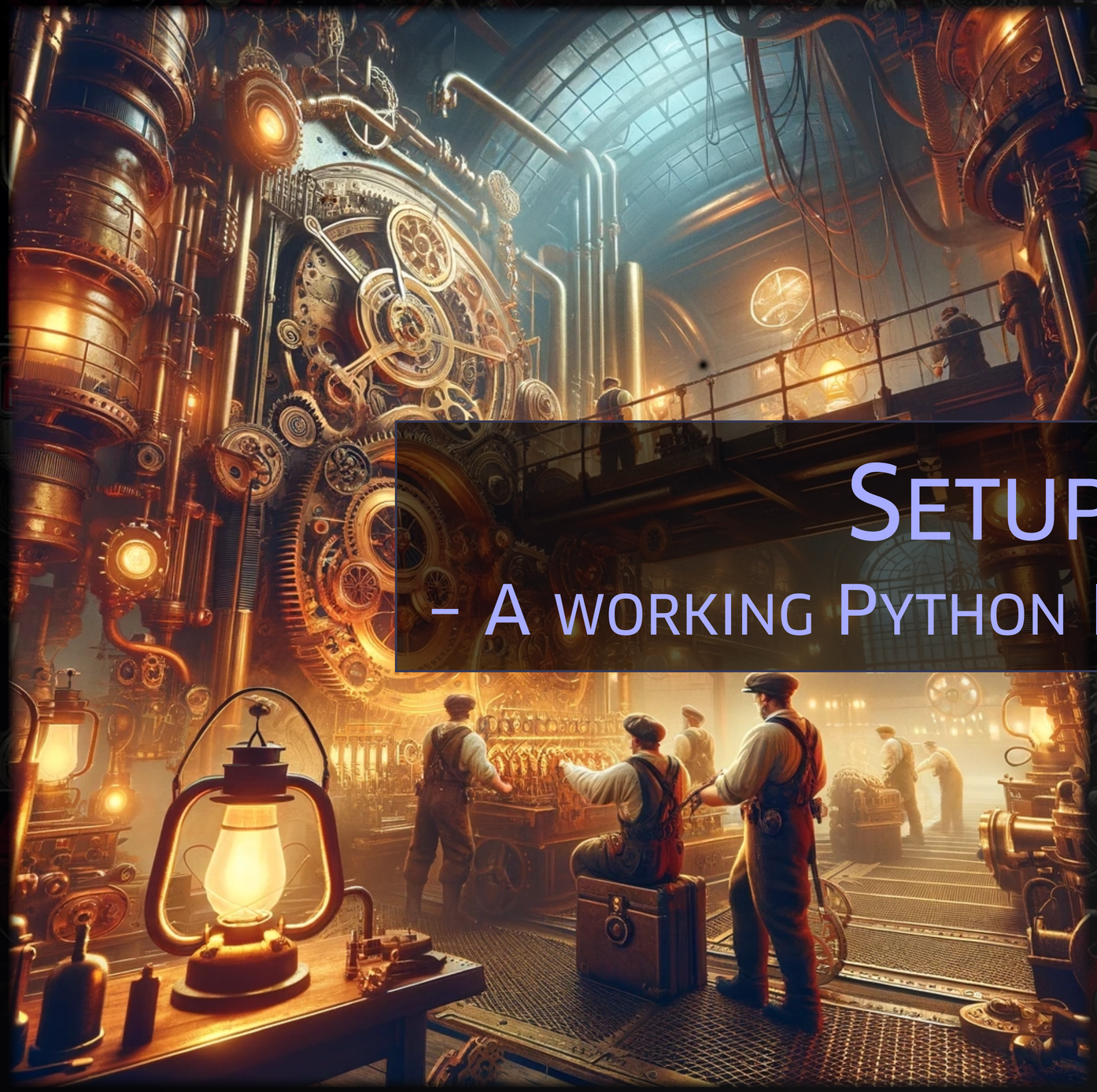
Python-Docs

– Setup –

Code Based Learning

Prof. Dr. rer. nat.
Alexander Voß

Inform-Professur
FH Aachen



Code Based Learning

Setup

– A working Python Installation –

- Git, clone
- Python Interpreter
- pyenv
- pip3
- venv

git | Working Directory

Setup

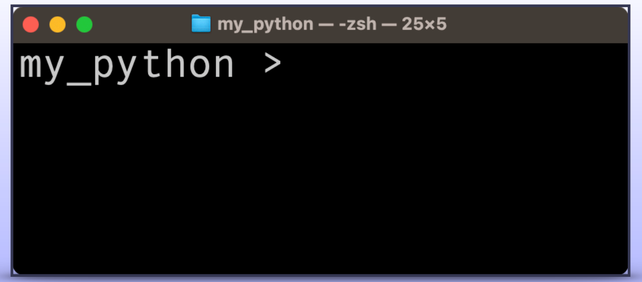
Throughout the course, you will need the version control program **git**. Chances are you have **git** already installed and are familiar with it. If not, you should definitely invest some time. You can start e.g. here.

```
> git --version
```

So let's assume from here on that git is available.

