
Development Environments

Christian Drefke

Apr 22, 2022

CONTENTS:

1	What is a Development Environment?	3
2	Components	5
3	Setup	9
4	Sources	29
5	ToDo	31
6	Indices and tables	33

Docs Homepage: <https://devenvironment.readthedocs.io/en/latest/>

WHAT IS A DEVELOPMENT ENVIRONMENT?

1.1 Basics

Think of a development environment like the workbench of a handyman or the setup of an operation room with all its instruments and tools. Each tool has its function or use and its fixed place. But different kinds of operations or crafting projects might need a different toolset. And every handyman or surgeon might even use their own preferred tools in their own preferred way.

Other professions that come in mind when thinking about such environments: Graphic Designers, Financial Traders, Emergency Responders, Software Developers.

The environment of a software developer also needs to be structured and set up well to be able to jump right into an existing or new project without struggling to get up to speed. It's necessary to have a collection of tools and methods, working together with your operating system in both, development and production stage.

1.2 Environment Location

An environment can run local on your notebook or workstation, in a virtual machine or even remote.

1.2.1 Local Development Environment

The first option for the development environment is to install and configure the development software and tools directly on your local workstation. This provides a high degree of customization and personalization for the developer to choose exactly how they want to work, and because it's all installed locally, it provides a highly portable option.

1.2.2 Hosted Development Environments

In the hosted development environment option, rather than installing the software and tools directly on your local workstation, the developer leverages a virtual machine (shared or dedicated) from a hosting provider where the tools are installed and made available. Developers can then use remote access technologies such as remote desktop or SSH to connect to the hosted environment to do their development.

1.2.3 Cloud Development Environments

A cloud development environment is a type of hosted environment where developers will often access their tools from within a web application.

Definitions from: [What is a Development Environment and why do you need one](#)

1.3 Tiers - Stages of development

There are usually three tiers when developing an application. In a perfect world every developer is running new code through all of these tiers, manually or automatically. When talking about running the code, it basically means to run it against the server side or backend. Therefore we talk about tiers on the server side.

1.3.1 Development Tier

All development is done in the development tier. The developer is running and testing the application against a backend that is not running and production critical processes. Depending on the project this tier needs a whole setup of server backends, hardware and other things to test the application. In many cases additional tools or scripts or logging features are used to debug code more efficiently during development. These additions are not necessary or purposely deactivated during production.

1.3.2 Staging Tier

A staging tier is basically a copy of the production tier. The application is tested against this environment. Long running performance and stability tests are usually run against this. If something is wrong with the application, the developer is going back to the development tier to fix identified issues.

1.3.3 Production Tier

If all tests are ok and the application is cleared for production, this is where it's activated and running. There should be no development, bugfixing whatsoever to this tier before being tested through the other tiers.

COMPONENTS

Components of a development environment

2.1 Operating System

Not going into details here, we need at least some kind of operating system to run our environment on.

2.1.1 Linux

Probably the most powerful OS for a developer but also the one that many beginners struggle with. At least when they are used to Windows environments before. But give it a try, though it might not be your first choice on the client side it will definitely be the OS that you will see most on the server side.

2.1.2 Windows

Developing on Windows was quite a pain with earlier versions of the OS. At least as long as you didn't develop for the Windows ecosystem working with languages like .NET or VBScript. Especially developing with interpreter languages like Python or Perl had quite a few drawbacks. I personally had at least one Linux virtual machine available to get around the issues with Windows based developing.

But these times are over since we have Windows 10. It made a large step towards a developer friendly environment. There are still a few gaps and hiccups (especially when running Docker containers locally) but the requirements are satisfied.

See this article from [Alexander Lockshyn](#) as example for someone who has the same feeling: [Surprisingly Software Development on Windows is Awesome Now](#)

2.1.3 Mac OS

Can't really say that much from a personal point of view about Mac OS but I know that it's great for developers. Plenty out there are using it for a long time. And its relationship and common features to Unix/Linux are probably the main reason for this success. It might be the perfect OS for having both, a robust development environment and a well designed UI operating system.

2.2 Terminals and Shells

I always prefer a good CLI over a good UI and sometimes even a great UI. There are plenty out there and they can be used more and more universally between the different operating systems:

- **Bourne shell** - Good old ‘bash’ from Unix/Linux based systems. Still going strong.
- **Z shell** - This extended bash, shortly called ‘zsh’, is used as the default shell on Macs since macOS Catalina.
- **PowerShell** - Open Source since 2016, this shell from Microsoft can be installed not only on Windows but also Unix/Linux based systems. Powershell 7, stable since March 4, 2020, is the universal replacement for older 5.x and 6.x versions.

Even if you prefer to use your mouse for all operations, you’ll need to run some commands here and there to start your scripts, applications, containers, ... through a terminal.

2.3 Programming Languages

Programming languages are divided in two major families, compiler and interpreter languages.

2.3.1 Compiler Languages

An application written in a compiler language needs to be translated by a compiler into “machine code”/“assembly” before it can be executed on a computer system. The compiler is only needed for the translation, the compiled application can run standalone afterwards.

Examples: C++, Go, Java

2.3.2 Interpreter Languages

Code written in an interpreted language needs an interpreter to be executed. The interpreter is processing the code “on the fly”.

Examples: Perl, Python, Ruby on Rails

2.4 Editors

2.4.1 Text Editors

A text editor is, well, a text editor. There are plenty out there and in many cases a text editor can be upgraded through add-ons to support developing features like syntax highlighting, code completion or even version control integration.

Examples: Atom, Notepad++, Sublime, Visual Studio Code

2.4.2 Integrated Development Environment

An “Integrated Development Environment” or short IDE is a “batteries included” software that helps you to write, test and deliver your source code and applications in a specific programming language. Main differentiations between an IDE and even sophisticated text editors like Notepad++ are:

- Integration in version control systems like Git
- Dependency checking
- Running your application through an interpreter for debugging directly in your IDE window
- Visual programming

And of course the features that a good text editor can do too like syntax highlighting or code search.

Let's set up an environment for you. This section is divided into guides for different environment and deployments. And at the moment only one guide long. Others will hopefully follow, depending on my time and possible contributions (always welcome).

3.1 Python on Windows 10

3.1.1 Overview

After successfully implementing this guide you should have a development environment for writing and running python applications directly through a Windows 10 terminal.

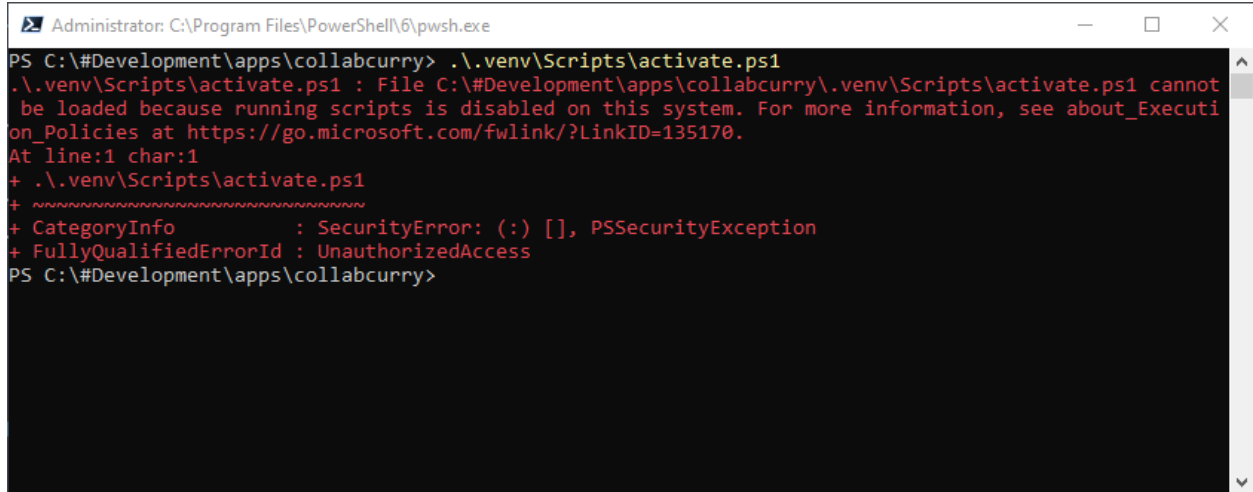
This setup includes: - Version Control with Git - Python 3.8+ - Virtual Environment for Python - Microsoft Visual Studio Code as Editor

3.1.2 OS Requirements

This setup is written for a Windows 10 platform.

