

Part 0: Setting up your development environment

0.1 Install the necessary software

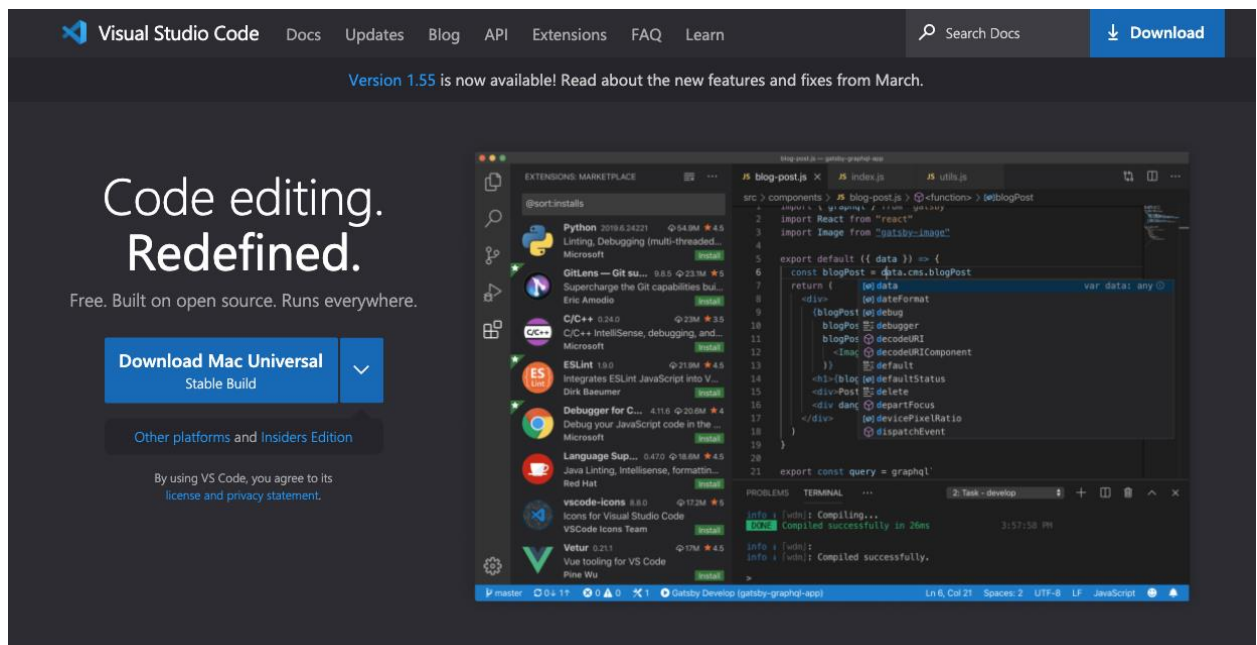
In this section, you will set up your computer to install the necessary software for developing Python.

1.1.1 Install Visual Studio Code

Visual Studio Code is a code editor for software engineers. It is built by Microsoft and is available for everyone for free. The idea of “code editor for software” is similar to “Microsoft Word to documents”.

Steps:

1. Go to the official website(<https://code.visualstudio.com/>) of Visual Studio Code to download the latest version of Visual Studio Code installer.

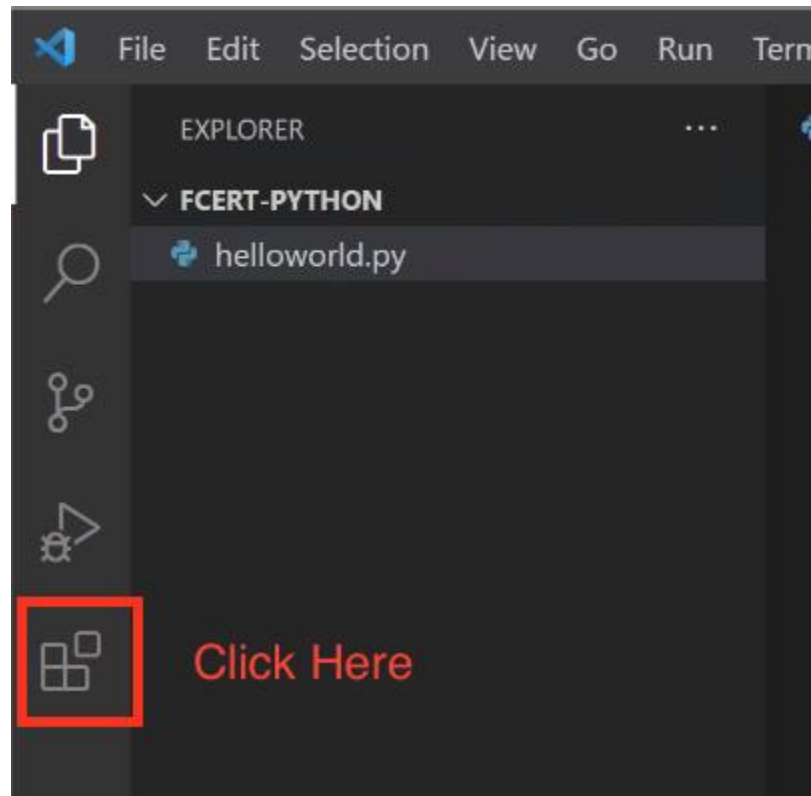


2. Once it is downloaded, run the installer (VSCodeUserSetup-{version}.exe). This will only take a minute. You can just leave all the options as-is.
3. By default, VS Code is installed under `C:\users\{username}\AppData\Local\Programs\Microsoft VS Code`.