Furthermore, working from the command line (**terminal**, **shell**) is quite useful, at least for installation. Later, you can then use an IDE such as **PyCharm** to run Python scripts.

Now open a terminal and create a **working directory** of your choice for the teaching and exercise units.
Here in the example this is **my_python**.



zsh (shell) / terminal window

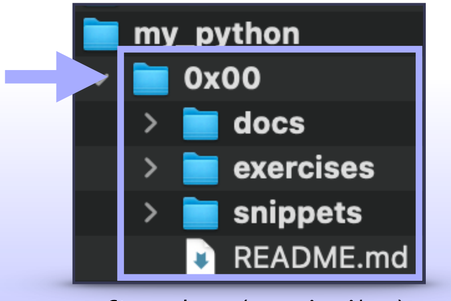
Repository python_course

Now clone the repository **python_course** into the folder you just created. Here you will find, among other things, Python lectures (snippets), background information, exercises and, last but not least, this document.

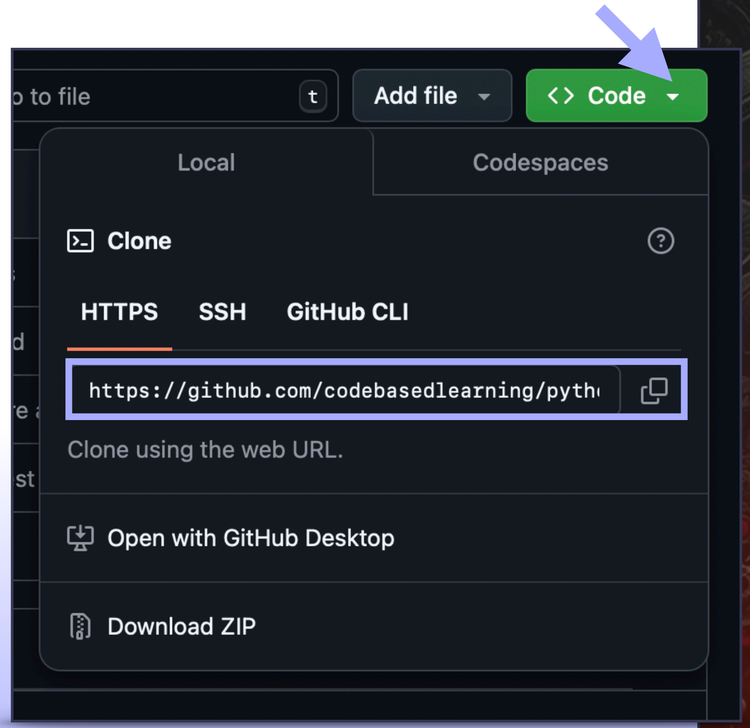
The repository is available on GitHub. Under **Code** you will find the **path** to the repository for the **git clone** command.

Let's clone with this command...

```
> git clone <repo path>
```



after **clone** (or similar)



Project **python_course** in GitHub

Python Program

Similar to Java, we need a **Python program to run Python scripts**.

To avoid confusion, we always write **python3** (and **pip3**), although on systems where only Python 3.x is installed, **python** (or **pip**) usually works as an alias as well. We will go into the structure of the code example later.

If Python is installed, this will run a script, e.g. **hello_world.py** in **0x00**, from the snippets folder:

```
> python3 <script>
> python3 --version
```

```
3  import platform
4
5  print(f"Hello World! (python {platform.python_version()})")
```

hello_world.py in PyCharm

```
snippets –
snippets > python3 hello_world.py
Hello World! (python 3.9.13)
snippets >
snippets > python3 --version
Python 3.9.13
snippets >
```

hello_world and python3 show version

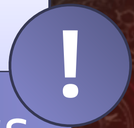
Compiler | Interpreter | Scripts

Python is often referred to as an *interpreter* and the language is compared with *compiled* languages such as C++ in terms of speed, for example.

> **Basically, however, *interpreted* or *compiled* is not a property of the language at all, but of the implementation!**

The exact differentiation between these two properties is fuzzy. What can be said is that the most commonly used Python implementation, namely the standard distribution `CPython`, combines the properties of compiler and interpreter: Python source code is compiled into byte code instructions, i.e. translated for the so-called *Python Virtual Machine (PVM)*. This byte code is then executed there, i.e. *interpreted*. This is basically similar to `Java` or `C#`.

The `python3` application performs both steps when we run a `*.py` program, aka. *script*.



We speak, well aware of this vagueness, in the following nevertheless of the *Python interpreter* and *scripts*.

Working with Python – Three Options

Options:

- *The Purist:* An editor, an installed Python, a terminal and done.
- *Mr. Comfort:* An IDE and an installed Python.
- *VM Lover:* An IDE and Python, but Python installed in Docker.

And everything in between…

Common problems encountered:

- Your operating system uses the supplied Python version 3.9.13, but your project requires 3.11.1.
- One project needs a library in version 2.1, but another project needs version 3.1.

You should not just install Python!

