Computational Photography 2024

Setting up the environment

This guide leads you through the setup of the environment needed for the computational photography exercises.

We will use jupyter notebooks for the exercises. Each exercise consists of at least one notebook which you can directly work in.

If you used python before this is not going to be very different from the usual workflow to set up a development environment. But this guide will give you a detailed step-by-step approach taking into account different operating systems, too in case this is your first contact with python. So don't be afraid of the length and feel free to skip to the relevant parts.

1. Download the first exercise from Ilias.

You will find it in the Exercise section.

2. Extract the archive at a convenient place.

The first exercise comes with a directory structure like this:

```
cp_2024_exercises/
                    exercise_##/
                    ...
```

The recommended way to deal with it is to place this folder somewhere on your local hard drive (e.g. home) and add more directories to it as you go. We will refer to the *cp_2024_exercises* parent directory as the **exercises root** and the *exercise_##* as the **exercise folder**.

You will need to upload a zip - archive containing the whole *exercise_##* folder for the submission. You are encouraged to use git for version control if you like.

3. Install python.

Ok, this is the most complex part, actually.

The exercises were designed for `python3.9`, so we need to install that for your distribution.
If you know what you are doing you can set your system python to that version, but we generally recommend using either `pyenv` or `conda` to manage your python versions. This also allows you to switch between versions or install python versions on a machine on which you do not have root access.

If you are already working with conda this would be the easiest way but to reduce the necessary overhead, the `pyenv` route will be the preferred option here.

3.1 Using pyenv

In general, pyenv is a tool that lets you use multiple python versions next to each other on a system. You can associate a python version with a given directory or switch your global python as often as you like with a single command.

3.1.1 Linux

In a shell run:

```
curl https://pyenv.run | bash
```

Now restart your shell for the changes to take effect. You can also run

```
exec $SHELL
```

or source your `~/.bashrc` file.

Next you can use pyenv to install python 3.9.

Have a look at the available versions using:

```
pyenv install -l
```

and install one of the 3.9 versions.

```
pyenv install 3.9.x
```

`pyenv versions` gives you an overview over all installed python versions. Python 3.9.x should be one of them now.

We can associate this python version with your exercise root directory. Navigate into the exercise root or open a shell at this location.

Now run

```
pyenv local 3.9.x
```

The version number should match the one you installed before.

3.1.2 MacOS

We recommend using Homebrew as a package manager here. You can also install from the source directly. Please follow the guide at https://github.com/pyenv/pyenv#installation in this case.

Make sure you have homebrew and XCode command line tools (`xcode-select --install`) installed. In a terminal run the following.

```
brew update
brew install pyenv
```

Now we need to add pyenv init to your shell to enable shims and autocompletion. Please make sure eval "*$(pyenv init -)*" is placed toward the end of the shell configuration file since it manipulates PATH during the initialization.

The easiest way to achieve this is to execute the following in a terminal. Depending on your MacOS version you are either using a bash or Zsh shell. (You can find out with `echo $0`.)

```
echo -e 'eval "$(pyenv init -)"' >> ~/.bash_profile
```

for MacOs 10.14 and below or

```
echo -e 'eval "$(pyenv init -)"' >> ~/.zshrc
```

for newer versions.

Optionally, inspect your `~/.bash_profile` or `~/.zshrc` (i.e. open it in the text editor of your choice) if your new entry is in there.

Restart your terminal by running

```
exec $SHELL
```

Now you can use pyenv to install python 3.9.

Have a look at the available versions using:

```
pyenv install -l
```

and install one of the 3.9 versions.

```
pyenv install 3.9.x
```

`pyenv versions` gives you an overview over all installed python versions. Python 3.9.x should be one of them now.

Now we can associate this python version with your exercise directory. Navigate into the exercise root or open a shell at this location.

Next run

```
pyenv local 3.9.x
```

The version number should match the one you installed before.