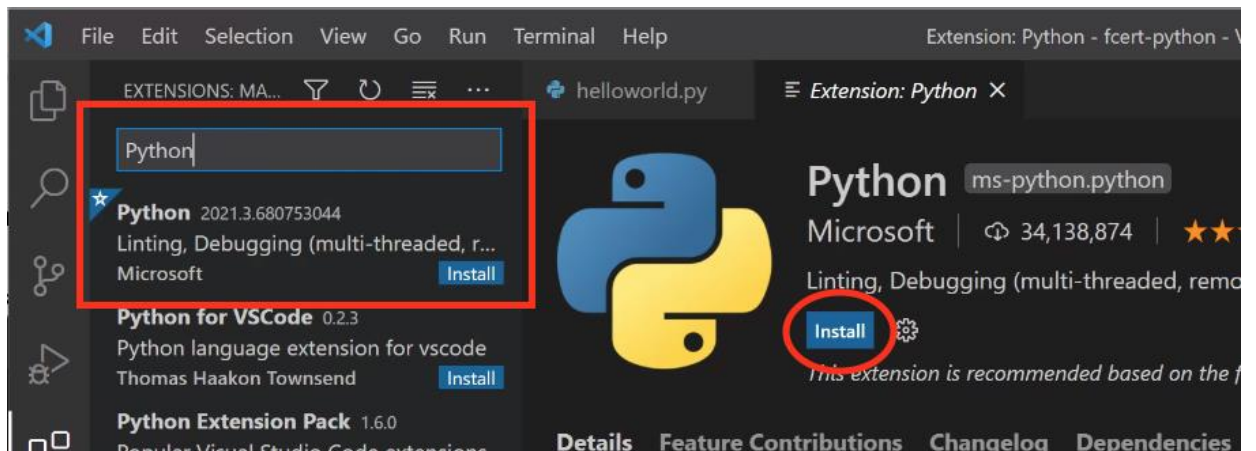
1.1.2 Setup your Visual Studio Code for Python

Steps:

1. Open the “Extension” panel by clicking on the fifth icon on the left.



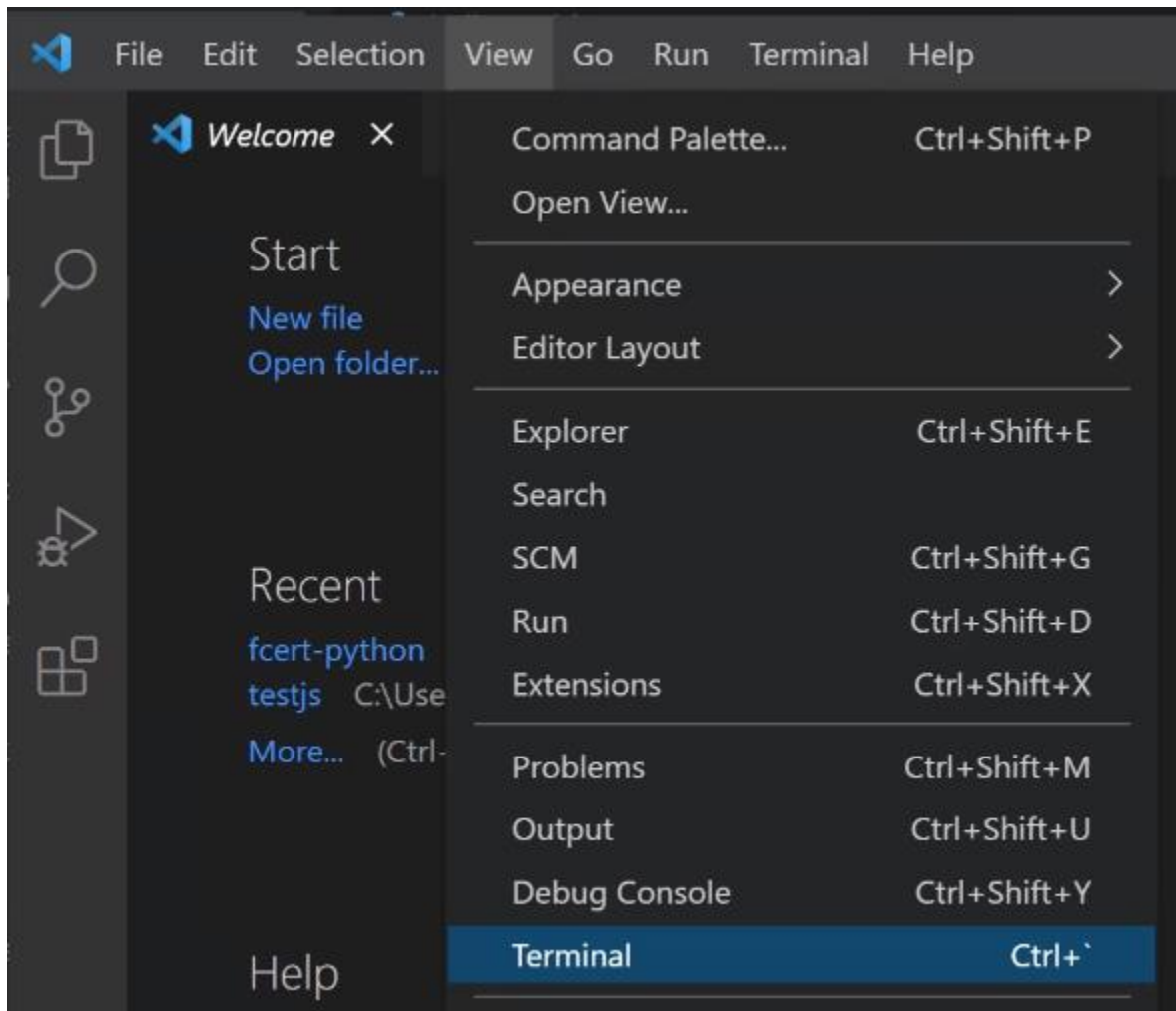
2. In the search bar, type “Python” and install the “Python Extension”.



