How to Python in VS Code

Redowan Delowar

Oct 25, 2019

CONTENTS

Although I've been tinkering with python for my Data Science projects since 2016, I only started coding professionally at the end of 2018. It wasn't easy to get accustomed to the workflow by any means and the rigor of a production and test driven environment was something completely different from what I was used to . I had only been using all the tools and packages integrated into the amazing Anaconda Distribution which means Spyder as the IDE and the trusted old Jupyter Notebook for experimentation. I still use jupyter notebook, however, I've picked up **VS Code** as my primary editor not only due to the fact that everyone at my workplace uses that but also in my opinion it's one of the best language agnostic code editor, period . So this post is an assortment of all the tools and practices that I've picked up throughout my development journey.

Here I'm using Linux as my primary development OS and many of the instructions apply directly to MacOS also. However, if you are doing python in VS Code on MacOS or Windows, I encourage you contribute and extend this guideline. Now let's jump in .

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Setting up VS Code on Your Machine

1.1.1 Installing VS Code

Installing VS code is easy across all operating systems.

- Just go to the link here and select the distribution that corresponds to your OS.
- Download the file and install

1.1.2 Running VS Code

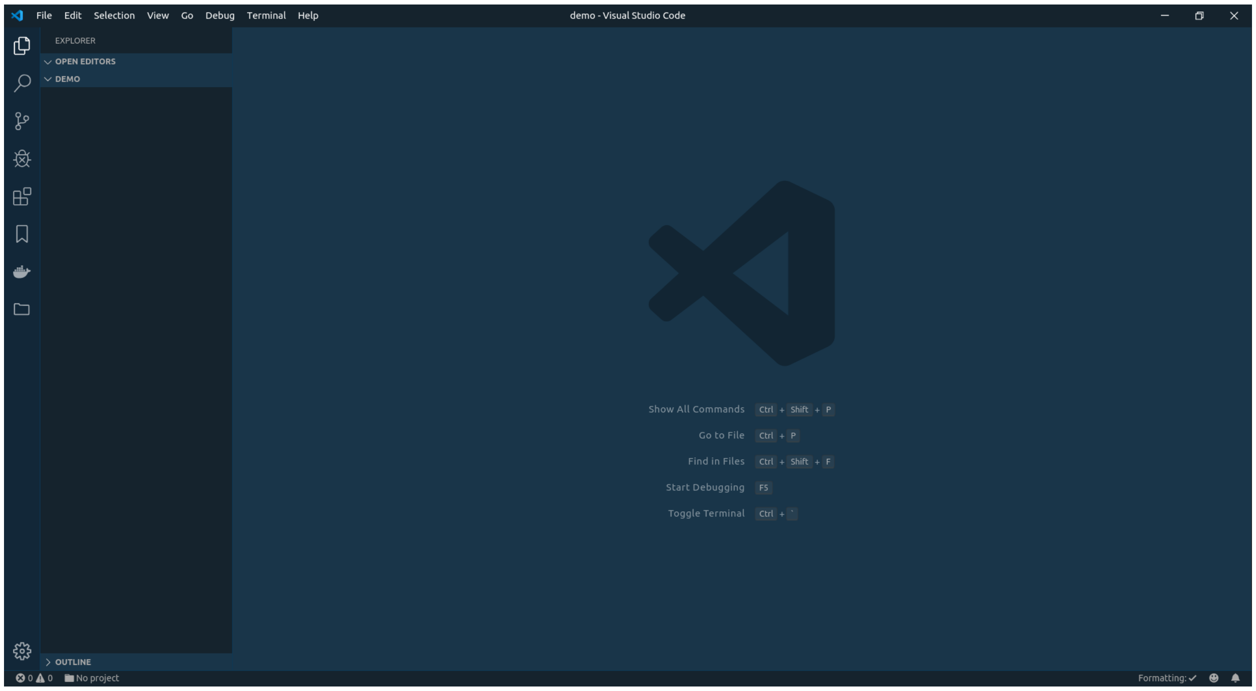
- After you've completed the installation, open terminal and `cd` to your project folder. Then type:

```
$ code .
```

This will open your project in VS Code. Familiarize yourself with the GUI if you're using this for the first time. The left most stripe gives you multiple options and hovering over the icons will show you their corresponding functionalites. If you're using any third party extensions, their icons may also appear. Clicking on each of them will bring up extra actions in the succeeding column.

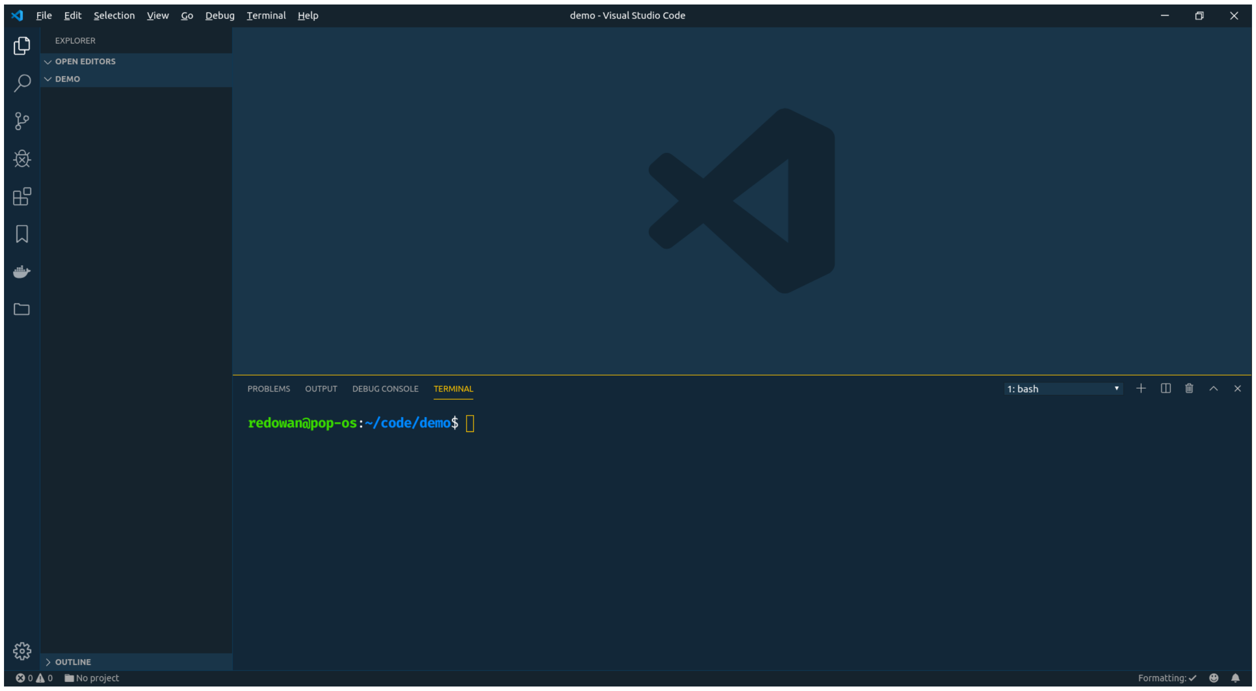
The default icons are (from top to bottom):

- Explorer
- Search
- Source Control (Git)
- Debug
- Extensions


File Edit Selection View Go Debug Terminal Help
demo - Visual Studio Code
EXPLORER
OPEN EDITORS
DEMO
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl + `
OUTLINE
No project
Formatting: ✓

1.1.3 Using the Integrated Terminal

If you aren't migrating from any other terminal and haven't set up your preferred `keybindings` then you can open the integrated terminal by pressing `ctrl + ~` on your keyboard. This terminal is an exact replica of your `bash`/`zsh` terminal and can perform almost anything that you'd normally do in those. From now an on, unless explicitly mentioned otherwise, we'll be using the integrated terminal for the versatility and convenience it provides.



```
redowan@pop-os:~/code/demo$
```

1.2 Setting up Environments

1.2.1 Why Environments are Necessary

The main purpose of using environments is to create a segregation between the dependencies of different python projects. It eliminates (at least tries to) dependency conflicts since each project has it's own set of dependencies, isolated from one another.

Suppose you are working on two projects, `Project_1` and `Project_2`, both of which have a dependency on the same library, let's say the awesome `Requests` library. Dependency conflict will arise, if for some reason, the two projects need different versions of `Request` library. For example, maybe `Project_1` needs `v1.0.0`, while `Project_2` requires the newer `v2.0.0`.