3.1.3 Windows

There is no official pyenv for Windows unfortunately. If you still want to use this workflow you can use Pyenv-win which seems to be pretty stable by now. You'll find a detailed installation guide in the github readme.

To test your setup you can run `pyenv version` as described in their documentation. Should you get an output reading something like *"The environment path is not set"* this is fine as long as you get the version number displayed, too. If there is a *"Command not found"* error you should inspect your `PATH` variable in the GUI (see 5.1 on how to this).

Next you can use pyenv to install python 3.9.

Have a look at the available versions using:

```
pyenv install -l
```

and install one of the 3.9 versions.

```
pyenv install 3.9.x
```

An install wizard may pop up for some non-silent installs. You'll need to click through the wizard during installation. There's no need to change any options in it.

Now we can associate this python version with your exercise directory. Navigate into the "cp_2024_exercises" folder in the current cmd window or open a cmd window at this location and run

```
pyenv local 3.9.x
```

The version number should match the one you installed before.

3.1.4 Further information

The version set in the `pyenv local` command will be used whenever python is called from within this folder. The recommended workflow is to always start your work (i.e. jupyter lab) inside this folder.

3.2 Using conda

Install Conda (anaconda or miniconda both work) if you haven't yet.
https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html gives you all the information you need.

If you need a video for that you'll find the basic variants including some info on how to manage environments here:

- Linux: https://www.youtube.com/watch?v=eVMvbtdUnlg
- MacOs: https://www.youtube.com/watch?v=YJC6ldI3hWk
- Windows: https://www.youtube.com/watch?v=dgjEUcccRwM

Additional information for Windows:
Generally, it is assumed you install conda for your user only (not needing admin rights) and it is added to your `PATH` variable either automatically during installation (be sure to tick the box in the graphical installer or add the AddToPath=1 flag when installing from command line) or manually afterwards. We do not recommend to make conda's python your default system python, so uncheck that option in the installer unless you are certain you want this.

After the successful installation we can create an environment for the exercises.

1. Open a shell / terminal / command window. On Windows you can use the Anaconda Prompt (we will generally refer to it as the command window here).

2. Create a new environment using python3.9:

   ```
   conda create -n p39_cp -c conda-forge python=3.9 -y
   ```

   calling it `p39_cp` here (you can call it anyway you like).

3. Activate the environment. In your terminal / shell / cmd window navigate (cd) to the *cp_2024_exercises* directory you extracted earlier.

   Now execute:

   ```
   conda activate p39_cp
   ```

   You can test if you are using the correct python version by opening an interactive python session by calling `python`. Exit again (exit()).

4. Install Poetry

4.1 Pyenv

Poetry is the package manager for python we are going to use and we will use it to install all other dependencies. You can find more information on https://python-poetry.org/docs/.

If you have poetry already installed on your system you can skip to step 6.

Make sure you have the correct python version active (i.e. you are in the correct directory when using pyenv) during install.

4.1.1 Linux / MacOS

```
curl -sSL https://install.python-poetry.org | python3 -
```

4.1.2 Windows Powershell

```
(Invoke-WebRequest -Uri https://install.python-poetry.org -UseBasicParsing).Content | py -
```

4.2 Conda

Run the following in an activated environment.

```
conda install -c conda-forge poetry -y
```

5. Add poetry to your path.

Execute the following command to check whether `poetry` is correctly installed:

```
poetry --version
```

If you get a "Command not found" error, you probably need to add poetry to your **PATH** variable.

These are the locations poetry gets installed to according to the documentation and you would need to add to **PATH** consequently:

- `$HOME/.local/bin` on Unix.
- `%APPDATA%\Python\Scripts` on Windows.
- `$POETRY_HOME/bin` if **$POETRY_HOME** is set.

Assuming `your-path` is the path you want to add, here is an overview on how to do this.