Powershell Excecution-Policy

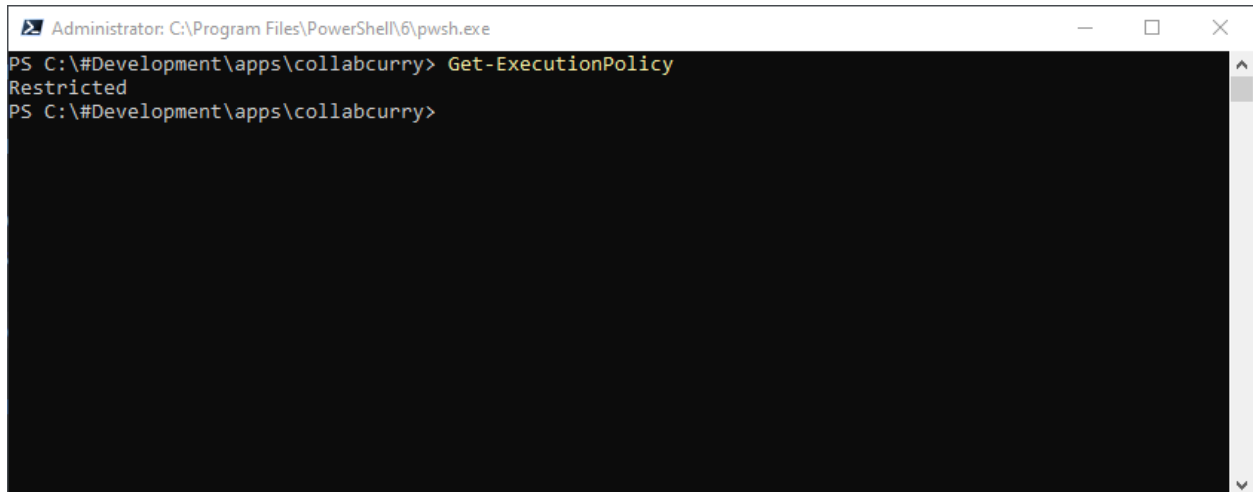
Windows 10 default security settings prevent you from running PowerShell scripts. In the “Virtual Environment” a PowerShell script is executed to activate the virtual environment. You might run into an error which points to the Excecution-Policy.

A screenshot of a PowerShell console window titled "Administrator: C:\Program Files\PowerShell\6\pwsh.exe". The prompt is "PS C:\#Development\apps\collabcurry>". The user enters ".\.\.env\Scripts\activate.ps1". The output shows an error: ".\.\.env\Scripts\activate.ps1 : File C:\#Development\apps\collabcurry\.env\Scripts\activate.ps1 cannot be loaded because running scripts is disabled on this system. For more information, see about_Execution_Policies at https://go.microsoft.com/fwlink/?LinkID=135170. At line:1 char:1 + .\.\.env\Scripts\activate.ps1 + ~~~~~ + CategoryInfo : SecurityError: (:) [], PSSecurityException + FullyQualifiedErrorId : UnauthorizedAccess". The prompt returns to "PS C:\#Development\apps\collabcurry>".

```
Administrator: C:\Program Files\PowerShell\6\pwsh.exe
PS C:\#Development\apps\collabcurry> .\.\.env\Scripts\activate.ps1
.\.\.env\Scripts\activate.ps1 : File C:\#Development\apps\collabcurry\.env\Scripts\activate.ps1 cannot
be loaded because running scripts is disabled on this system. For more information, see about_Executi
on_Policies at https://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\.\.env\Scripts\activate.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\#Development\apps\collabcurry>
```

You can check the current excecution policy by starting a PowerShell and running the command:

```
Get-ExecutionPolicy
```

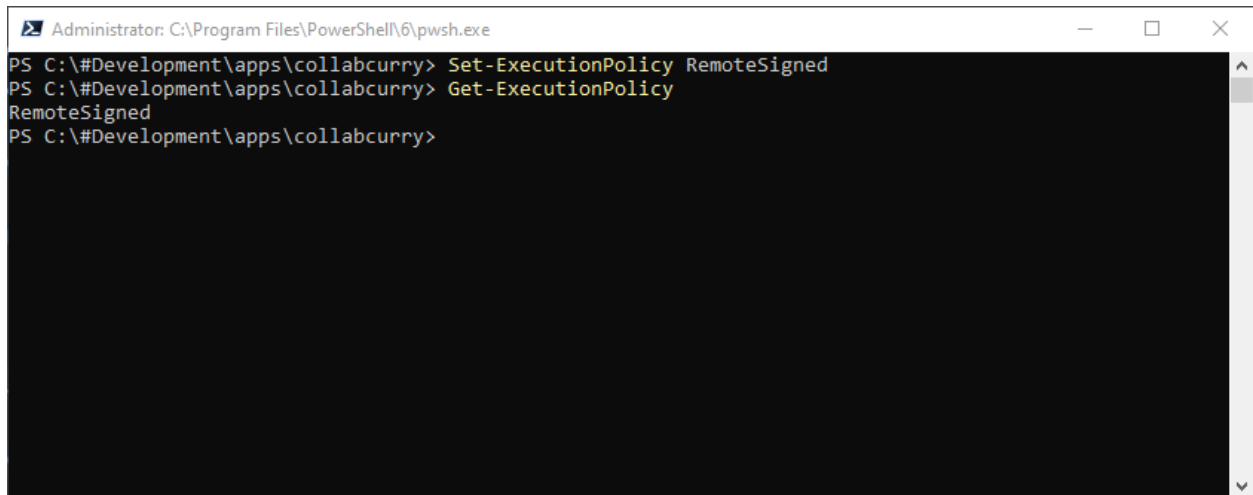
A screenshot of a PowerShell console window titled "Administrator: C:\Program Files\PowerShell\6\pwsh.exe". The prompt is "PS C:\#Development\apps\collabcurry>". The user enters "Get-ExecutionPolicy". The output is "Restricted". The prompt returns to "PS C:\#Development\apps\collabcurry>".

```
Administrator: C:\Program Files\PowerShell\6\pwsh.exe
PS C:\#Development\apps\collabcurry> Get-ExecutionPolicy
Restricted
PS C:\#Development\apps\collabcurry>
```

There are several ways to overcome these restrictions. Quite a few of them are described in a blog article by Scott Sutherland which you can find under the [Sources](#) section.

For ease of use we will set the Policy to ‘RemoteSigned’. This means that remotely executed PowerShell scripts need to be signed. Locally executed scripts don’t need to meet this requirement. Set the policy and check the result afterwards.

```
Set-ExecutionPolicy RemoteSigned
Get-ExecutionPolicy
```



```
Administrator: C:\Program Files\PowerShell\6\pwsh.exe
PS C:\#Development\apps\collabcurry> Set-ExecutionPolicy RemoteSigned
PS C:\#Development\apps\collabcurry> Get-ExecutionPolicy
RemoteSigned
PS C:\#Development\apps\collabcurry>
```

Warning: Be aware that this allows the execution of local PowerShell scripts without them being signed.

3.1.3 Project Directory

Set up a base directory for your development projects. It doesn't really matter if you have a structure below the base directory itself but it might help organizing certain things.

My current structure is:

```
#Development/
├─apps/
│   └─ <-- contains my different applications, most projects are here -->
├─containers/
│   └─ <-- repositories for my docker container repositories -->
├─docs/
│   └─ <-- documentation projects like this one -->
├─snippets/
│   └─ <-- small code snippets, single methods, ... -->
```

3.1.4 Version Control - Git

There are a few version control systems but we are using Git.

Download and Installation

Download the Windows Installer from: <https://git-scm.com/downloads>

Run the Installer as usual.



After the installation is finished, an html page with notes and current issues for Git on Windows opens up. It's never a bad idea to read such notes.

You might have to reboot at this point since the git command shell will integrate into Path.

Initialize your first repository

You can run git commands from both, CMD and PowerShell, or through Git Bash which comes with the installation. I'll use PowerShell for this section, you just use your terminal of choice.

We won't get into details of the usage of git but check a basic feature of it.

Change to your projects base directory and check the version of your git installation.

```
git --version
```

You should see the current git version. If not, restart your machine to update your Path.

Git requires a basic configuration before you can push your first code to a remote repository so let's do this.

```
git config --global user.name "Christian Drefke"
```

```
git config --global user.email "christian.drefke@bechtle.com"
```

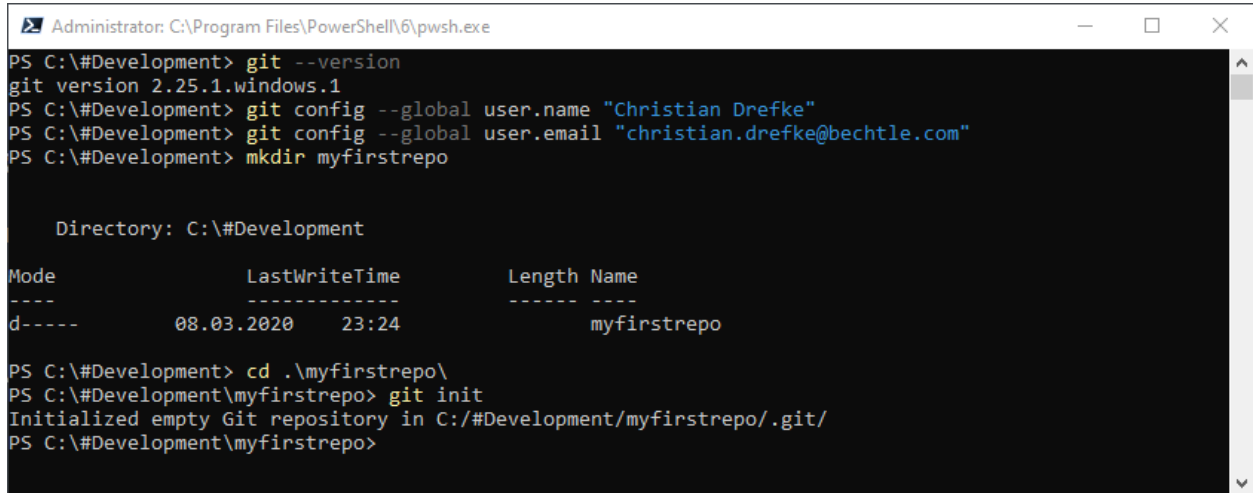
Create a directory, change into it and initialize your first local repository.

```
mkdir myfirstrepo
```

```
cd myfirstrepo
```



```
git init
```