This can easily be avoided by using individual environment for each project where all the dependencies of the corresponding project will reside. There are multiple ways you can create environment. We'll mainly focus on creating python3 based `conda environment` and native `virtual environment`.

1.2.2 Conda Environment

Installing Anaconda Distribution

Install anaconda on your machine. I personally prefer miniconda over the full fledged anaconda. The installation guide can be found here:

- Linux
- MacOS

Creating Conda Environment

After installing anaconda, to create a python3 environment with a specific version of python, type the following command. This will create an environemnt named `myenv` with python 3.7:

```
$ conda create -n myenv python=3.7
```

Activating Conda Environment

After creating the conda environment, type the folling command to activate the `myenv` environment:

```
$ conda activate myenv
```

Deactivating Conda Environment

To deactivate, simply type:

```
$ conda deactivate
```

1.2.3 Virtual Environment

Installing python3-venv

To create virtual environment, first you need to install `python3-venv`. Run:

```
$ sudo apt update
$ sudo apt-get install python3-venv
```

Creating Virtual Environments

Create a virtual environment named `myenv` via running:

```
$ python3 -m venv myenv
```

You should see a folder named `myenv` in your current directory. This is the folder where all your project-specific dependencies are going to reside.

Activating Virtual Environments

To activate `myenv`, run:

```
$ source myenv/bin/activate
```

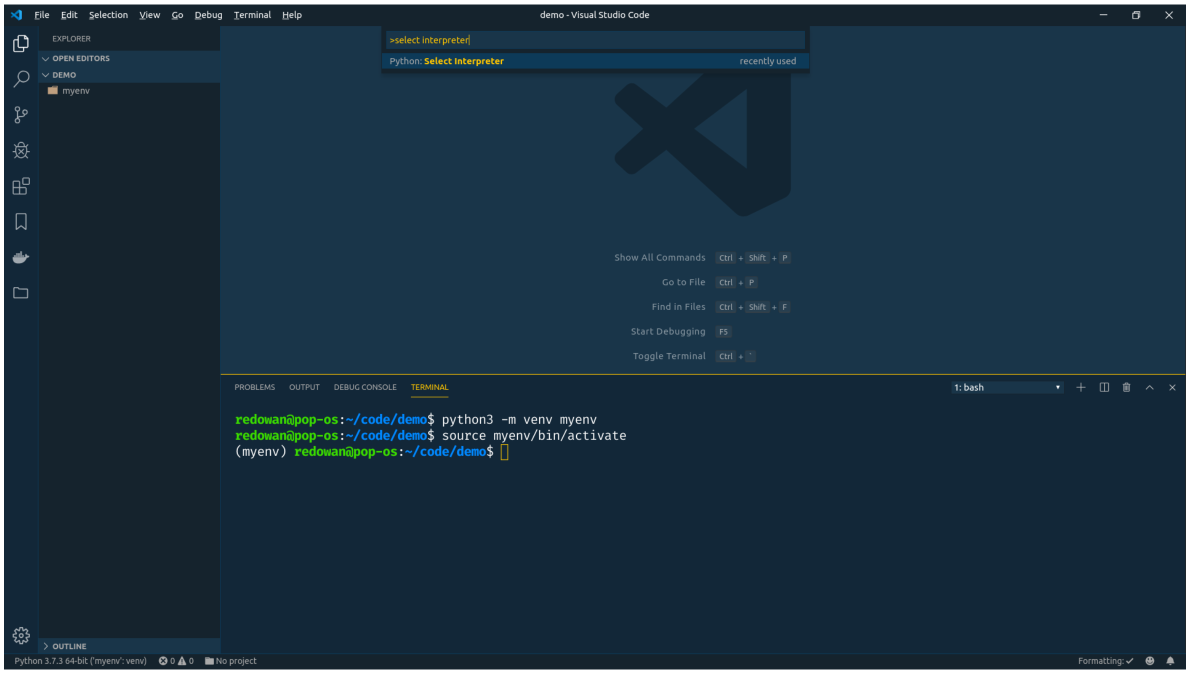
Deactivating Virtual Environment

To deactivate, simply type:

```
$ deactivate
```

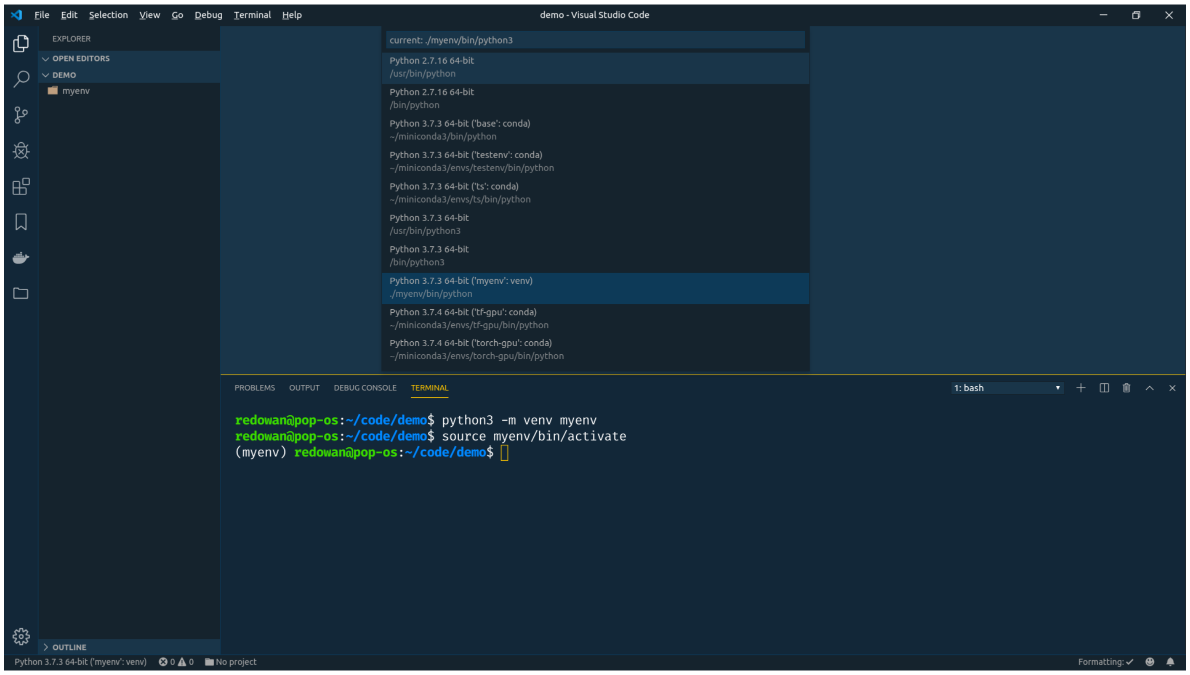
1.2.4 Selecting & Switching Between the Environments in VS Code

- Press `ctrl+shift+P` to open VS Code's `command palette`. You should be seeing something like this:


File Edit Selection View Go Debug Terminal Help
demo - Visual Studio Code
>select interpreter
Python: Select Interpreter
recently used
EXPLORER
OPEN EDITORS
DEMO
myenv
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl + `
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
OUTLINE
Python 3.7.3 64-bit ('myenv': venv)
No project
Formatting: ✓

```
redowan@pop-os:~/code/demo$ python3 -m venv myenv
redowan@pop-os:~/code/demo$ source myenv/bin/activate
(myenv) redowan@pop-os:~/code/demo$
```

- Type `interpreter` in the search box. And select the `Python: Select Interpreter` option. You should see a list of all the available (both conda and virtual environments are shown) python environments. You should also see your recently created `myenv` environment there. Toggle and select your environment and you are good to go.


