
Python Scientific Development Environments

Release 0.1

Jonas Lindemann

Oct 13, 2022

CONTENTS:

| | | |
|----------|---|-----------|
| 1 | Python and Visual Studio Code | 1 |
| 1.1 | Installing Visual Studio Code | 1 |
| 1.2 | Installing required extensions for Python | 1 |
| 1.3 | Creating, saving and opening Python source files | 3 |
| 1.4 | Activating the Python extension | 3 |
| 1.5 | Selecting the correct Python interpreter | 4 |
| 1.6 | Executing a Python source file (.py) | 4 |
| 1.7 | Using Visual Studio Code with Anaconda | 4 |
| 2 | Python and Spyder | 7 |
| 3 | Using and creating Jupyter Notebooks | 9 |
| 4 | How to use Python virtual environments | 11 |
| 4.1 | Using a virtual enviroments (venv) | 11 |
| 4.2 | Managing packages in virtual environment | 12 |
| 4.3 | Using virtualenv to create environments | 12 |
| 4.4 | Creating reproducible environments | 13 |
| 5 | Using pipenv to manage environments and packages | 15 |
| 5.1 | Installing pipenv | 15 |
| 5.2 | Setting up a pipenv project | 15 |
| 5.3 | Installing packages | 17 |
| 5.4 | Running a project using pipenv | 17 |
| 5.5 | Executing a shell in the created environment | 19 |
| 5.6 | Listing package dependencies | 20 |
| 5.7 | Updating packages in a project | 20 |
| 6 | How to use Conda environments | 23 |
| 6.1 | Creating a Anaconda environment | 23 |
| 6.2 | Activating and deactivating Anaconda environments | 24 |
| 6.3 | Removing an environment | 24 |
| 6.4 | Cloning an existing environment | 25 |
| 6.5 | Exporting an Anaconda environment | 25 |
| 6.6 | Conda and pip | 25 |
| 6.7 | Using conda environments in Jupyter Notebooks | 26 |
| 7 | Indices and tables | 27 |

PYTHON AND VISUAL STUDIO CODE

Visual Studio Code is Microsoft's effort to create a platform independent code editor and development environment. The Editor is available for macOS, Windows and Linux. One of the key features of the editor is that it is very flexible and extensible. It supports almost all programming languages and integrates with source versioning systems such as git.

For Python development Microsoft provides several plugins that make Visual Studio Code to an excellent environment for Scientific Programming in Python as well as using Python Notebooks. This chapter describes how to setup and configure Visual Studio as a Python development environment.

1.1 Installing Visual Studio Code

Visual Studio code is available from Microsoft at the following location:

<https://code.visualstudio.com/Download>

At this location Visual Studio Code can be downloaded for all platforms.

The installers are quite self explanatory, so we won't document the steps for installing this on all platforms.

1.2 Installing required extensions for Python

To be able to use Python efficiently in Visual Studio Code a couple of extensions are required. The easiest way to install this is to install the Microsoft extension, **Python**, which automatically installs the other Microsoft extensions **Pylance** and **Jupyter**.

These extensions are installed directly from within the application by selecting the extension tab on the left side in the application as shown in the following image:

To find an extension type the name of the extension in the text box in the top of the tab. More information on the plugin can be found by clicking on the plugin in the list. Installing the plugin is done by clicking on **Install** either directly in the extension list or on the extension information page.

The extension information page for the **Python** extension with the install button is shown in the figure below:

Click the **install** button to install the required extensions.

To see the installed extensions. Clear the search box in the extension tab and the installed extensions will be displayed under **INSTALLED**.

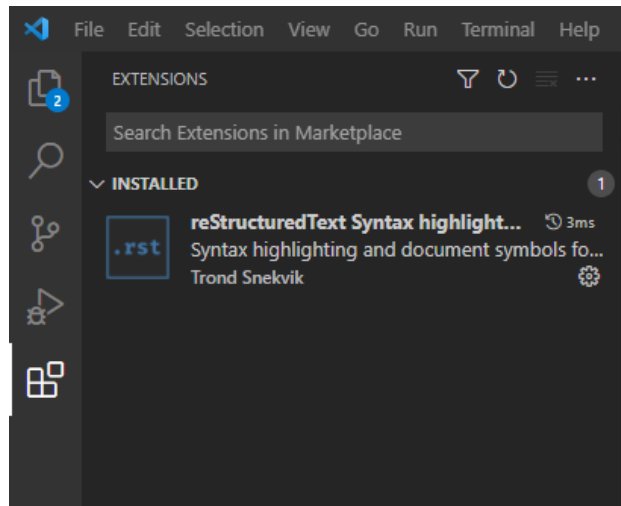


Fig. 1: Extension tab in Visual Studio Code.