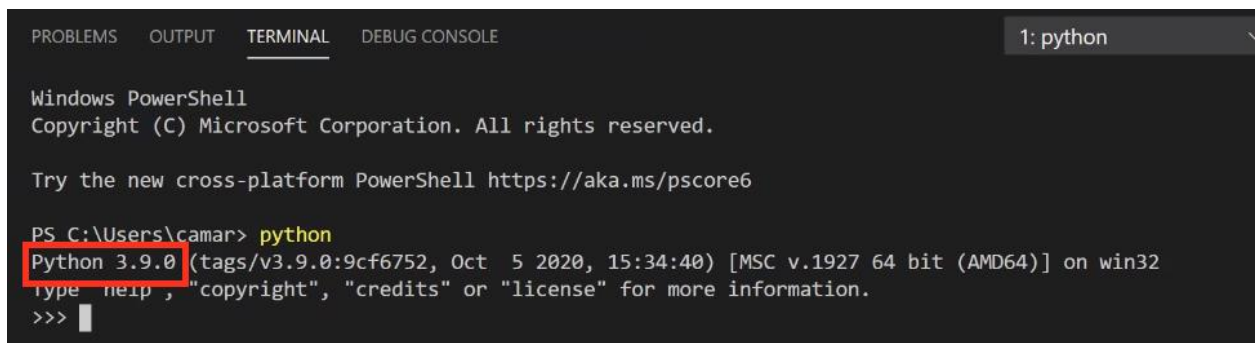
3. Wait for the installation process to finish.

</talentlabs>

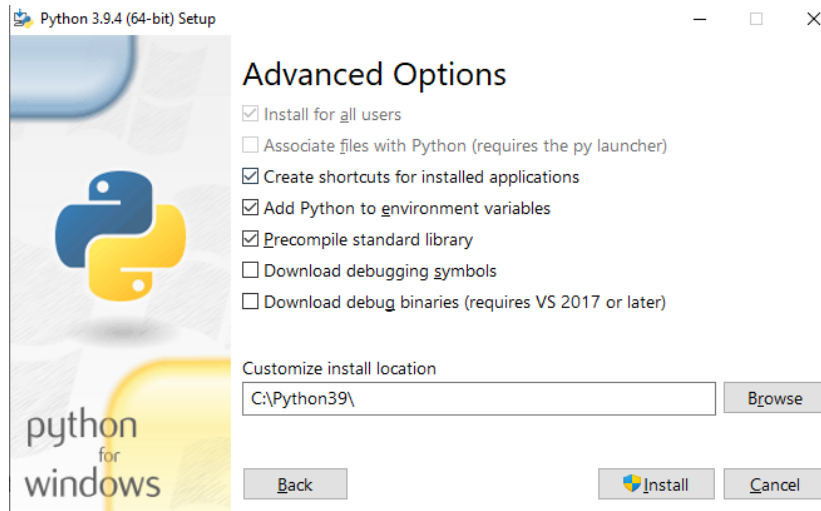
4. Open a terminal in Visual Studio Code by clicking “View -> Terminal”



5. In the terminal opened at the bottom of VSCode, type “python” and press enter. If the setup is successful, then you should see something like the following screenshot (Python 3.x.x).