File Edit Selection View Go Debug Terminal Help
demo - Visual Studio Code
EXPLORER
OPEN EDITORS
DEMO
myenv
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
OUTLINE
Python 3.7.3 64-bit ('myenv': venv)
No project
Formatting: ✓

```
current: ./myenv/bin/python3
Python 2.7.16 64-bit
/usr/bin/python
Python 2.7.16 64-bit
/bin/python
Python 3.7.3 64-bit ('base': conda)
~/miniconda3/bin/python
Python 3.7.3 64-bit ('testenv': conda)
~/miniconda3/envs/testenv/bin/python
Python 3.7.3 64-bit ('ts': conda)
~/miniconda3/envs/ts/bin/python
Python 3.7.3 64-bit
/usr/bin/python3
Python 3.7.3 64-bit
/bin/python3
Python 3.7.3 64-bit ('myenv': venv)
./myenv/bin/python
Python 3.7.4 64-bit ('tf-gpu': conda)
~/miniconda3/envs/tf-gpu/bin/python
Python 3.7.4 64-bit ('torch-gpu': conda)
~/miniconda3/envs/torch-gpu/bin/python
```

```
redowan@pop-os:~/code/demo$ python3 -m venv myenv
redowan@pop-os:~/code/demo$ source myenv/bin/activate
(myenv) redowan@pop-os:~/code/demo$
```

1.2.5 Installing Third Party Packages

To install third party packages/libraries/moduels from `pip` or `conda`,

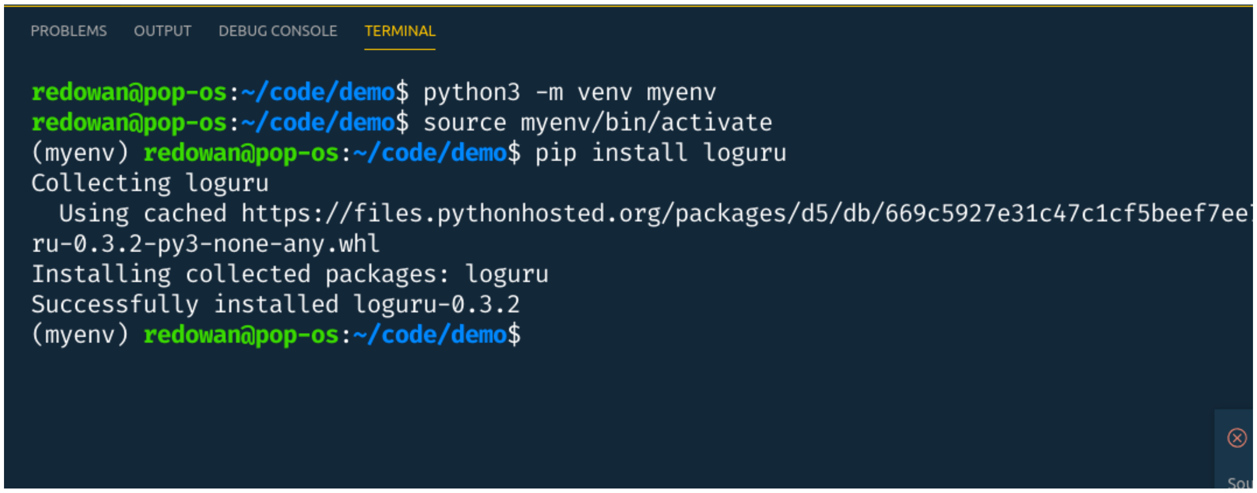
- Activate your respective environments

- Depending on the package manager you want to use, type either:

```
$ pip install <package_name>
```

or

```
conda install <package_name>
```

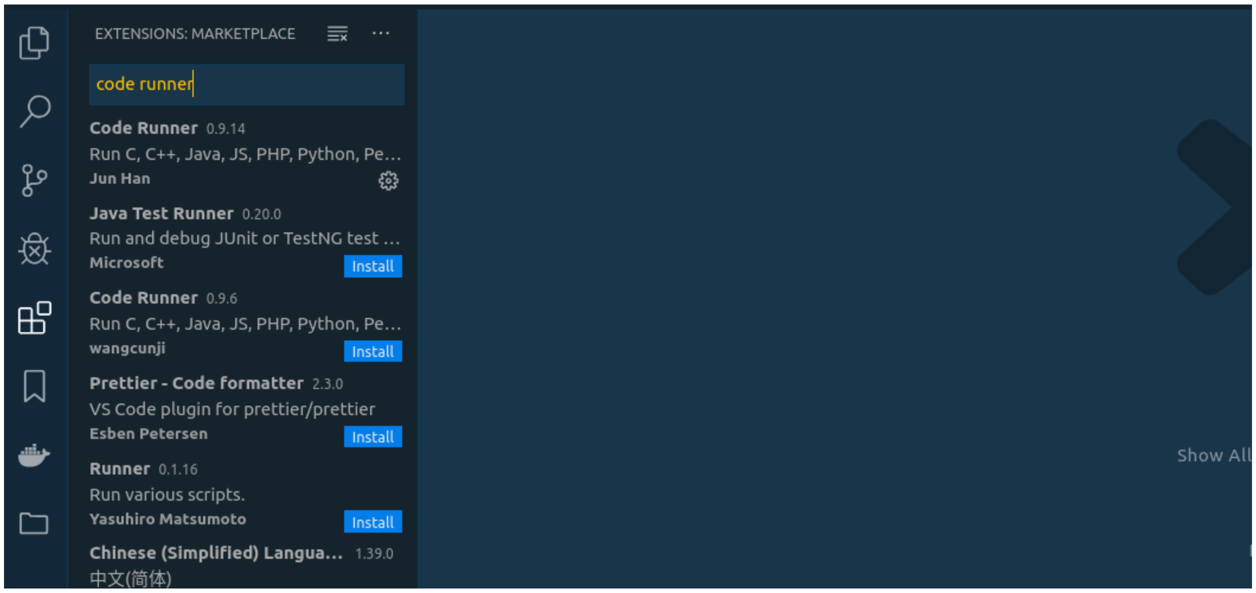


```
redowan@pop-os:~/code/demo$ python3 -m venv myenv
redowan@pop-os:~/code/demo$ source myenv/bin/activate
(myenv) redowan@pop-os:~/code/demo$ pip install loguru
Collecting loguru
  Using cached https://files.pythonhosted.org/packages/d5/db/669c5927e31c47c1cf5beef7ee
ru-0.3.2-py3-none-any.whl
Installing collected packages: loguru
Successfully installed loguru-0.3.2
(myenv) redowan@pop-os:~/code/demo$
```

1.3 Running Python Scripts with Code Runner

1.3.1 Installing Code Runner Extension

Click the extension icon on the left most stripe and type `code runner` in the search bar. You should see the extension popping up in the first row. Click and install.



Code Runner 0.9.14
Run C, C++, Java, JS, PHP, Python, Pe...
Jun Han

Java Test Runner 0.20.0
Run and debug JUnit or TestNG test ...
Microsoft

Code Runner 0.9.6
Run C, C++, Java, JS, PHP, Python, Pe...
wangcunji

Prettier - Code formatter 2.3.0
VS Code plugin for prettier/prettier
Esben Petersen

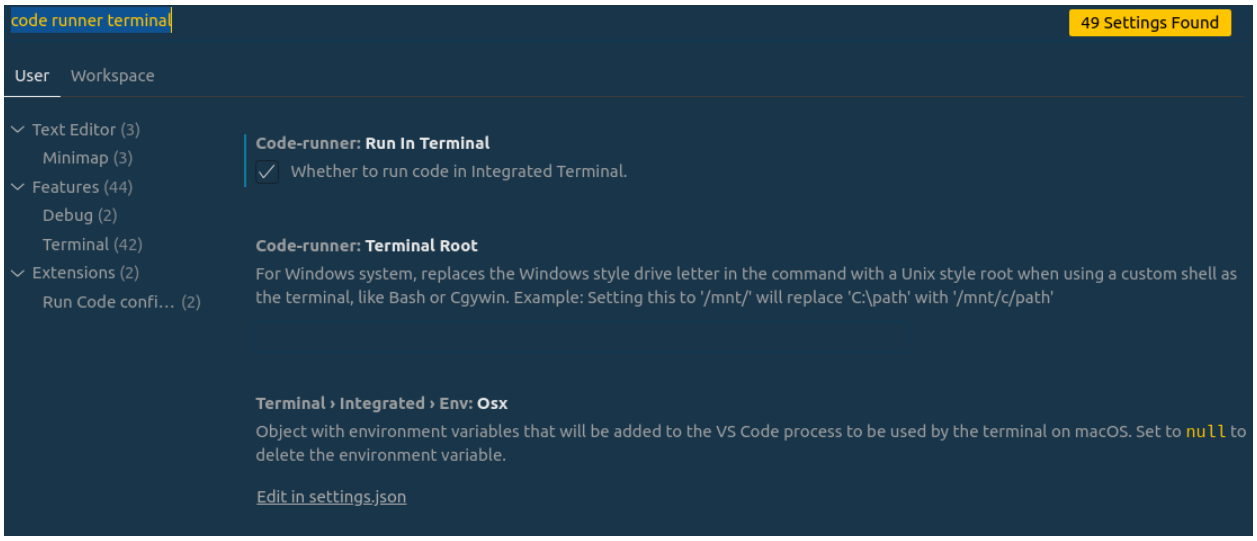
Runner 0.1.16
Run various scripts.
Yasuhiro Matsumoto

Chinese (Simplified) Langua... 1.39.0
中文(简体)

1.3.2 Setting up Code Runner

By default, code runner uses its own panel for showing the results after you run your python script. However, it's better to set it up in a way that it will show the results in the integrated terminal.

