ACIT 3855 – Service Oriented Architectures – Winter 2024

Lesson 7 Pre-Reading

Reading

In Lesson 8, we will be containerizing our services and deploying them using Docker and Docker Compose.

For a review of Docker (assuming you have or are currently taking a Docker course), please read the following:

- Docker Overview: https://docs.docker.com/get-started/overview/
- Docker Compose Overview: https://docs.docker.com/compose/

Because each of our Python Connexion applications will be independently deployable as a Docker container, this brings up the importance of ensuring we know which Python libraries are used by each. So Virtual Environments become important as well as having a requirements.txt file containing all the required dependences and versions.

PYTHON VIRTUAL ENVIRONMENTS - Tutorial

When developing with Python, it is recommended to keep you development environments isolated. This helps protect your main Python installation from clutter and potential corruption that might be introduced by installed packages. You will need different packages for different projects, and virtual environments enable you to keep these dependencies separate per project. This also make your projects' environments easily replicable.

Pipenv is one among a host of tools that let you create virtual environments for your projects. Others include `venv` and `virtualenv`, but `pipenv` may be the easiest to use. **Pipenv** maintains your project's dependencies in a file called **Pipfile** and can lock your dependencies to ensure deterministic builds. To install `pipenv`, you will need both ***Python*** and ***Pip*** installed on your system.

1. Installing PIPENV

```
pip install –user pipenv
```

2. Using PIPENV

Pipenv manages dependencies on a per project basis. Typically you would want to create a directory for you project and run `pipenv` command at the root of your project. When you run `pipenv` for the first time, it will create a file called **Pipfile** at the root of your project which will contain your project's dependencies.

```
mkdir my_project
cd my_project
```

```
pipenv install requests
```

Create a module inside your project:

```
touch my_app.py
```

Run the module using pipenv:

```
pipenv run python my_app.py
```

Running you application this way ensures that your dependencies are available to your code. Alternative, you can run: `pipenv shell` which will spawn a new shell that loads your dependencies and makes them available to your commands. Then you can run your modules as you would in a normal python shell:

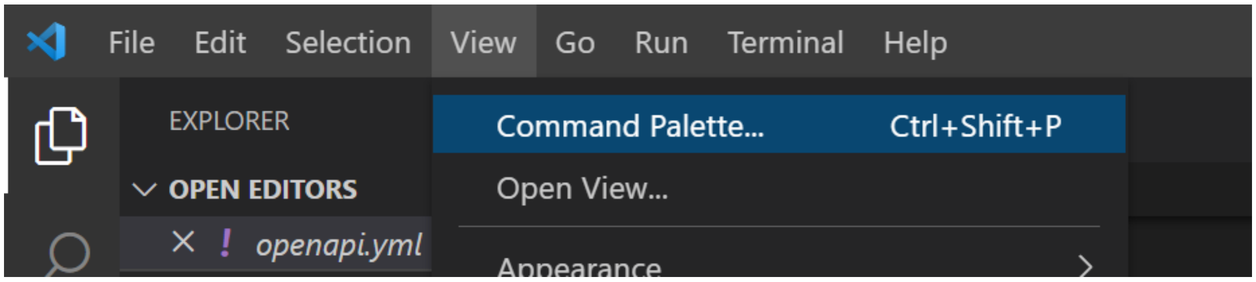
```
python my_app.py

exit
```

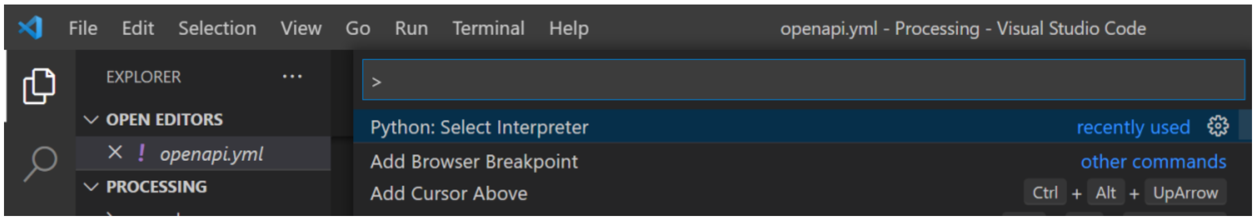
The exit command lets you terminate the shell, effectively deactivating the virtual environment. You can still run your programs by prefixing `pipenv` run to the python command.

Visual Studio Code configuration

In order to configure a virtual environment created with `pipenv` for your project in VSCode, first ensure you have the "*Python Extension Pack*" installed.