6. If the above step is not successful, install python in the step 7
7. Download the Python Windows installer here:
<https://www.python.org/ftp/python/3.9.4/python-3.9.4-amd64.exe>
8. In the installation step, click “**Customized Installation**”
9. Make sure you selected “**Add Python to environment variables**”



10. After the installation, **restart the VSCode** and try steps 4 ~ 5 again, and you should see this:

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  1: python

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\camar> python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
type help, "copyright", "credits" or "license" for more information.
>>> 
```

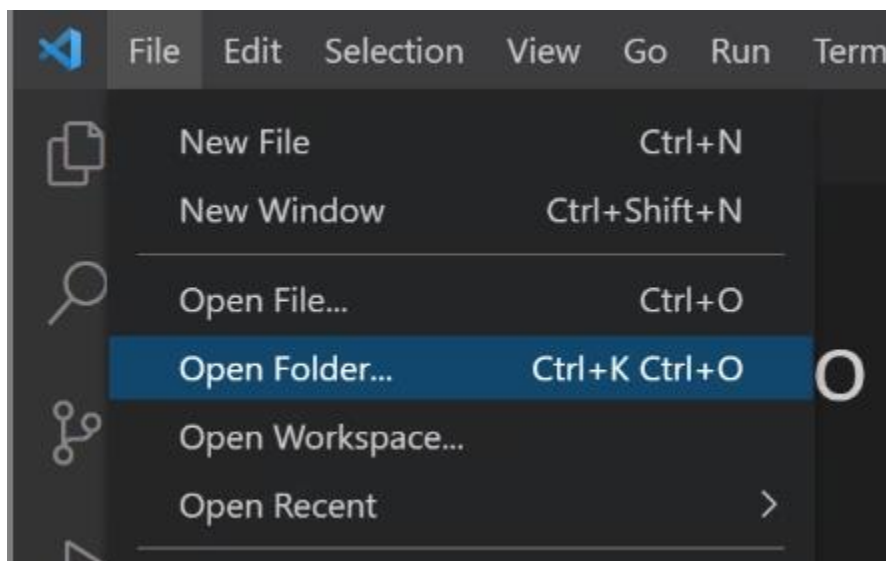
Task 1.1.3 Try running your first Python program in VSCode