- Press `ctrl+,` to open up the settings panel
- On the search bar type `code runner terminal`
- You should be able to see an option named `Code-runner: Run In Terminal`
- Check off the option and you are good to go


code runner terminal

49 Settings Found

User Workspace

- Text Editor (3)
  - Minimap (3)
- Features (44)
  - Debug (2)
  - Terminal (42)
- Extensions (2)
  - Run Code confi… (2)

Code-runner: Run In Terminal
✓ Whether to run code in Integrated Terminal.

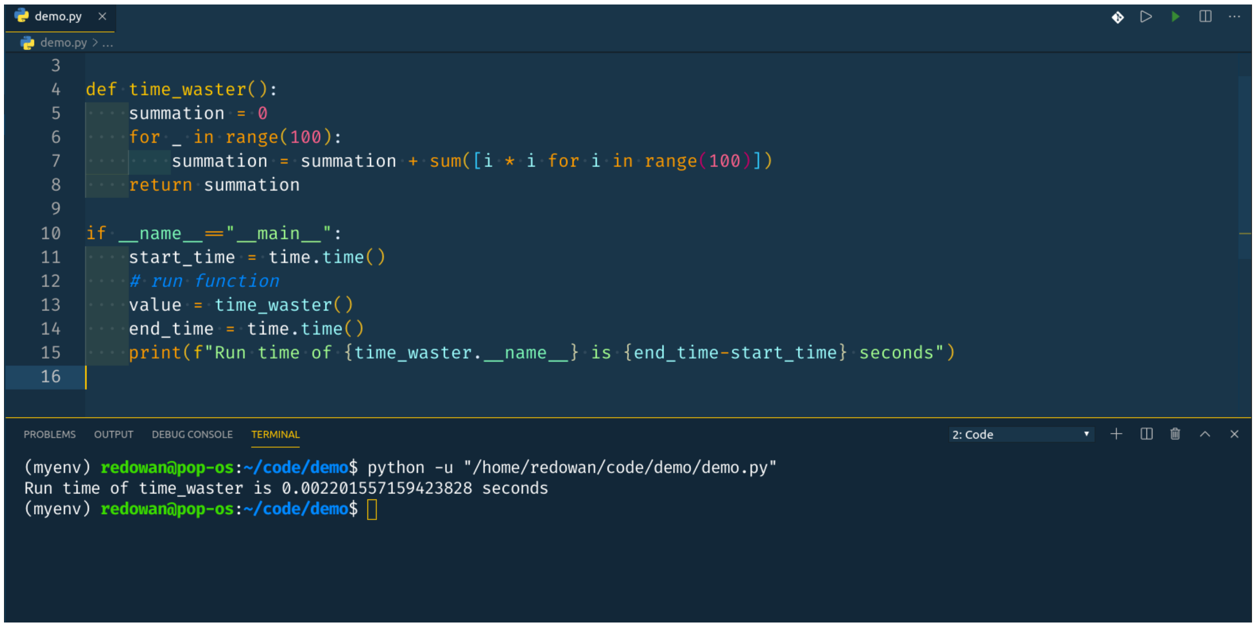
Code-runner: Terminal Root
For Windows system, replaces the Windows style drive letter in the command with a Unix style root when using a custom shell as the terminal, like Bash or Cgywin. Example: Setting this to '/mnt/' will replace 'C:\path' with '/mnt/c/path'

Terminal › Integrated › Env: Osx
Object with environment variables that will be added to the VS Code process to be used by the terminal on macOS. Set to `null` to delete the environment variable.

Edit in settings.json

1.3.3 Running Python Scripts

- Create and select your python environment (See the instructions here.)
- Create a new file via `ctrl+N`
- Press `ctrl+s` to save the file and give it a name with `.py` extension
- Write down your python code in the file
- Press `ctrl+alt+N` to run the code via Code Runner
- You should see your results in the integrated terminal
- To run only a selected lines of codes, select the lines you want to run and press `ctrl+alt+N`.



```python

def time_waster():
    summation = 0
    for _ in range(100):
        summation = summation + sum([i * i for i in range(100)])
    return summation

if __name__=="__main__":
    start_time = time.time()
    # run function
    value = time_waster()
    end_time = time.time()
    print(f"Run time of {time_waster.__name__} is {end_time-start_time} seconds")

```

```
(myenv) redowan@pop-os:~/code/demo$ python -u "/home/redowan/code/demo/demo.py"
Run time of time_waster is 0.002201557159423828 seconds
(myenv) redowan@pop-os:~/code/demo$
```

1.4 Linting & Formatting

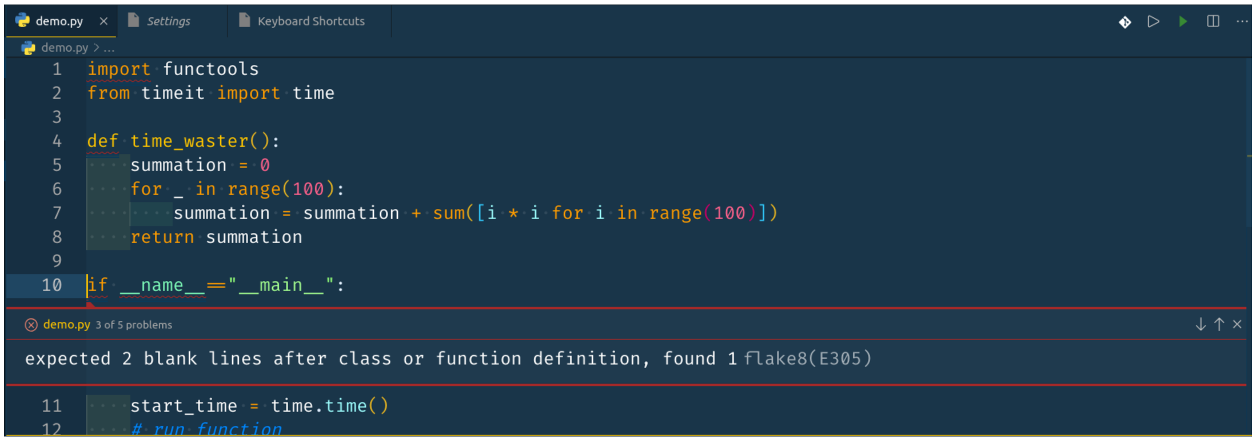
Linters perform static analysis of source codes and check for symantic discrepancies. When you lint your code, it's passed through a basic quality checking tool that provides instructions on how eliminate basic syntactic inconsistencies.

Formatters are similar tools that tries to restructure your code spacing, line length, argument positioning etc to ensure that your code looks consistent across different files or projects.

Python offers you a plethora of linters and formatters to choose from. Flake8, pyflakes, pycodestyle, pylint are some of the more widely used linters and black, yapf are two newer members in the code formatting space. However, not to bombard you with a deluge of information, we are taking an opinionated route that gets the job done without a hitch. Let's talk about `Flake8` and `Black`.