The screenshot shows a PowerShell terminal window titled "Administrator: C:\Program Files\PowerShell\6\pwsh.exe". The user is in the directory C:\#Development. They run the following commands: `git --version` (output: git version 2.25.1.windows.1), `git config --global user.name "Christian Drefke"`, `git config --global user.email "christian.drefke@bechtle.com"`, and `mkdir myfirstrepo`. A directory listing for C:\#Development shows a new directory named "myfirstrepo" with a last write time of 08.03.2020 23:24. Then, they navigate to `cd .\myfirstrepo\` and run `git init`, which outputs "Initialized empty Git repository in C:/#Development/myfirstrepo/.git/".

```
Administrator: C:\Program Files\PowerShell\6\pwsh.exe
PS C:\#Development> git --version
git version 2.25.1.windows.1
PS C:\#Development> git config --global user.name "Christian Drefke"
PS C:\#Development> git config --global user.email "christian.drefke@bechtle.com"
PS C:\#Development> mkdir myfirstrepo

Directory: C:\#Development

Mode                LastWriteTime         Length Name
----                -
d-----           08.03.2020    23:24             myfirstrepo

PS C:\#Development> cd .\myfirstrepo\
PS C:\#Development\myfirstrepo> git init
Initialized empty Git repository in C:/#Development/myfirstrepo/.git/
PS C:\#Development\myfirstrepo>
```

Git did now initialize a local repository in your directory. All necessary data for this repository is stored in the hidden directory `‘.git’`.

```
ls -Force
```

3.1.5 Python

Using Python on Windows 10 is quite straight forward nowadays.

Download and Installation

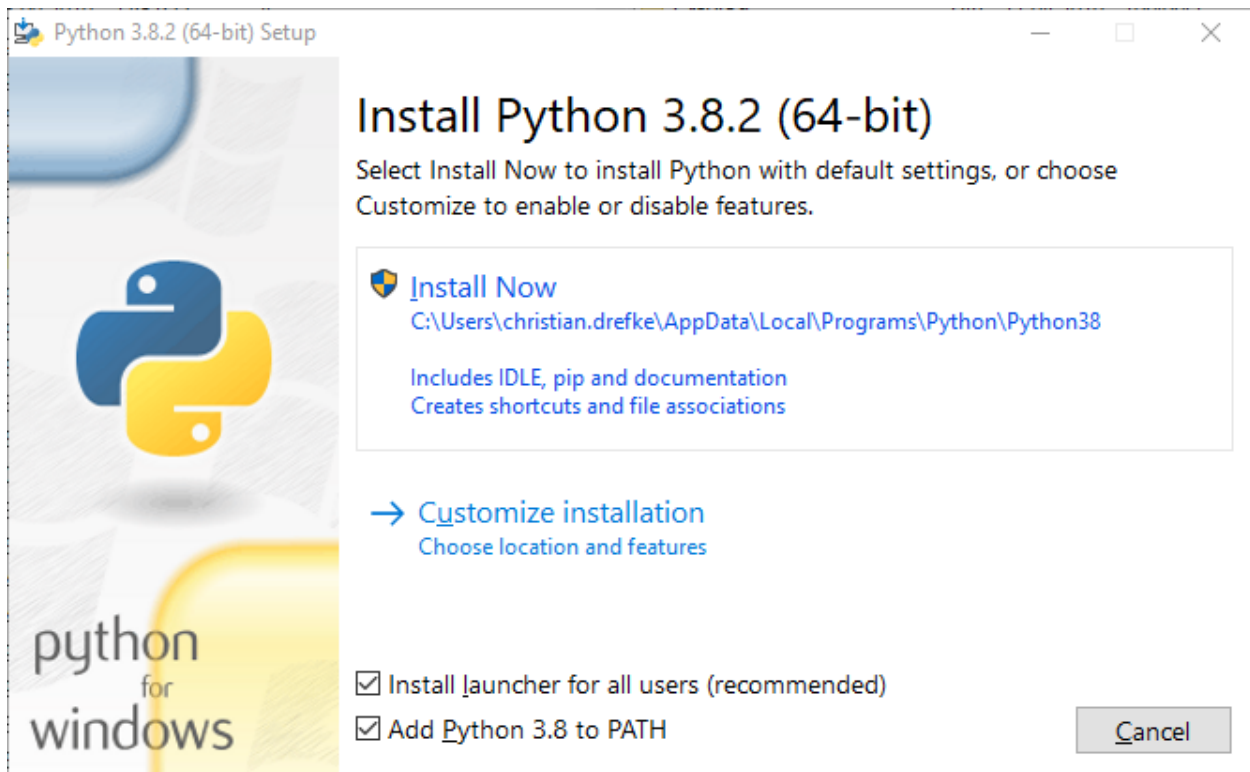
Download the Windows Installer from: <https://www.python.org/downloads/>

- Click on the ‘Download Python’ button
- Use the ‘Windows x86-64 executable installer’ for this setup

Note: You can also use the embeddable zip file for installation but this requires more manual work than just using the installer.

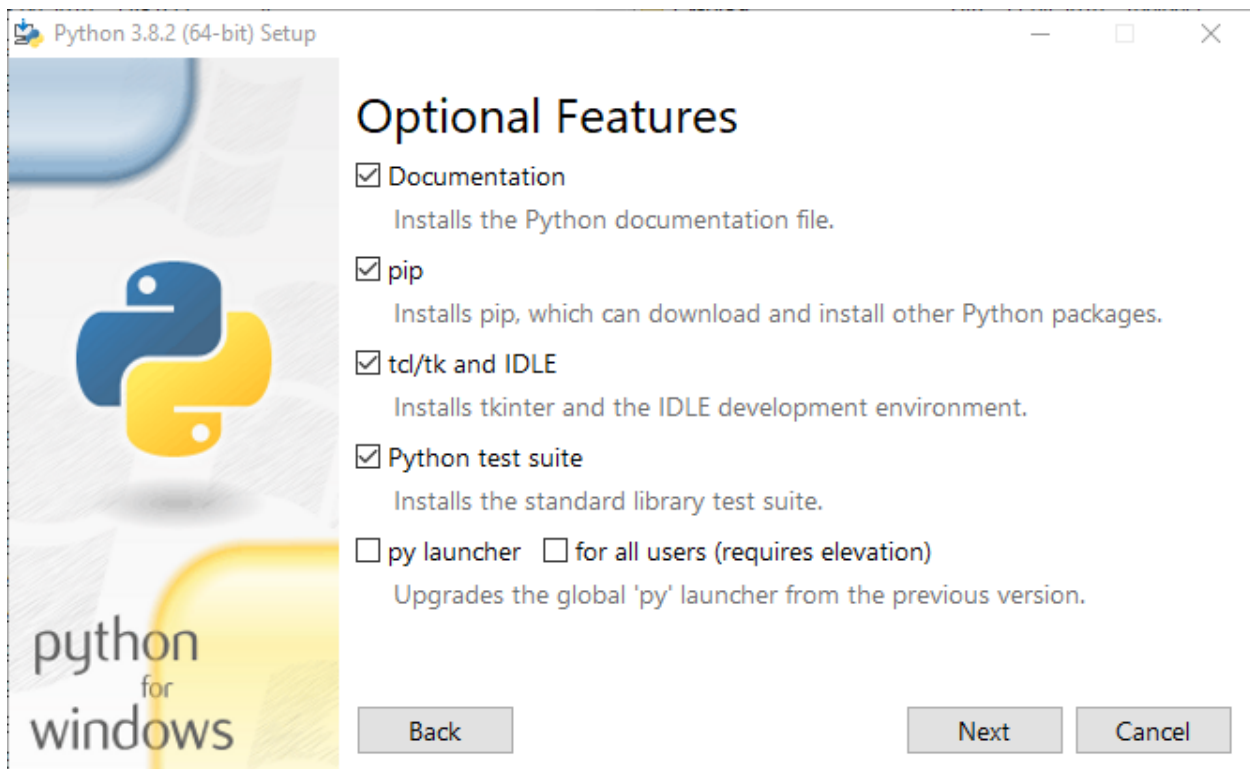
Run the installer.

Make sure you have selected ‘Add Python 3.8 to PATH’.



If you choose 'Custom':