Steps:

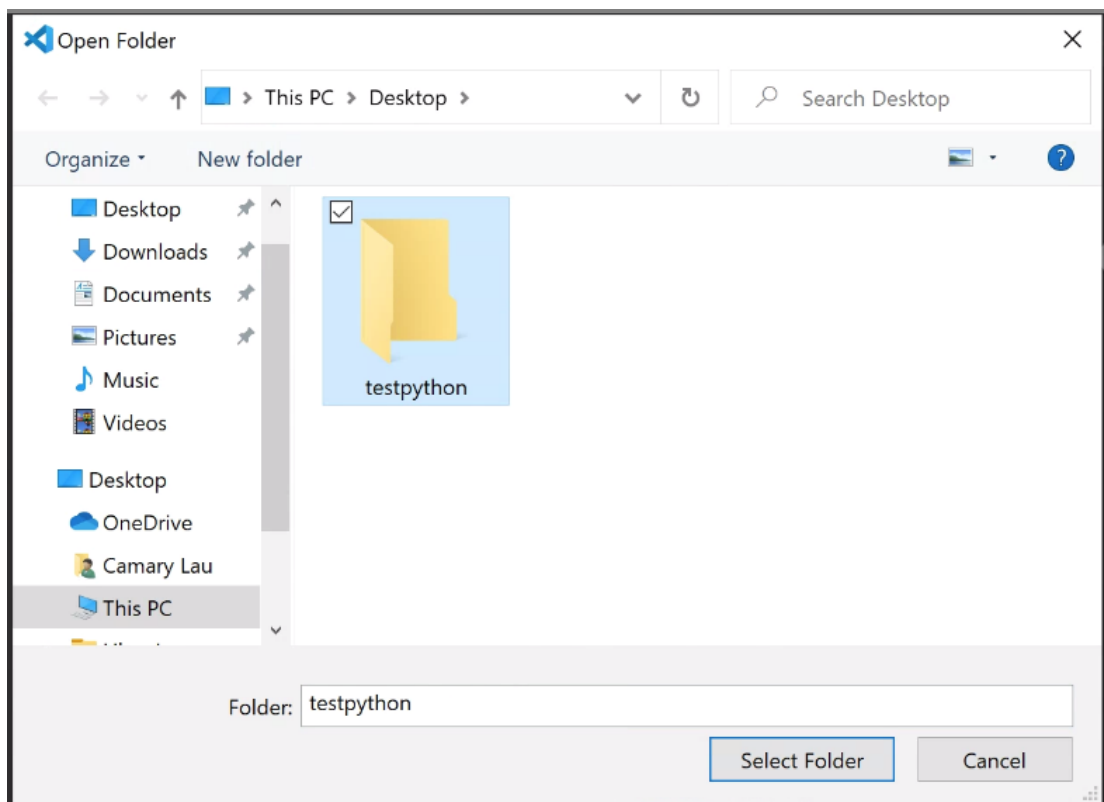
1. Pick any place on your computer, create a new folder, and name it as “testpython”.

</talentlabs>

2. Open the Visual Studio Code software and pick “Open Folder”

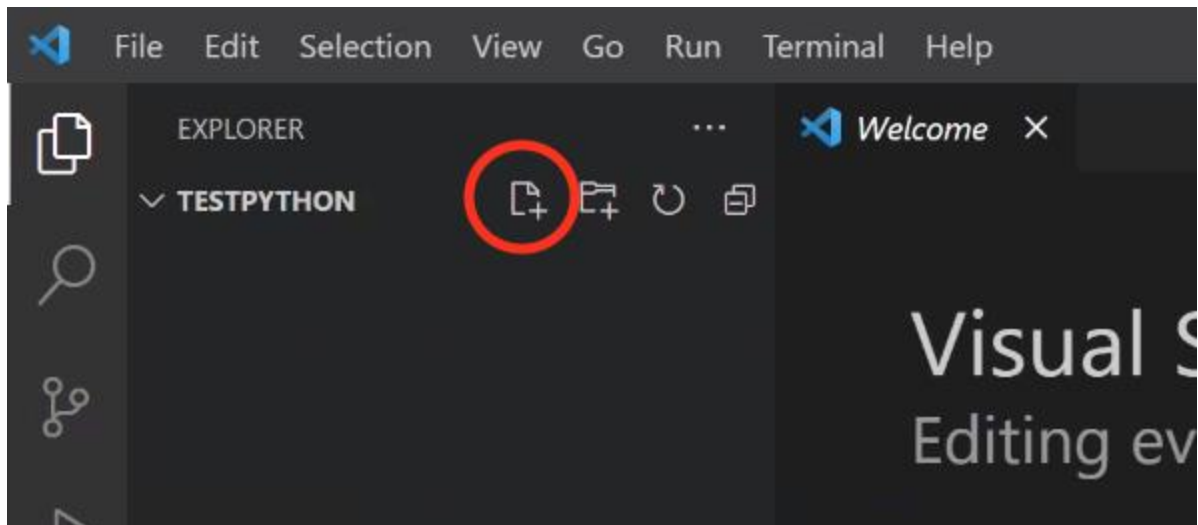


3. Then pick the folder you just created.

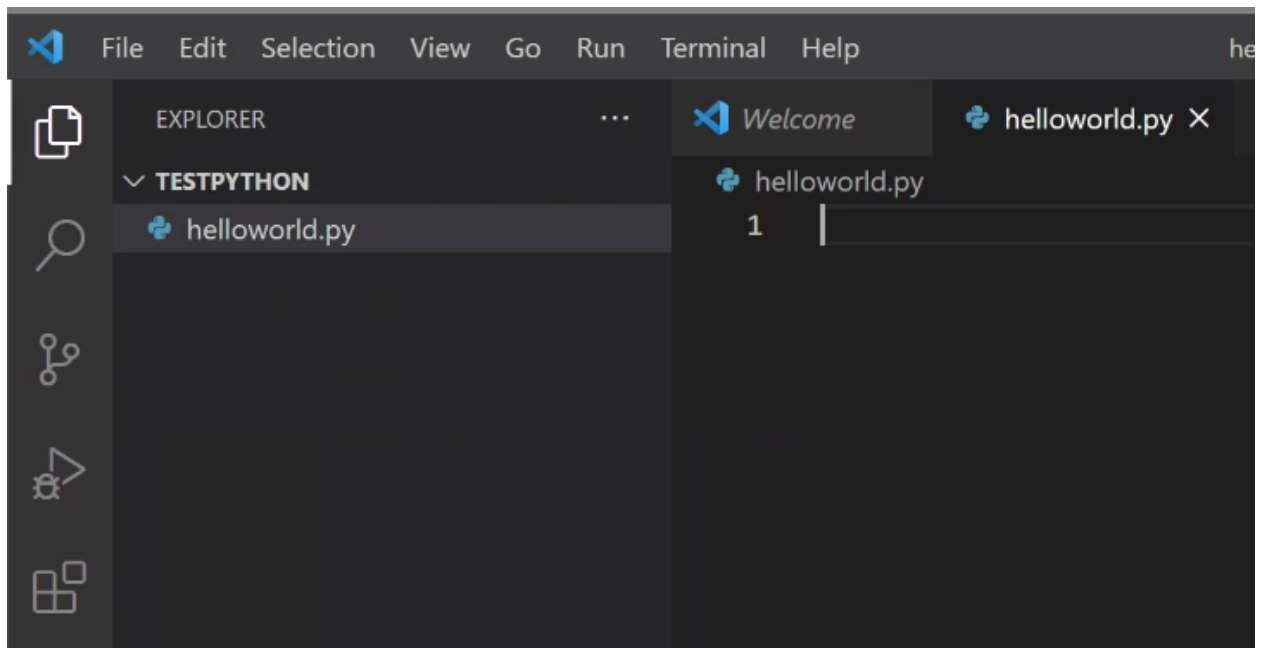


</talentlabs>

4. Create a new file in this folder by clicking the “New File” button.

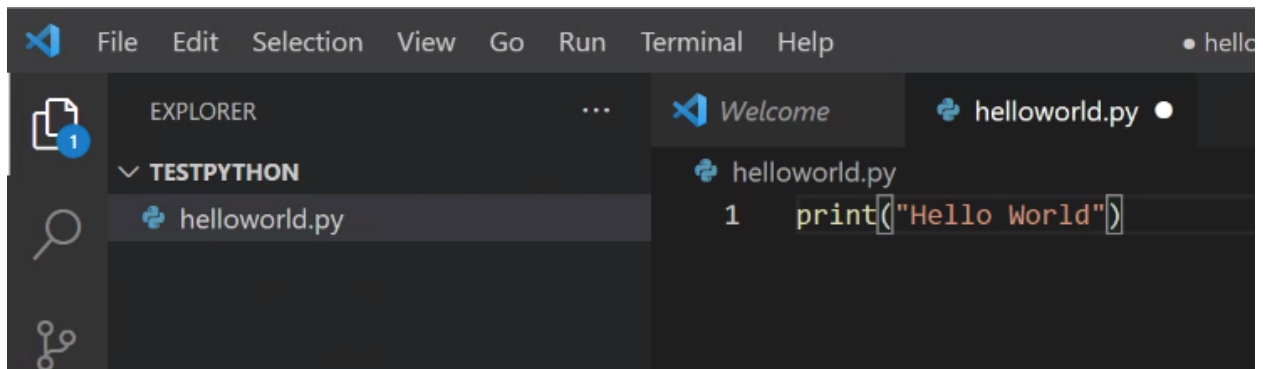


5. Give the new file name “helloworld.py”. It should look like something like the screenshot below.

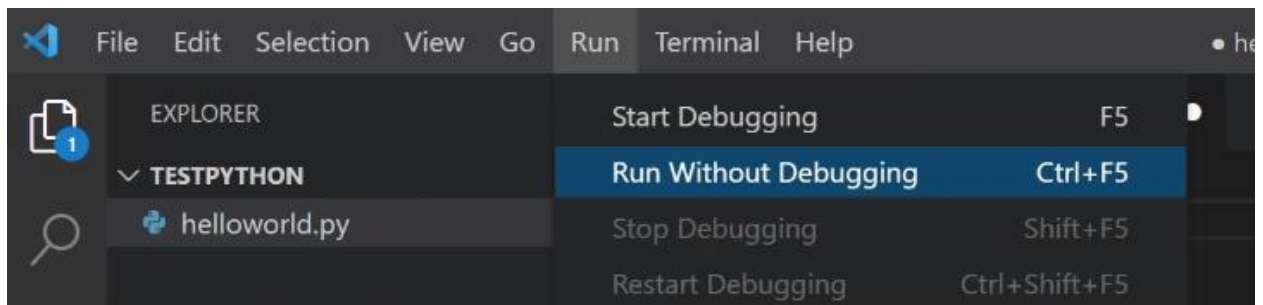


</talentlabs>

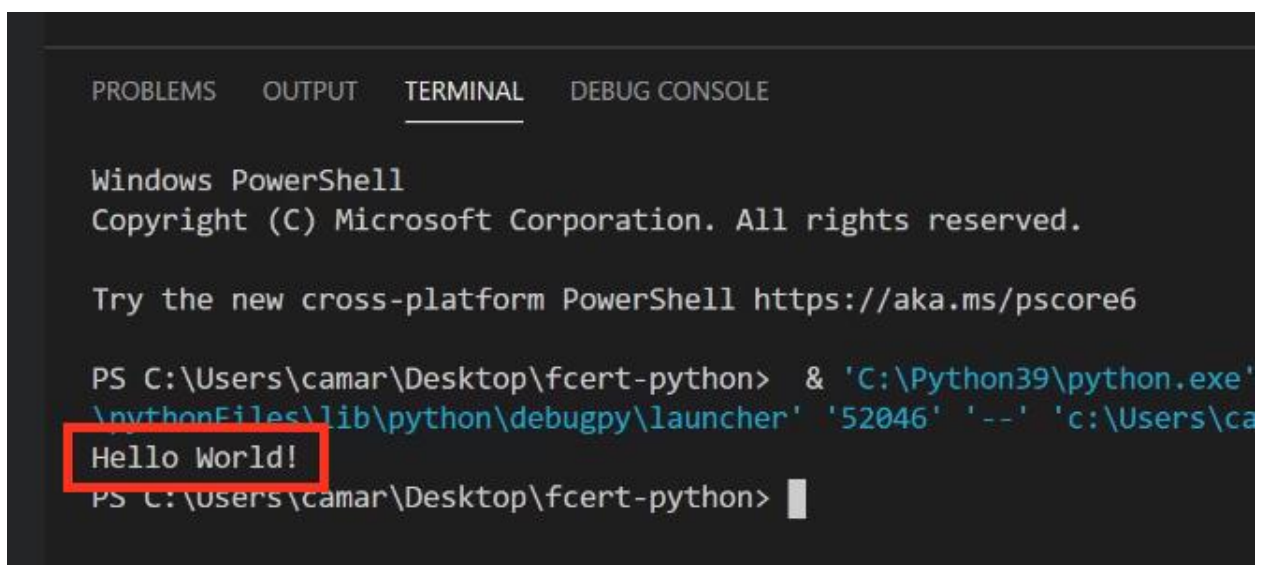
6. In the helloworld.py file, type in the following code.



7. Then click “Run -> Run Without Debugging”



8. If it is successful, then you should see the output at the bottom of the window.



Part 1: Project Introduction

In this project, you are going to build a Tic-Tac-Toe game in the command line interface. Two players, X and O are going to take turns picking their next move by entering the numbers in the command line (See the screenshot on the right-hand side.)

To help you in building the project, we have broken down the project into 2 parts:

- Part 1: To build all the utility functions for this game including:
 - markBoard
 - printBoard
 - validateMove
 - checkWin
 - checkFull, and
 - A constant variable storing all the winning combinations (winCombinations)
- Part 2: To complete the gameplay using the utility functions you built in Part 1.

