Python Setup Guide

This guide will teach you how to set up Git, Python, and your practical folder so you're all set for the **Python Programming Language** course. Depending on your prior knowledge this should take 20min - 1.5 hours!

Prerequisites:

- Basic understanding of the terminal (change directory, make directory, move files), if you have no idea check e.g. this video or check out one of the various online guides.
- You have a vague idea of what Python is, if not check e.g. this video
- You have a clean Windows/Linux/Mac system available. If you've installed Python before skip/adjust the steps as needed. However note that the cleaner your system the better we will be able to help you.
- MacOS: Homebrew installed

Git Setup

"Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency."

Linux

`git` should come pre-installed with almost all Linux distributions. Verify via

```
git --version
```

MacOS

`git` might come pre-installed, check via:

```
git --version
```

If you don't get a version install `git` via

```
brew install git
```

Restart the console and verify git is installed as above.

Windows

1. In the PowerShell enter

```
winget install --id Git.Git -e --source winget
```

2. Restart the PowerShell
3. Run

```
git --version
# Should return something like git version 2.52.0.windows.1
```

GitHub Setup

GitHub is a service for hosting remote git repositories. You will learn more during the practical. Go to the GitHub webpage and create an account.

Configuring git

Run

```
 git config --global user.name "Your Name"
git config --global user.email "you@example.com"
# Replace the fields with your name and email
# Verify via
git config --global --list
```

Installing Python and pip

pip is the standard package installer and management tool for Python. It is used to install 3rd party Python packages (package = bundle of code).

Linux/MacOS:

Python

1. Python usually comes pre-installed. Open the console and try:

```
python --version
# if not working try
python3 --version
```

If you get something like `Python 3.12.3` you're good to go.

2. If you instead get `Command 'python'/'python3' not found` or a version <3.12 try:

```
# Linux:
sudo apt update
sudo apt install python3

# Mac:
brew install python
```

3. Restart your terminal afterwards. For non-Ubuntu distributions look for help online.

Installing pip

1. Might be pre-installed

```
pip --version
# or
pip3 --version
sudo apt install python3-pip
```

2. If not found

```
# Linux:
sudo apt install python3-pip

# MacOS:
# Should automaticall be installed via brew install python, might need to reinstall
```

3. Restart your terminal and verify again

Windows:

1. Open the Windows PowerShell (Win key -> Search for PowerShell), if you can't find PowerShell follow this
2. Run

```
winget install Python.Python.3.12
```

Press `Y` and enter to accept the license agreement. Restart the PowerShell. 3. Check Python is installed. Pip should also be automatically installed:

```
python3 --version
# or
python --version
# and
pip --version
```

Virtual Environments

An "environment" is the collection of your Python interpreter and installed libraries. By default, `pip` installs new packages into your **global environment**, making them available everywhere. Over time this can bloat your installation and create conflicts, especially when different projects require different package versions.
To avoid these issues, we use *virtual environments*: Isolated spaces with their own project-specific packages.

⚠ It is **strongly recommended** to install packages only inside virtual environments, to keep your global environment as clean as possible.

Here we will use a package called `venv` to create and manage Virtual Environments, which is part of the Python standard library.

Your first virtual environment

1. Create a new folder at a location of your choice called `venv_test`
2. Change directory to venv_test
3. Run

```
 # Create a virtual environment named .venv_test
python -m venv .venv_test
# On some Linux distributions you may first need to run
sudo apt install python3-venv

# Activate it:

# macOS / Linux:
source .venv_test/bin/activate
# Windows:
.venv_test/Scripts/Activate.ps1
# You may need to allow execution via PowerShell:
Set-ExecutionPolicy Unrestricted -Force



# You should notice ".venv_test" appearing at the beginning of the terminal prompt
```

4. Install a package and verify you can import it

```
 pip install numpy
python
# This opens the Python terminal.
# Now enter
import numpy
# and press enter
# exit the Python terminal:
exit()
# and press enter
```

