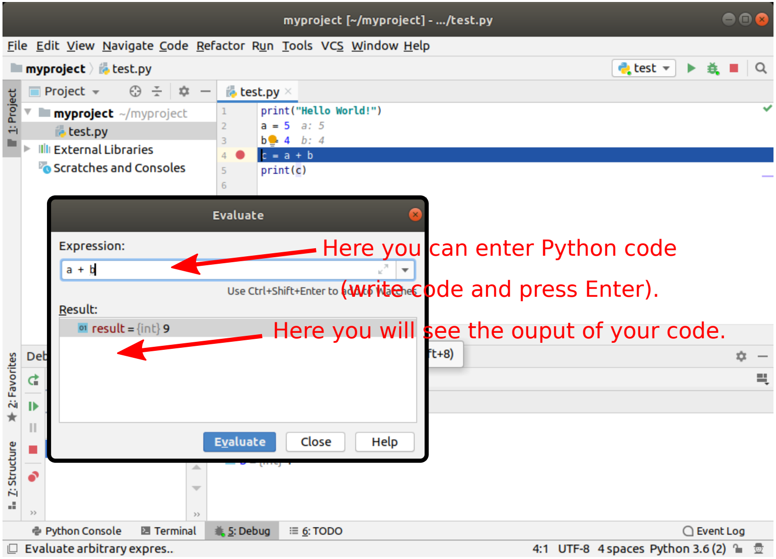
```
print("Hello World!")
a = 5  a: 5
b = 4  b: 4
c = a + b
print(c)
```

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)

a = {int} 5
b = {int} 4

Task 4: Debugging a Python Program (18)

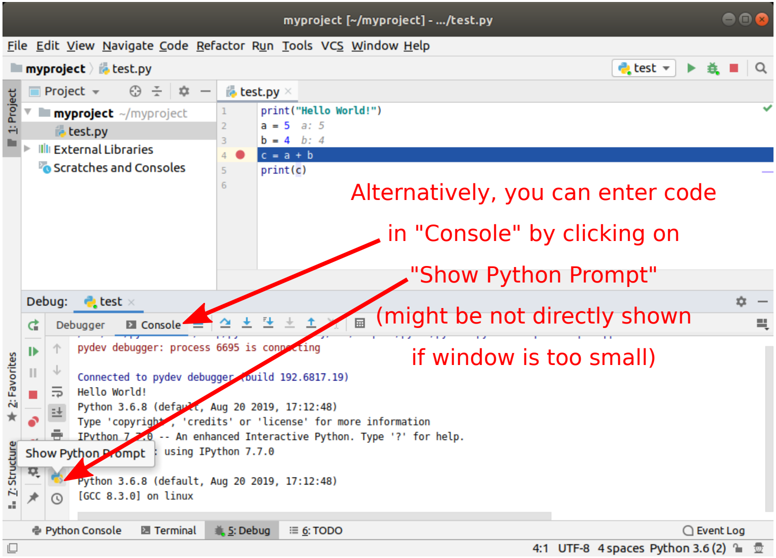


```
print("Hello World!")
a = 5   a: 5
b = 4   b: 4
c = a + b
print(c)
```

Here you can enter Python code (write code and press Enter).

Here you will see the ouput of your code.

Task 4: Debugging a Python Program (19)

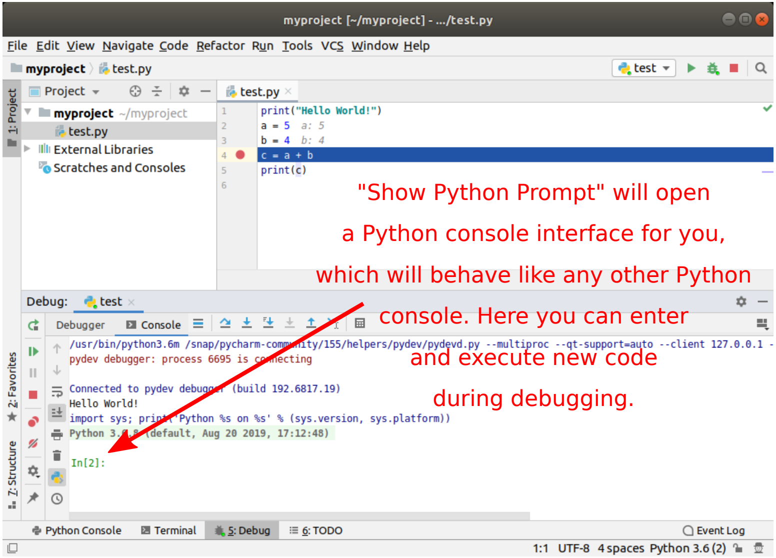


Alternatively, you can enter code in "Console" by clicking on "Show Python Prompt" (might be not directly shown if window is too small)

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

```
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
```

Task 4: Debugging a Python Program (20)