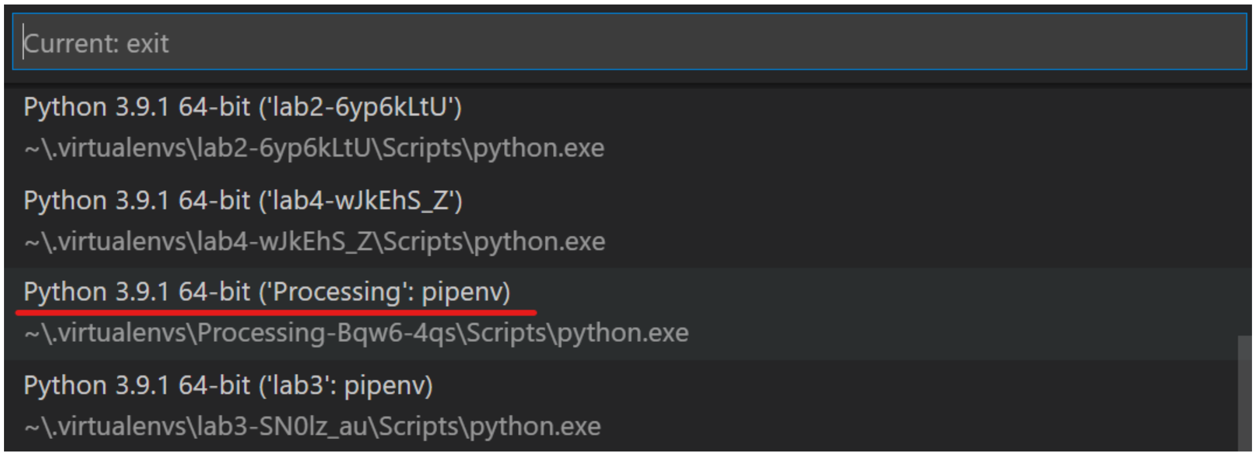
To activate your project's environment, open the project folder in VSCode, go to *View > Command Palette* and select *Python: Select Interpreter*.





Python: Select Interpreter — recently used
Add Browser Breakpoint — other commands
Add Cursor Above — Ctrl + Alt + UpArrow

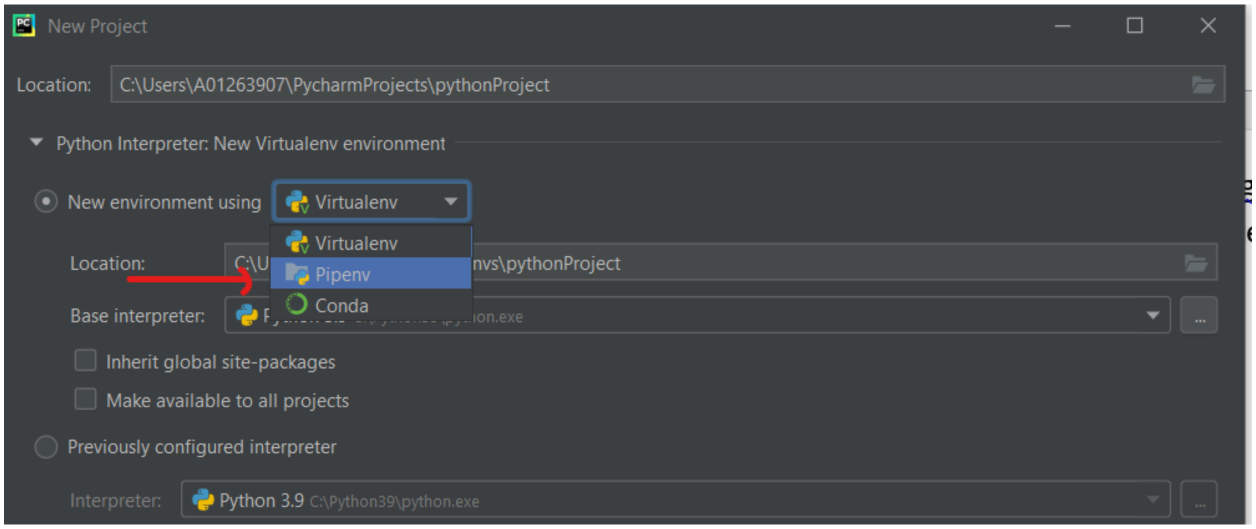
You will be presented with a list of paths to environments you have created. Select the one you created for your project. Afterwards, when you run your modules, VSCode will use your virtual environment.



```
Current: exit
Python 3.9.1 64-bit ('lab2-6yp6kLtU')
~\.virtualenvs\lab2-6yp6kLtU\Scripts\python.exe
Python 3.9.1 64-bit ('lab4-wJkEhS_Z')
~\.virtualenvs\lab4-wJkEhS_Z\Scripts\python.exe
Python 3.9.1 64-bit ('Processing': pipenv)
~\.virtualenvs\Processing-Bqw6-4qs\Scripts\python.exe
Python 3.9.1 64-bit ('lab3': pipenv)
~\.virtualenvs\lab3-SN0lz_au\Scripts\python.exe
```

