

PROGRAMMING IN PYTHON I

Editor and Debugger



Michael Widrich
Institute for Machine Learning

Copyright statement:

This material, no matter whether in printed or electronic form, may be used for personal and non-commercial educational use only. Any reproduction of this material, no matter whether as a whole or in parts, no matter whether in printed or in electronic form, requires explicit prior acceptance of the authors.

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 scripts, ...)
 - 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

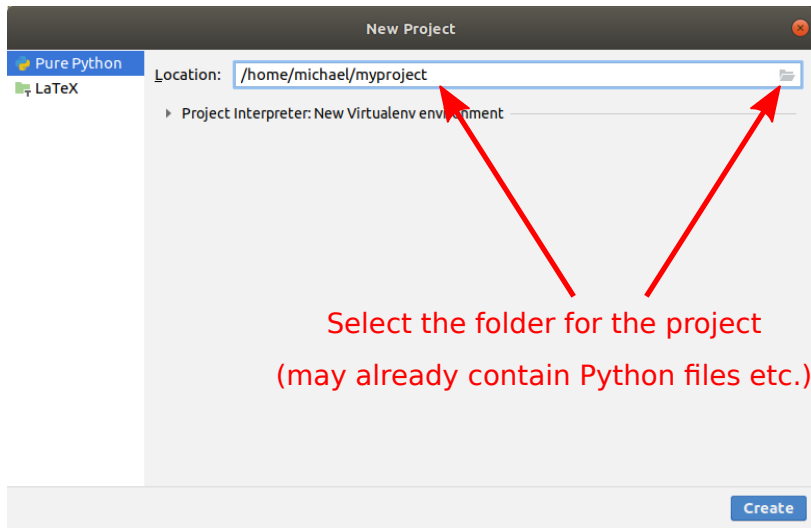
PYCHARM – EDITOR



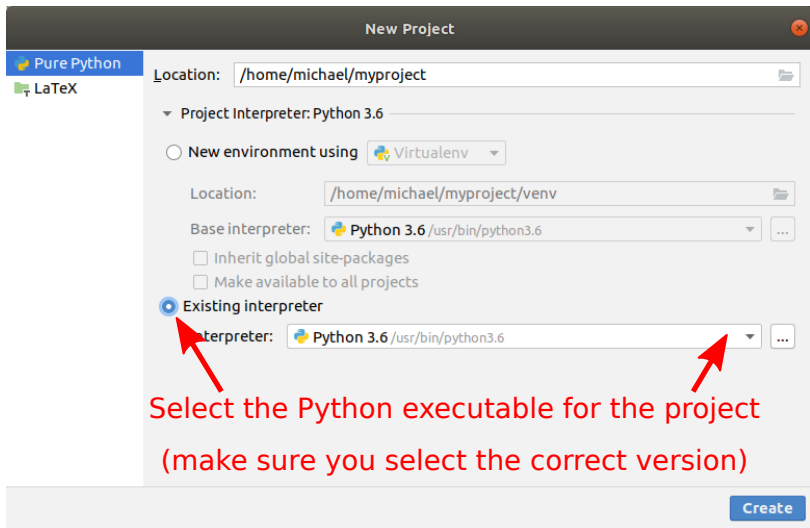
Task 1: Create a new Pycharm project (1)

- 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

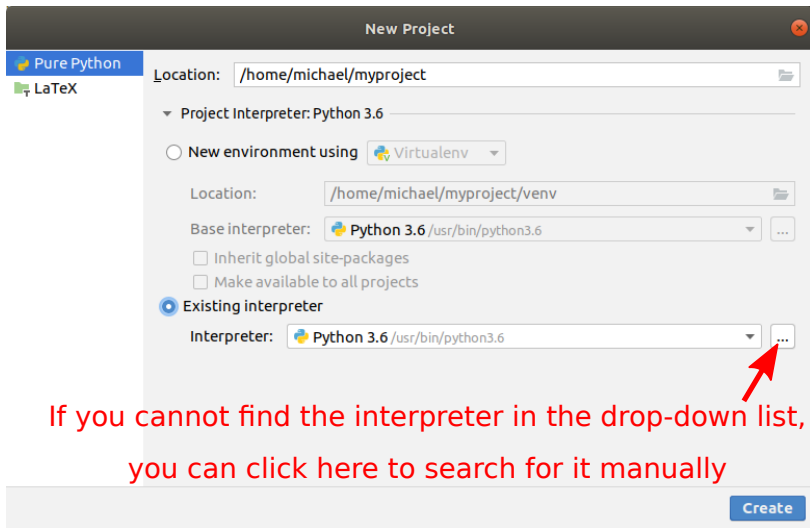
Task 1: Create a new Pycharm project (2)



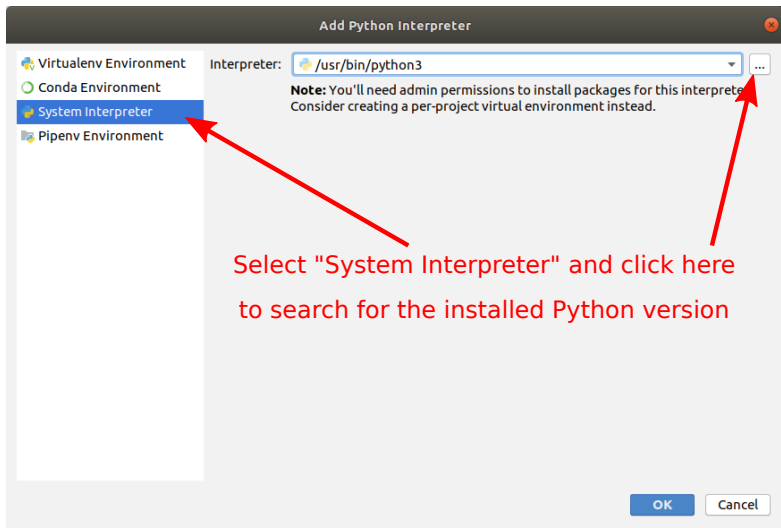
Task 1: Create a new Pycharm project (3)



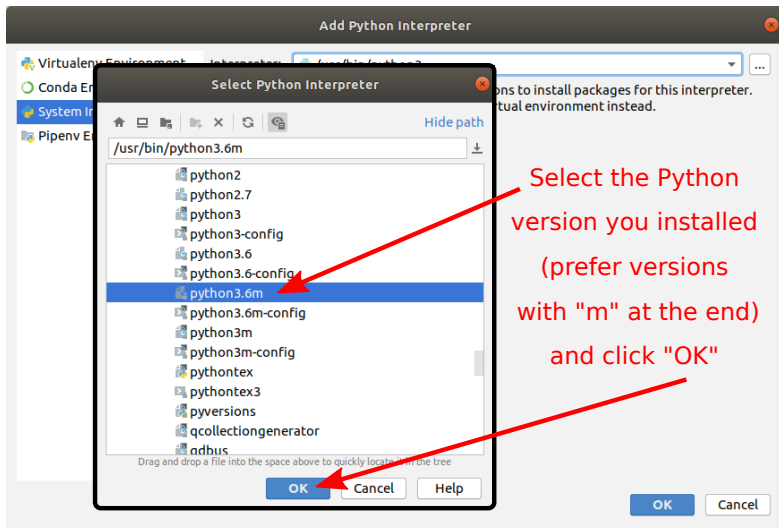
Task 1: Create a new Pycharm project (4)