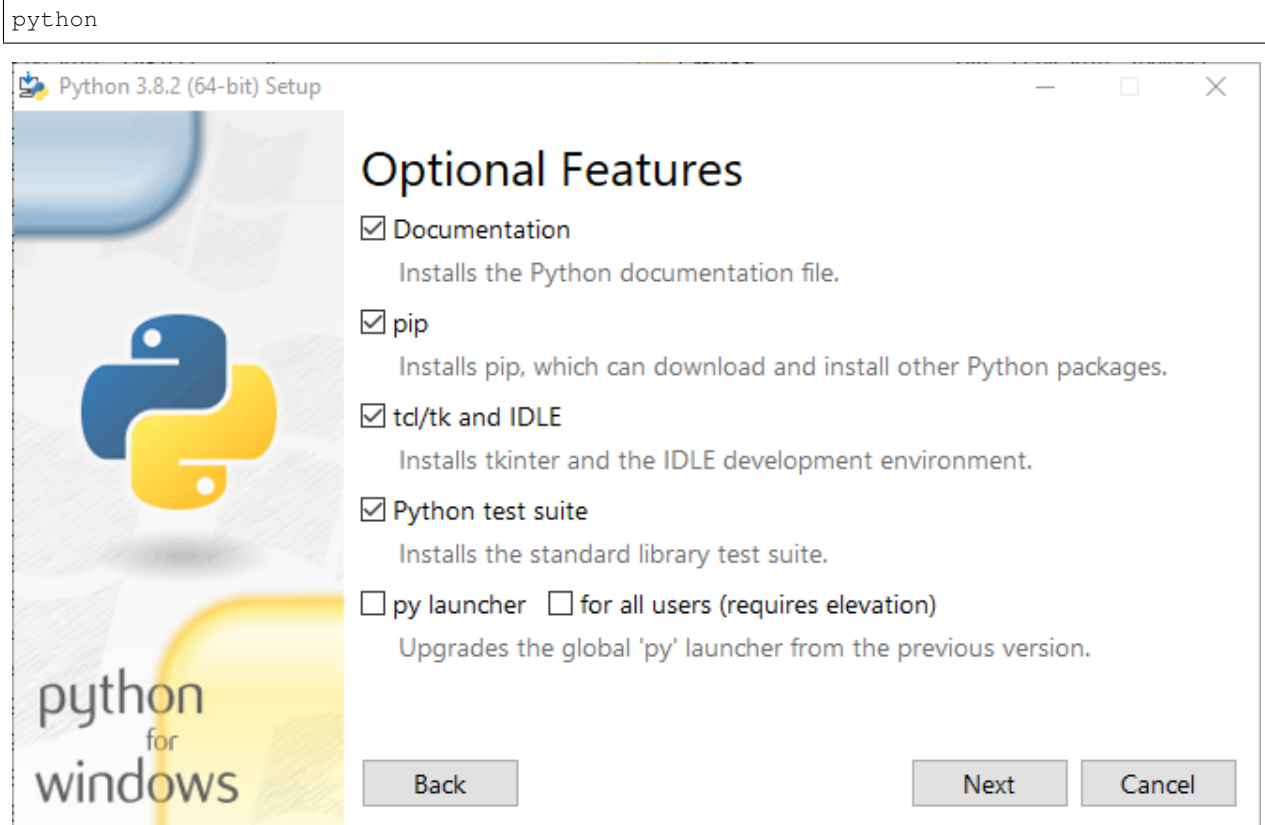
- Select 'pip'
- (optional) Select Documentation and/or tcl/tk



In the next screen, select your preferred installation directory and hit 'Install'. The installation might take a few minutes.

Run Python

To check if the installation was successful, start a PowerShell and start a python interpreter.



When the interpreter started and you see a version number together with the interpreter prompt '>>>', your Python installation is ready to use.

Note: You can leave the interpreter by typing in 'exit()' and Enter

Virtual Environment

Why virtual environments?

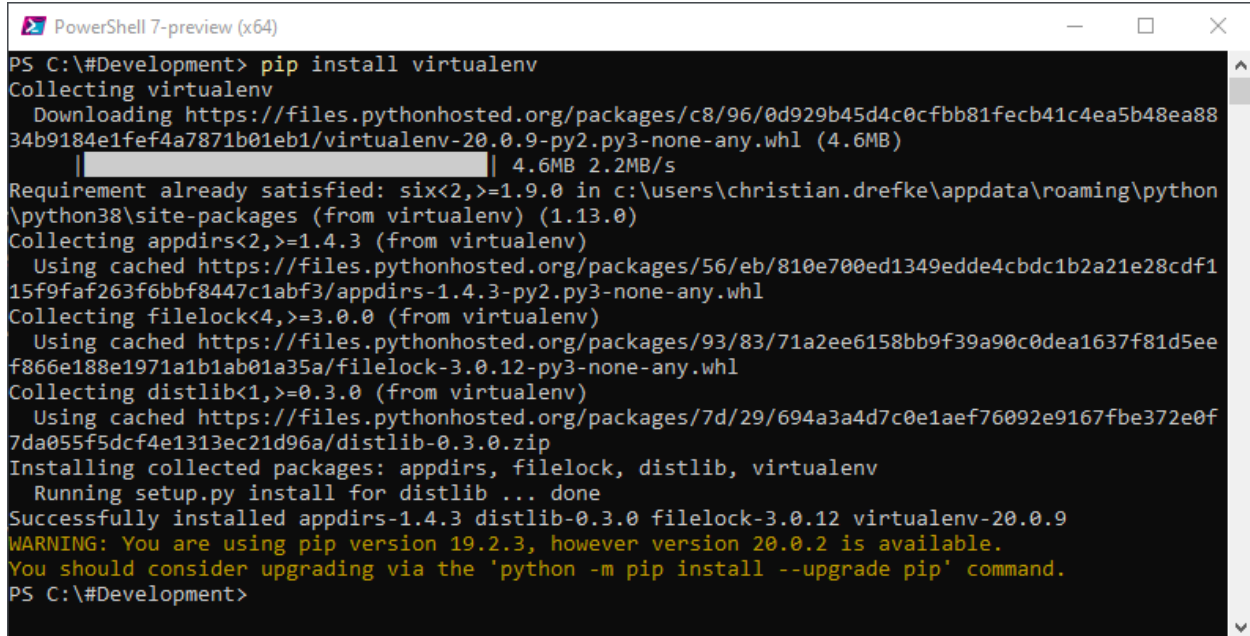
Python is bringing quite a lot of useful modules with it but there are many projects out there that use external modules, usually installed from the [Python Packaging Index](#). But it might happen that you are using different versions of such module in different projects. And a newer version might have a certain method or class name changed in comparison to an earlier version. This might brake your applications and you would have to adapt these module updates in our code.

To avoid such dependency problems a pretty common practice is to use virtual environments. It is a good habit to initiate a venv right after creating the project directory. And after the virtual environment got set up, activating it each time before you start working on the project.

Installation

Let's install virtualenv for python first. Start a PowerShell terminal and install virtualenv through the pip installer.

```
pip install virtualenv
```



```
PowerShell 7-preview (x64)
PS C:\#Development> pip install virtualenv
Collecting virtualenv
  Downloading https://files.pythonhosted.org/packages/c8/96/0d929b45d4c0cfbb81fecb41c4ea5b48ea8834b9184e1fef4a7871b01eb1/virtualenv-20.0.9-py2.py3-none-any.whl (4.6MB)
    |#####| 4.6MB 2.2MB/s
Requirement already satisfied: six<2,>=1.9.0 in c:\users\christian.drefke\appdata\roaming\python\python38\site-packages (from virtualenv) (1.13.0)
Collecting appdirs<2,>=1.4.3 (from virtualenv)
  Using cached https://files.pythonhosted.org/packages/56/eb/810e700ed1349edde4cbdc1b2a21e28cdf115f9faf263f6bbf8447c1abf3/appdirs-1.4.3-py2.py3-none-any.whl
Collecting filelock<4,>=3.0.0 (from virtualenv)
  Using cached https://files.pythonhosted.org/packages/93/83/71a2ee6158bb9f39a90c0dea1637f81d5ee f866e188e1971a1b1ab01a35a/filelock-3.0.12-py3-none-any.whl
Collecting distlib<1,>=0.3.0 (from virtualenv)
  Using cached https://files.pythonhosted.org/packages/7d/29/694a3a4d7c0e1aef76092e9167fbe372e0f7da055f5dcf4e1313ec21d96a/distlib-0.3.0.zip
Installing collected packages: appdirs, filelock, distlib, virtualenv
  Running setup.py install for distlib ... done
Successfully installed appdirs-1.4.3 distlib-0.3.0 filelock-3.0.12 virtualenv-20.0.9
WARNING: You are using pip version 19.2.3, however version 20.0.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
PS C:\#Development>
```

You might get a notice that your pip version is out of date. This is no big issue but let's update pip at this point as well.

```
python -m pip install --upgrade pip
```

Pip is now up to date.

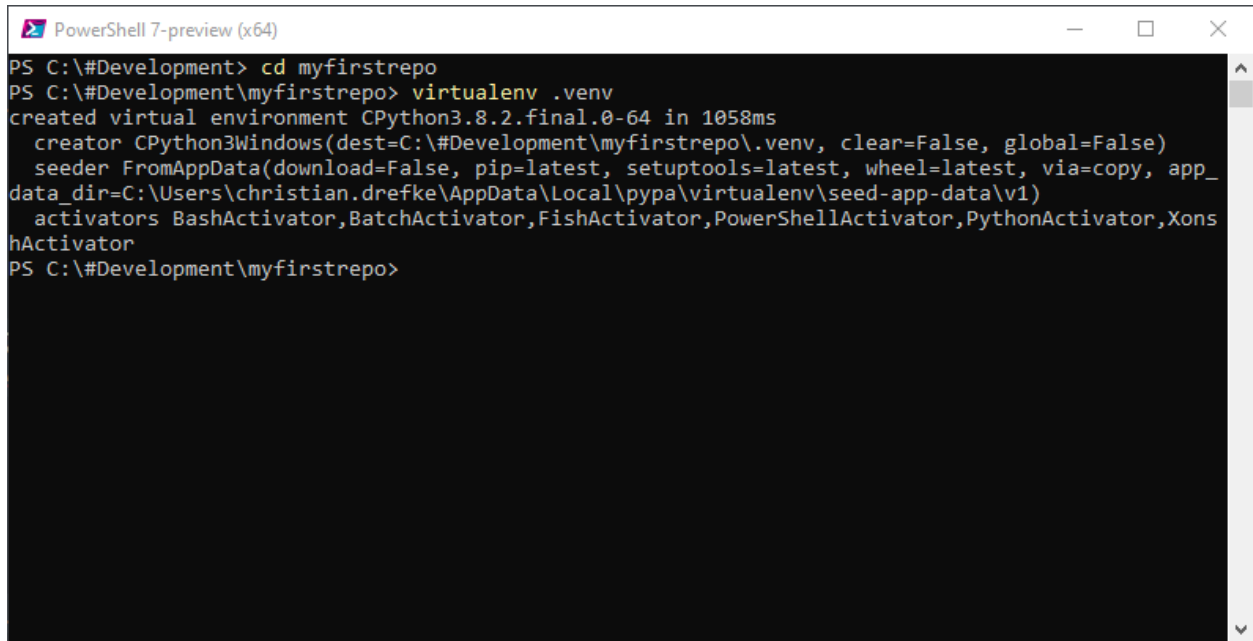
Initialize and activate a Virtual Environment