We need a solution to this version hell - and this is called a *virtual environment*, e.g. managed by `pyenv` and `venv`.

To be fair, there are other tools, such as `Conda` or `Poetry`, but we will focus on `pyenv` and `venv` for now.

Manage Python Versions | pyenv | pip | cbl | venv

Leaving aside the *VM lover's* expert approach for now, we need Python installed in both first cases!

Using the **pyenv** program, we can not only leave an installed version alone, but also install and manage different versions of Python. This is useful, for example, if you want to try out a language feature that is currently under development.

Installation of **pyenv** is sometimes a bit tricky and varies depending on the operating system (OS). In the following we show:

- how to get access to the program **pyenv** including installation help,
- use **pyenv** to install and activate different Python versions, and
- use **pip3** to install the **Code Based Learning** test library **cbl** in different versions in so-called *virtual environments*, previously created with **venv**.

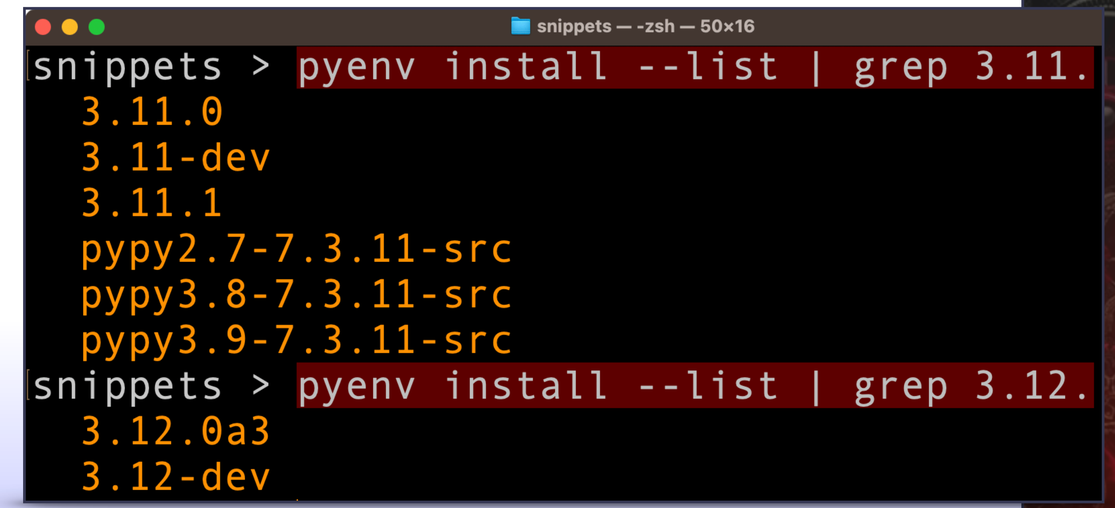
Install pyenv | Python Versions available

Setup

Familiarise yourself with the installation of **pyenv** for your OS (Linux, macOS and Windows) and install **pyenv** from GitHub. You can skip the explanations about the *path*, *shims* etc. for now and have a look at *Getting Pyenv* and *Set up your shell environment* (see *Table of Contents*). In the text you will also find a link to the pyenv Windows fork.

You have done it if **pyenv** responds to the request for *available* Python installations (ignore the **pypy*** lines):

```
> pyenv install --list | grep <prefix>
> pyenv --version
```

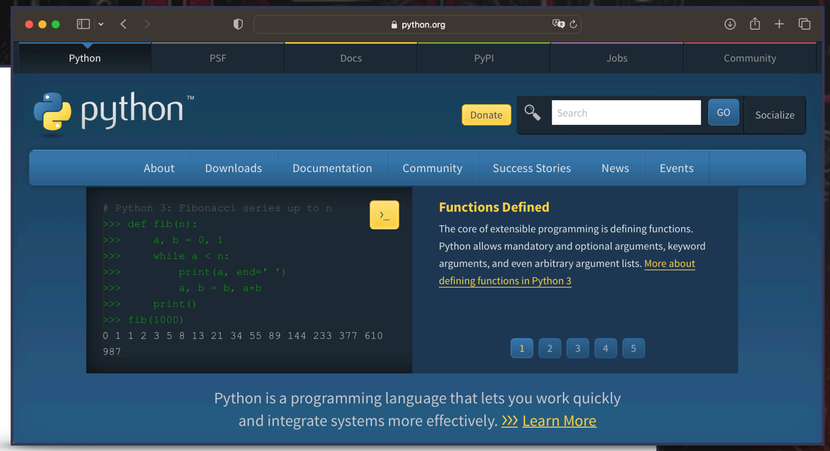


pyenv lists Python versions available (or newer)

For **Powershell** it is **pyenv-win** or **pyenv.bat** or **pyenv.ps1**.