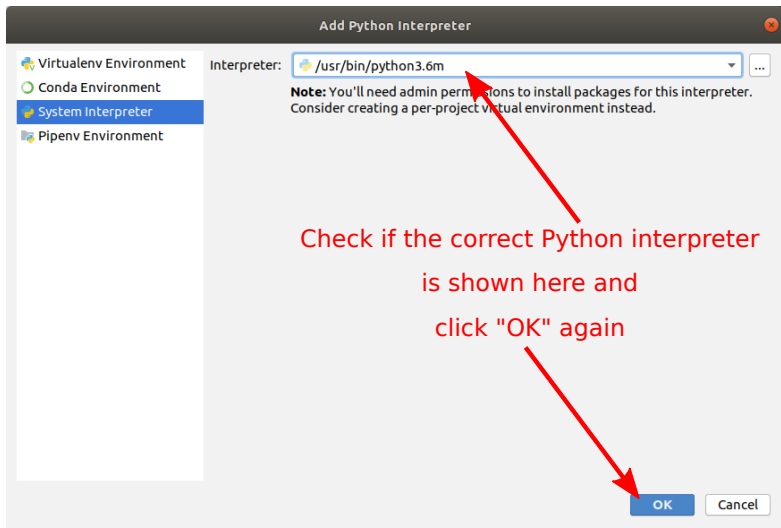
Task 1: Create a new Pycharm project (5)



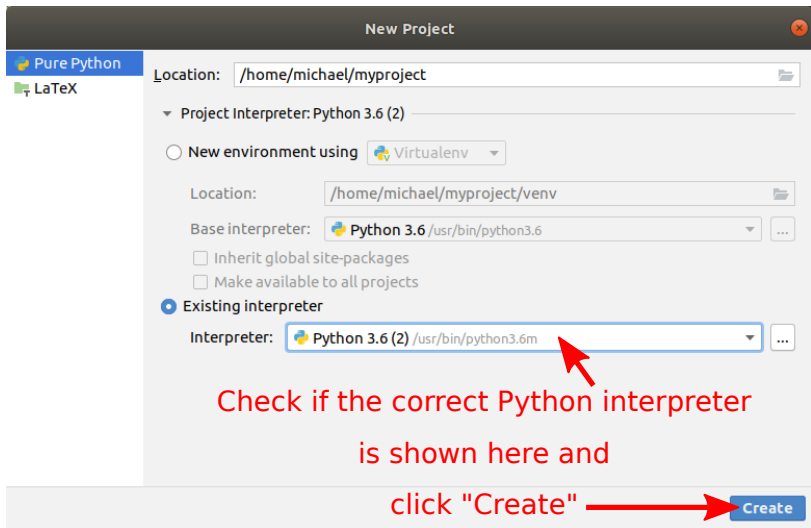
Task 1: Create a new Pycharm project (6)



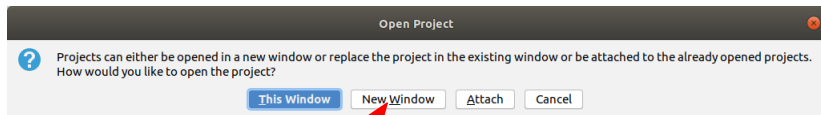
Task 1: Create a new Pycharm project (7)



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)

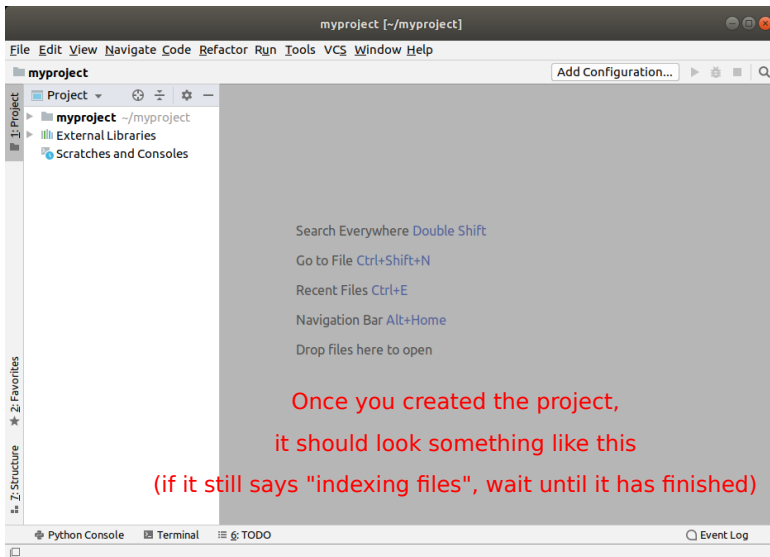
PYCHARM – PYTHON CONSOLE



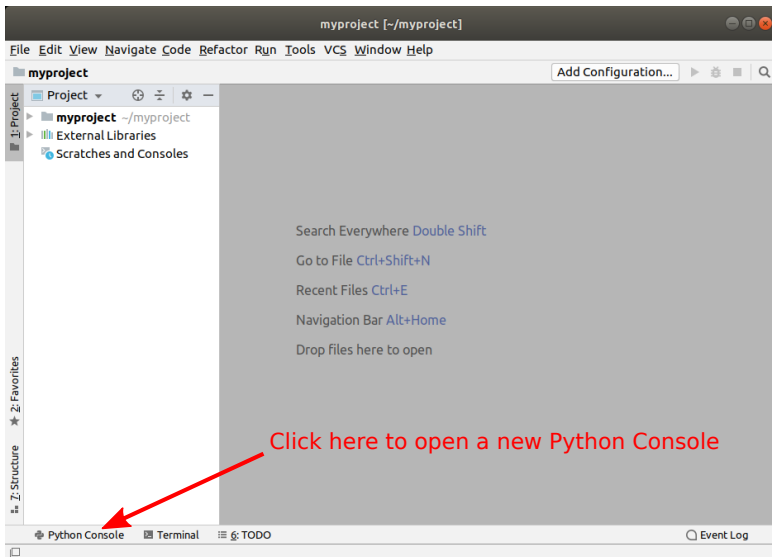
Task 2: Use the project Python console (1)

- 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)



Task 2: Use the project Python console (3)



Task 2: Use the project Python console (4)

The screenshot shows the 'myproject' IDE window. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The left sidebar shows the Project view with 'myproject' selected. The main editor area displays a list of shortcuts: Search Everywhere Double Shift, Go to File Ctrl+Shift+N, Recent Files Ctrl+E, Navigation Bar Alt+Home, and Drop files here to open. The bottom panel shows the Python Console with the following code: `import sys; print('Python %s on %s' % (sys.version, sys.platform))` and `sys.path.append(['/home/michael/myproject'])`. The console output shows 'Python 3.6.1 (default, Aug 20 2019, 17:12:48)' and the prompt 'In[2]: |'. A red arrow points from the text 'Here you can use the console as you would in the terminal/command line' to the Python Console. The bottom status bar shows 'Python Console', 'Terminal', 'TODO', and 'Event Log'.

myproject [~/myproject]

File Edit View Navigate Code Refactor Run Tools VCS Window Help

myproject

Add Configuration...