To initialize a virtual environment, switch the directory that we created earlier. From our base development directory:

```
cd myfirstrepo
```

Now initialize the venv. We will use '.venv' as the name for the environment data directory.

```
virtualenv .venv
```



```
PS C:\#Development> cd myfirstrepo
PS C:\#Development\myfirstrepo> virtualenv .venv
created virtual environment CPython3.8.2.final.0-64 in 1058ms
  creator CPython3Windows(dest=C:\#Development\myfirstrepo\.venv, clear=False, global=False)
  seeder FromAppData(download=False, pip=latest, setuptools=latest, wheel=latest, via=copy, app_
data_dir=C:\Users\christian.drefke\AppData\Local\pypa\virtualenv\seed-app-data\v1)
  activators BashActivator,BatchActivator,FishActivator,PowerShellActivator,PythonActivator,Xon
shActivator
PS C:\#Development\myfirstrepo>
```

We now have an additional directory in our project which contains a copy of our python interpreter including it's scripts like pip. But to make use of this new venv we need to activate it first.

```
.venv/Scripts/activate.ps1
```

You should now see a prefix '(venv)' in front of your command prompt. This tells us that we have successfully activated the virtual environment.

Note: Actually there's an 'deactivate.bat' under the venv 'Scripts' directory as well but at least in my case this has never really worked. Simply close the terminal session or use the 'activate.ps1' in a different project folder to switch to it.

Note: Use the 'activate.bat' file if you are using CMD as your terminal.

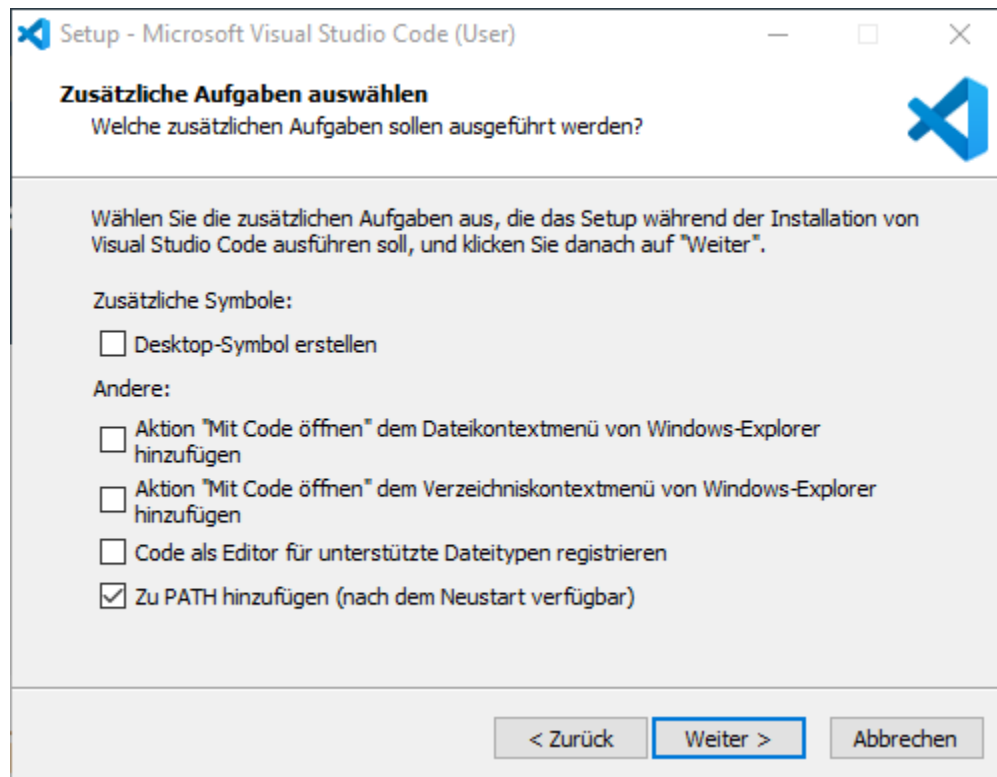
3.1.6 Visual Studio Code

Download

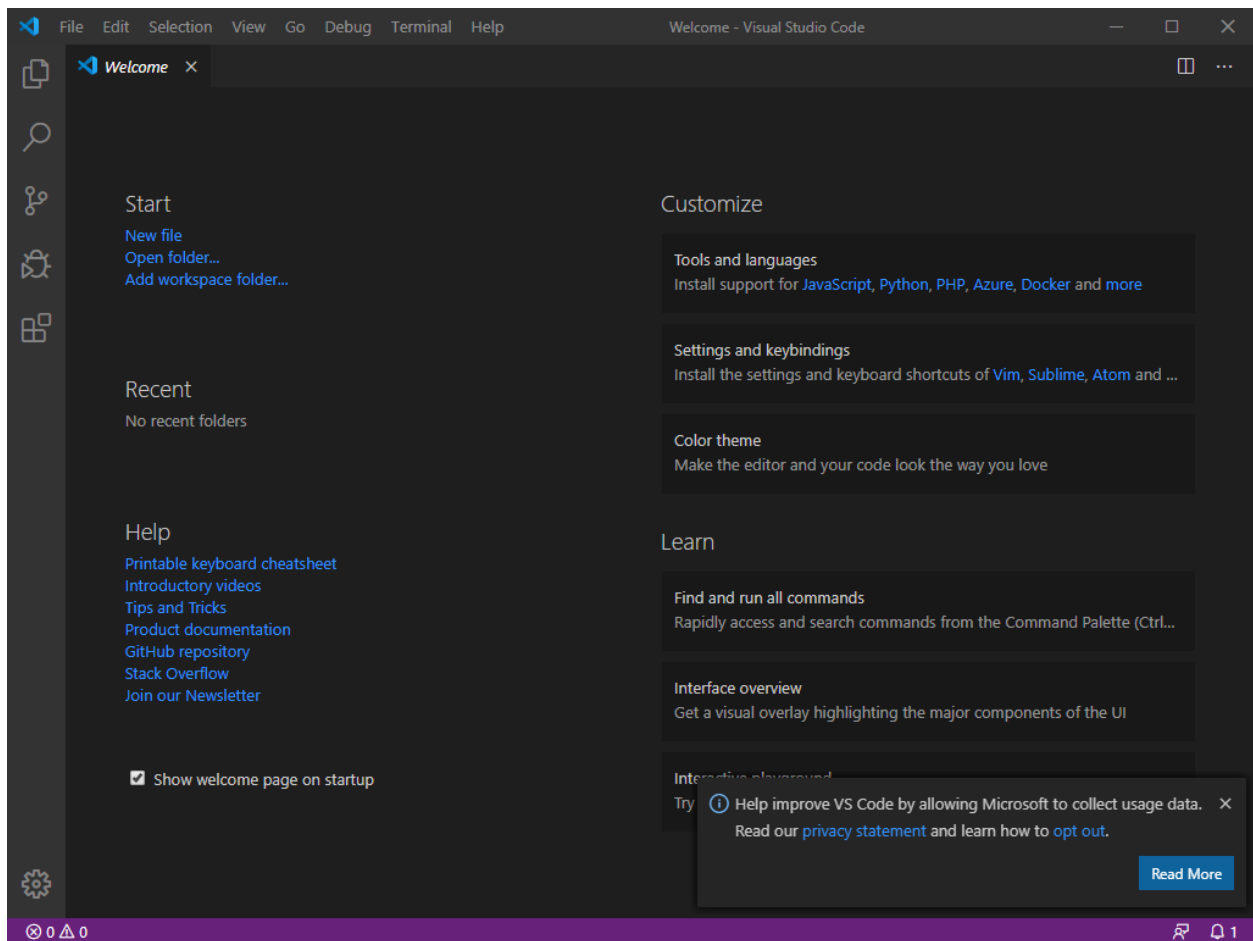
Download the Windows Installer from: <https://code.visualstudio.com/>

Installation

The installation is basically straight forward but you need to lookout for the PATH variable. Check the box so that VSCode is included.



After installation start the program to see if it's running.



