01. [40p] Python environment

Different *Python* projects usually employ not only different modules, but even different versions of the same module. The reason is that module developers push out new versions very rapidly and project maintainers may be slow on the uptake. Ideally, updates to any module (or package) should not break tools using previous implementations. Each update should add bug fixes and extend the API with new features, but not change it! However, programs are sometime built around some anomalous behavior of an API and bug fixes actually break said program.

Click GIF to maximize.

As you work on different *Python* projects, you may need different versions of the same module. Probably even a module that you already have installed system-wide. For this, we use virtual environments [https://docs.python.org/3/library/venv.html]. These environments allow you to install specific module versions in a local directory and alters your shell's environment to prioritize using them. Switching between environments can be as easy as **source**ing another setup script.

The problem with virtual environments is that they don't mesh well with **apt**. In stead of **apt**, we will use a *Python* module manager called **pip3**. Our suggestion is to use **pip** only in virtual environments. Yes, it can also install modules system-wide, but most modules can be found as **apt** packages anyway. Generally, it is not a good idea to mix package managers!

[10p] Task A - Dependency installation

First things first, we need **python3**, the **venv** module, and **pip**. These, we can get with **apt**

```
$ sudo apt install python3 python3-venv python3-pip
```

[10p] Task B - Creating the environment

Assuming that you are in your project's root directory already, we can set up the virtual environment:

```
$ python3 -m venv .venv
```

The `-m` flag specifies to the *Python* interpreter a module. **python3** searches its known install paths for said module (in this case, **venv**) and runs it as a script. `.venv` is the script's argument and represents the name of the storage directory. Take a look at its internal structure:

```
$ tree -L 3 .venv
```

Notice that in *.venv/bin/* we have both binaries and activation scripts. These scripts, when sourced, will force the current shell to prioritize using these. The modules you install will be placed in *.venv/lib/python3.*/site-packages/.* Try to activate your environment now. Once active, you will have access to the **deactivate** command that will restore your previous environment state:

```
$ source .venv/bin/activate
$ deactivate
```

If you are still using the setup from the first lab, you may get an ugly `(.venv)` prompt that looks nothing like that in the GIF above. Add this to your *.zshrc*:

```
VIRTUAL_ENV_DISABLE_PROMPT="yes"
```

The display function depends on your selected theme. For *agnoster*, you can fiddle with the **prompt_virtualenv()** function in the *agnoster.zsh-theme* source file.

[10p] Task C - Fetching modules with pip

Same as **apt**, **pip** used to have a search function for modules. Unfortunately, they removed this feature due to a high number of queries. Now, to search for modules, you will need to use the web interface [https://pypi.org/project/pip/].

Let us install the modules needed for this laboratory. After that, let us also check the versions of all modules (some will be added implicitly). Can you also find their installation path in the *.venv/* directory?

```
$ sudo apt install libffi-dev libnacl-dev python3-dev

$ pip3 install 'py-cord[voice]' PyNaCl ipython
$ pip3 list
```

So, what are these, exactly?

- **py-cord[voice]**: a *Python* module that interacts with discord [https://discord.com/]. The previous discord.py [https://discordpy.readthedocs.io/en/stable/] project has been discontinued and pycord [https://docs.pycord.dev/en/master/index.html] remains the best alternative. We'll be using this in the following exercise to write a discord bot.
- **PyNaCl**: wrapper of the NaCl [https://nacl.cr.yp.to/] library. This library offers a high-level interface for networking and cryptographic operations. Note that the library must be installed using **apt** and that which we install with **pip** is only a wrapper.
- **ipython**: a more interactive *Python*. See the next task for some details, but there's really not much to it.

This list of dependencies can be exported (together with the exact version) in a way that allows another user to install the exact same modules in their own virtual environments. The file holding this information in named by convention *requirements.txt*:

```
$ pip3 freeze > requirements.txt
$ pip3 install -r requirements.txt
```

[10p] Task D - Testing that it works, with ipython

In the previous lab, you used the **python3** interpreter in interactive mode. Now, we upgrade to **ipython**. This new interpreter offers an enhanced user experience by means of color coding, tab completion, multi-line editing, etc. For scripting purposes, **python3** should remain your interpreter of choice. But for prototyping, we suggest using this. For now, let's see if the **discord** module in the **py-cord[voice]** package is available to import.

```
$ ipython
Python 3.9.7 (default, Oct 10 2021, 15:13:22)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.29.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import discord

In [2]: discord?

In [3]: discord??

In [4]: help(discord)
```

Note how in stead of **help**, in **ipython** we can ? or ?? to something to access a brief / more complete documentation.

If you can't import the **discord** module, try to source the activation script again after installing the packages with **pip**. Some versions of **venv** / **pip** might act up.