1.4.1 Flake8

Flake8 is a Python linting library that basically wraps three other linters, **PyFlakes**, **pycodestyle** and **Ned Batchelder's McCabe** Script. It's one of the better linters out there that has very low false positive rate. It checks your code base against PEP8 programming style, programming errors (like "library imported but unused" and "Undefined name") and cyclomatic complexity.



```
import functools
from timeit import time

def time_waster():
    summation = 0
    for _ in range(100):
        summation = summation + sum([i * i for i in range(100)])
    return summation

if __name__=="__main__":
```

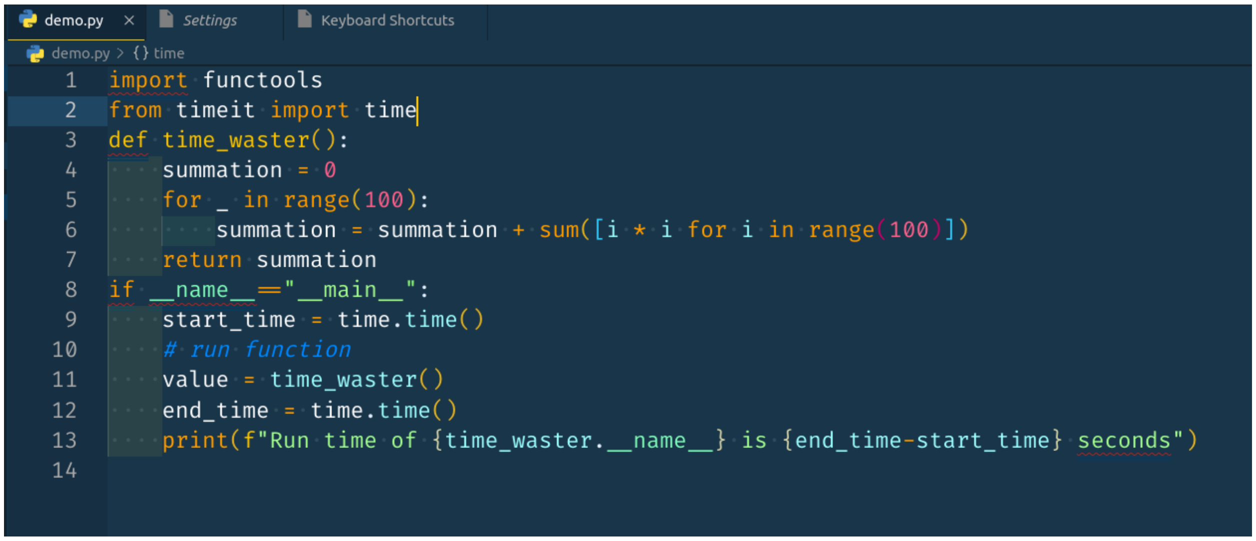
```
expected 2 blank lines after class or function definition, found 1 flake8(E305)
```

```
    start_time = time.time()
    # run function
```

For more details on the nitty gritties of flake8, check out their github project here.

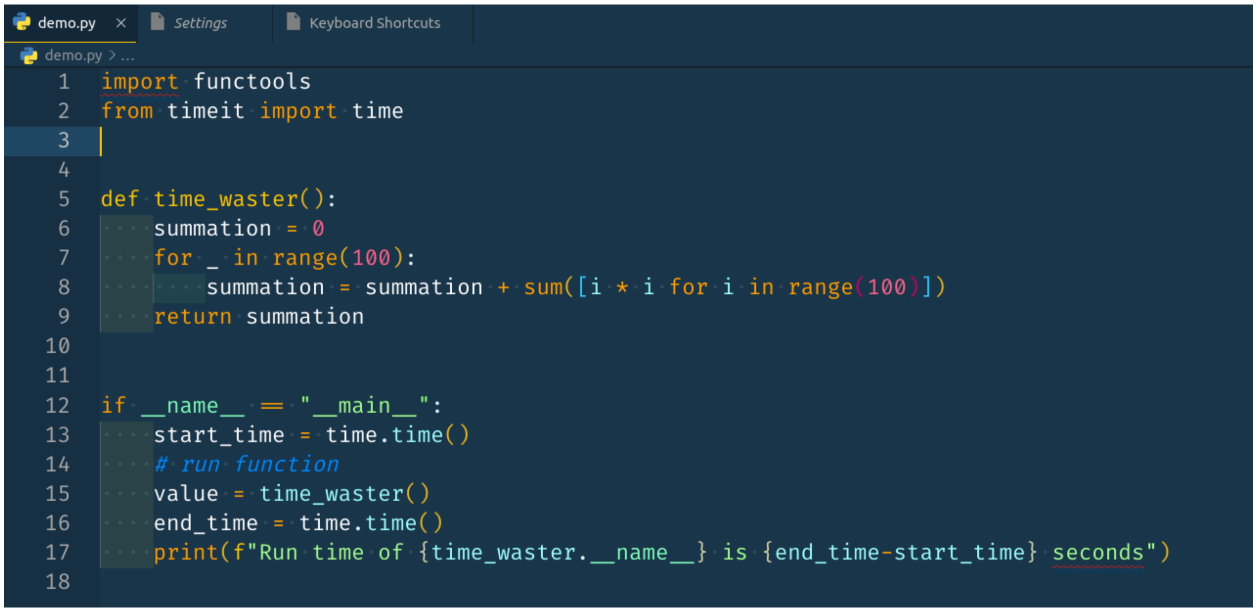
1.4.2 Black

Black is known as the uncompromised Python code formatter. Unlike flake8 or pycodestyle, it doesn't nag you when there are style inconsistencies. It just fixes them for you. Black does not have a lot of options to tinker with and has a lot of opinion on how your code should look and feel. You might not always agree with the decisions that black takes for you but if you can get along with the style that black imposes on you, it can take care of the unnecessary hassles of formatting your codes to keep it conistent across multiple projects or organization.



```
import functools
from timeit import time
def time_waster():
    summation = 0
    for _ in range(100):
        summation = summation + sum([i * i for i in range(100)])
    return summation
if __name__=="__main__":
    start_time = time.time()
    # run function
    value = time_waster()
    end_time = time.time()
    print(f"Run time of {time_waster.__name__} is {end_time-start_time} seconds")
```

Before formatting with black



demo.py × | Settings | Keyboard Shortcuts

demo.py > ...

```python
import functools
from timeit import time


def time_waster():
    summation = 0
    for _ in range(100):
        summation = summation + sum([i * i for i in range(100)])
    return summation


if __name__ == "__main__":
    start_time = time.time()
    # run function
    value = time_waster()
    end_time = time.time()
    print(f"Run time of {time_waster.__name__} is {end_time-start_time} seconds")
```

After formatting with black

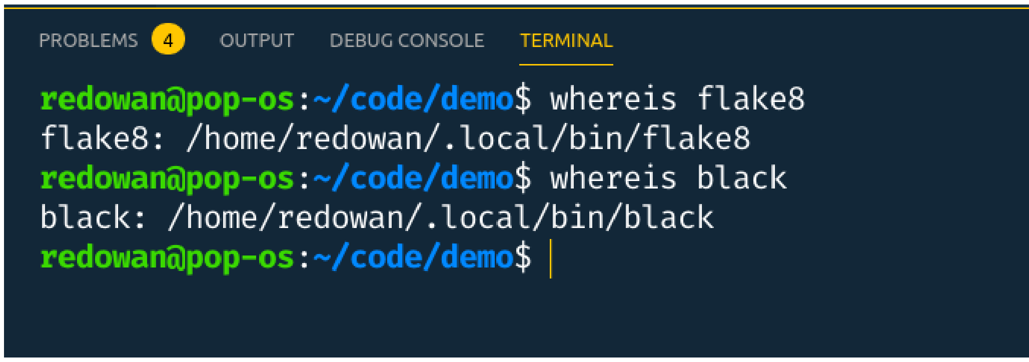
1.4.3 Setting Up Linters in VS Code

Luckily VS Code comes with both `flake8` and `black` formatter lurking in the settings. To set them up:

- Press `ctrl+,` to fire up the settings panel
- Search for `flake8` in the search panel
- Enable the option `Python>Linting:Flake8 Enabled`
- Search for black and select `black` from the dropdown called `Python>Formatting:Provider`

Doing the above will set `flake8` and `black` to lint and format your script on a project basis. You have to install `flake8` and `black` in your environment via `pip install flake8` and `pip install black` respectively. If you want to set them up globally and don't want to worry about formatting ever again, you have set up their global paths. To do so:

- Deactivate your environment
- Install `flake8` and `black` globally via `pip3 install flake8` and `pip3 install black`
- On the terminal write `whereis flake8` and `whereis black`