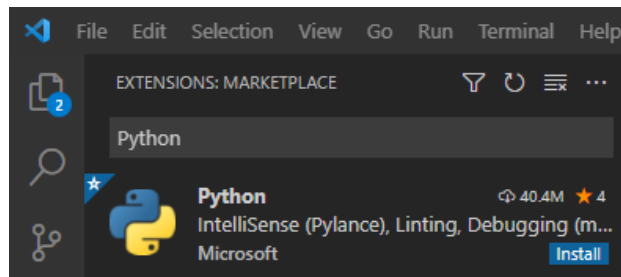


Fig. 2: Installing a Python extension.

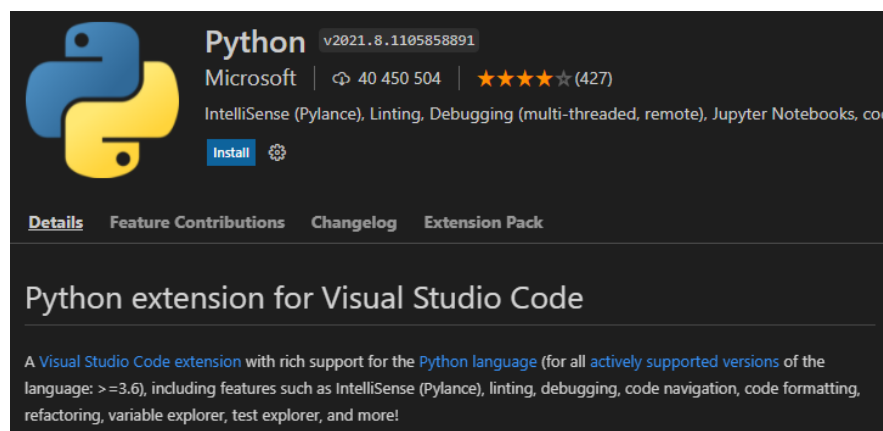


Fig. 3: Extension information page.

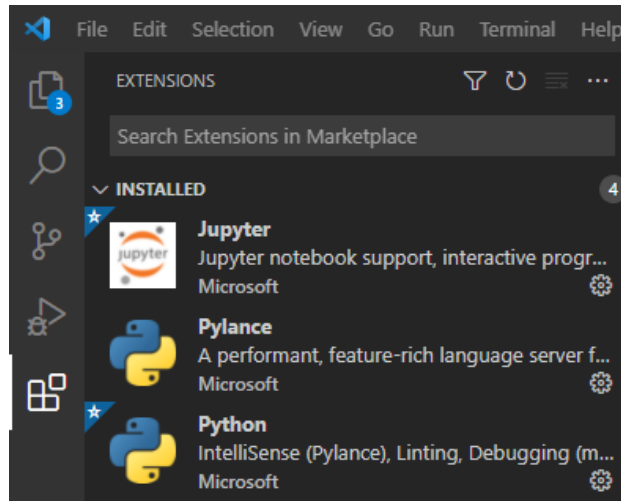


Fig. 4: Installed Python extensions.

1.3 Creating, saving and opening Python source files

Creating, saving and opening Python source files are done using the normal **New**, **Save** and **Open** item in the file menu.

When creating a new file in Visual Studio Code it is possible to tell the editor what language you want to use in by clicking on the link **Select a language**. Another alternative is to directly save the file with the extension `.py`. Visual Studio Code then automatically detects the language and enables Python highlighting and the Python specific extensions.

1.4 Activating the Python extension

The Python extension is activated every time a Python source file is opened in the editor. Status of the extension is displayed in the lower left of the window.



Fig. 5: Extension status information.

By default the extension status displays shows the default Python interpreter installed on the system. In this case Python 3.7.8 64-bit is the current default interpreter.

1.5 Selecting the correct Python interpreter

To run Python in Visual Studio code with the extensions installed, the correct Python interpreter must be selected. To change the default Python interpreter click in the extension status information display in the lower left of the window.