5. Create a sample Python script

```
 # Linux/MacOS:
touch first_python_file.py

# Paste as one command:
cat << 'EOF' > first_python_file.py
import numpy as np
test_array=np.linspace(1, 10, 10)
print('Hello World!')
print(f'Numpy array: {test_array}')
EOF

# Windows (Past as one command):
@"
import numpy as np
test_array = np.linspace(1, 10, 10)
print("Hello World!")
print(f"Numpy array: {test_array}")
"@ | Set-Content -Encoding utf8 first_python_file.py


# Verify file content:
cat first_python_file.py

# Run the file:
python first_python_file.py

# Should output:
# Hello World!
# Numpy array: [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

6. Deactivate the virtual environment either by restarting the console or via

```
deactivate
```

7. Verify `numpy` is not installed

```
 python
# This opens the Python terminal.
# Now enter
import numpy
# and press enter, should raise ModuleNotFoundError
```

⚠ Remember to manually change your virtual environment when switching between projects.

Setting up a folder for the practical

1. Create a directory at a path of your choice called `python_practical_2026`
2. Enter the directory
3. Run

```
 # Create a new virtual env
python -m venv .venv_practical

# Activate it
# macOS / Linux:
source .venv_practical/bin/activate

# Windows:
.venv_practical/Scripts/Activate.ps1
```

4. Installing packages The practical will make use of interactive files called `Jupyter notebooks`. They mix Markdown (which is what was used to write this file) with Python code that can be run in the file itself. They're a bad tool for actual projects but very useful for guides and tutorials.
   ⚠ Make sure that your venv is active and run:

```
pip install jupyter
```

   Under Windows this may fail because Windows has a default limit on the length of file names (pretty dumb imo). To fix this you may need to do some spooky changes in the Registry:

   1. Windows key -> Type `regedit` -> Enter
   2. Navigate to `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`
   3. Find the value `LongPathsEnabled` and double click it
   4. Change the value to "1"
   5. Retry installing Jupyter

5. Before each practical day create a folder called `week x`, replace x with current week
6. Download the Jupyter notebook for the day from Canvas and place it into the folder

Your final folder structure should look something like:

```
python_practical/
├── .venv
├── week1
│   ├── exercise1.py
│   └── solutions1.py
└── week2
...
```

Running the practical exercises

1. Navigate to the practical folder and activate your venv.
2. Start a local `jupyter notebook` server by running

```
jupyter notebook
```

   This will produce a bunch of cryptic looking output. Towards the end there should be a line like `http://localhost:8888/tree?tokenSOME_CRYPTIC_SEQUENCE`.
3. Copy the line by marking and right-clicking it
4. Paste the line as a URL into a browser of your choice
5. Tadaa, the jupyter notebook interface should load
6. Navigate to the exercises and you're good to go!

Installing an IDE

You could write Python code with a basic text editor, however there exist editors specifically designed for programming, called IDEs (Integrated development environment).
At their core they are text editors that provide comfort functions, such as automatic syntax highlighting, autocomplete, git integration, and debugging.
⚠ We highly recommend using one for your group projects.
I personally use VSCode, since it's easy to set up, relatively lightweight, works for multiple programming languages, and is well maintained and documented.
For installation please refer to the official installation instructions. Python specific setup is found here.
Start VSCode from the console inside your project folder while your virtual Environment is active, VSCode should automatically select the correct interpreter.
There is a lot of stuff VSCode can do to make your life easier, check out one of the multiple guides on YouTube or the official VSCode documentation to learn what you can do! Other common IDEs include PyCharm, Atom, and Spider. Which one you choose comes down to personal preference!

Further information

This guide is intended to provide you with the minimal instructions while maintaining good practices to participate in the practical. Therefore I will add some further information here, that is **not relevant for the practical**.

Package and environment management

pip and venv are the most common and basic tools for package and environment management. However there exist a plethora of other tools that cover tasks that go beyond the scope of this course.
I will mention some of them here and some other cool tools.
`conda` is part of the `Anaconda` Python suite. It's fairly common, especially in Biology and DataScience. It's both a package manager and environment manager, and even works for non-Python software.
`poetry` is a dependency manager and environment manager better suited for publishing projects as packages.
`pyenv` is a tool for managing and switching between multiple Python versions on the same machine.
`black` provides automatic formatting to your code.
`flake8` is a common linter, meaning it catches syntactic errors in your code and warns you of e.g. unused imports. `pytest` is used to generate tests for Python

General

You can set this venv up such that when you enter your project folder the virtual environment is automatically activated. You can find guides online or use the LLM of your choice.