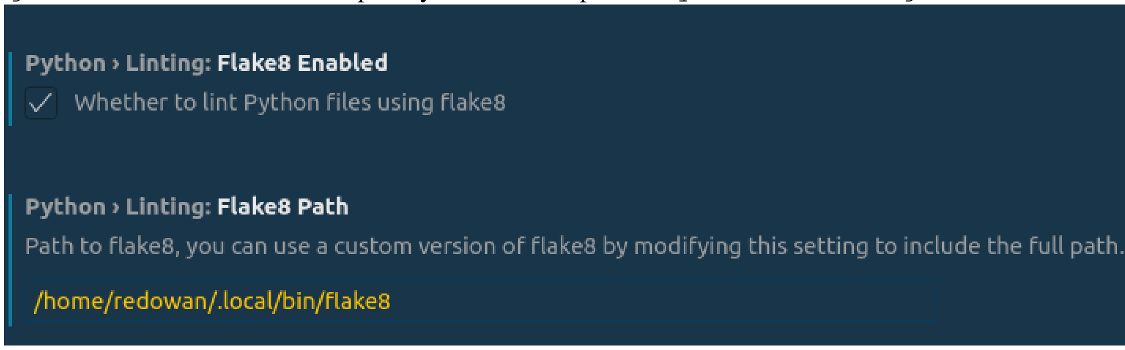


PROBLEMS 4   OUTPUT   DEBUG CONSOLE   TERMINAL

```
redowan@pop-os:~/code/demo$ whereis flake8
flake8: /home/redowan/.local/bin/flake8
redowan@pop-os:~/code/demo$ whereis black
black: /home/redowan/.local/bin/black
redowan@pop-os:~/code/demo$ 
```

- You should see their global paths

- Now go to the `settings` and search for `flake8` and paste your `flake8` path in `Python › Linting:`



Python › Linting: **Flake8 Enabled**

✓ Whether to lint Python files using flake8

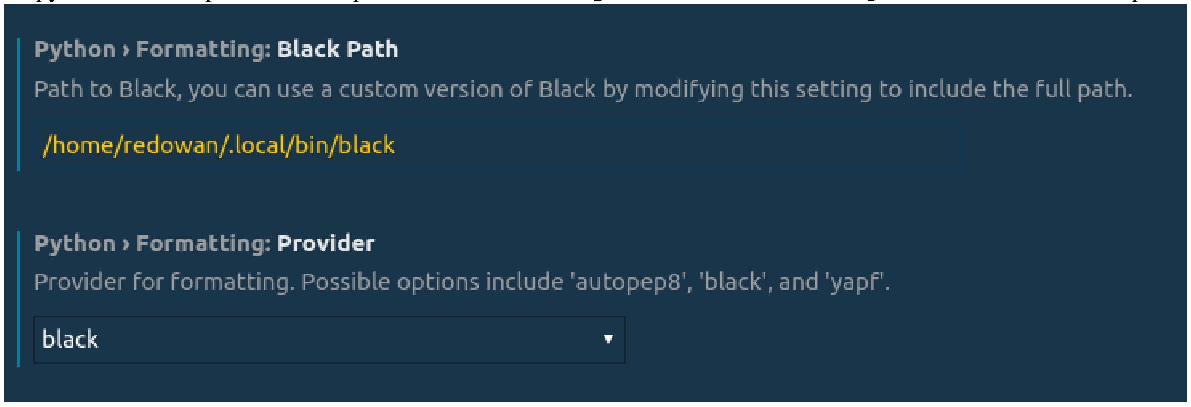
Python › Linting: **Flake8 Path**

Path to flake8, you can use a custom version of flake8 by modifying this setting to include the full path.

/home/redowan/.local/bin/flake8

  `Flake8 Path` option

- Copy `black` path and paste them in `Python › Formatting: Black Path` option



Python › Formatting: **Black Path**

Path to Black, you can use a custom version of Black by modifying this setting to include the full path.

/home/redowan/.local/bin/black

Python › Formatting: **Provider**

Provider for formatting. Possible options include 'autopep8', 'black', and 'yapf'.

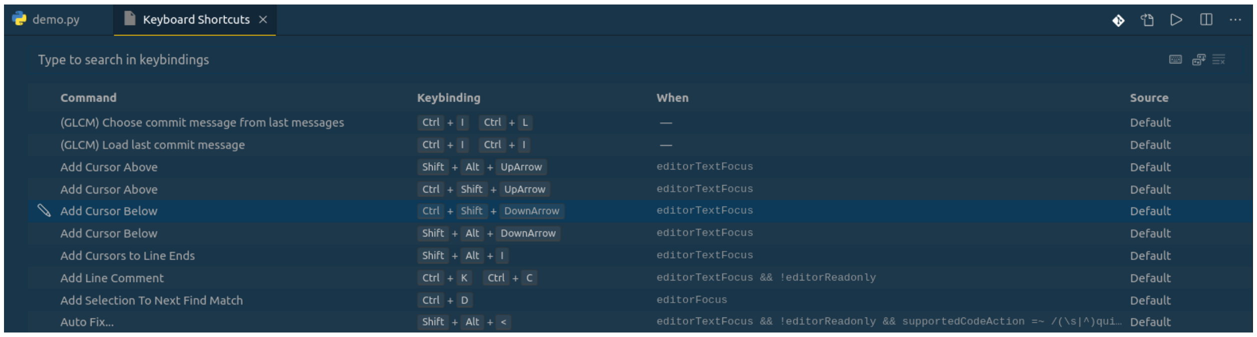
black

1.5 Shortcuts and Keymaps

VS Code is loaded with customization options and shortcuts. Default shortcuts can vary between different operating systems. However, you can always change those according to your likings. You can even adopt other editors' keybindings into VS Code.

1.5.1 Changing the Default Keymaps

- Open the command palette by pressing `ctrl+shift+p`.
- Type `Keyboard shortcuts`.
- Select `Preferences: Open Keyboard Shortcuts`. This will show you a list of all the keyboard shortcuts available to you.
- You can hover over a keybinding and a pencil icon should pop up.
- Click on the pencil icon to add your preferred key combination and press `enter`.



Command	Keybinding	When	Source
(GLCM) Choose commit message from last messages	Ctrl + I Ctrl + L	—	Default
(GLCM) Load last commit message	Ctrl + I Ctrl + I	—	Default
Add Cursor Above	Shift + Alt + UpArrow	editorTextFocus	Default
Add Cursor Above	Ctrl + Shift + UpArrow	editorTextFocus	Default
Add Cursor Below	Ctrl + Shift + DownArrow	editorTextFocus	Default
Add Cursor Below	Shift + Alt + DownArrow	editorTextFocus	Default
Add Cursors to Line Ends	Shift + Alt + I	editorTextFocus	Default
Add Line Comment	Ctrl + K Ctrl + C	editorTextFocus && !editorReadonly	Default
Add Selection To Next Find Match	Ctrl + D	editorFocus	Default
Auto Fix...	Shift + Alt + <	editorTextFocus && !editorReadonly && supportedCodeAction =~ /(\s|^)qui…	Default

Adopting Keymaps from Other Editors

If you are migrating from any other editors and you want to use you previous keymaps, you can do that too.

- Go to the extension panel.
- Type `<yourdesirededitor> keymaps`.
- Select and install. This should replace the default keymap with your desired one.

1.5.2 A Few Important Shortcuts (On Linux)

Although the abundance of shorcuts can be a plus,it can be intimidating for someone who's just starting out. Here's a short list of a few important shortcuts that appear more frequently.

- `ctrl+~` : Opens the integrated terminal
- `ctrl+N` : Opens a new empty file
- `ctrl+Shift+P` : Opens the command palette
- `ctrl+,` : Opens settings
- `ctrl+B` : Toggles between shrinked and extended side panel
- `F11` : Full screen mode
- `ctrl+alt+N` : Running python code
- `F5` : Running python code in debugging mode

1.6 Themes

Here is a list of some awesome vs code themes that I have encountered over the year. Currently I'm using Cobalt2. You can use this website for searching and watching previews of the themes.