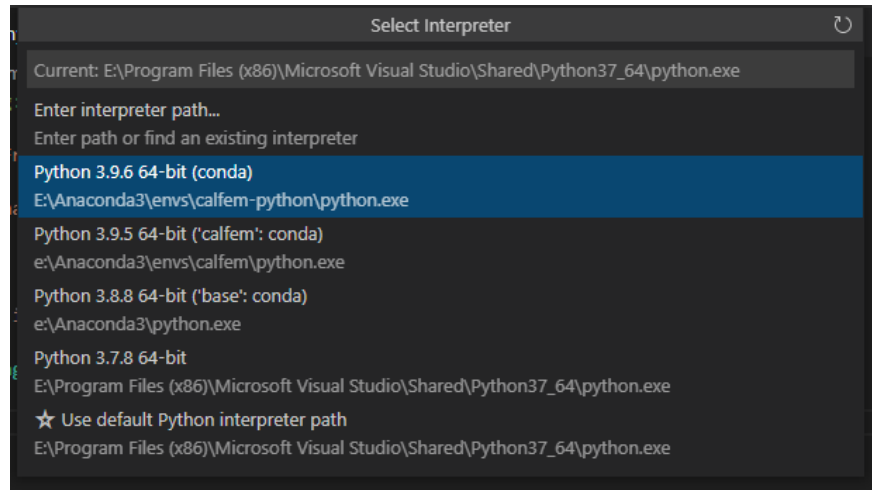


Fig. 6: Selecting the default Python interpreter.

1.6 Executing a Python source file (.py)

When a Python source file is open it is possible to execute the file using the currently selected Python interpreter. In the top right there is a button to execute the current source code.

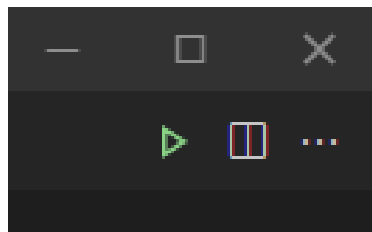


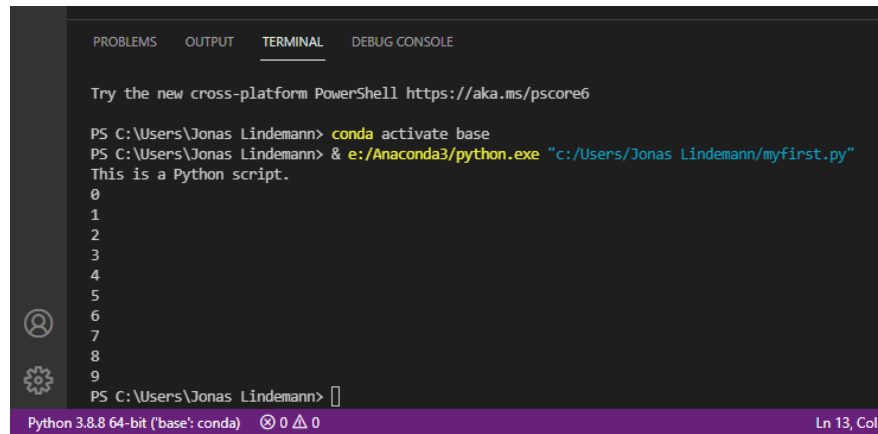
Fig. 7: Executing Python.

Output from the running code is shown in the **Terminal** view in the lower part of the editor window.

1.7 Using Visual Studio Code with Anaconda

To be able to use the Anaconda distribution fully in Visual Studio Code you should start the editor from a **Anaconda Command Prompt** which is found in the start menu in Windows or Mac. On Linux it works without any special changes.

In the example below Visual Studio Code is started from the **test** directory. This will instruct the editor to use that directory as the working directory.



The screenshot shows a Visual Studio Code terminal window with the 'TERMINAL' tab selected. The terminal displays the following text:

```
Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\Users\Jonas Lindemann> conda activate base
PS C:\Users\Jonas Lindemann> & e:/Anaconda3/python.exe "c:/Users/Jonas Lindemann/myfirst.py"
This is a Python script.
0
1
2
3
4
5
6
7
8
9
PS C:\Users\Jonas Lindemann>
```

The status bar at the bottom indicates 'Python 3.8.8 64-bit (base: conda)' and 'Ln 13, Col 1'.

Fig. 8: Output from running code.

```
(base) E:\...\>cd test
```

```
(base) E:\...\>code .
```


PYTHON AND SPYDER

USING AND CREATING JUPYTER NOTEBOOKS

HOW TO USE PYTHON VIRTUAL ENVIRONMENTS

Python can use a lot of third-party extension modules. Complex projects can depend on a lot of these modules. Installing extension modules in the base installation can work for smaller projects and scripts. However different projects can have dependencies on different incompatible libraries, making it hard to maintain an single Python installation for multiple projects. To solve this, Python contains a mechanism for creating isolated python environments that each can have it's own set of extension modules installed. This mechanics is called virtual environments and is accessible through the venv module.