- Linux:

  ```
  echo -e 'export PATH="${PATH}:your-path"' >> ~/.bashrc
  source ~/.bashrc
  ```

- MacOS: Again, depending on your MacOs version you might have to use `.bash_profile` (MacOS < 10.15) in place of `.zshrc` (MacOS >= 10.15).

  ```
  echo -e 'export PATH="${PATH}:your-path"' >> ~/.zshrc
  source ~/.zshrc
  ```

- Windows:
  - Open the "System (Control Panel)"
  - Click "Advanced system settings".
  - Click "Environment Variables".
  - Under "System Variables", find the `PATH` variable, select it, and click "Edit". If there is no `PATH` variable, click "New".
  - Add your directory to the beginning of the variable value followed by `;` (a semicolon). For example, if the value was `C:\Windows\System32`, change it to `your-path;C:\Windows\System32`
  - Click "OK".
  - Restart your command window.

6. Finally, install the environment

We bundled all dependencies required for the exercises into a poetry package. You find a `pyproject.toml` and a `poetry.lock` file in the root of the extracted folder of the first exercise. With `poetry` you can create a python environment which contains all the dependencies defined in these files in one go.

(We assume you still have the environment active or have the correct python version selected as explained above.)

In the directory containing the `pyproject.toml` (still the same as above) call

```
poetry install
```

This will install all we need.

7. Use the environment

7.1 Pyenv

In order to get into the environment with all the installed packages you have to run from the current directory (i.e. the one where the `pyproject.toml` is):

```
poetry shell
```

7.2 Conda

To use the installed packages you have to activate your environment. If you followed the guide from the start your environment will already be active at this point. Otherwise run:

```
conda activate p39_cp
```

8. Jupyter Lab

Jupyter Lab is a web based user interface for working with jupyter projects. It is the closest thing to a Jupyter-IDE out there and a modern way of manipulating `jupyter notebooks`. The exercise sheets are jupyter notebook files, which you can work on in `jupyter lab` or `jupyter notebook`. Jupyter Lab has some additional features, like multiple tabs with interactive shells and many plugins that simplify programming compared to working in `jupyter notebook`. Visual Studio Code also has plugins for notebook support, however there are also some limitations. Please always test your submissions in `jupyter lab` before submission.

This document will guide you through the most important things you need to know to use Jupyter Lab to solve our exercises. For more information check out the official documentation.

8.1 Starting Jupyter Lab

After successfully installing the environment you should be able to launch Jupyter Lab via

```
jupyter lab
```

This will start a webserver in your terminal and open a Jupyter Lab client in your browser. If the lab-client does not open automatically in your browser you'll find the link in the terminal log of the server.

8.2 Jupyter Lab extensions

The provided `poetry` project comes with a bunch of python packages that help when developing in python, such as code formatting refactoring and auto completion. It also contains some official jupyter lab extensions, that will be used in the exercises.

However some extensions only can be installed from within `jupyter lab`.

In the extension manager (left side puzzle-icon) activate extensions (you might need to restart jupyter lab afterwards - see 9 for how to stop jupyter lab) and make sure that the following extensions are installed:

- `jupyter-matplotlib` will be used to show generated images and plots.

Feel free to try other extensions that you deem useful like themes or a vim-mode...

You can open the `verify_installation.ipynb` notebook also found in the exercise root folder to check whether everything is setup correctly now.

9. Exit the session and resuming work

Be sure to save your work if you have autosave disabled. You can stop juypter lab from within the menu in the browser or by typing `CTRL + c` (twice) in the terminal / console / cmd.

Use `exit` or `CTRL + d` to exit the pipenv shell. To resume your work at a later time proceed with **7.1**.

9.1 Conda:

Additionally use `conda deactivate` to leave the conda environment.

Use the `conda activate p39_cp` command from above to activate it again (as explained in **7.1**). Proceed with **8.1** to resume your work.

10. Known Issues

- It might be necessary to uninstall the virtualenv package from your conda environment if it is automaitcally install. Run `pip3 uninstall virtualenv` to accomplish this.