Basic Setup

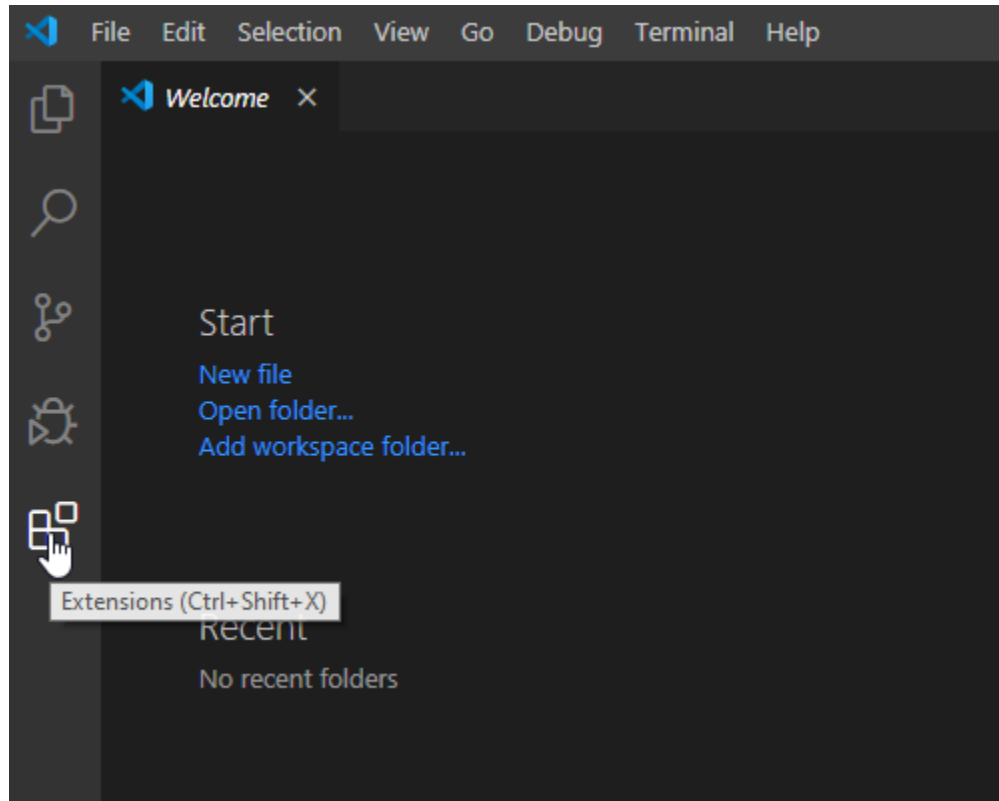
Every code editor comes with a sophisticated set of features but don't necessarily have support for syntax highlighting, remote server access (FTP, SFTP, ...) already included.

We'll make a very basic setup of VSCode for Python and Git.

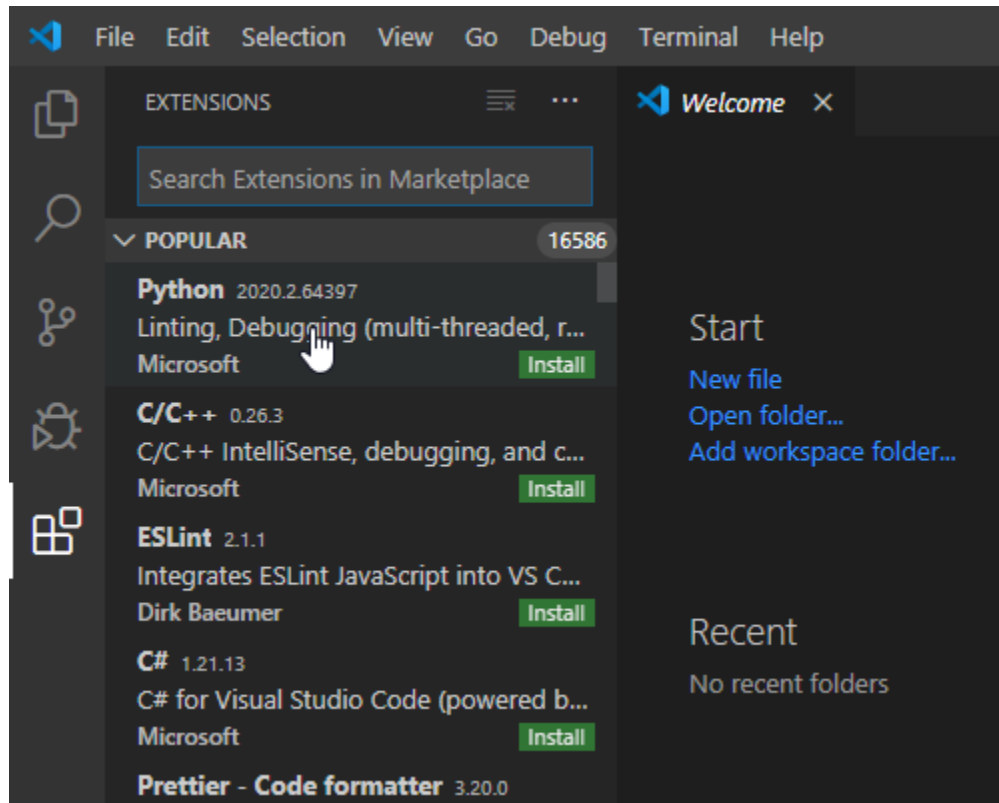
Syntax Highlighting

Not having proper syntax highlighting when writing code is a major pain and definitely not fun. Especially syntax errors will happen way more often than with proper highlighting.

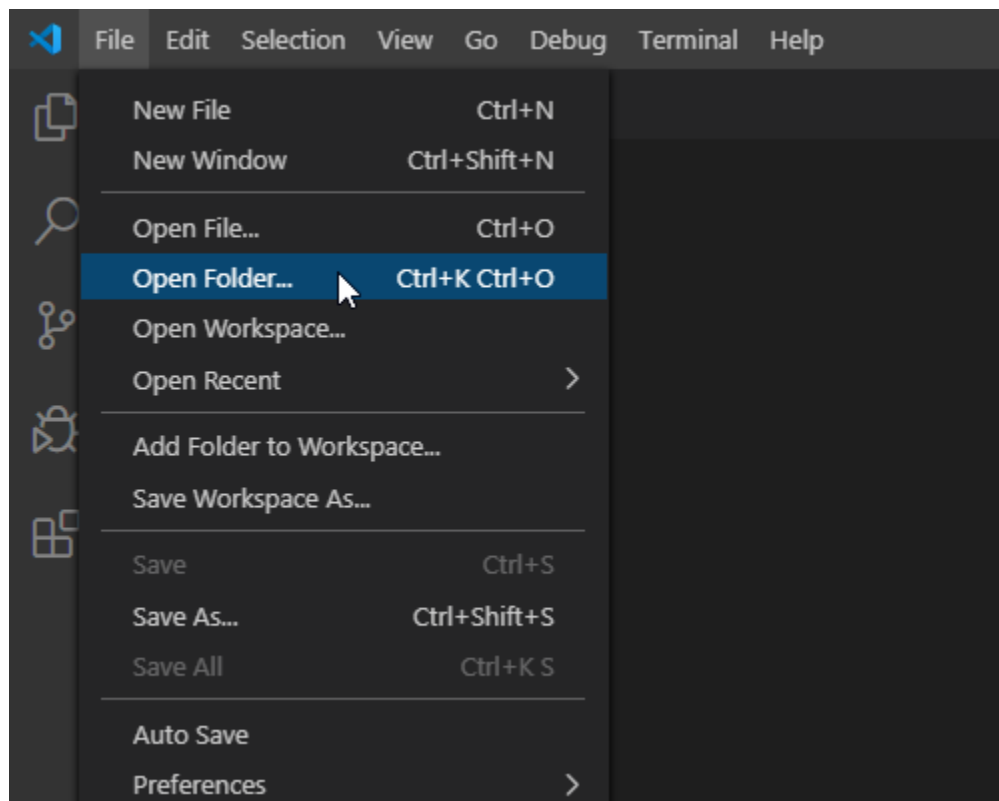
To enable highlighting for the Python syntax all we need to do is to enable the extension for it. On the left side of the VSCode window select Extensions (or Ctrl+Shift+X).



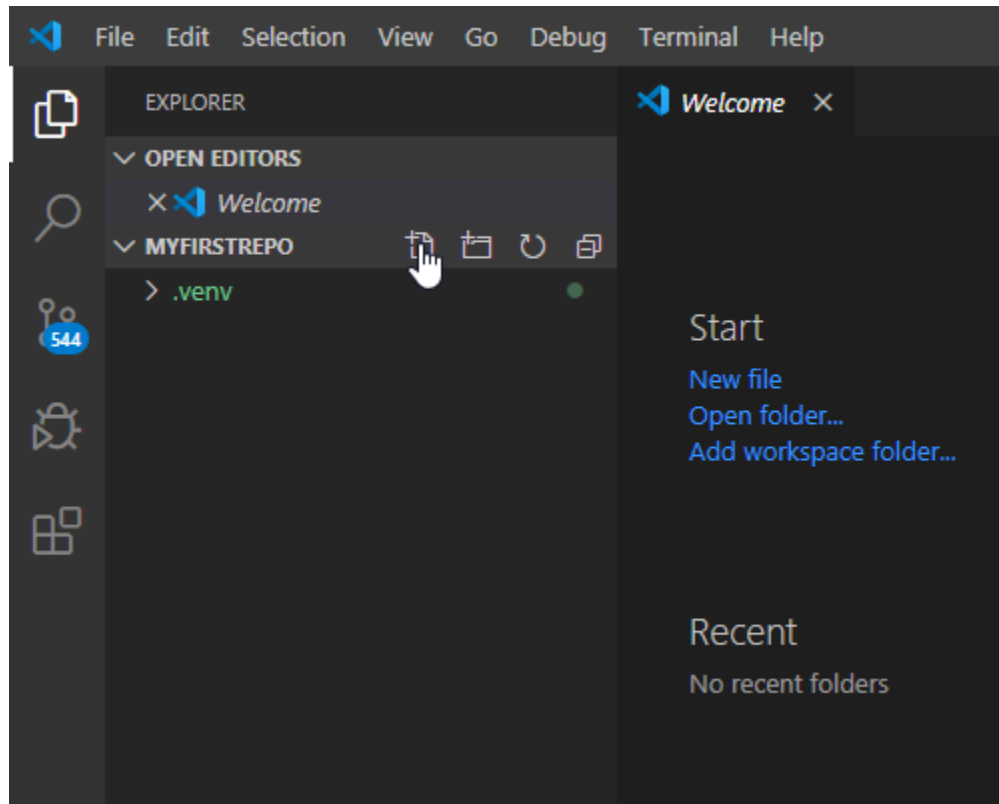
You should see Python right away. Otherwise use the search field. Click “Install” to install the extension.