```
Game started:
1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 9

X's turn, input: 1

X | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 9

O's turn, input: 2

X | O | 3
-----
4 | 5 | 6
-----
7 | 8 | 9

X's turn, input: 
```

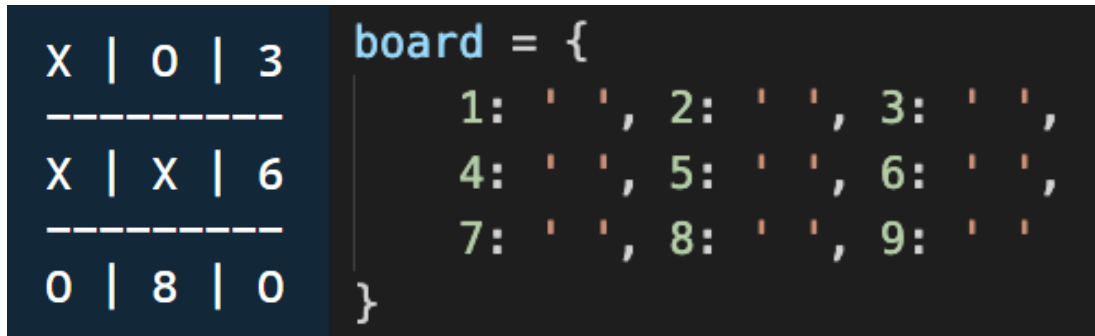
Part 2: Build All the Utility Functions for the Game

2.1 Get familiar with the data structure

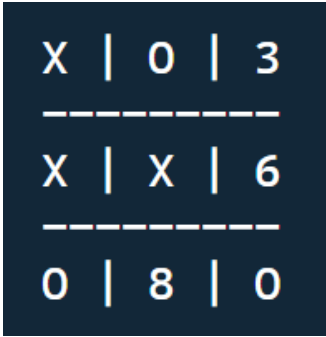
In this project, we are going to represent the tic-tac-toe game board using an object, with 9 properties. Each property represents 1 box in the game board, in sequential order. (See below for the usage of the data structure)

The name of the property represents the position of the box, while the value of the property represents who put their mark in the box (either “X” or “O” or “”).

</talentlabs>



2.2 Get familiar with the utility functions that you are going to build

Name of the Function	Requirements
markBoard(position, mark)	<p>Takes 2 inputs:</p> <ul style="list-style-type: none">position - which is a number representing the box numbermark - the player just picked the box (X or O) <p>This function should fill in the box picked with the mark X or O.</p>
printBoard()	<p>Print out the game board in the format below. If a box is unoccupied, then you should print the number of that box.</p> 
validateMove(position)	<p>This function takes in the position that the player picked as an input, and validates for the following error. This function should return true or false, with true denoting correct input.</p> <ul style="list-style-type: none">Wrong Input (invalid position, not entering a position)Out of bound position (smaller than 1, or larger than 9)The position is already occupied
checkWin	<p>Check if the input user is the winner by returning true or false.</p>

	True denotes the player wins.
checkFull	Check if the game is in a tie situation, i.e. all boxes are occupied with no winner.
winCombinations (not a function, but a constant variable)	List out all 8 winning combinations as an array of arrays. The first one is already done for you

2.3 Learn how to test your functions

We have already created some automated tests for the above function. All you need to do is run the code file, and the output in the console will tell you which test you are failing. It would report one failed test every time.

The test case specifications start from line 62 of the tic-tac-toe_Part1.py file. We built the automated tests using a Python Module named “unittest”. You don’t have to worry about the setup of it, you only need to know how to read the test cases.

Reading the test cases:

The diagram illustrates how to read a test case in Python code. It shows a code snippet with three annotations:

- Test the return value:** A green arrow points from the text to the `checkWin('X')` expression in the `tc.assertEqual` call.
- Compare it to the expected value:** A blue arrow points from the text to the `False` value in the `tc.assertEqual` call.
- If they are not equal, raise an error with error message:** A red arrow points from the text to the error message string `"checkWin() didn't work as expected with input : 'X'"` in the `tc.assertEqual` call.

```

74 ~ testBoard = {
75     1: 'X', 2: 'O', 3: 'X',
76     4: 'O', 5: 'X', 6: 'O',
77     7: ' ', 8: ' ', 9: ' '
78 }
79 tc.assertEqual(checkWin('X'), False, "checkWin() didn't work as expected with input : 'X'")

```

Reading the failing test cases:

The diagram shows a traceback of a failing test case with three annotations:

- The value returned by your function:** A blue arrow points from the text to the `None` value in the `AssertionError: None != False` message.
- The expected value returned by your function:** A green arrow points from the text to the `False` value in the `AssertionError: None != False` message.
- error: Line 68:** An orange box highlights the line number `line 68` in the traceback.