Python Versions installed

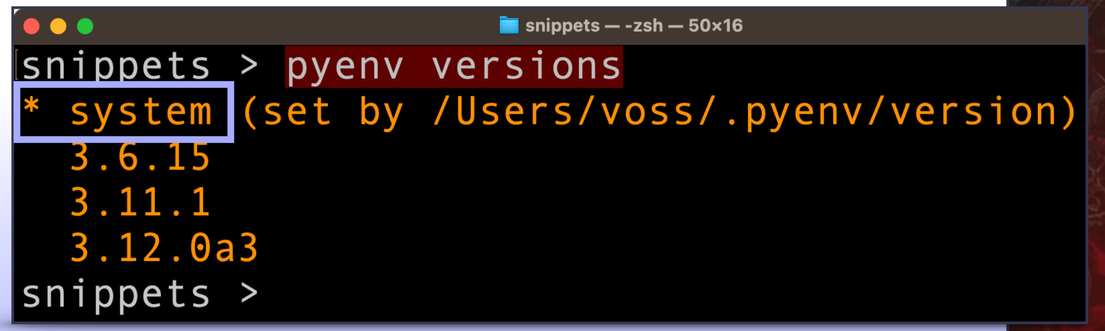
Below we discuss some **pyenv** commands, including those for installing and activating specific Python versions. In February 2025 the last stable Python version is 3.13.2 and the last supported Python pre-release is 3.14.0a5. See Python for more information on the current development status.



www.python.org

There may already be Python versions installed. You can get the list of *installed versions* with:

```
> pyenv versions
```



pyenv lists Python versions installed (or newer)

The currently used or active one is the one marked with *.

Install Python

Setup

Install a specific Python version with:

```
> pyenv install <version>
```

If you don't know which version, take the latest release (Feb. 2025: 3.13.2).

!

After installation this version is not yet active.

```
snippets — -zsh — 50×23
snippets > pyenv versions
* system (set by /Users/voss/.pyenv/version)
  3.6.15
  3.11.1
  3.12.0a3
snippets > pyenv install 3.9.13
python-build: use openssl@1.1 from homebrew
python-build: use readline from homebrew
Downloading Python-3.9.13.tar.xz...
-> https://www.python.org/ftp/python/3.9.13/Python
-3.9.13.tar.xz
Installing Python-3.9.13...
Installed Python-3.9.13 to /Users/voss/.pyenv/vers
ions/3.9.13
snippets > pyenv versions
* system (set by /Users/voss/.pyenv/version)
  3.6.15
  3.9.13
  3.11.1
  3.12.0a3
```

installing a version (example)

Activate Python

Activating a specific global Python version (explanations about global and local later) is done this way:

```
> pyenv global <version>
```

Here (version) system stands for the standard version installed in the OS. This example uses version 3.11.1, please replace it with your version.

On macOS, this setting is stored in the ~/.pyenv/version file.

```
snippets — -zsh — 50×26
snippets > pyenv versions
* system (set by /Users/voss/.pyenv/version)
  3.6.15
  3.11.1
  3.12.0a3
snippets > python3 hello_world.py
Hello World! (python 3.9.13)
snippets > python3 --version
Python 3.9.13
snippets > pyenv global 3.11.1
snippets > pyenv versions
  system
  3.6.15
* 3.11.1 (set by /Users/voss/.pyenv/version)
  3.12.0a3
snippets > python3 hello_world.py
Hello World! (python 3.11.1)
snippets > python3 --version
Python 3.11.1
snippets > pyenv global system
snippets > pyenv versions
* system (set by /Users/voss/.pyenv/version)
  3.6.15
  3.11.1
  3.12.0a3
```