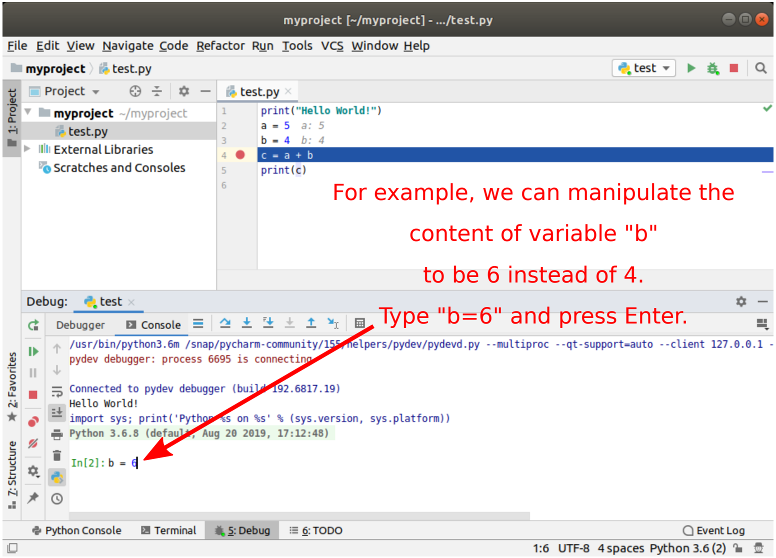

myproject [~/myproject] - .../test.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
myproject test.py
test
Project
myproject ~/myproject
test.py
External Libraries
Scratches and Consoles
Debug: test
Debugger Console
Python Console Terminal 5: Debug 6: TODO Event Log
1:1 UTF-8 4 spaces Python 3.6 (2)

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

```
/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]:
```

"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)


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
Debug: test
Debugger Console
Python Console Terminal 5: Debug 6: TODO Event Log
1:6 UTF-8 4 spaces Python 3.6 (2)

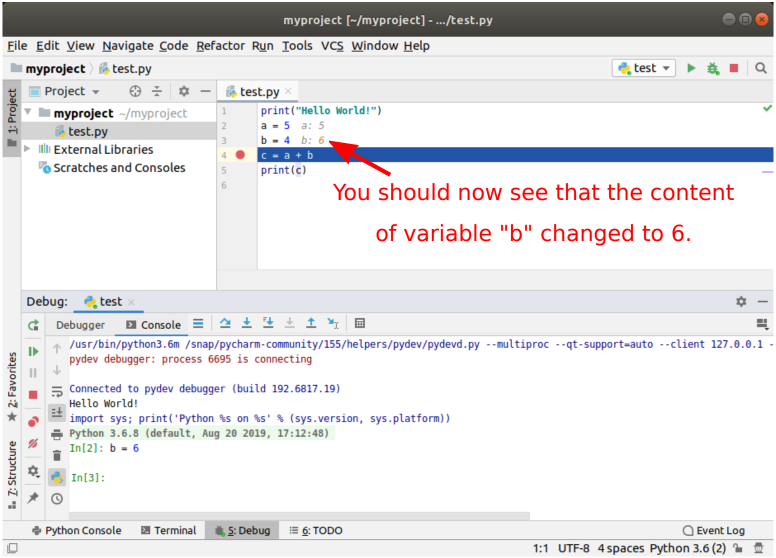
```
print("Hello World!")
a = 5  a: 5
b = 4  b: 4
c = a + b
print(c)
```

For example, we can manipulate the content of variable "b" to be 6 instead of 4.

Type "b=6" and press Enter.

```
/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
```

Task 4: Debugging a Python Program (22)

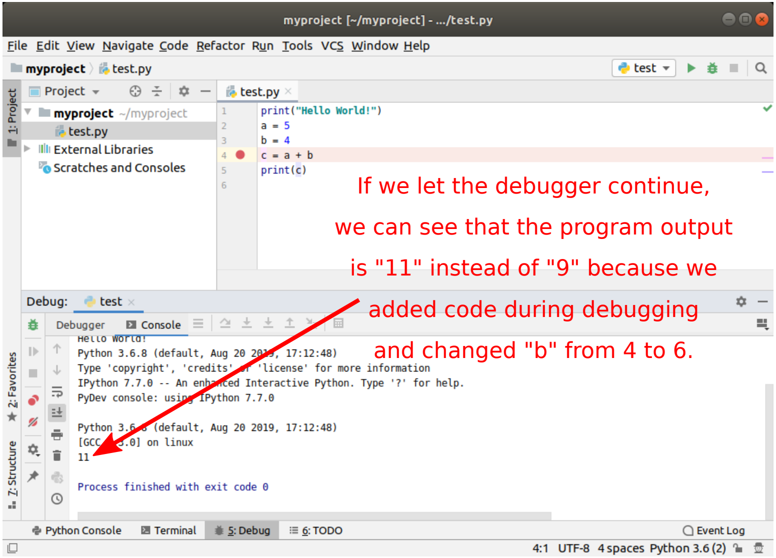

myproject [~/myproject] - .../test.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
myproject test.py
test
Project
myproject ~/myproject
test.py
External Libraries
Scratches and Consoles
Debug: test
Debugger Console
Python Console Terminal 5: Debug 6: TODO Event Log
1:1 UTF-8 4 spaces Python 3.6 (2)

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

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

```
/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]:
```

Task 4: Debugging a Python Program (23)


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
Debug: test
Debugger Console
Python Console Terminal 5: Debug 6: TODO Event Log
4:1 UTF-8 4 spaces Python 3.6 (2)

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

```
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.
PyDev console: using IPython 7.7.0

Python 3.6.8 (default, Aug 20 2019, 17:12:48)
[GCC 8.3.0] on linux
11

Process finished with exit code 0
```

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!