```

Traceback (most recent call last):
  File "/Users/darrenchiu/Development/fcert-python/project1/tic-tac-toe_Part1.py", line 68, in <module>
    tc.assertEqual(validateMove(0), False, "validateMove() didn't work as expected with input : 0")
  File "/usr/local/Cellar/python@3.9/3.9.5/Frameworks/Python.framework/Versions/3.9/lib/python3.9/unittest
    assertion_func(first, second, msg=msg)
  File "/usr/local/Cellar/python@3.9/3.9.5/Frameworks/Python.framework/Versions/3.9/lib/python3.9/unitt
    raise self.failureException(msg)
AssertionError: None != False

```

2.4 Implementing the Part 1 of this project

Steps:

1. Open the `project1` file.
2. Start implementing the functions in `tic-tac-toe_Part1.py` following the sequence below.
3. After implementing each of the following, run the test to make sure all the test cases of that function are passed. You can run the tests by running `tic-tac-toe_Part1.py` in the command line.
 - `validateMove`
 - `markBoard`
 - `winCombinations` and `checkWin`
 - `checkFull`
 - `printBoard` (No automated test cases)
4. If you are seeing "All tests passed! Congratulations!" in the console output, then you have successfully implemented all the functions. Congratulations!

Part 3: Build the Game Play

In this part, you are going to implement the tic-tac-toe gameplay using the utility functions that you implemented in Part 1. Please follow the following steps for the setup.

3.1 Copy the utility functions you created to the project file

Steps:

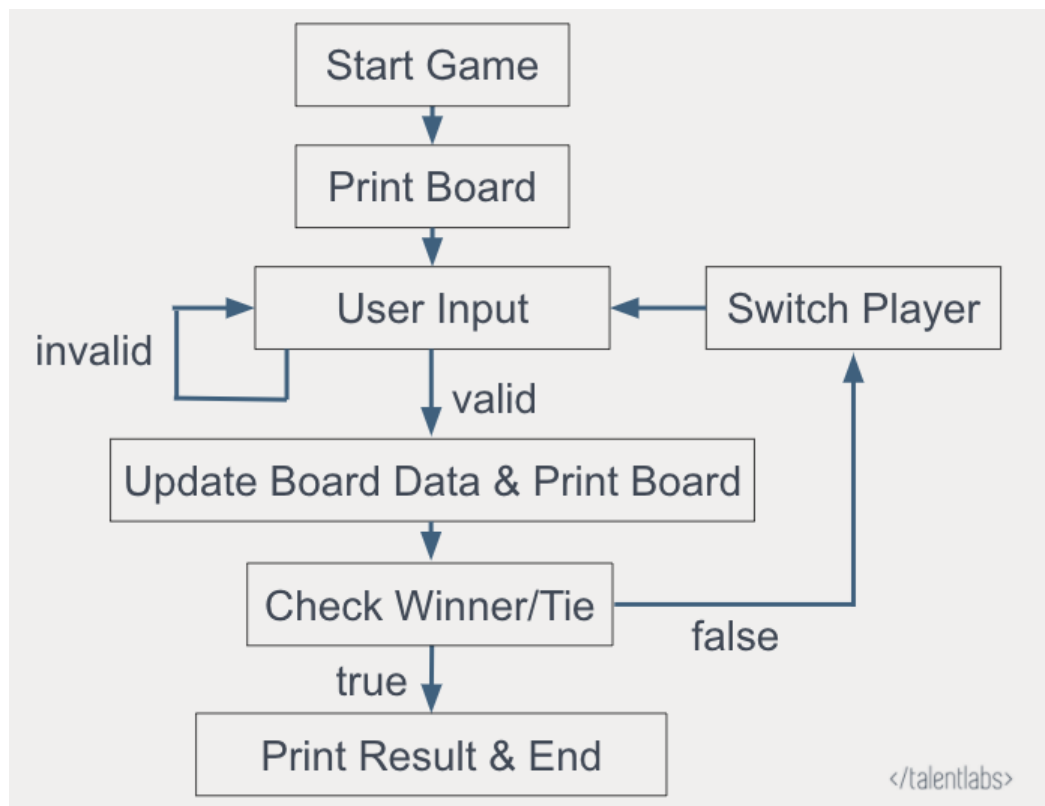
1. Open the `tic-tac-toe.py` file in the project folder in VSCode.
2. Replace all the functions in this file with the functions you implemented in Part 1 (i.e. remove lines 1-63, and replace with the code from `tic-tac-toe_Part1.py` without test cases)

3.2 Implement the gameplay

Basically, you will need to complete the gameplay implementation using the functions you created in Part 1.

Try playing around with the game after finishing the game to make sure it works for all cases (wrong input, winning scenarios, and tie scenarios)

If you are not too sure about the logic, you can take a look at the flowchart below. The flow chart described a recommended control flow of the game. We have already included some hints in the while loop at line 82.



3.3 Bonus Point (Optional)

If you want to challenge yourself a bit, feel free to implement the restart feature for your game. Your game should ask users if they want to restart the game or not. Users could respond by typing Y or N, and your program should respond accordingly.