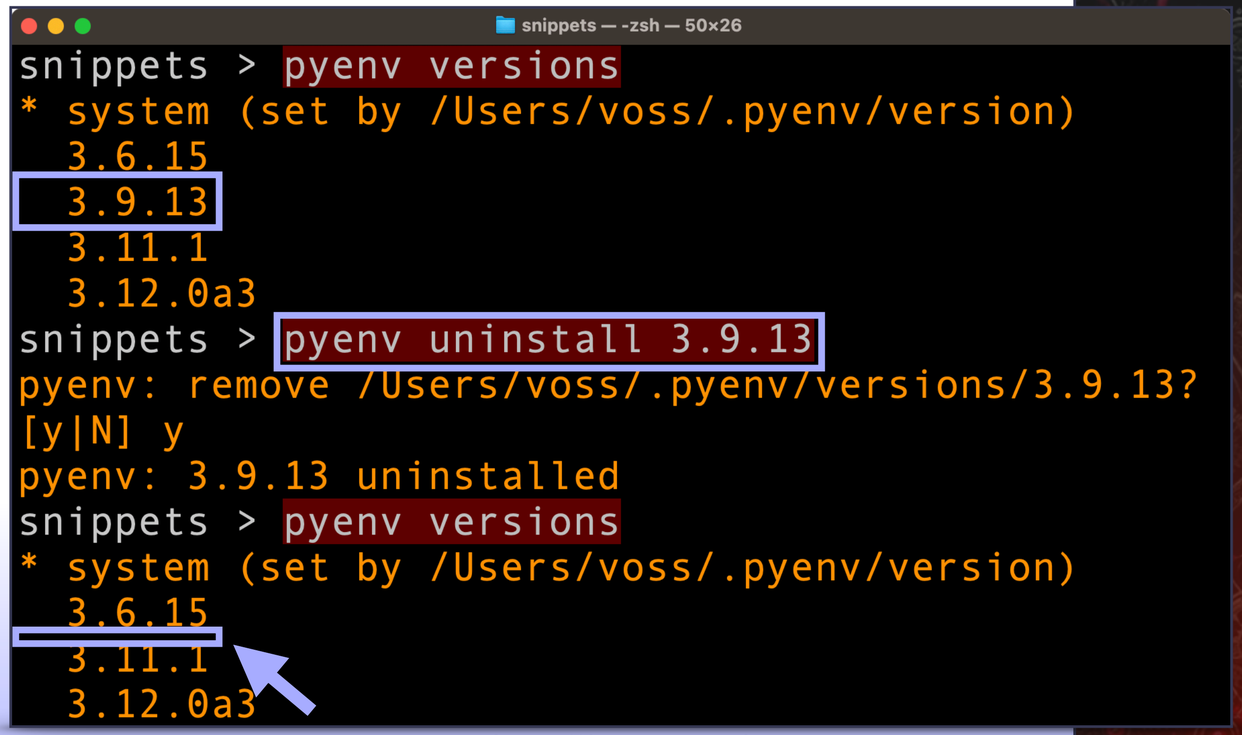
setting global version

Uninstall Python

If you want to remove a version, then with this command:

```
> pyenv uninstall <version>
```

Attention: Do not uninstall a version that is still needed in a project.



uninstalling a version

Combine Python and Packages | cbl

Setup

We will now install different versions of the cbl package, *locally* and *globally*. *Locally* means in a so-called *virtual environment* - we'll get to that later. The program tries to import this package.

The following examples show how we can choose and combine the version(s) of Python and libraries for each project.

```
21      print(f"{_m}python {platform.python_version()}{_M} {pretty_pa
22      try:
23          import cbl
24          print(f"{_m}cbl {cbl.setup.about_package().version}{_M} {
25      except ModuleNotFoundError:  # Error message replaced for dida
26          return f"{_m}'cbl' package not installed{_M} ('> pip3 ins
```

about_setup.py in PyCharm, implementing a success and a failure case

```
snippets — -zsh — 55×26
snippets > python3 about_setup.py
python 3.9.13 /usr/local/opt/python@3.9/bin/python3.9
'cbl' package not installed ('> pip3 install cbl')
```

about_setup shows, that cbl is not available

Combine globally | Choose Python

Setup

First, we set the global Python version to a non-system version, e.g. 3.11.1.

As expected, and as we have already seen, the cbl package cannot be loaded because we have not installed anything yet.

Next will show you how to implement project-specific installations.

```
snippets — -zsh — 55×22
snippets > pyenv global 3.11.1
snippets > pyenv versions
  system
  3.6.15
* 3.11.1 (set by /Users/voss/.pyenv/version)
  3.12.0a3
snippets > python3 about_setup.py
python 3.11.1 ~/.pyenv/versions/3.11.1/bin/python3
'cbl' package not installed ('> pip3 install cbl')
```

setting Python 3.11.1 as global version

Again, it is important to note that all projects using this version of Python are affected, for better or worse.

Combine globally | Install cbl Package | pip3

Setup

Now we install the package in a specific version, here 0.2.3, using the pip3 tool – with success.

If you omit the version specification, the latest package version is used.

```
snippets — -zsh — 55×22
snippets > pip3 install cbl==0.2.3
Collecting cbl==0.2.3
  Using cached cbl-0.2.3-py3-none-any.whl (4.7 kB)
Installing collected packages: cbl
Successfully installed cbl-0.2.3
snippets > python3 about_setup.py
python 3.11.1 ~/.pyenv/versions/3.11.1/bin/python3
cbl 0.2.3 ~/.pyenv/versions/3.11.1/lib/python3.11/site-packages/cbl
```

about_setup shows, that cbl is now available

> pip3 install <package>==<version>

In addition to the installed version, the script also shows where the library is installed, namely next to the Python 3.11.1 version in the lib folder.

cbl is now installed *globally* for Python 3.11.1.

Manage Packages | pip3 | show | uninstall

Setup

You can also get more information about installed packages using **pip3**.

```
> pip3 show <package>
```

Of course, you can also uninstall the package.

```
> pip3 uninstall <package>
```

```
snippets — -zsh — 58×22
snippets > pip3 show cbl
Name: cbl
Version: 0.2.3
Summary: A small example package
Home-page:
```

pip lists package information

```
snippets — -zsh — 58×22
snippets > pip3 uninstall cbl
Found existing installation: cbl 0.2.3
Uninstalling cbl-0.2.3:
  Would remove:
    /Users/voss/.pyenv/versions/3.11.1/lib/python3.11/site
-packages/cbl-0.2.3.dist-info/*
    /Users/voss/.pyenv/versions/3.11.1/lib/python3.11/site
-packages/cbl/*
Proceed (Y/n)? Y
  Successfully uninstalled cbl-0.2.3
snippets > pip3 show cbl
WARNING: Package(s) not found: cbl
snippets > python3 about_setup.py
python 3.11.1 ~/.pyenv/versions/3.11.1/bin/python3
'cbl' package not installed ('> pip3 install cbl')
```

uninstalling cbl

Combine locally | Choose Python

Setup

Enter the project directory (one level up from snippets to 0x00) because the following structures are usually created there.