Project

myproject ~/myproject

External Libraries

Scratches and Consoles

Here you can use the console as you would in the terminal/command line

Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

Python Console

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

Python 3.6.1 (default, Aug 20 2019, 17:12:48)

In[2]: |

Special Variables

Python Console Terminal TODO Event Log

1:1

Task 2: Use the project Python console (5)

myproject [~/myproject]

File Edit View Navigate Code Refactor Run Tools VCS Window Help

myproject

Project

myproject ~/myproject

External Libraries

Scratches and Consoles

Let's start by typing some Python code
(type `print("Hello world")` as shown here)

Search Everywhere Double Shift

Go to File `Ctrl+Shift+N`

Recent Files `Ctrl+E`

Navigation Bar `Alt+Home`

Drop files here to open

Python Console

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

Python 3.6.1 (default, Aug 20 2019, 17:12:48)

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

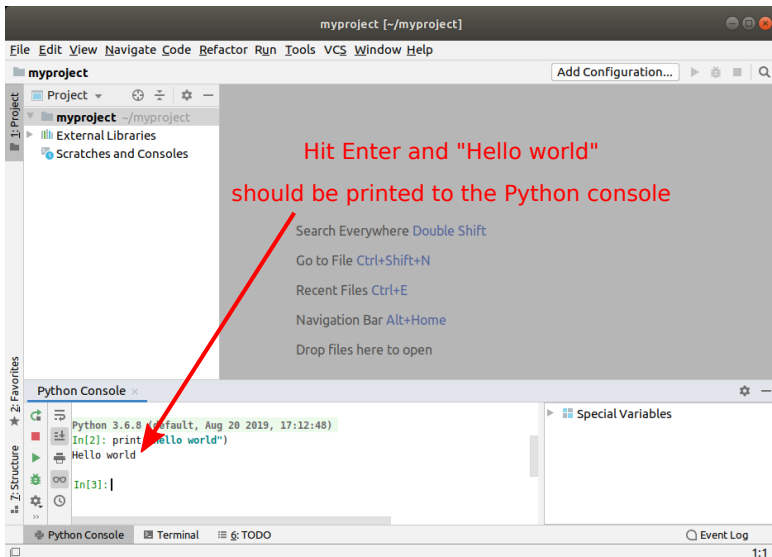
Special Variables

Python Console Terminal TODO

Event Log

1:21

Task 2: Use the project Python console (6)



Task 2: Use the project Python console (7)

The screenshot shows the 'myproject' IDE window. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The left sidebar shows the Project Explorer with 'myproject' and 'External Libraries'. The main editor area displays a list of shortcuts: 'Search Everywhere Double Shift', 'Go to File Ctrl+Shift+N', 'Recent Files Ctrl+E', 'Navigation Bar Alt+Home', and 'Drop files here to open'. The bottom panel shows the 'Python Console' with the following output:

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

A red arrow points from the text 'Create a variable "a" that is 5 by typing "a=5" (we'll see what that means later)' to the input field of the Python Console where 'a = 5' is being typed. The bottom status bar shows 'Python Console', 'Terminal', 'TODO', and 'Event Log'.

Task 2: Use the project Python console (8)

The screenshot shows the PyCharm IDE interface for a project named 'myproject'. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The left sidebar shows the Project view with 'myproject' and 'External Libraries'. The main editor area displays a search bar and a list of shortcuts: Search Everywhere Double Shift, Go to File Ctrl+Shift+N, Recent Files Ctrl+E, Navigation Bar Alt+Home, and Drop files here to open. The bottom panel is split into two sections: the Python Console on the left and the Special Variables panel on the right. The Python Console shows the following code and output:

```
Python 3.6.8 (default, Aug 20 2019, 17:12:48)
In[2]: print("Hello world")
Hello world
In[3]: a = 5
In[4]:
```

The Special Variables panel shows a list of variables, with 'a' highlighted and its value '{int} 5' displayed. A red arrow points from the text 'Here we see all current variables and their contents.' to the Special Variables panel. Another red arrow points from the text 'Hit Enter and the variable will be created.' to the Python Console.

Hit Enter and the variable will be created.

Here we see all current variables and their contents.

Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

Python Console

Python 3.6.8 (default, Aug 20 2019, 17:12:48)

```
In[2]: print("Hello world")
Hello world
In[3]: a = 5
In[4]:
```

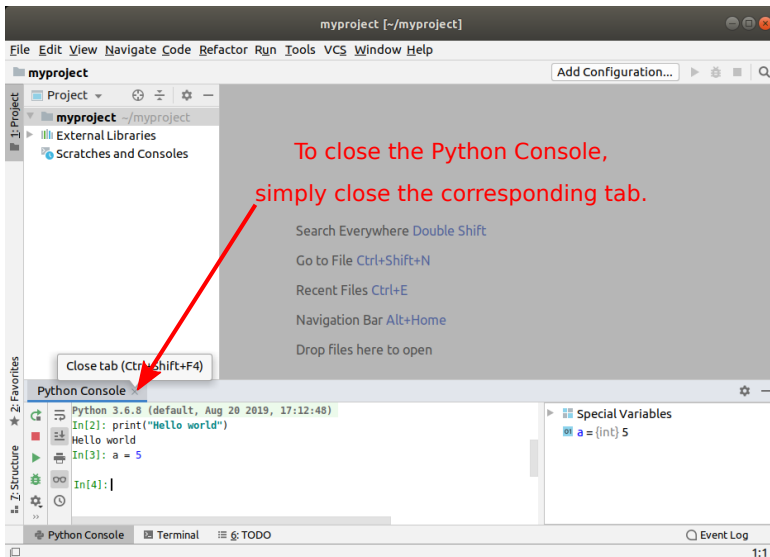
Special Variables

a = {int} 5

Python Console Terminal TODO Event Log

1:1

Task 2: Use the project Python console (9)



PYCHARM – RUNNING A PYTHON PROGRAM

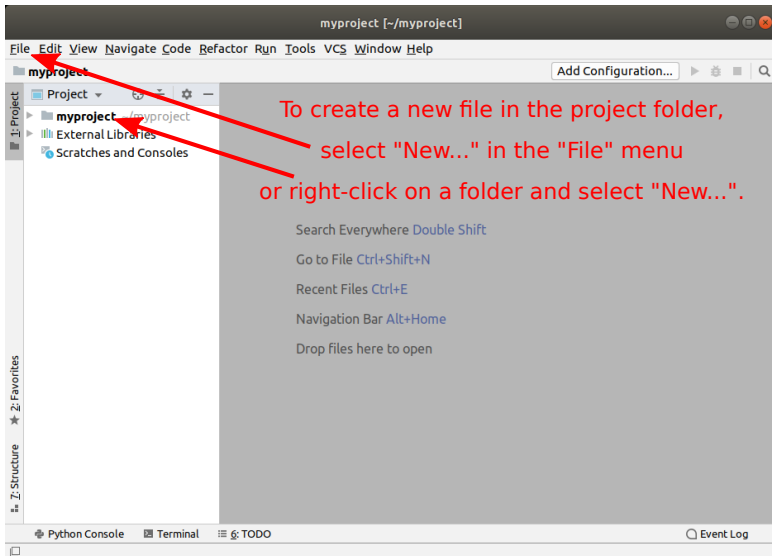


Task 3: Running a Python program (1)