A virtual environment contains is a copy of the Python base installation with only the absolute minimum of installed packages.

4.1 Using a virtual enviromnents (venv)

Creating a virtual environment is done by the following command:

```
python -m venv myenv
```

This will create a separate directory, myenv, where copy of the Python base install will be installed. To use this new environment it has to be activated. This is done by calling/sourcing a special script that is available inside the environment, activate. On a windows installation a virtual environment is activated by calling:

```
myenv\Scripts\activate.bat
```

On a Linux/Mac environment the environment is activated by sourcing:

```
source myenv/bin/activate
```

When an enviromnent is activte the prompt is changed to display the currently active environment:

```
(myenv) (base) E:\Users\Jonas\Development\python_dev_doc\examples>
```

If the environment is no longer used it can be deactivated using the following commands:

```
myenv\Scripts\deactivate.bat
```

On a Linux/Mac environment the environment is activated by sourcing:

```
source myenv/bin/deactivate
```

4.2 Managing packages in virtual environment

Packages or third-party packages in a virtual environment are installed and managed using the **pip** command just like in a base Python installation. The difference is that a newly created environment only holds just the required packages for the environment to work.

Available packages in an environment can be listed using the **pip list** command:

```
$ pip list
```

Running this command in a base installation will produce a long list of packages:

```
$ pip list
Package                                Version
-----
alabaster                             0.7.12
anaconda-client                       1.7.2
anaconda-navigator                   2.0.3
anaconda-project                     0.9.1
anyio                                 2.2.0
appdirs                              1.4.4
argh                                  0.26.2
argon2-cffi                          20.1.0
asn1crypto                           1.4.0
astroid                              2.5
astropy                              4.2.1
async-generator                       1.10
atomicwrites                         1.4.0
attrs                                20.3.0
autopep8                             1.5.6
Babel                                 2.9.0
```

Doing the same thing in a newly created environment produces the following output.

```
pip list
Package    Version
-----
pip        20.2.3
setuptools 49.2.1
```

4.3 Using virtualenv to create environments

virtualenv is a tool that provides additional options and also makes it easier to create virtual environments. This should preferably be installed in the base Python installation. The tool is installed using the following command:

```
pip install virtualenv
```


4.4 Creating reproducible environments

In many scientific workflows it is important to create reproducible workflows. This also extends to scientific software. Virtual environments are excellent to create a reproducible set of dependencies for a scientific workflow.

When a environment has been created and packages have been installed, it is possible to create a list of required packages that can be used to recreate the exact environment. Using the **pip freeze** command it is possible to create a list of requirements that can be used as input in a **pip install** command.

In the following example we will create a *requirements.txt* file containing the needed modules in the myenv environment. Listing the installed packages produces the following output:

```
$ pip list
Package            Version
-----
calfem-python      3.5.10
cyclor              0.10.0
gms                 4.8.4
kiwisolver          1.3.1
matplotlib          3.4.3
numpy               1.21.2
Pillow              8.3.1
pip                 21.2.4
PyOpenGL            3.1.5
pyparsing            2.4.7
PyQt5               5.15.4
PyQt5-Qt5           5.15.2
PyQt5-sip            12.9.0
PyQtWebEngine        5.15.4
PyQtWebEngine-Qt5   5.15.2
python-dateutil      2.8.2
PyVTK                0.5.18
scipy                1.7.1
setuptools           49.2.1
six                  1.16.0
visvis               1.13.0
wheel                0.37.0
```

Using the **pip freeze** command we can create a list of requirements.

```
$ pip freeze > requirements.txt
$ cat requirements.txt
calfem-python==3.5.10
cyclor==0.10.0
gms==4.8.4
kiwisolver==1.3.1
matplotlib==3.4.3
numpy==1.21.2
Pillow==8.3.1
PyOpenGL==3.1.5
pyparsing==2.4.7
PyQt5==5.15.4
PyQt5-Qt5==5.15.2
PyQt5-sip==12.9.0
```

(continues on next page)

(continued from previous page)

```

PyQtWebEngine==5.15.4
PyQtWebEngine-Qt5==5.15.2
python-dateutil==2.8.2
PyVTK==0.5.18
scipy==1.7.1
six==1.16.0
visvis==1.13.0

```

On Windows use **type requirements.txt**.