Now we select the Python version 3.12.0a3 *locally*, i.e. for the project only. This creates or updates a .python-version file containing the local version.

```
0x00 — -zsh — 55×24
0x00 > pyenv local 3.12.0a3
0x00 > cat .python-version
3.12.0a3
0x00 > python3 snippets/about_setup.py
python 3.12.0a3 ~/.pyenv/versions/3.12.0a3/bin/python3
'cbl' package not installed ('> pip3 install cbl')
0x00 > python3 --version
Python 3.12.0a3
0x00 > pyenv versions
  system
  3.6.15
  3.11.1
* 3.12.0a3 (set by /Users/voss/Projects/code-based-lear
ning/python/course/my_python/0x00/.python-version)
```

choosing Python 3.12.0a3 as local version

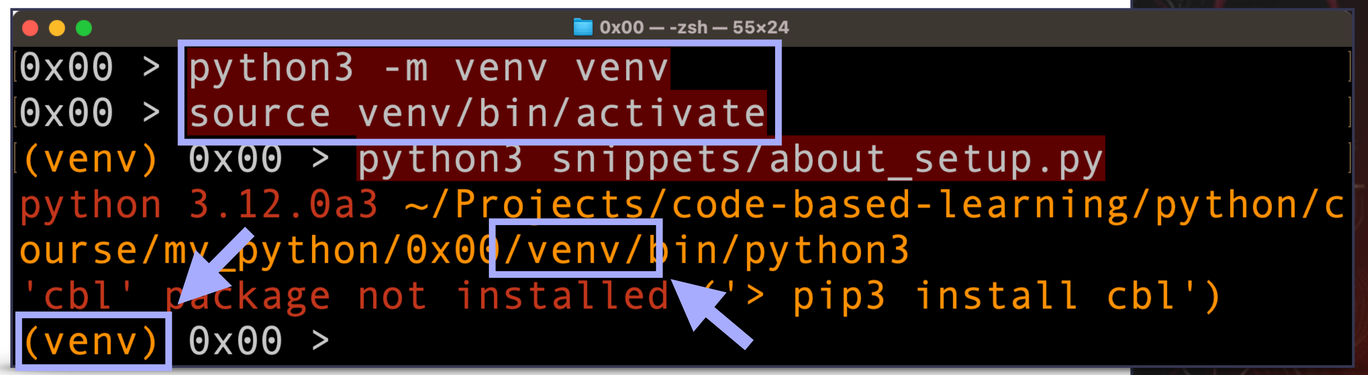
> pyenv local <version>

about_setup also respects this configuration and pyenv displays it too. cbl is not yet installed.

Combine locally | Manage Virtual Environment | venv

Setup

The Python module **venv** manages *virtual environments*. The idea behind this is to assemble a specific version of Python and the necessary libraries and run a Python program in exactly this environment.



environment **venv** is activated on Linux/macOS

First *create* the virtual environment **venv** via the *module* **venv** and *activate* it:

```
> python3 -m venv <folder >
> source <folder>/bin/activate
> <folder>/Scripts/Activate.ps1
```

Linux/macOS

Powershell Windows

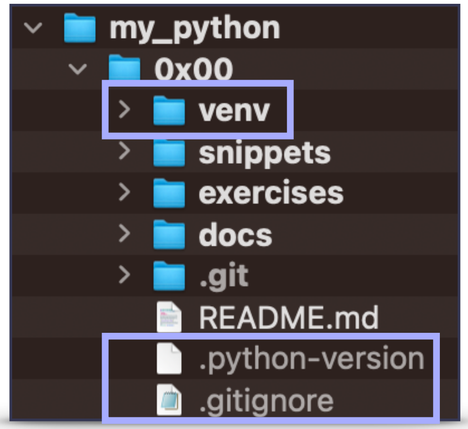
Note that the local environment is displayed in the prompt.

Actually, it's not so clever to call the folder venv as well, but that's quite common.

Now we have a *local* Python installation based on version 3.12.0a3, as you can also see from the Python-path.

Virtual Environment | Remarks | venv

- Calling Python with parameter `-m` executes `venv` as a module. This is the preferred way for virtual environments from Python 3.5.
- The name and location of the folder `venv` varies depending on the project or policy. Some call it `.venv` and/or do not place it inside the project folder at all. Here it is called `venv`, although `.venv` would better reflect its internal nature (like `.git` or `.idea`). However, depending on the OS, these folders are *not always visible* and if you want to select a local Python version in an IDE, then invisible folders are impractical…



folder structure with `venv` (or similar)

- The folder `venv` should not be versioned, so it is listed in `.gitignore`.
- Whether you version file `.python-version` depends on your overall project structure, there are pros and cons.