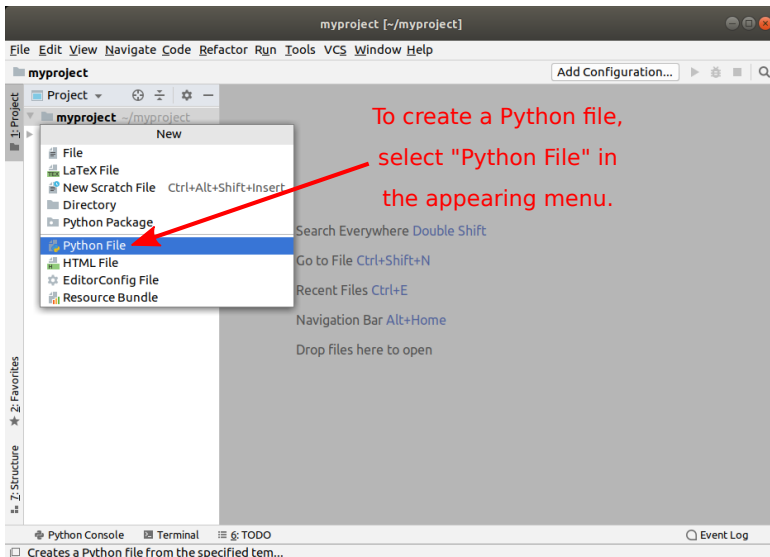
- We will now execute (=run) a Python program in PyCharm in these steps (see following slides for help):
 1. Create a new Python file `test.py` with content

```
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!"

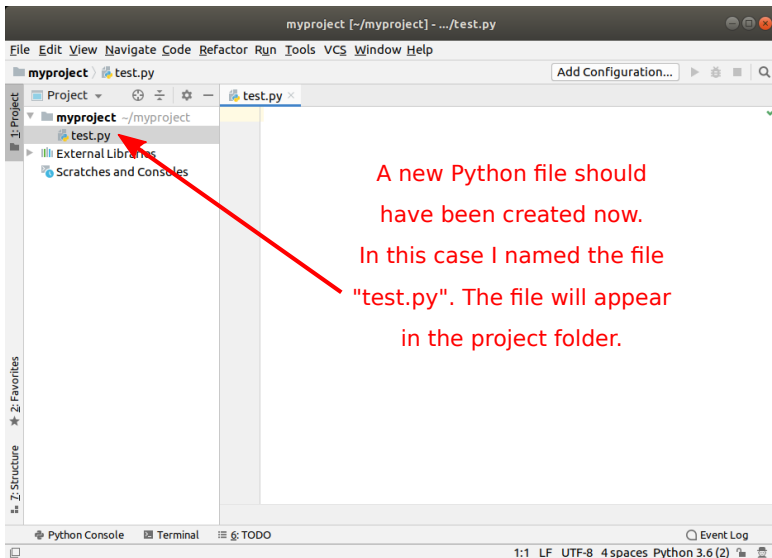
Task 3: Running a Python program (2)



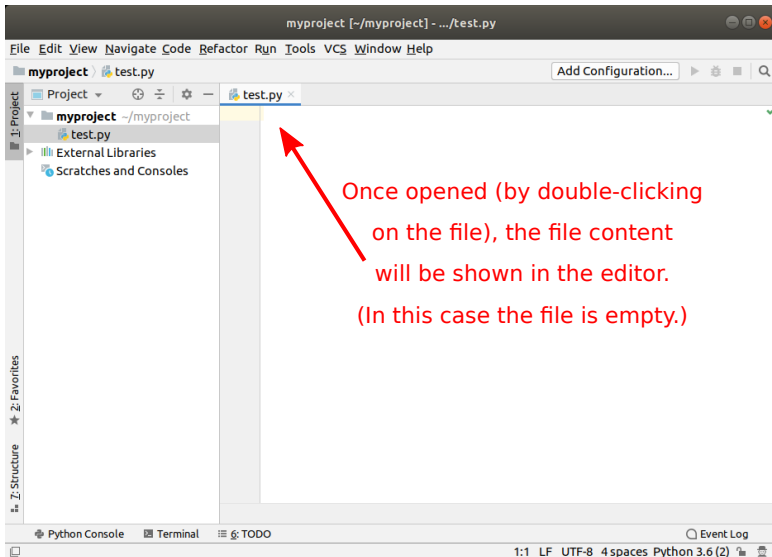
Task 3: Running a Python program (3)



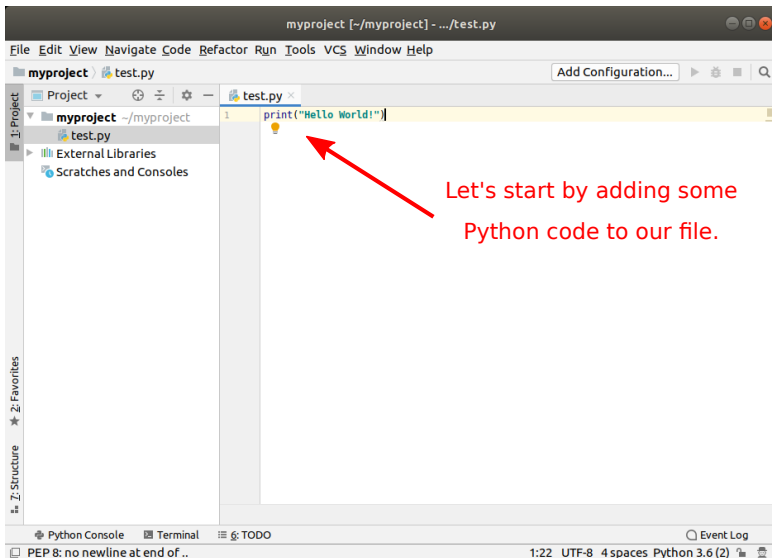
Task 3: Running a Python program (4)



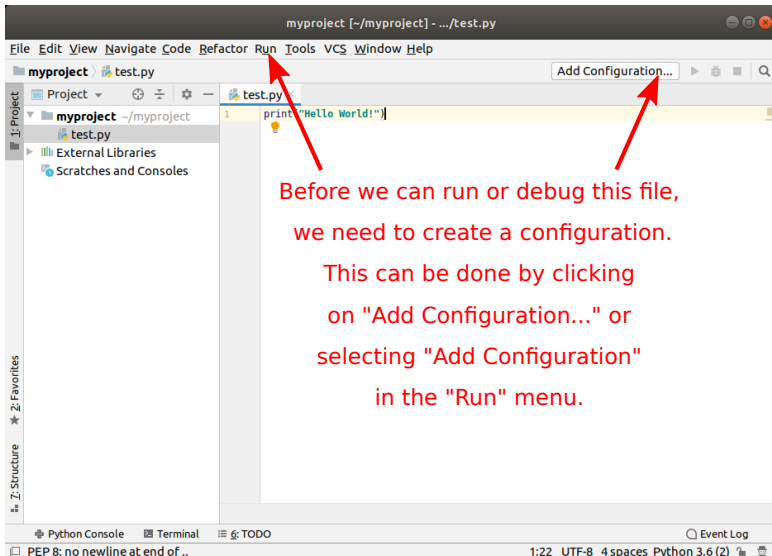
Task 3: Running a Python program (5)



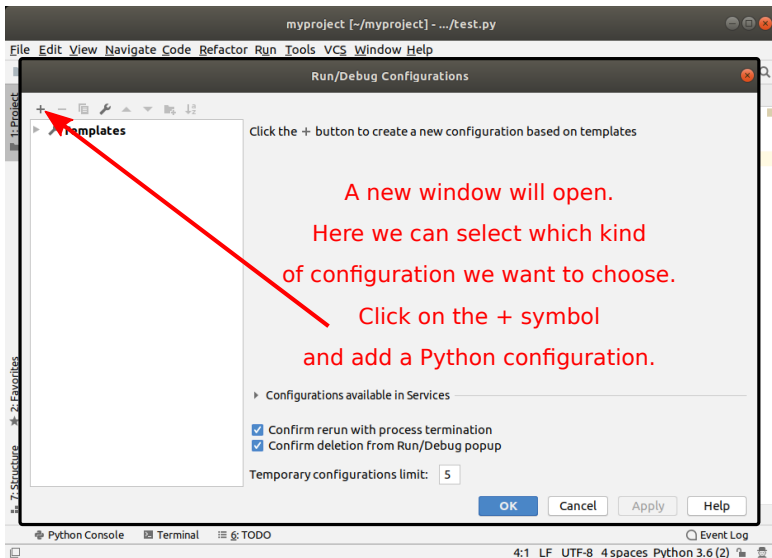
Task 3: Running a Python program (6)



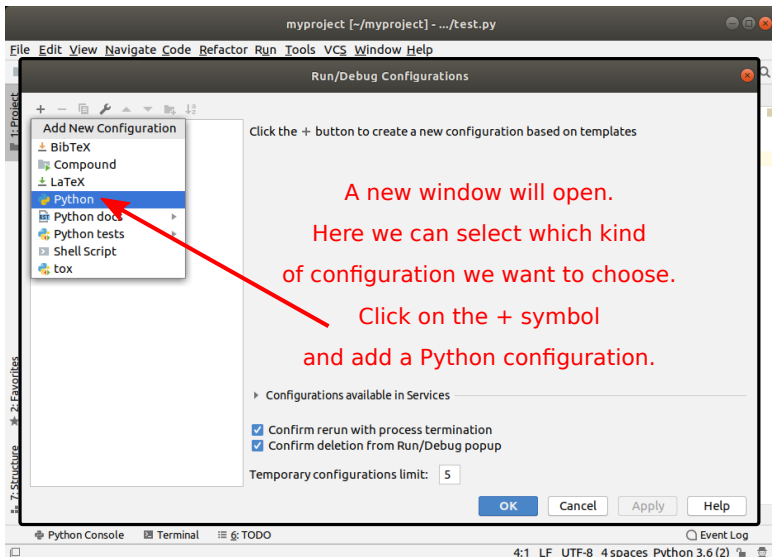
Task 3: Running a Python program (7)



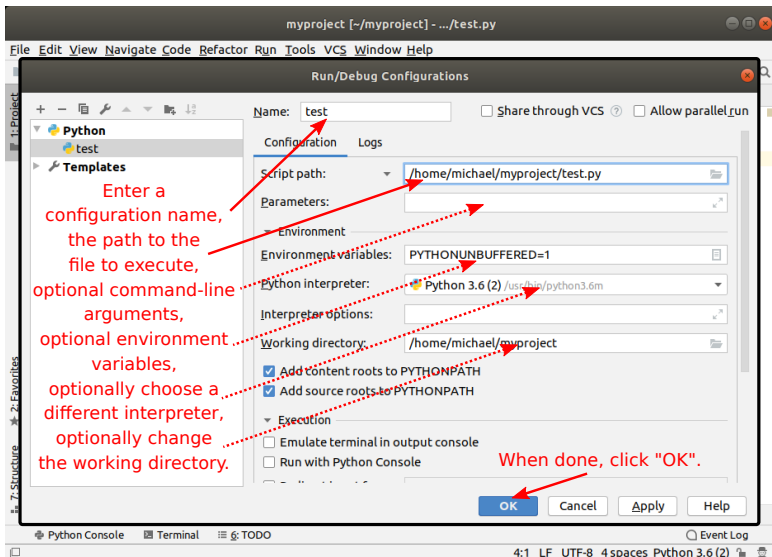
Task 3: Running a Python program (8)



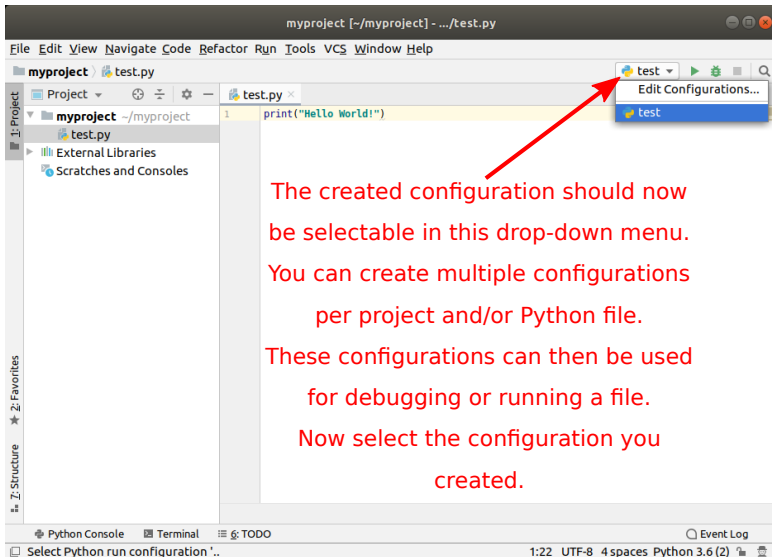
Task 3: Running a Python program (9)



Task 3: Running a Python program (10)

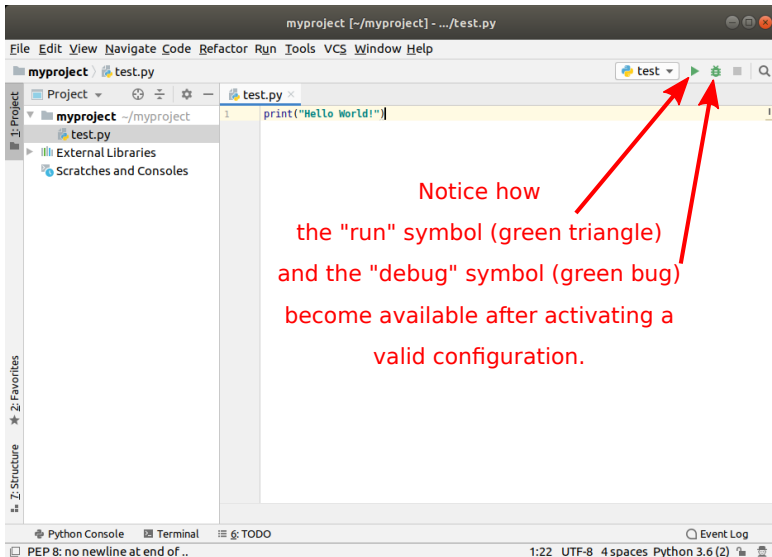