Using this file it is now possible to recreate a new environment using the following commands:

```

$ python -m venv newenv
$ newenv/Scripts/activate.bat
(newenv) $ pip install -r myenv\requirements.txt
Collecting caldem-python==3.5.10
Using cached caldem_python-3.5.10-py3-none-any.whl (70 kB)
Collecting cycder==0.10.0
...
Successfully installed Pillow-8.3.1 PyOpenGL-3.1.5 PyQt5-5.15.4 PyQt5-Qt5-5.15.2 PyQt5-
sip-12.9.0 PyQtWebEngine-5.15.4 PyQtWebEngine-Qt5-5.15.2 PyVTK-0.5.18 caldem-python-3.
5.10 cycder-0.10.0 gms-4.8.4 kiwisolver-1.3.1 matplotlib-3.4.3 numpy-1.21.2 pyparsing-
2.4.7 python-dateutil-2.8.2 scipy-1.7.1 six-1.16.0 visvis-1.13.0

```

If we activate and list the packages we should get the same packages in **newenv** as in **myenv**.

```

$ pip list
Package            Version
-----
caldem-python      3.5.10
cycder             0.10.0
gms               4.8.4
kiwisolver         1.3.1
matplotlib         3.4.3
numpy             1.21.2
Pillow            8.3.1
pip               20.2.3
PyOpenGL           3.1.5
pyparsing         2.4.7
PyQt5             5.15.4
PyQt5-Qt5         5.15.2
PyQt5-sip         12.9.0
PyQtWebEngine     5.15.4
PyQtWebEngine-Qt5 5.15.2
python-dateutil   2.8.2
PyVTK             0.5.18
scipy             1.7.1
setuptools        49.2.1
six              1.16.0
visvis           1.13.0

```

We now have an exact copy of the myenv environment. This can be useful to recreate the requirements for a scientific software package on a different system or resource.

USING PIPENV TO MANAGE ENVIRONMENTS AND PACKAGES

virtualenv, **venv** and **pip** are essential tools for creating a reproducible environment for Python applications. However, they are separate tools which require many manual steps to setup a working environment for an application.

Pipenv is a tool that combines package and environment creation in a single tool and automatically handles requirements.

Advantages with *Pipenv* are:

- Single tool to manage packages and virtual environments
- Better handling of requirements
- Everything is hashed to ensure security.
- Make sure the latest packages are used.

5.1 Installing pipenv

Pipenv can be installed using the following commands:

```
$ pip install pipenv
```

It can also be installed in your home directory using:

```
$ pip install --user pipenv
```

5.2 Setting up a pipenv project

Pipenv is best used in a project setting. That is all project related source code and files are located in a single directory. This directory will be the base for the **Pipenv** environment and package data. An example of how to setup a project with **Pipenv** is shown below:

```
$ mkdir myproj
$ cd myproj
$ pipenv install

Creating a virtualenv for this project...
Pipfile: E:\Users\Jonas\Development\python_dev_doc\examples\myproj\Pipfile
Using E:/Program Files (x86)/Microsoft Visual Studio/Shared/Python37_64/python.exe (3.7.
→8) to create virtualenv...
```

(continues on next page)

(continued from previous page)

```
[ =] Creating virtual environment...created virtual environment CPython3.7.8.final.0-
↳64 in 4414ms
creator CPython3Windows(dest=C:\Users\Jonas Lindemann\.virtualenvs\myproj-jNDwD6z3,
↳clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy,
↳ app_data_dir=C:\Users\Jonas Lindemann\AppData\Local\pypa\virtualenv)
    added seed packages: pip==21.1.3, setuptools==57.4.0, wheel==0.36.2
activators BashActivator,BatchActivator,FishActivator,PowerShellActivator,
↳PythonActivator,XonshActivator

Successfully created virtual environment!
Virtualenv location: C:\Users\Jonas Lindemann\.virtualenvs\myproj-jNDwD6z3
Creating a Pipfile for this project...
Pipfile.lock not found, creating...
Locking [dev-packages] dependencies...
Locking [packages] dependencies...
Updated Pipfile.lock (a65489)!
Installing dependencies from Pipfile.lock (a65489)...
===== 0/0 - 00:00:00
To activate this project\'s virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
```

Please note that on Windows it used the default Python installation on your system and not the Python interpreter in the **PATH**. If you want to build the environment using a specific Python interpreter use the following command instead:

```
$ pipenv --python [path to python interpreter] install
```

This will create 2 files in the project directory. A *Pipfile* and a *Pipfile.lock*. The *Pipfile* contains the dependencies for the project and the *Pipfile.lock* contains the latest tested combination of packages.