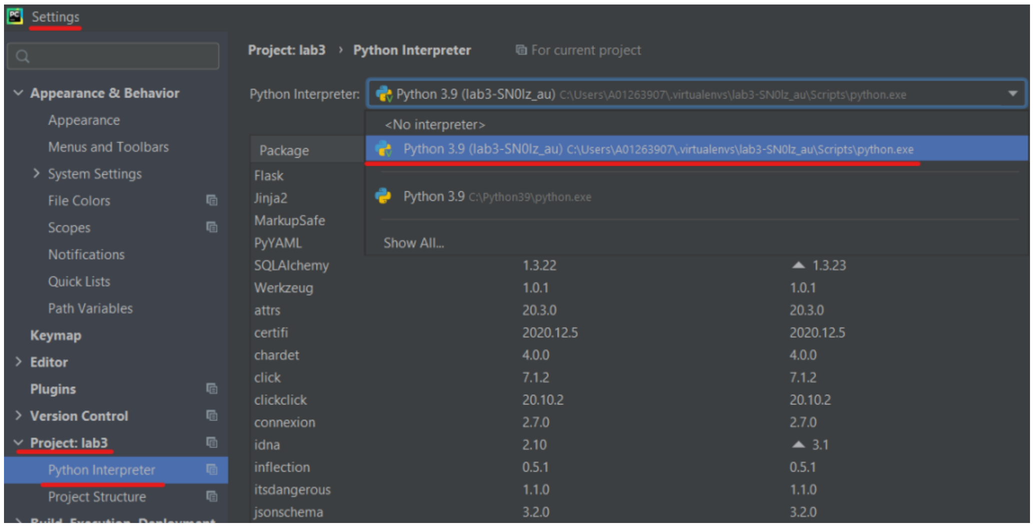
Pycharm

You can create a virtual environment with `pipenv` when creating a new project in Pycharm.


New Project
Python Interpreter: New Virtualenv environment
New environment using
Virtualenv
Virtualenv
Pipenv
Conda
Inherit global site-packages
Make available to all projects
Previously configured interpreter

Location: C:\Users\A01263907\PycharmProjects\pythonProject
Location: C:\U…nvs\pythonProject
Base interpreter: P…on.exe
Interpreter: Python 3.9 C:\Python39\python.exe

For an existing project, you will need to tell Pycharm where to find you virtual environments. **Pipenv** stores environments in a folder called *.virtualenvs* in your home directory.



Project: lab3 › Python Interpreter

Python Interpreter: Python 3.9 (lab3-SN0lz_au) C:\Users\A01263907\.virtualenvs\lab3-SN0lz_au\Scripts\python.exe

Python 3.9 (lab3-SN0lz_au) C:\Users\A01263907\.virtualenvs\lab3-SN0lz_au\Scripts\python.exe

Python 3.9 C:\Python39\python.exe

Package		
Flask		
Jinja2		
MarkupSafe		
PyYAML		
SQLAlchemy	1.3.22	▲ 1.3.23
Werkzeug	1.0.1	1.0.1
attrs	20.3.0	20.3.0
certifi	2020.12.5	2020.12.5
chardet	4.0.0	4.0.0
click	7.1.2	7.1.2
clickclick	20.10.2	20.10.2
connexion	2.7.0	2.7.0
idna	2.10	▲ 3.1
inflection	0.5.1	0.5.1
itsdangerous	1.1.0	1.1.0
jsonschema	3.2.0	3.2.0

PYTHON requirements.txt

Here is an example of a requirements.txt file in Python:

```
connexion==2.7.0
swagger-ui-bundle==0.0.8
SQLAlchemy==1.3.22
mysql-connector-python==8.0.23
pymysql==1.0.2
pykafka==2.4.0
```

Notice we have a line for each package, then a version number. This is important because as you start developing your python applications, you will develop the application with specific versions of the packages in mind. Later on, the package maintainer might make changes which would break your application. It is too much work to keep track of every downstream package change. Especially if it is a larger project. So you want to keep track of what version of each package you are using to prevent unexpected changes. (Source: https://www.idkrtm.com/what-is-the-python-requirements-txt)

The command **pip freeze** generates output in the format of a requirements.txt showing all the Python packages installed in your environment with their version numbers. You can copy and paste this output into a requirements.txt file, and you now have all of these packages documented.

The command **pip install** allows you to install individual Python packages in your environment. In addition, **pip install -r requirements.txt** will install all the Python packages of the given version numbers defined in the requirements.txt file. The above applies to the entire Python installation on our computer.

If we are using a **virtual environments**, we can also use pip freeze to get the content for a requirements.txt file specifically for that environment. Then we can then install those Python packages with pip install –r requirements.txt in another Python environment (virtual or not). This will be especially useful when building the Docker images for our services.