Task 3: Running a Python program (11)

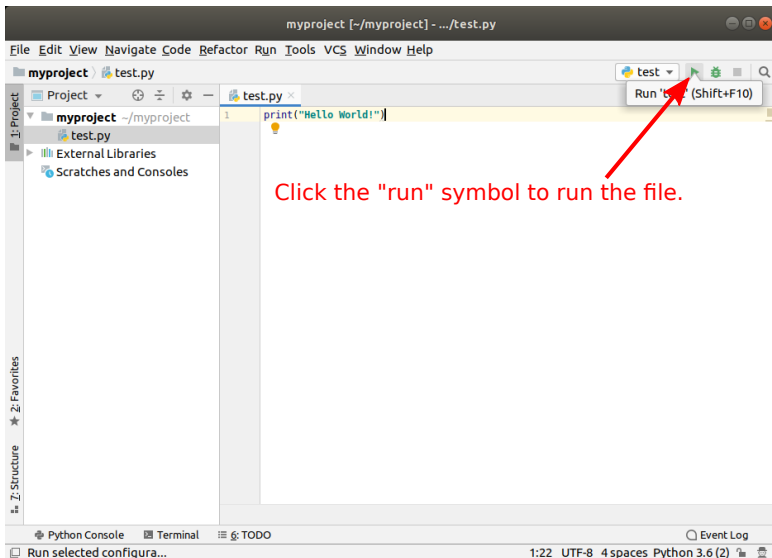


The created configuration should now be selectable in this drop-down menu. You can create multiple configurations per project and/or Python file. These configurations can then be used for debugging or running a file. Now select the configuration you created.

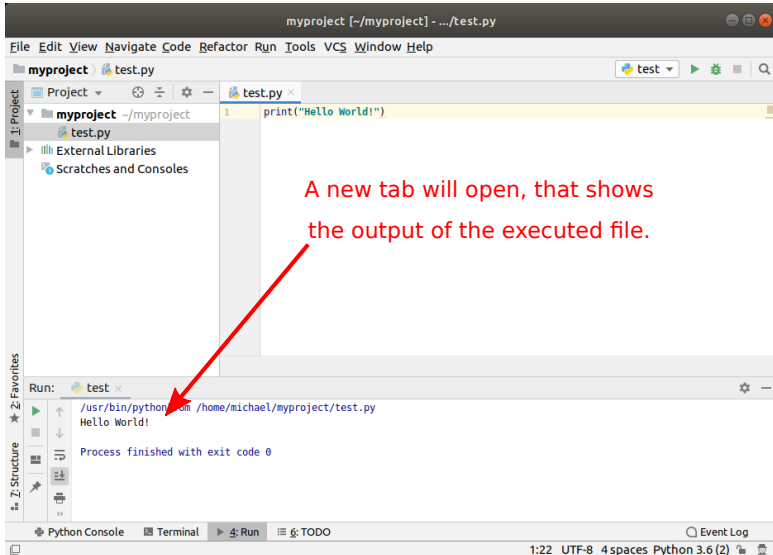
Task 3: Running a Python program (12)



Task 3: Running a Python program (13)



Task 3: Running a Python program (14)



A new tab will open, that shows
the output of the executed file.

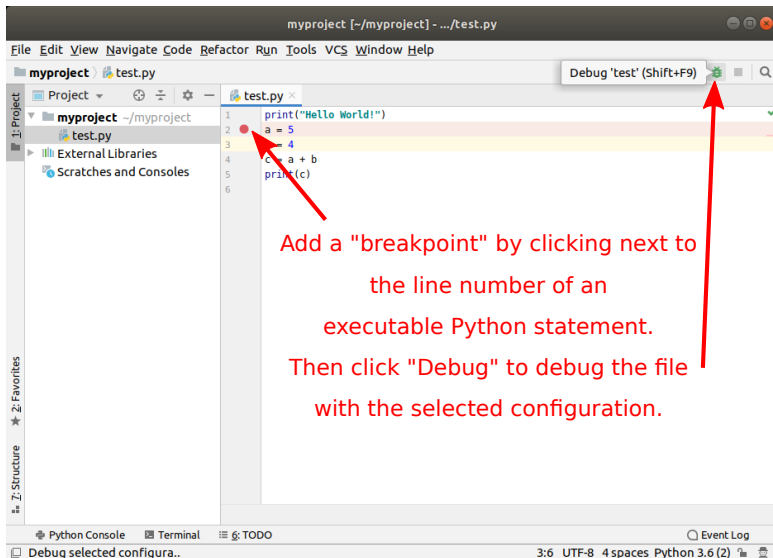
PYCHARM – DEBUGGING A PYTHON PROGRAM



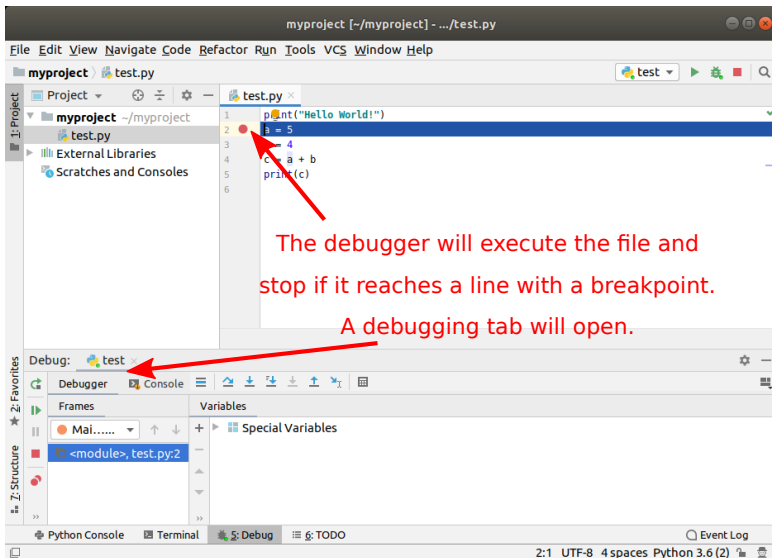
Task 4: Debugging a Python program (1)

- 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

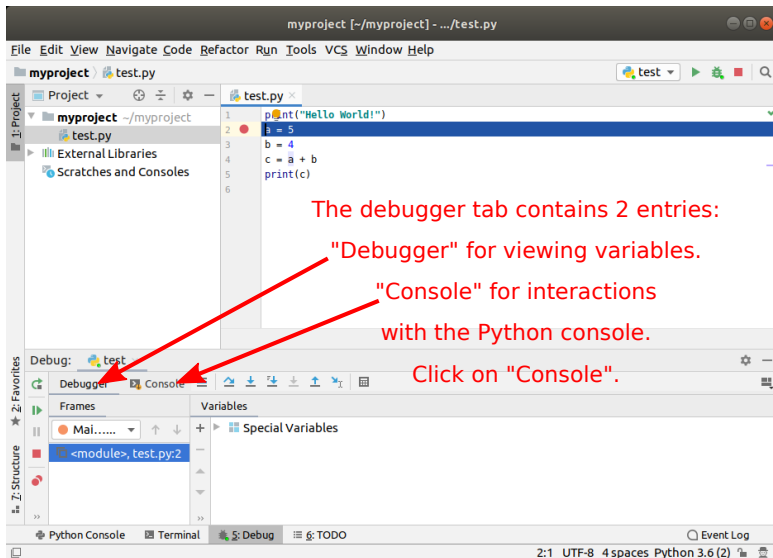
Task 4: Debugging a Python program (3)



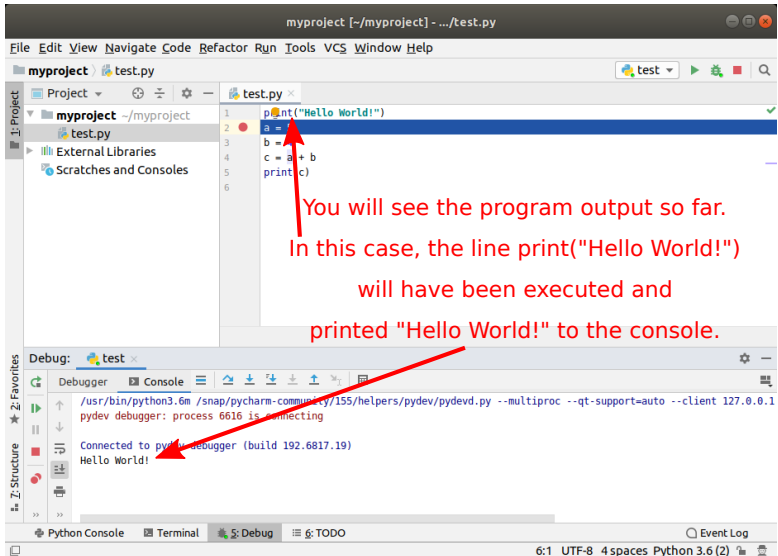
Task 4: Debugging a Python program (4)