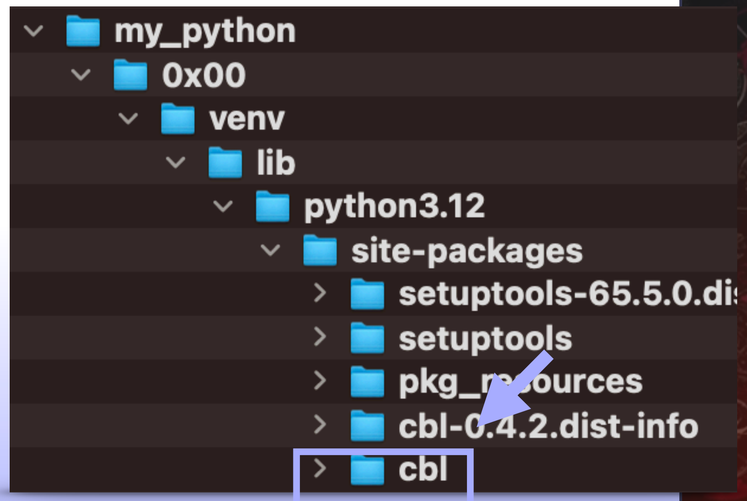
Combine locally | Install cbl Package | pip3

Setup

Equipped with the new **venv** virtual environment, we can now install project-specific libraries locally, e.g. a different version of **cbl** (0.4.2).

```
0x00 — -zsh — 55×24
(venv) 0x00 > pip3 install cbl==0.4.2
Collecting cbl==0.4.2
  Using cached cbl-0.4.2-py3-none-any.whl (4.4 kB)
Installing collected packages: cbl
Successfully installed cbl-0.4.2
(venv) 0x00 > python3 snippets/about_setup.py
python 3.12.0a3 ~/Projects/code-based-learning/python/course/my_python/0x00/venv/bin/python3
cbl 0.4.2 ~/Projects/code-based-learning/python/course/my_python/0x00/venv/lib/python3.12/site-packages/cbl
(venv) 0x00 > pip3 show cbl
Name: cbl
Version: 0.4.2
Summary: A small example package
```

cbl installed locally



folder structure with cbl (or similar)

Manage Virtual Environment | source | activate | deactivate

The following command deactivates the virtual environment.
However, it is not deleted but can be reactivated with source or Activate (Windows, see before) at any time...

```
(venv) 0x00 > python3 snippets/about_setup.py
python 3.12.0a3 ~/Projects/code-based-learning/python/course/my_python/0x00/venv/bin/python3
cbl 0.4.2 ~/Projects/code-based-learning/python/course/my_python/0x00/venv/lib/python3.12/site-packages/cbl
(venv) 0x00 > deactivate
0x00 > python3 snippets/about_setup.py
python 3.12.0a3 ~/.pyenv/versions/3.12.0a3/bin/python3
'cbl' package not installed ('> pip3 install cbl')
```

about_setup shows the current setting, cbl is no longer available

```
> deactivate
```

You will also see that the command prompt changes again to reflect the current setting.

Manage Packages | pip3 | freeze

Setup

When you create a virtual environment, the installed libraries of the underlying Python version are not initially taken along. To *transfer* the libraries, you can collect them using

```
> pip3 freeze > <filename>
```

Note that the '>' between **freeze** and **filename** is no typo and redirects the output to the file, e.g. **pip3 freeze > requirements.txt**

This will create a file containing all the libraries you need. Then use this as a source to install them locally with **pip3** as seen before.

```
> pip3 install -r <filename>
```

Freezing alone, i.e. without the redirection, just lists the packages.

Switch Virtual Environments

It is also no problem to switch between two virtual environments, e.g. venv1 and venv2.

In this example cbl is only installed in one environment (venv1).

```
0x00 — -zsh — 61×25
0x00 % pyenv local 3.11.1
0x00 % python3 -m venv venv1
0x00 % python3 -m venv venv2
0x00 % source venv1/bin/activate
(venv1) 0x00 % pip3 install cbl
Collecting cbl
  Using cached cbl-0.4.2-py3-none-any.whl (4.4 kB)
Installing collected packages: cbl
Successfully installed cbl-0.4.2
(venv1) 0x00 % python3 snippets/about_setup.py
python 3.11.1 .../0x00/venv1/bin/python3
cbl 0.4.2 .../0x00/venv1/lib/python3.11/site-packa
(venv1) 0x00 % source venv2/bin/activate
(venv2) 0x00 % python3 snippets/about_setup.py
python 3.11.1 .../0x00/venv2/bin/python3
'cbl' package not installed ('> pip3 install cbl')
```

switch between venv1 and venv2 on Linux/macOS

Setup

– STATIC CODE ANALYSIS –

Code Based Learning

Pylint

mypy

Static Code Analysis | Pylint

Setup

A *linter* is a tool that analyses your source code for potential errors, stylistic issues, and coding standard violations without actually running the code. One example is Pylint, so choose your favourite virtual environment or install it globally with

```
> pip3 install pylint
```

and call it with

```
> pylint <script>
```

```
voss@Mac python_course % pylint 0x01/solutions/poke_peach_one_solution.py
************* Module poke_peach_one_solution
0x01/solutions/poke_peach_one_solution.py:6:0: C0116: Missing function or
 method docstring (missing-function-docstring)


-------------------------------------------------------------------
Your code has been rated at 9.47/10 (previous run: 10.00/10, -0.53)
```