Nothing is better for checking syntax highlighting than writing a few lines of code. To do so, open your projects directory that we created earlier. File -> Open or Ctrl+K, Ctrl+O.



After opening the directory you should see it on the left side in the Explorer section (else Ctrl+Shift+E to open). We'll now create a new file (Ctrl+N).

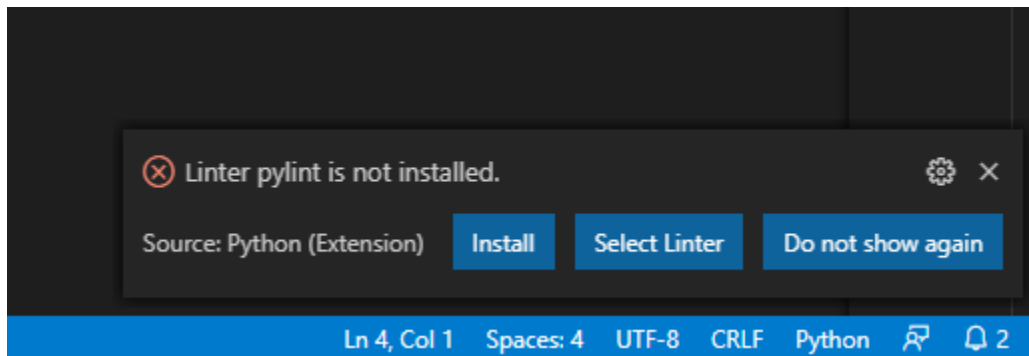


And add some content to it.

```
answer = input("The answer to the ultimate question of life, the universe and  
→everything is ")  
  
print(answer)
```

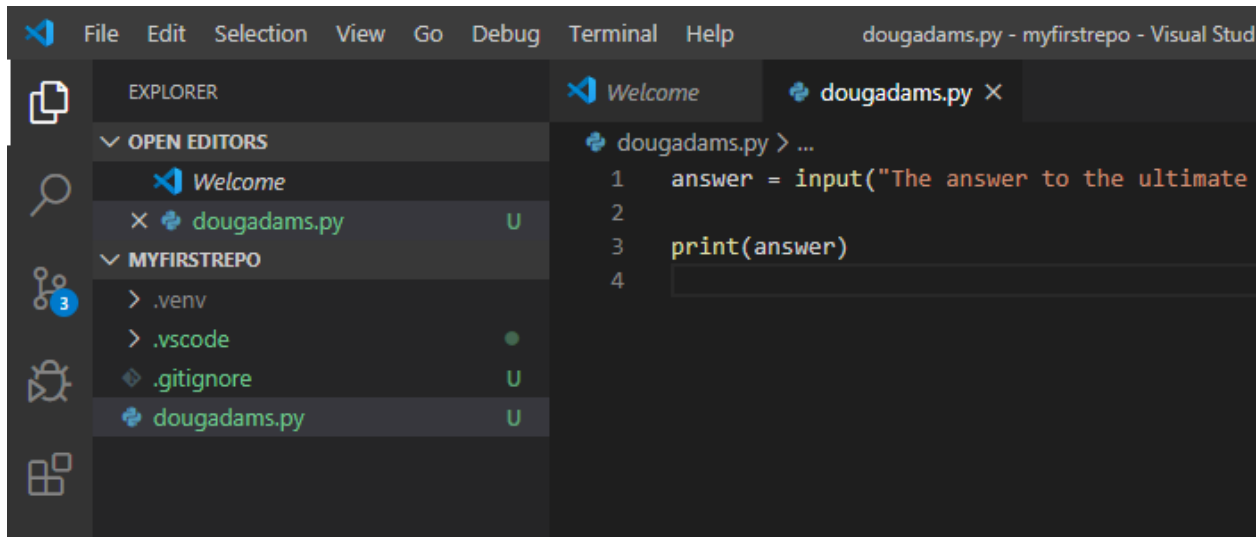
Now save (Ctrl+S) your file and give it a proper filename with file extension “.py” which is the typical file extension for Python files.

If you did a fresh installation of VSCode you will probably see the following info message at the lower right corner.

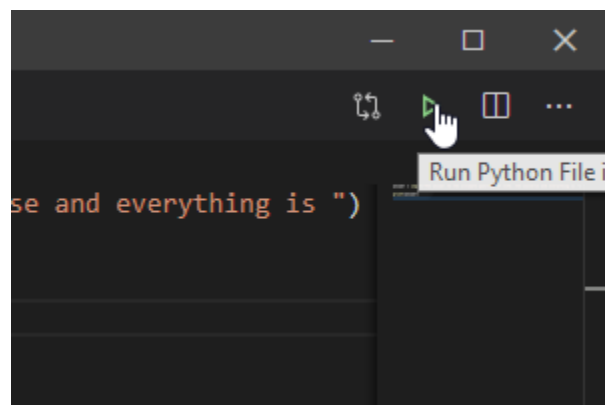


A linter, here specifically pyLint are extensions or modules that check your written code against syntax and code quality rules. If you like to, install this extension. You can skip this for now but you can't start too early to write well formatted code.

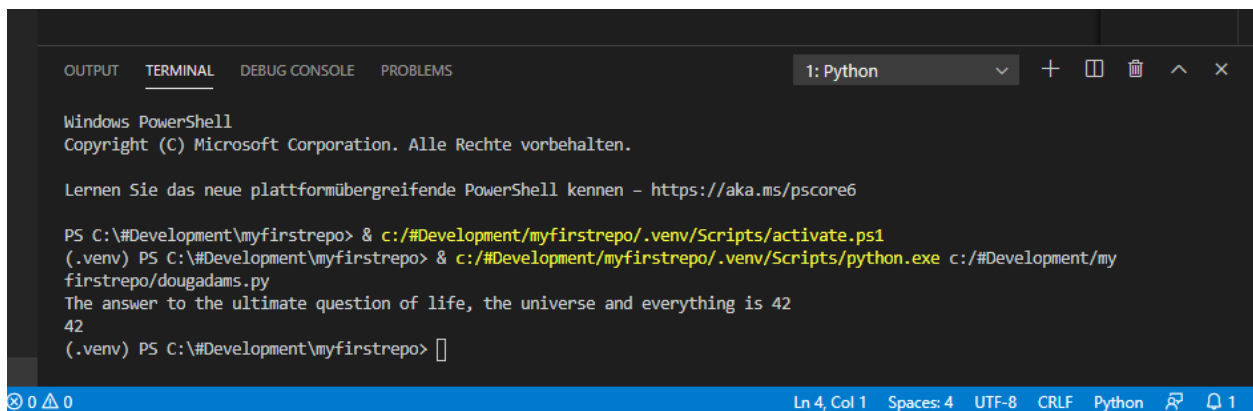
Back to our repository you should see the new file in your explorer section.



VSCode gives you the possibility to directly run you newly written script.



You'll see the script output in the terminal section of VSCode. And in this case the script even expects an input from your side. So provide an answer to finish the script.



Et voilà, if we did everything right you just ran your first Python script.

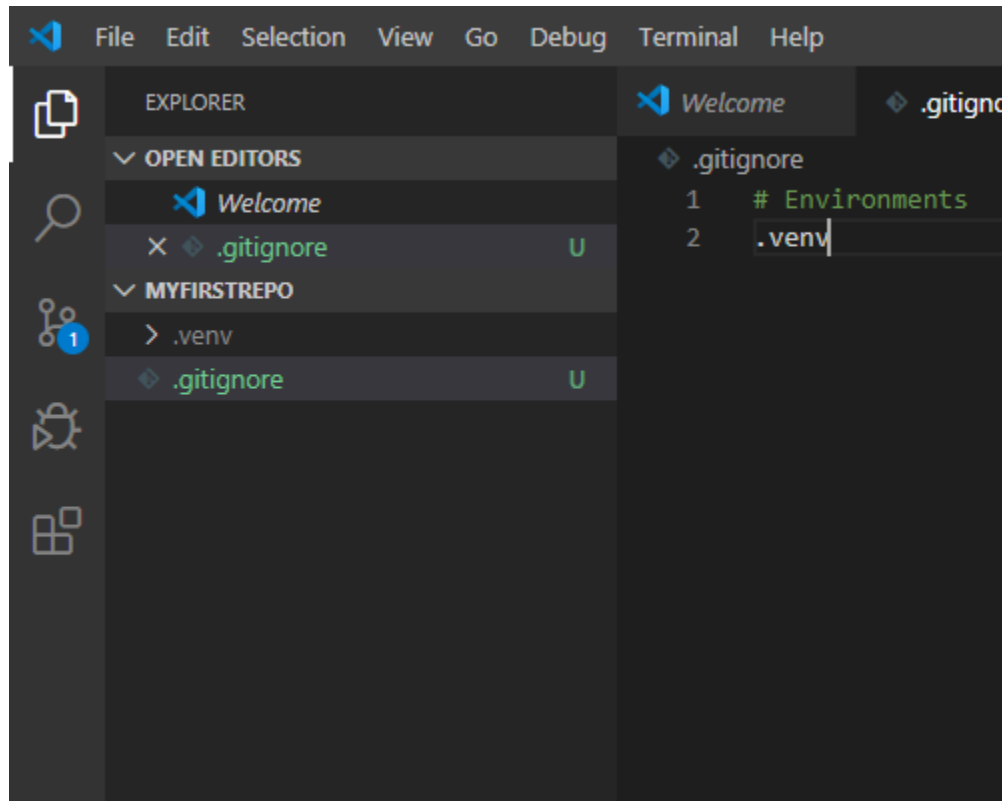
Integration with Git

VSCode already has an integration for Git and should recognize the hidden `.git` directory from our repository automatically.