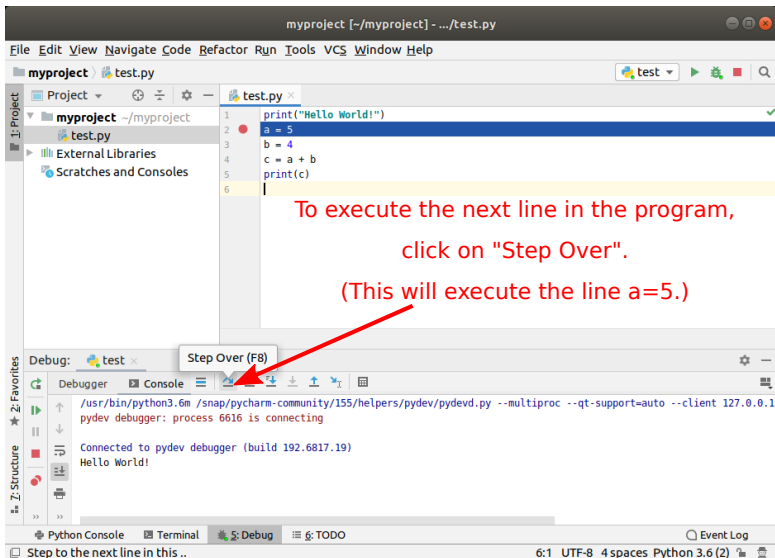
Task 4: Debugging a Python program (5)



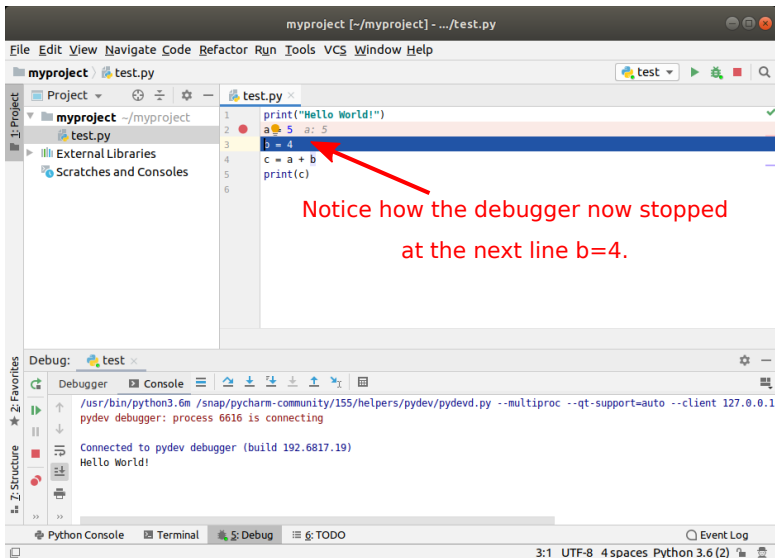
Task 4: Debugging a Python program (6)



Task 4: Debugging a Python program (7)



Task 4: Debugging a Python program (8)



Task 4: Debugging a Python program (9)

The screenshot shows the PyCharm IDE interface. The top toolbar includes buttons for 'test', 'Run', 'Debug', and 'Stop'. The left sidebar shows the project structure with 'myproject' and 'test.py'. The main editor window displays the code in 'test.py':

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

A red circle highlights the line `a = 5`, and a red arrow points to it from the text below. The 'Debugger' tab is active, showing the following output:

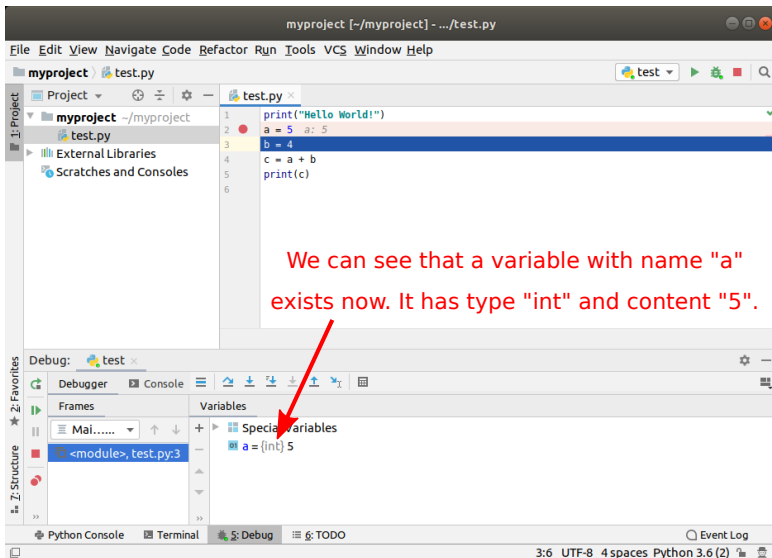
```
/usr/bin/python3.6m /snap/pycharm-community/155/helpers/pydev/pydevd.py --multiproc --qt-support=auto --client 127.0.0.1
pydev debugger: process 6616 is connecting