1.6.1 Awesome Dark Themes

1. Cobalt2
2. Panda Syntax
3. Dracula
4. Darcula

5. TruBoo
6. Night Owl
7. Ayu Mirage
8. Noctis
9. Shades of Purple
10. Noctis

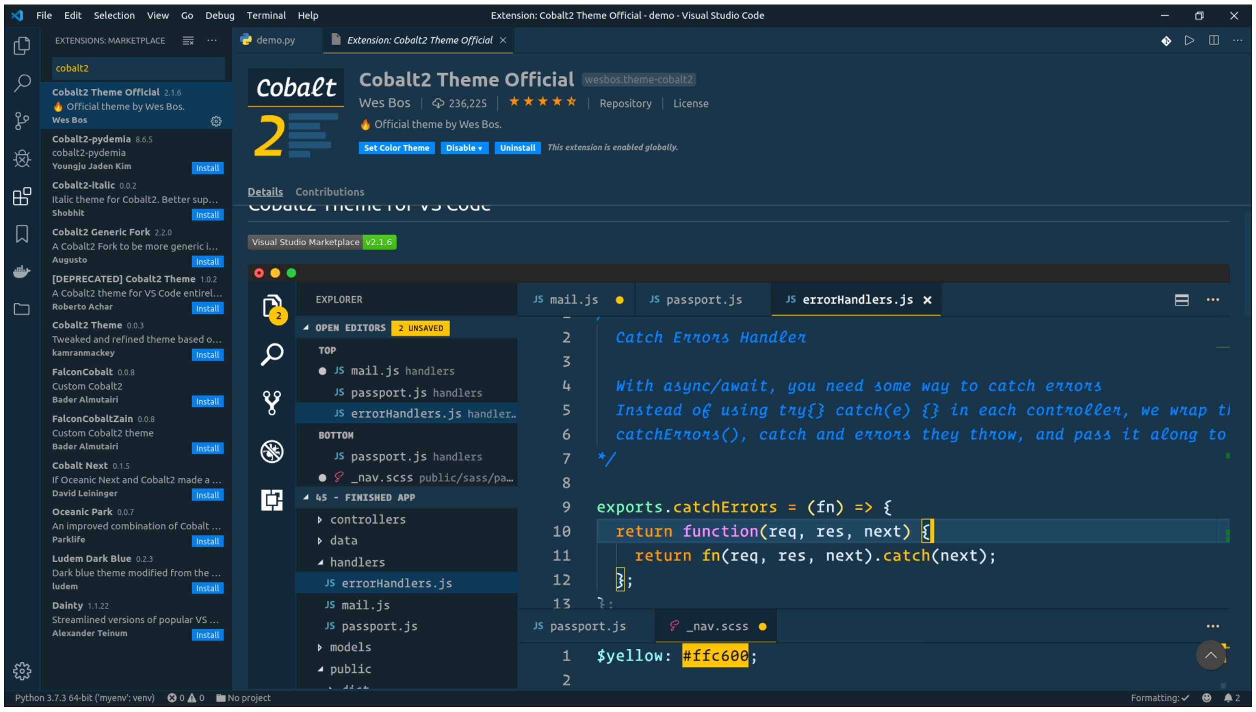
1.6.2 Awesome Light Themes

1. Winter is Coming
2. Github Plus
3. Min Light
4. Braver Solerized
5. Boxy Theme Kit

1.6.3 Installing Themes

Installing procedure for a theme is same as installing any other extension.

- From the left most stripe select extension icon
- Type your desired extension name
- Press install



cobalt2

Cobalt2 Theme Official 2.1.6
Official theme by Wes Bos.
Wes Bos

Cobalt2-pydemia 8.6.5
cobalt2-pydemia
Youngju Jaden Kim — Install

Cobalt2-italic 0.0.2
Italic theme for Cobalt2. Better sup...
Shobhit — Install

Cobalt2 Generic Fork 2.2.0
A Cobalt2 Fork to be more generic i...
Augusto — Install

[DEPRECATED] Cobalt2 Theme 1.0.2
A Cobalt2 theme for VS Code entirel...
Roberto Achar — Install

Cobalt2 Theme 0.0.3
Tweaked and refined theme based o...
kamranmackey — Install

FalconCobalt 0.0.8
Custom Cobalt2
Bader Almutairi — Install

FalconCobaltZain 0.0.8
Custom Cobalt2 theme
Bader Almutairi — Install

Cobalt Next 0.1.5
If Oceanic Next and Cobalt2 made a ...
David Leininger — Install

Oceanic Park 0.0.7
An improved combination of Cobalt ...
Parklife — Install

Ludem Dark Blue 0.2.3
Dark blue theme modified from the ...
ludem — Install

Dainty 1.1.22
Streamlined versions of popular VS ...
Alexander Teinum — Install

Cobalt2 Theme Official wesbos.theme-cobalt2
Wes Bos | 236,225 | ★★★★★ | Repository | License
Official theme by Wes Bos.
This extension is enabled globally.

Cobalt2 Theme for VS Code

```
TOP
● mail.js handlers
  passport.js handlers
  errorHandlers.js handler…
BOTTOM
  passport.js handlers
● _nav.scss public/sass/pa…
45 - FINISHED APP
  controllers
  data
  handlers
    errorHandlers.js
    mail.js
    passport.js
  models
  public
```

mail.js ● | passport.js | errorHandlers.js

```
  Catch Errors Handler

  With async/await, you need some way to catch errors
  Instead of using try{} catch(e) {} in each controller, we wrap t
  catchErrors(), catch and errors they throw, and pass it along to
*/

exports.catchErrors = (fn) => {
  return function(req, res, next) {
    return fn(req, res, next).catch(next);
  };
};
```

passport.js | _nav.scss ●

```
$yellow: #ffc600;
```

1.7 Fonts

Monospace fonts are better optimized for code readability and alignment. Here is a list of awesome coding fonts that I've used in the past. But I always come back to `Fira Code`.

1.7.1 List of Awesome Fonts

Free Fonts

1. Fira Code
2. Hack
3. Monoid
4. Hasklig
5. Input Mono
6. Office Code Pro
7. Inconsolata
8. DejaVu Sans Mono
9. Droid Sans Mono
10. Source Code Pro

Paid Fonts

1. Operator Mono
2. Dank Mono
3. PragmataPro

1.7.2 Installing Fonts

On Ubuntu

Even on same OS, installation procedure can vary between different fonts. On ubuntu, `open type` fonts generally live in `/usr/share/fonts/opentype/` folder.

- To install `Fira Code` you can simply type:

```
$ sudo apt install fonts-firacode
```

- However, this often downloads older version of the font. Recently they released a sleeker version 2.0 of the font. To install that, download the font from here.. Unzip the font folder and go to `otf` folder. Then simply run:

```
$ sudo cp -a . /usr/share/fonts/opentype/firacode
```

1.8 Extensions

You've already seen extensions like `themes`, `keymaps`, `code runner` etc in action. These modules can take your coding experience beyond what the builtins can offer. Here is an inexhaustive list of a few awesome extensions that will help you to make your python workflow more optimized.

1. **Code Runner:** Runs code snippet or code file of many popular languages like C, C++, Java, JavaScript, PHP, **Python**, Perl, Ruby, Go, Lua, Groovy, PowerShell etc.
2. **Better TOML:** Syntax highlighting for `.toml` files
3. **docs-yaml:** YAML schema validation and auto-completion
4. **DotENV:** Support for dotenv file syntax
5. **Githistory:** View git log, file history, compare branches or commits
6. **hide-gitignored:** Hide files from the file Explorer that are ignored by your workspace's `.gitignore` files
7. **Pytest IntelliSense:** Provides autocompletion for pytest
8. **pytest-snippets:** Snippet and templates for pytest
9. **Rainbow Brackets:** Provide rainbow colors for the round brackets, the square brackets and the squiggly brackets. This is particularly useful for Lisp or Clojure programmers, and of course, JavaScript, and other programmers. The isolated right bracket will be highlighted in red.
10. **Rainbow CSV:** Highlight CSV and TSV files in different colors, Run SQL-like queries
11. **Rainbow End:** This extension allows to identify keyword / end with colours.
12. **Indent-rainbow:** This extension colorizes the indentation in front of your text alternating four different colors on each step. Some may find it helpful in writing code for Nim or Python.
13. **reStructuredText:** reStructuredText language support (RST/ReST linter, preview, IntelliSense and more)
14. **Docker:** Adds syntax highlighting, commands, hover tips, and linting for Dockerfile and docker-compose files.
15. **Bookmarks:** Mark lines and jump to them
16. **TODO Highlight:** Highlight TODO, FIXME and other annotations within your code.
17. **autoDocstring** Generates python docstrings

1.9 Settings

All your settings and preferences are kept in `settings.json` file. You can directly manipulate that file to set your preference locally or globally. Locally the settings file can be found in `.vscode/settings.json` folder in your current project location. To see the global `settings.json`, head over to `$HOME/.config/Code/User/settings.json` and open it in VS Code. If you haven't changed any of the default settings, the file should be empty or almost empty.