A typical *Pipfile* is shown below:

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
calfem-python = "*"

[dev-packages]

[requires]
python_version = "3.8"
```

5.3 Installing packages

Packages can be installed in the project using the **pipenv install** command. In the following command we install the `calfem-python` package in the current `pipenv` project.

```
$ cd myproj
$ pipenv install calfem-python

Installing calfem-python...
Adding calfem-python to Pipfile's [packages]...
Installation Succeeded
Pipfile.lock (db4242) out of date, updating to (656452)...
Locking [dev-packages] dependencies...
Locking [packages] dependencies...
    Building requirements...
Resolving dependencies...
Success!
Updated Pipfile.lock (656452)!
Installing dependencies from Pipfile.lock (656452)...
===== 0/0 - 00:00:00
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
This will add calfem-python to the
```

5.4 Running a project using pipenv

We now have a project that can use the `calfem-python` module with all its dependencies. We now create a main python file, `fea_analysis.py` in our project directory with the following contents:

```
import numpy as np
import calfem.core as cfc

# ----- Topology -----

Edof = np.array([
    [1, 2, 3, 4, 5, 6],
    [4, 5, 6, 7, 8, 9],
    [7, 8, 9, 10, 11, 12]
])

# ----- Stiffness matrix K and load vector f -----

K = np.mat(np.zeros((12,12)))
f = np.mat(np.zeros((12,1)))
f[4] = -10000.

# ----- Element stiffness matrices -----

E = 2.1e11
A = 45.3e-4
I = 2510e-8
```

(continues on next page)

(continued from previous page)

```

ep = np.array([E,A,I])
ex = np.array([0.,3.])
ey = np.array([0.,0.])

Ke = cfc.beam2e(ex,ey,ep)

print(Ke)

# ----- Assemble Ke into K -----

K = cfc.assem(Edof,K,Ke);

# ----- Solve the system of equations and compute support forces -

bc = np.array([1,2,11])
(a,r) = cfc.solveq(K,f,bc);

# ----- Section forces -----

Ed=cfc.extractEldisp(Edof,a);

es1, ed1, ec1 = cfc.beam2s(ex, ey, ep, Ed[0,:], nep=10)
es2, ed2, ec2 = cfc.beam2s(ex, ey, ep, Ed[1,:], nep=10)
es3, ed3, ec3 = cfc.beam2s(ex, ey, ep, Ed[2,:], nep=10)

# ----- Results -----

print("a=")
print(a)
print("r=")
print(r)
print("es1=")
print(es1)
print("es2=")
print(es2)
print("es3=")
print(es3)

print("ed1=")
print(ed1)
print("ed2=")
print(ed2)
print("ed3=")
print(ed3)

```

It is possible to run the project by issuing the following command:

```

$ pipenv run fea_analysis.py
[[ 3.17100000e+08  0.00000000e+00  0.00000000e+00 -3.17100000e+08
  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  2.34266667e+06  3.51400000e+06  0.00000000e+00
 -2.34266667e+06  3.51400000e+06]

```

(continues on next page)

(continued from previous page)

```

...

[ 0.          -0.01992032]
[ 0.          -0.01823785]]
ed3=
[[ 0.00000000e+00 -1.99203187e-02]
 [ 0.00000000e+00 -1.82378462e-02]
 [ 0.00000000e+00 -1.63679985e-02]
 [ 0.00000000e+00 -1.43341976e-02]
 [ 0.00000000e+00 -1.21598653e-02]
 [ 0.00000000e+00 -9.86842362e-03]
 [ 0.00000000e+00 -7.48329434e-03]
 [ 0.00000000e+00 -5.02789938e-03]
 [ 0.00000000e+00 -2.52566063e-03]
 [ 0.00000000e+00  1.73472348e-18]
 [ 0.00000000e+00  2.52566063e-03]]

```

5.5 Executing a shell in the created environment

It is also possible to create a shell with the created project environment. From this shell it is possible to examine the installed packages using the **pipenv graph**:

```

$ pipenv shell
Launching subshell in virtual environment...
(myproj-jNDwD6z3) (base) $ pip list
Package            Version
-----
calfem-python      3.5.10
cyclor             0.10.0
gmsh               4.8.4
kiwisolver         1.3.1
matplotlib         3.4.3
numpy              1.21.2
Pillow             8.3.1
pip                21.1.1
PyOpenGL           3.1.5
pyparsing          2.4.7
PyQt5              5.15.4
PyQt5-Qt5          5.15.2
PyQt5-sip          12.9.0
PyQtWebEngine      5.15.4
PyQtWebEngine-Qt5  5.15.2
python-dateutil    2.8.2
PyVTK              0.5.18
scipy              1.7.1
setuptools         56.0.0
six                1.16.0
visvis             1.13.0
wheel              0.36.2

```

When you don't want to use the environment anymore just type **exit** and you are back in your previous environment.

5.6 Listing package dependencies

Using the **pipenv graph** it is also possible to list the dependencies of a project:

```
$ pipenv graph
calfem-python==3.5.10
- gmsh [required: Any, installed: 4.8.4]
- matplotlib [required: Any, installed: 3.4.3]
  - cycler [required: >=0.10, installed: 0.10.0]
  - six [required: Any, installed: 1.16.0]
  - kiwisolver [required: >=1.0.1, installed: 1.3.1]
  - numpy [required: >=1.16, installed: 1.21.2]
  - pillow [required: >=6.2.0, installed: 8.3.1]
  - pyparsing [required: >=2.2.1, installed: 2.4.7]
  - python-dateutil [required: >=2.7, installed: 2.8.2]
  - six [required: >=1.5, installed: 1.16.0]
- numpy [required: Any, installed: 1.21.2]
- pyqt5 [required: Any, installed: 5.15.4]
  - PyQt5-Qt5 [required: >=5.15, installed: 5.15.2]
  - PyQt5-sip [required: >=12.8,<13, installed: 12.9.0]
- pyqtwebengine [required: Any, installed: 5.15.4]
  - PyQt5 [required: >=5.15.4, installed: 5.15.4]
  - PyQt5-Qt5 [required: >=5.15, installed: 5.15.2]
  - PyQt5-sip [required: >=12.8,<13, installed: 12.9.0]
  - PyQt5-sip [required: >=12.8,<13, installed: 12.9.0]
  - PyQtWebEngine-Qt5 [required: >=5.15, installed: 5.15.2]
- pyvtk [required: Any, installed: 0.5.18]
  - six [required: Any, installed: 1.16.0]
- scipy [required: Any, installed: 1.7.1]
  - numpy [required: >=1.16.5,<1.23.0, installed: 1.21.2]
- visvis [required: Any, installed: 1.13.0]
  - numpy [required: Any, installed: 1.21.2]
  - pyOpenGL [required: Any, installed: 3.1.5]
```

This command does not require the project environment to be activated.

5.7 Updating packages in a project

Installed packages in a project can be updated using the **pipenv update** command in your project directory.

```
$ pipenv update --outdated
Locking...Building requirements...
Resolving dependencies...
Success!
Skipped Update of Package visvis: 1.13.0 installed,, 1.13.0 available.
Skipped Update of Package six: 1.16.0 installed,, 1.16.0 available.
Skipped Update of Package scipy: 1.7.1 installed,, 1.7.1 available.
Skipped Update of Package PyVTK: 0.5.18 installed,, 0.5.18 available.
```

(continues on next page)

(continued from previous page)

```
Skipped Update of Package python-dateutil: 2.8.2 installed,, 2.8.2 available.
Skipped Update of Package PyQtWebEngine: 5.15.4 installed,, 5.15.4 available.
Skipped Update of Package PyQtWebEngine-Qt5: 5.15.2 installed,, 5.15.2 available.
Skipped Update of Package PyQt5: 5.15.4 installed,, 5.15.4 available.
Skipped Update of Package PyQt5-sip: 12.9.0 installed,, 12.9.0 available.
Skipped Update of Package PyQt5-Qt5: 5.15.2 installed,, 5.15.2 available.
Skipped Update of Package pyparsing: 2.4.7 installed,, 2.4.7 available.
Skipped Update of Package PyOpenGL: 3.1.5 installed,, 3.1.5 available.
Skipped Update of Package Pillow: 8.3.1 installed,, 8.3.1 available.
Skipped Update of Package numpy: 1.21.2 installed,, 1.21.2 available.
Skipped Update of Package matplotlib: 3.4.3 installed,, 3.4.3 available.
Skipped Update of Package kiwisolver: 1.3.1 installed,, 1.3.1 available.
Skipped Update of Package gmsh: 4.8.4 installed,, 4.8.4 available.
Skipped Update of Package cyclcr: 0.10.0 installed,, 0.10.0 available.
Skipped Update of Package caldem-python: 3.5.10 installed, 3.5.10 required (Unpinned in
↳Pipfile), 3.5.10 available.
All packages are up to date!
```


HOW TO USE CONDA ENVIRONMENTS

Anaconda is one of the largest commercial Python distributions. It has become a defacto standard for packaging Scientific Software and comes with it's own packaging system, conda. If you are developing software using Anaconda the preferred way of installing software is by using the conda command line tool. If the packages are available in Anaconda's package-repos they are tested to work together. It is often also not a good idea to mix conda and pip package-repositories.

In many situations it is often required to use both pip and conda packages. In these situations it is possible within Anaconda to create conda environments. These are derived from the builtin virtual environment tools in Python, but extended and made easier to use.

6.1 Creating a Anaconda environment

An anaconda environment is created using the **conda create** command. As shown in the following example:

```
$ conda create -n cal fem-test
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: e:\anaconda3\envs\cal fem-test

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate cal fem-test
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

This creates a new empty environment ready for use.

This environment doesn't come with a Python interpreter, so that has to be installed by using:

```
$ conda install python
```

This will install the latest Python interpreter in the new environment

It is also possible to do this in a single command:

```
$ conda create -n calfem-test python
```

It is also possible to specify more packages to install by adding them on the command line.

The version of Python or packages can be specified by using an equal sign and a version number as shown below:

```
$ conda create -n calfem-test python=3.8
```

This will create an environment with the latest version of Python 3.8.

6.2 Activating and deactivating Anaconda environments

To use an environment it has to be activated. Activating an environment is done using the **conda activate** command. The command takes name of the environment as an additional parameter. In the following example we activate the previously created environment:

```
$ conda activate calfem-test
```

```
(calfem-test) $
```

This will setup all the search paths for the environment. The prompt is also modified to indicate which environment is active.

Please note that creating an empty environment does not come with a Python interpreter by default.

An environment can be deactivated by using the **conda deactivate** command:

```
(calfem-test) $ conda deactivate  
(base) $
```

No environment name is required for deactivating an environment.

6.3 Removing an environment

Removing a created environment is done using the **conda env remove** command:

```
conda env remove -n calfem-test
```

Remove all packages in environment e:\anaconda3\envs\calfem-test:

```
(base) $ conda remove -n calfem-test
```

```
Remove all packages in environment e:\anaconda3\envs\calfem-test:  
...
```

6.4 Cloning an existing environment

An exact copy of an existing environment can be created using the **—clone** option:

```
(base) $ conda create -n cal fem-dev-2 --clone cal fem-test
Source:      e:\anaconda3\envs\cal fem-dev
Destination: e:\anaconda3\envs\cal fem-dev-2
Packages: 158
Files: 13121
```

6.5 Exporting an Anaconda environment

You can list all packages and their versions using the **conda list** command:

```
(base) $ conda list cal fem-dev --explicit
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: win-64
@EXPLICIT
https://repo.anaconda.com/pkg s/main/win-64/ca-certificates-2021.7.5-haa95532_1.conda
https://repo.anaconda.com/pkg s/main/noarch/tzdata-2021a-h5d7bf9c_0.conda
https://repo.anaconda.com/pkg s/main/noarch/pyopenssl-20.0.1-pyhd3eb1b0_1.conda
...
https://repo.anaconda.com/pkg s/main/noarch/urllib3-1.26.6-pyhd3eb1b0_1.conda
https://repo.anaconda.com/pkg s/main/noarch/requests-2.26.0-pyhd3eb1b0_0.conda
https://repo.anaconda.com/pkg s/main/noarch/sphinx-4.0.2-pyhd3eb1b0_0.conda
```

This list can also be saved to a file that then can be used to recreate the environment.

```
(base) $ conda list cal fem-dev --explicit > spec-file.txt
```

Using this file it is now possible to create an environment with the exact set of packages.

```
(base) conda create --name cal fem-dev-3 --file spec-file.txt
Preparing transaction: done
Verifying transaction: done
Executing transaction:
...
```

6.6 Conda and pip

Pip can be used to install software in a conda environment. However, package information for Pip-packages are not exported using the **conda list** command. Pip-packages must be handled separately for example using the **pip freeze** command.

If possible it is always better to use the packages that are available in the conda repositories instead of using packages from the pip-package repository.

6.7 Using conda environments in Jupyter Notebooks

If you want to use a conda environment in a Jupyter Notebook it has to be added to the list of available kernels. This can be done by using the **ipykernel** package. This package is installed with the following commands:

```
(base) $ conda activate your-environment
(your-environment) $ conda install ipykernel
(base) $ conda deactivate
```

It is now possible to switch environment from within a jupyter-notebook started from the base-environment:

```
(base) $ jupyter-notebook
```

It is now possible to create a new notebook using the environment by selecting **New / Python [conda env:your-environment]**

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`