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

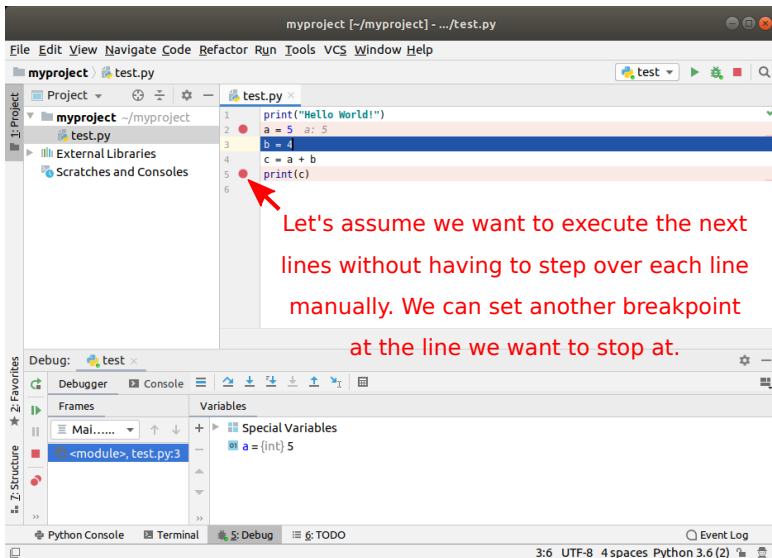
At the bottom of the IDE, the status bar shows '3:1 UTF-8 4 spaces Python 3.6 (2)'.

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"

Task 4: Debugging a Python program (10)



Task 4: Debugging a Python program (11)



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

File Edit View Navigate Code Refactor Run Tools VCS Window Help

myproject test.py

Project myproject test.py

External Libraries

Scratches and Consoles

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

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.

Debug: test

Debugger Console

Frames Variables

Mal... Special Variables

<module>, test.py:3

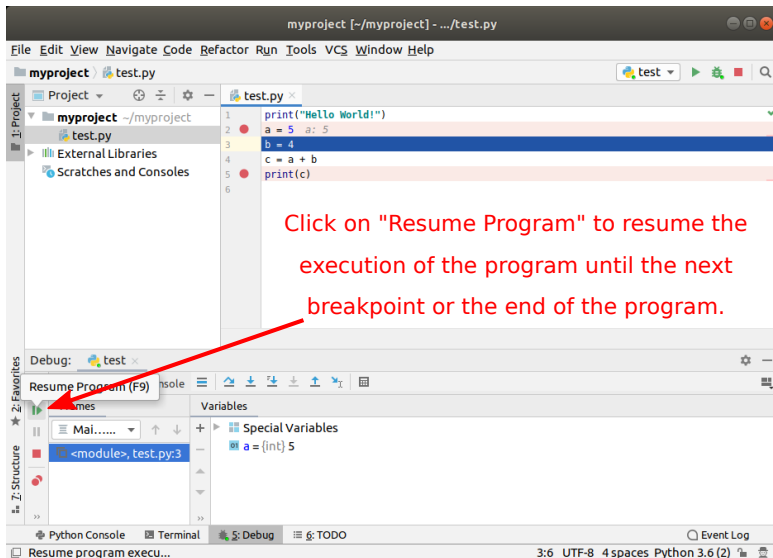
a = {int} 5

Python Console Terminal Debug TODO

Event Log

3:6 UTF-8 4 spaces Python 3.6 (2)

Task 4: Debugging a Python program (12)



Task 4: Debugging a Python program (13)

The screenshot shows an IDE window titled "myproject [-/myproject] - .../test.py". The editor displays the following code:

```
1 print("Hello World!")
2 a = 5 a: 5
3 b = 4 b: 4
4 c = a + b c: 9
5 print(c)
6
```

A red arrow points to the breakpoint at line 5. A text box with a red border contains the following text:

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

The bottom panel shows the "Debug" window with the "Variables" tab selected. It displays the following variables:

- Special Variables
- a = (int) 5
- b = (int) 4
- c = (int) 9

Another red arrow points to the variable "c".

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

Project myproject test.py

1: Project myproject test.py

2: test.py

3: External Libraries

4: Scratches and Consoles

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

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.