1.9.1 Syncing Your Settings & Extensions

You don't want to set up VS Code from scratch every time you change your machine or do a fresh installation of your OS. In such scenarios, `Settings Sync` comes to rescue. You can set it up once, sync your settings and restore the settings with a few click. This will restore all of your settings, extensions, themes and other preferences. To do so,

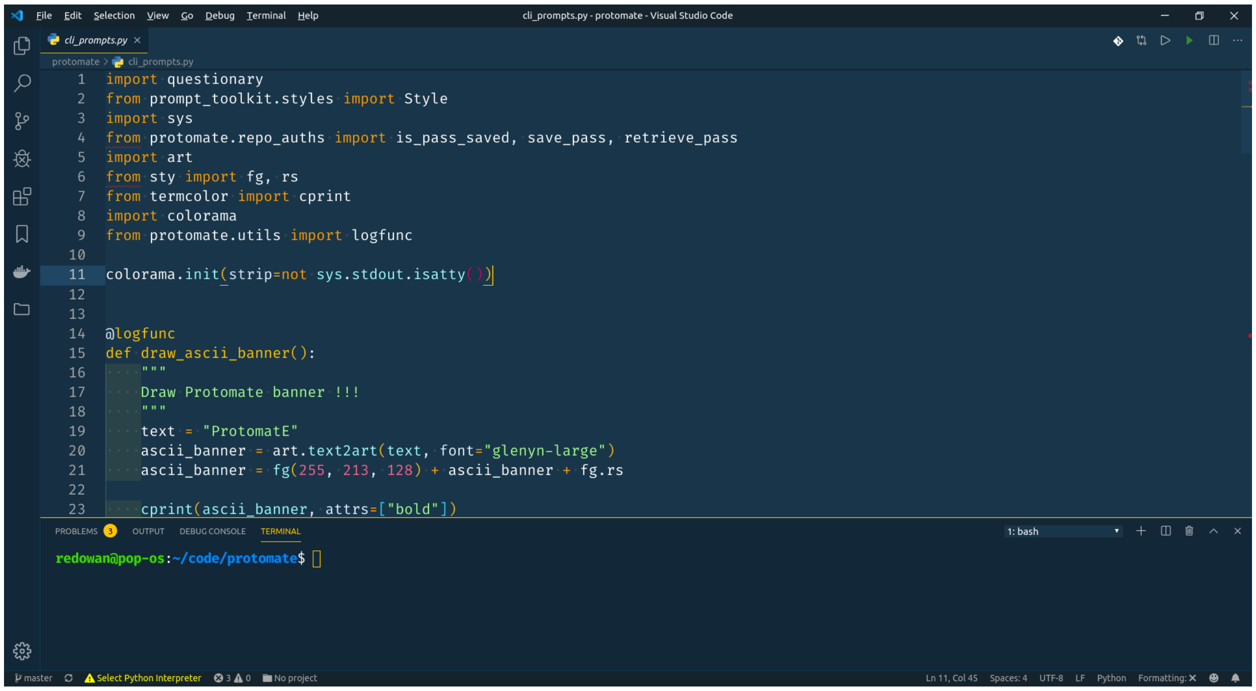
- Search and install `Settings Sync` from the extension panel

- Login with your github credentials
- Press `ctrl+Shift+P` to open the command prompt and select `Sync: Update/Upload Settings` option to upload your settings and save to a github gist
- If you are restoring the settings to a freshly installed VS Code, just select `Sync:Download Settings` and you should see your VS Code getting restored

1.9.2 Shut up & Let Me Replicate Your Settings

If you don't want to go through the hassle of manually installing all these extensions and like my settings. You can replicate my settings with the help of `Settings Sync` too. To do so:

- Go to the settings panel and search `setting sync`
- Find `Sync:Gist` option and replace it with `eec019bccd9c49388eaf9eeaf08c19ec` (This is the gist id of my settings)
- Then go to command prompt and select `Sync:Download Settings` option
- It will take some time to restore all the settings and you should see a setup similar to the following screenshots:


File Edit Selection View Go Debug Terminal Help
cli_prompts.py - protomate - Visual Studio Code
cli_prompts.py
protomate > cli_prompts.py
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
1: bash
master
Select Python Interpreter
3 0
No project
Ln 11, Col 45 Spaces: 4 UTF-8 LF Python Formatting: ✕

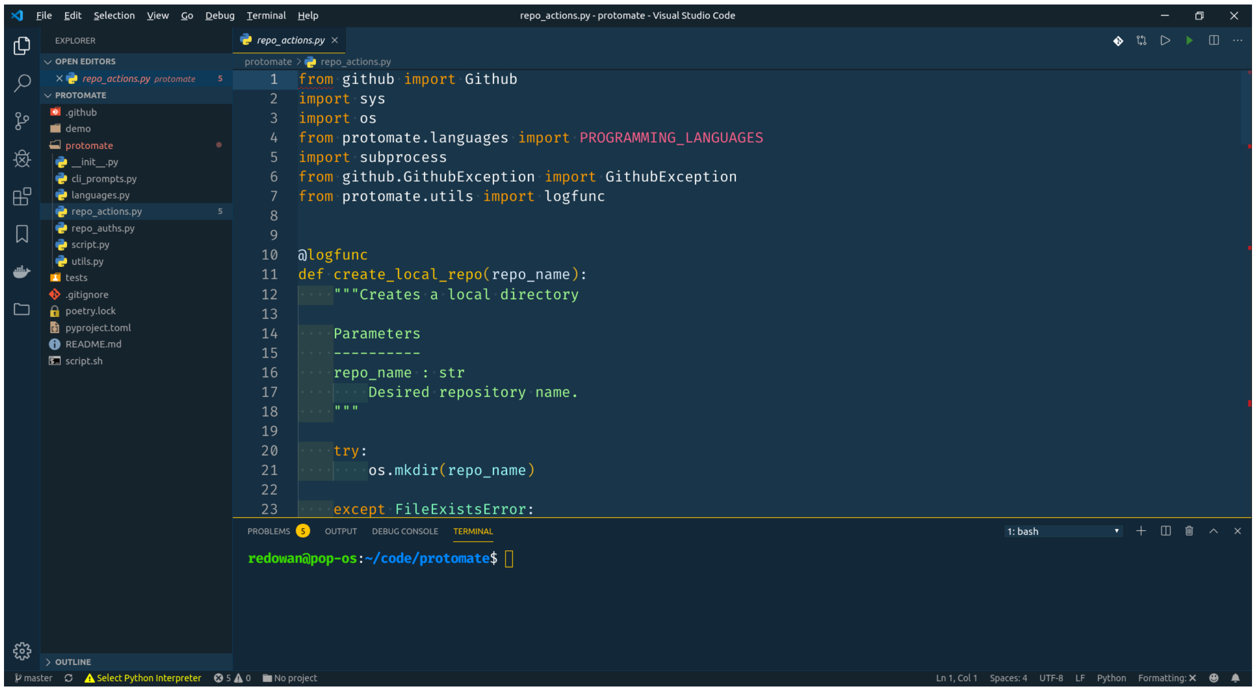
```python
import questionary
from prompt_toolkit.styles import Style
import sys
from protomate.repo_auths import is_pass_saved, save_pass, retrieve_pass
import art
from sty import fg, rs
from termcolor import cprint
import colorama
from protomate.utils import logfunc

colorama.init(strip=not sys.stdout.isatty())


@logfunc
def draw_ascii_banner():
    """
    Draw Protomate banner !!!
    """
    text = "ProtomatE"
    ascii_banner = art.text2art(text, font="glenyn-large")
    ascii_banner = fg(255, 213, 128) + ascii_banner + fg.rs

    cprint(ascii_banner, attrs=["bold"])
```

```
redowan@pop-os:~/code/protomate$
```



```python
from github import Github
import sys
import os
from protomate.languages import PROGRAMMING_LANGUAGES
import subprocess
from github.GithubException import GithubException
from protomate.utils import logfunc


@logfunc
def create_local_repo(repo_name):
    """Creates a local directory

    Parameters
    ----------
    repo_name : str
        Desired repository name.
    """

    try:
        os.mkdir(repo_name)

    except FileExistsError:
```

```
redowan@pop-os:~/code/protomate$
```

P.S.: This settings is a modified version of Kenneth Reitz's VS Code settings. Special thanks to him open sourcing that on twitter.

1.9.3 Customizing the Settings According to Your Need

After you've synced the above settings, you can easily change the themes, font sizes according to your liking. However, if you want to sync the settings via `Settings Sync`, you have to change back the github gist id and replace that with your own id. To do so:

- Go go to https://gist.github.com and find the gist name `cloudSettings`
- Check you url bar which should show something like this:

```
git.github.com/<username>/gist_id
```

- Copy your own gist id and replace that in `Sync:Gist` (In settings)

Now you can customize the workspace with your heart's content and sync accordingly.

A few Points to Note: If you have replicated my settings, you have to:

- Replace my github credentials with yours in the `settings.json` file
- Change and add your own `Flake8` and `Black` path for them to work (See it here.)

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search