To check our repository, change to the version control section (Ctrl+Shift+G).

In this section we see all files that git recognizes as changed.

You should see plenty of changed files at this stage cause by our virtual environment directory `.venv`. But since we definitely don't want to have these files in our repository, we need to make sure that Git won't include them. This is where we create a new file in our directory called `.gitignore`.



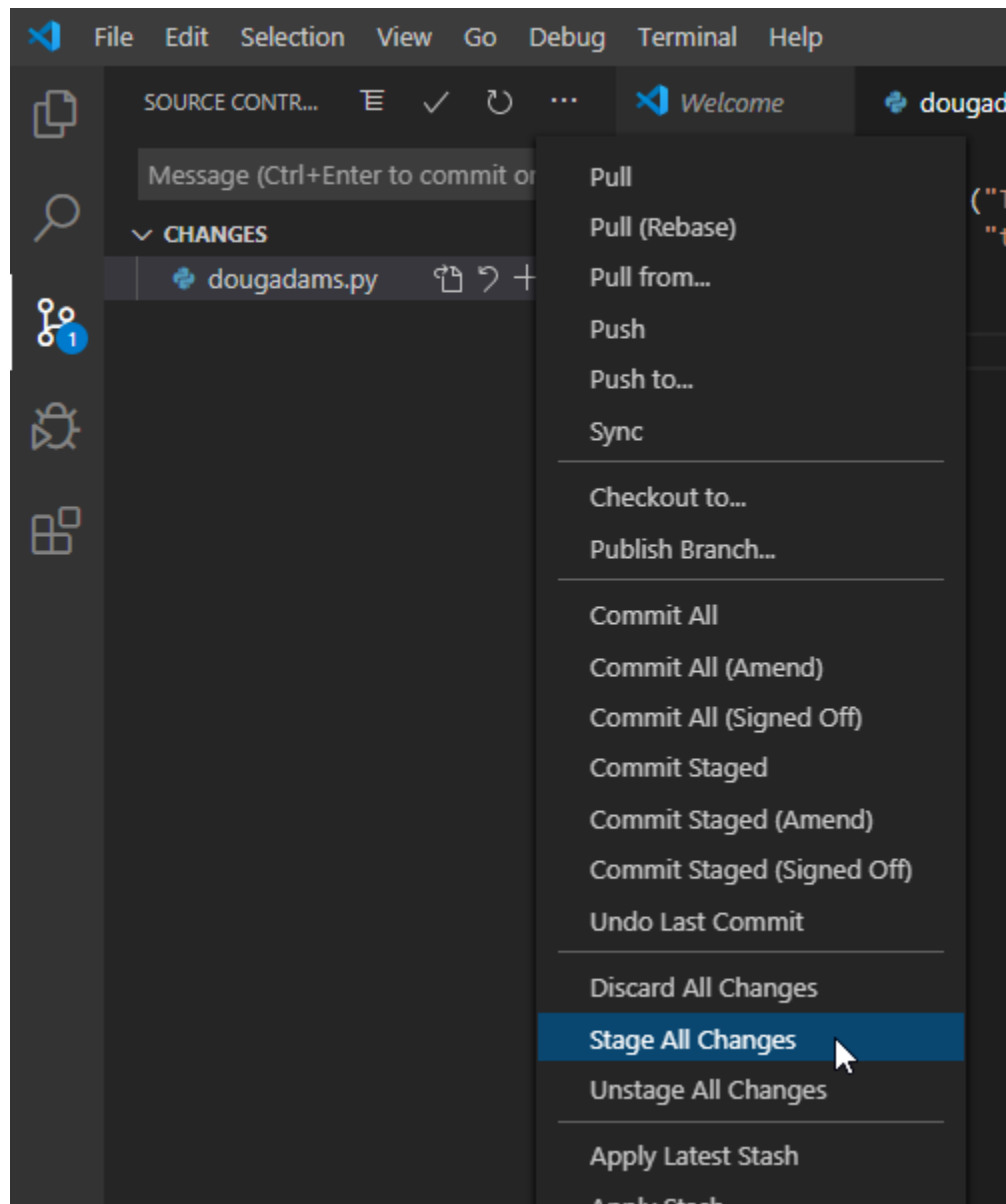
And add a single line to it.

```
# Environments
.venv

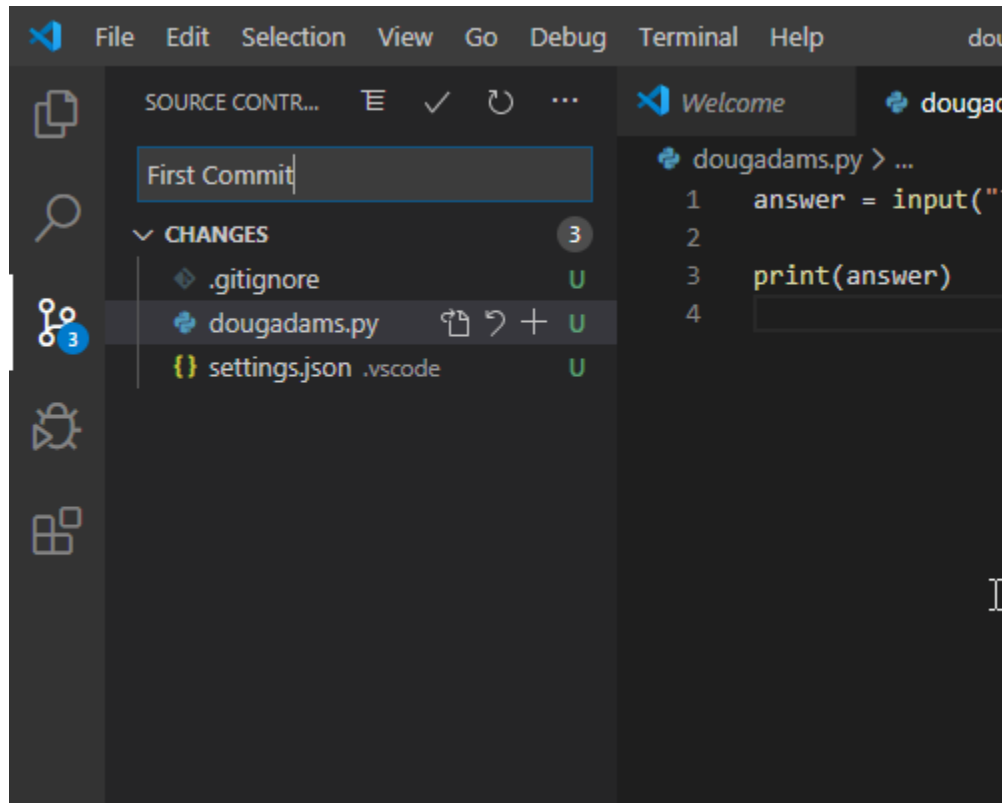
# Settings
.vscode
```

Git will ignore everything that is defined in such a `.gitignore` file. In our case the whole directory `.venv`. Save the file when ready. You should now see way less files than before. If not then something went wrong here.

Before we can commit our files to our repository, we need to stage them. This is part of a typical Git workflow and topic at some other time.



After all files are staged, write a commit message (mandatory) and commit (Ctrl+Enter).



I guess that's it for now. We have set up a small development environment on Windows 10 and you should be able to write and run your first applications without breaking them by dependency issues.

3.1.7 What next?

Well, there are still plenty of things you can do to optimize your environment. But also many things to learn for becoming more familiar with your tools and your personal workflow.

Some homework for the near future:

- Get to know Git a little bit better - Atlassian has a pretty nice [manual](#) for this
- Learn Python -
-

3.2 Structure and Requirements

All of the guides in this section need to include a minimum set of requirements to meet a setup that can be used by developers:

- Operating System specific commands
- Git as version control
- Use of a virtual environment if available for the uses language(s)
- At least one Text Editor or IDE with at least: * Syntax Highlighting * Git Integration
- Tests and checks to see if the instructions worked

The basic chapter structure for each guide is:

- Overview
- OS Requirements - Project Directory
- Version Control - Git
- [Language, e.g. Python] - (if available) Virtual Environment
- [Editor/IDE] - Download - Installation - Basic Setup - Integration with Git - Syntax Highlighting

To write your own guide you can make a copy of the 'setupguide_template.rst' which includes the structure above.

SOURCES

The content of this guide is not only based on personal experience and my own setup. This collection is also influenced and inspired by other blog posts and articles on this topic. This is just a list of the articles and posts I can recommend on this issue. You will find many more ways and ideas of setting up your personal environment, so please have a look on your own.

4.1 Cisco Devnet

Since my main career is being a Cisco collaboration solution engineer and consultant. And since I've written this guide primarily for a Devnet Express event session, this list of Leaning Labs from [Cisco Devnet Learning Labs](#) should not be omitted from this list.

- [What is a Development Environment and why do you need one?](#)

4.2 Others

- [15 Ways to Bypass the PowerShell Execution Policy](#) by Scott Sutherland

TODO

- Git Proxy Information

INDICES AND TABLES

- `genindex`
- `search`