Pylint output

Pylint analyses your Python code and outputs a score (out of 10) along with a list of warnings, errors, and suggestions. Customisations (e.g., suppressing unhelpful warnings) can be made in .pylintrc.

Static Code Analysis | mypy

Setup

mypy is a static type checker for Python. It analyses a Python program's type annotations to ensure type correctness without executing the code. So choose your favourite virtual environment or install it globally with

```
> pip3 install mypy
```

and call it with

```
> mypy <script>
```

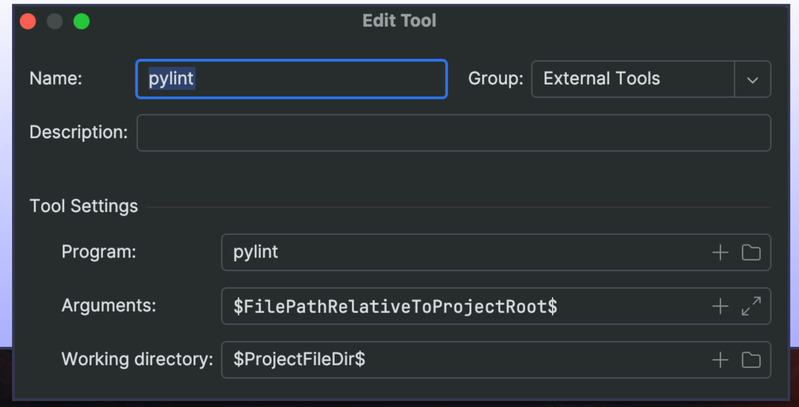
```
voss@Mac python_course % mypy 0x02/snippets/b_collections/b_shallow_and_deep.py
0x02/snippets/b_collections/b_shallow_and_deep.py:35: error: Unsupported target
for indexed assignment ("Sequence[object]")  [index]
0x02/snippets/b_collections/b_shallow_and_deep.py:41: error: Unsupported target
for indexed assignment ("Sequence[object]")  [index]
Found 2 errors in 1 file (checked 1 source file)
```

mypy output

Customisations can be made in mypy.ini.

Sometimes can also configure your IDE, e.g. PyCharm, to use mypy and/or Pylint as external tool.

PyCharm settings



Setup

– More Ways to run Code –

Code Based Learning

Executable Scripts

REPL

Executable Scripts

For the sake of completeness, it is also possible to start a Python script without calling python3 explicitly.

```
snippets — -zsh — 61×20
snippets % ls -la hello_script.py
-rw-r--r--  1 voss  staff  1028 Mar 20 00:23 hello_script
snippets % chmod +x hello_script.py
snippets % ls -la hello_script.py
-rwxr-xr-x  1 voss  staff  1028 Mar 20 00:23 hello_script
snippets % ./hello_script.py
Hello Script! (python 3.11.1)
snippets % chmod -x hello_script.py
```

running hello_script.py without python3 in Linux/macOS

The trick is to build the script accordingly (*shebang*) so that the shell starts python3 itself in the right environment. You will find the necessary information in the example hello_script.py.

The commands to make a script executable (Linux, macOS) and take it back again are

```
> chmod +x <file>
> chmod -x <file>
```

However, we will not pursue this approach any further.

For Windows start a script without opening a new window, assuming py Extension is handled by python:

```
> $env:pathext = $env:pathext+";.PY"
```

Interactive Python | REPL

Setup

The Python application **python3** has two modes: *script* and *interactive*.

In *normal* or *script mode*, the **.py** scripts are executed in the Python interpreter.

In *interactive mode*, or *interactive shell*, you can simply enter Python commands, e.g. this easter egg...

The whole process is also known as a **REPL**, standing for *Reading* (your input), *Evaluating* (your code), *Printing* (any output), *Looping* back to step one.

```
0x00 — python3 — 71×30
0x00 % python3
Python 3.11.1 (main, Jan 19 2023, 11:58:08) [Clang
.29.202)] on darwin
Type "help", "copyright", "credits" or "license"
>>> print("Hello")
Hello
>>>
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation t
There should be one-- and preferably only one --o
Although that way may not be obvious at first unl
Now is better than er.
Although never is better than *right* now.
is hard to explain, it's a
n is easy to explain, it may
onking great idea -- let's do
```

Licensed from envato

With a view to professional product development, we focus on working with an IDE.

n interactive mode

CODE BASED LEARNING

If you are missing something or have suggestions for improvement, please send me an E-Mail.