Debug: test

Debugger Console

Frames Variables

Frames are not available

Variables are not available

Python Console Terminal Debug TODO

Event Log

5:1 UTF-8 4 spaces Python 3.6 (2)

Task 4: Debugging a Python program (15)

The screenshot shows an IDE window titled "myproject [-/myproject] - .../test.py". The editor displays the following Python code:

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

Below the editor, the "Debug" panel is active, showing the "Console" tab. The output of the program is displayed:

```
pydev debugger: process 6616 is connecting
Connected to pydev debugger (build 192.6817.19)
Hello World!
9
Process finished with exit code 0
```

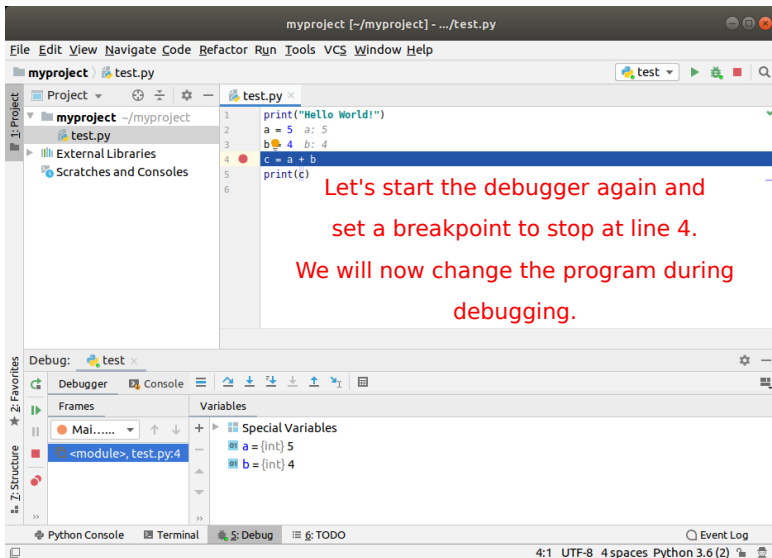
Red arrows point from the text annotations to the corresponding parts of the screenshot:

- An arrow points from the first part of the text to the "Console" tab in the Debug panel.
- An arrow points from the second part of the text to the output "Hello World!" in the console.
- An arrow points from the third part of the text to the output "9" in the console.
- An arrow points from the fourth part of the text to the output "Process finished with exit code 0" in the console.

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

Project 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 Variables

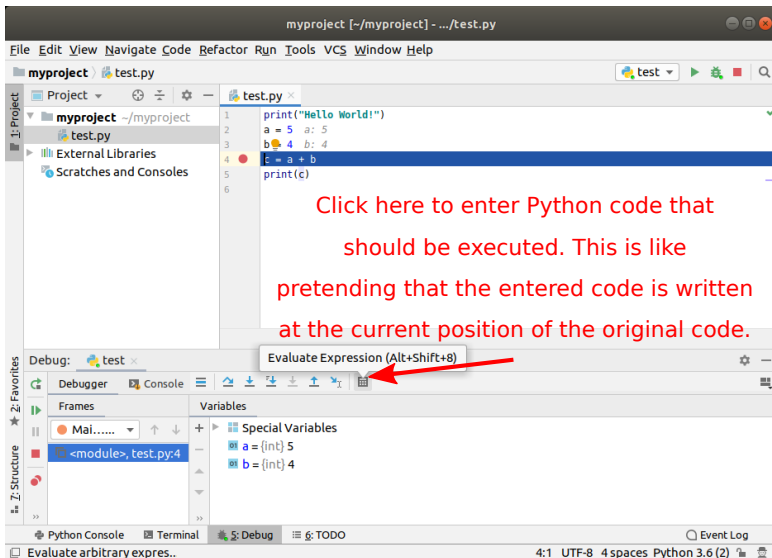
Special Variables

a = (int) 5
b = (int) 4

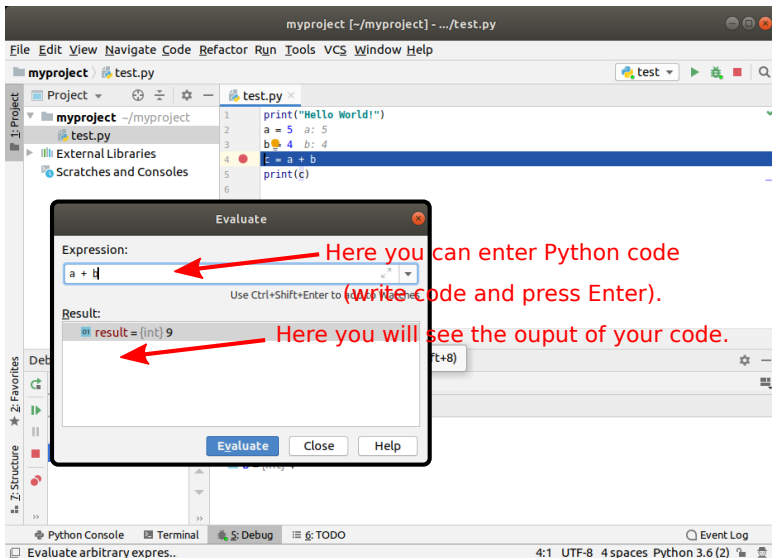
Python Console Terminal Debug TODO

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

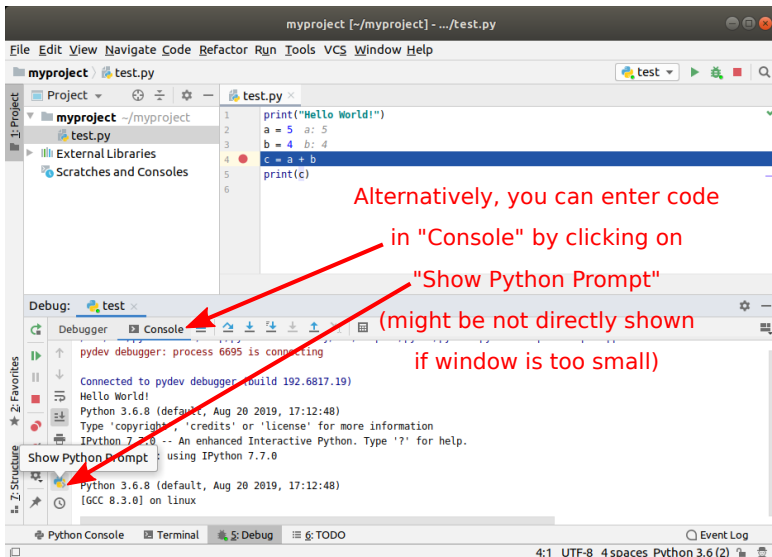
Task 4: Debugging a Python program (17)



Task 4: Debugging a Python program (18)



Task 4: Debugging a Python program (19)



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

Project test.py

test.py

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

"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.

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 3.6.5 (default, Aug 20 2019, 17:12:48)
In[2]:
```

Python Console Terminal Debug TODO

Event Log

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

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

Project test.py

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

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

Type "b=6" and press Enter.

Python Console Terminal Debug TODO

Event Log

1:6 UTF-8 4 spaces Python 3.6 (2)

Task 4: Debugging a Python program (22)

The screenshot shows the PyCharm IDE interface with a project named 'myproject'. The file 'test.py' is open, and the code is as follows:

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

A red arrow points to the line `b = 4 b: 6`, indicating that the value of variable 'b' has been updated to 6. The console output shows the following:

```
/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]:
```

The status bar at the bottom indicates the current state: '1:1 UTF-8 4 spaces Python 3.6 (2)'.

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

Project test.py

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

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.

Debug: test

Debugger Console

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 4.8.5] on linux

11

Process finished with exit code 0

Python Console Terminal Debug TODO

Event Log

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

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