PYTHON FOR INTERMEDIATES

By SNAP, presented by Angela Xue

TOPICS

- Virtual Environments
- How to run scripts from the terminal
- More complicated functions
- Creating and importing modules
- Dealing with large data.
- Debugging

VIRTUAL ENVIRONMENTS

Different projects require different packages.

+ Different packages may be incompatible with each other,
+ Some projects require a specific version of a package,
+ Your old code may be incompatible with newer package versions,

You will have conflicts in your libraries and be unable to do your work :(

→ Virtual environments can manage packages for different projects!

Popular package managers are

+ **conda** and **pip**, accessed within the command line,
+ and **Anaconda** which has a GUI.

Bash

```
$ python -m venv /path/to/my-venv
$ source /path/to/my-venv/bin/activate
(my-venv)$ deactivate
```

Windows

```
>python -m venv /path/to/my-venv
>/path/to/my-venv/Scripts/activate.bat
(my-venv)>deactivate
```

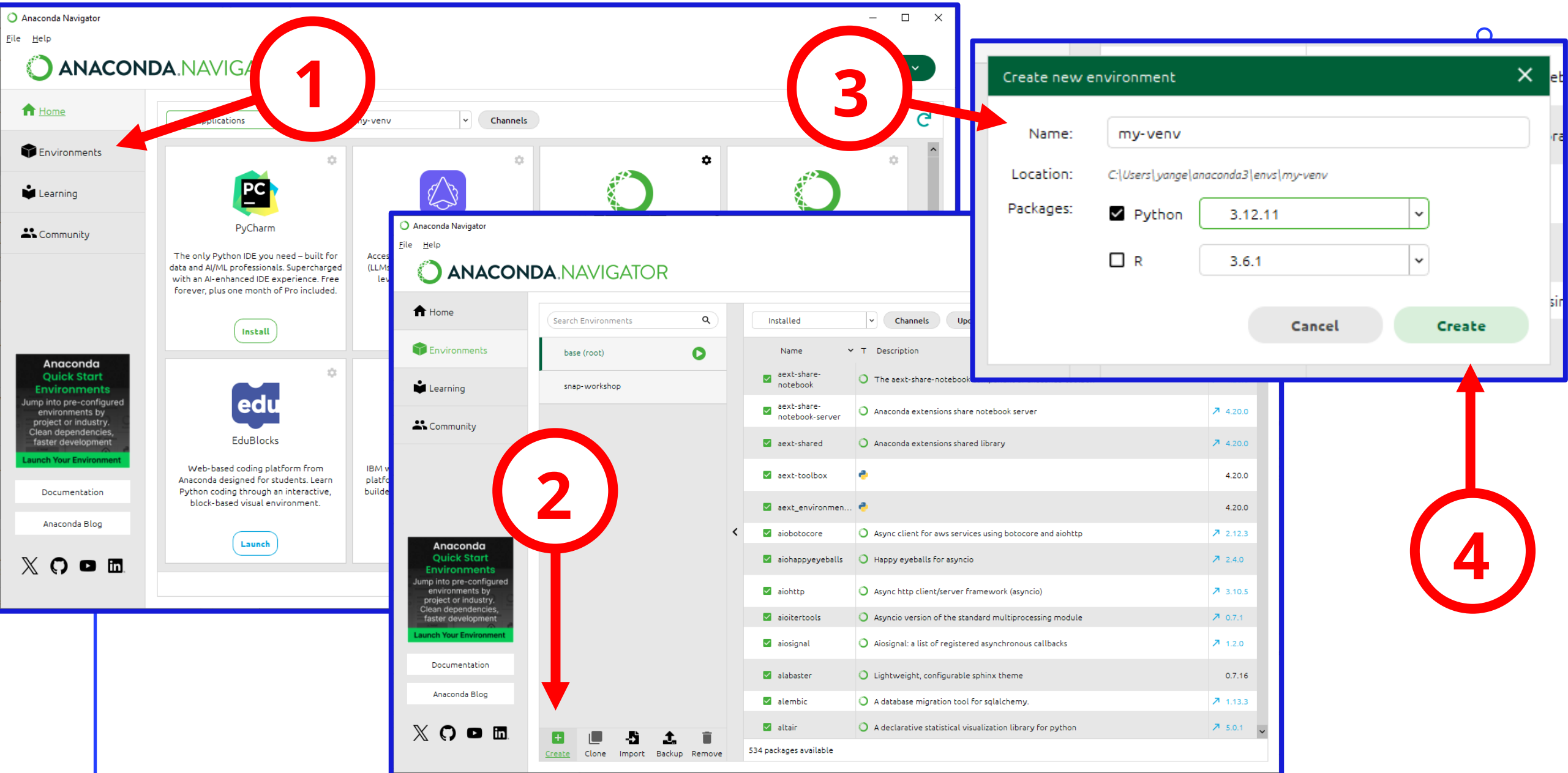
If using conda

```
conda create -n my-venv
conda activate mu-venv
(my-venv) conda deactivate
```

Link to more options:

https://docs.python.org/3/library/venv.htm

Anaconda


Anaconda Navigator
File Help
ANACONDA.NAVIGA
Home
Environments
Learning
Community
Channels
PyCharm
The only Python IDE you need – built for data and AI/ML professionals. Supercharged with an AI-enhanced IDE experience. Free forever, plus one month of Pro included.
Install
EduBlocks
Web-based coding platform from Anaconda designed for students. Learn Python coding through an interactive, block-based visual environment.
Launch
Anaconda Quick Start Environments
Jump into pre-configured environments by project or industry. Clean dependencies, faster development
Launch Your Environment
Documentation
Anaconda Blog
1
Anaconda Navigator
File Help
ANACONDA.NAVIGATOR
Home
Environments
Learning
Community
Search Environments
base (root)
snap-workshop
Installed
Channels
Name
Description
aext-share-notebook
aext-share-notebook-server
Anaconda extensions share notebook server
4.20.0
aext-shared
Anaconda extensions shared library
4.20.0
aext-toolbox
4.20.0
aext_environmen...
4.20.0
aiobotocore
Async client for aws services using botocore and aiohttp
2.12.3
aiohappyeyeballs
Happy eyeballs for asyncio
2.4.0
aiohttp
Async http client/server framework (asyncio)
3.10.5
aioitertools
Asyncio version of the standard multiprocessing module
0.7.1
aiosignal
Aiosignal: a list of registered asynchronous callbacks
1.2.0
alabaster
Lightweight, configurable sphinx theme
0.7.16
alembic
A database migration tool for sqlalchemy.
1.13.3
altair
A declarative statistical visualization library for python
5.0.1
534 packages available
Create
Clone
Import
Backup
Remove
2
3
Create new environment
Name:
my-venv
Location:
C:\Users\yange\anaconda3\envs\my-venv
Packages:
Python
3.12.11
R
3.6.1
Cancel
Create
4

Once your environment is activated, you can install packages with pip, conda or Anaconda

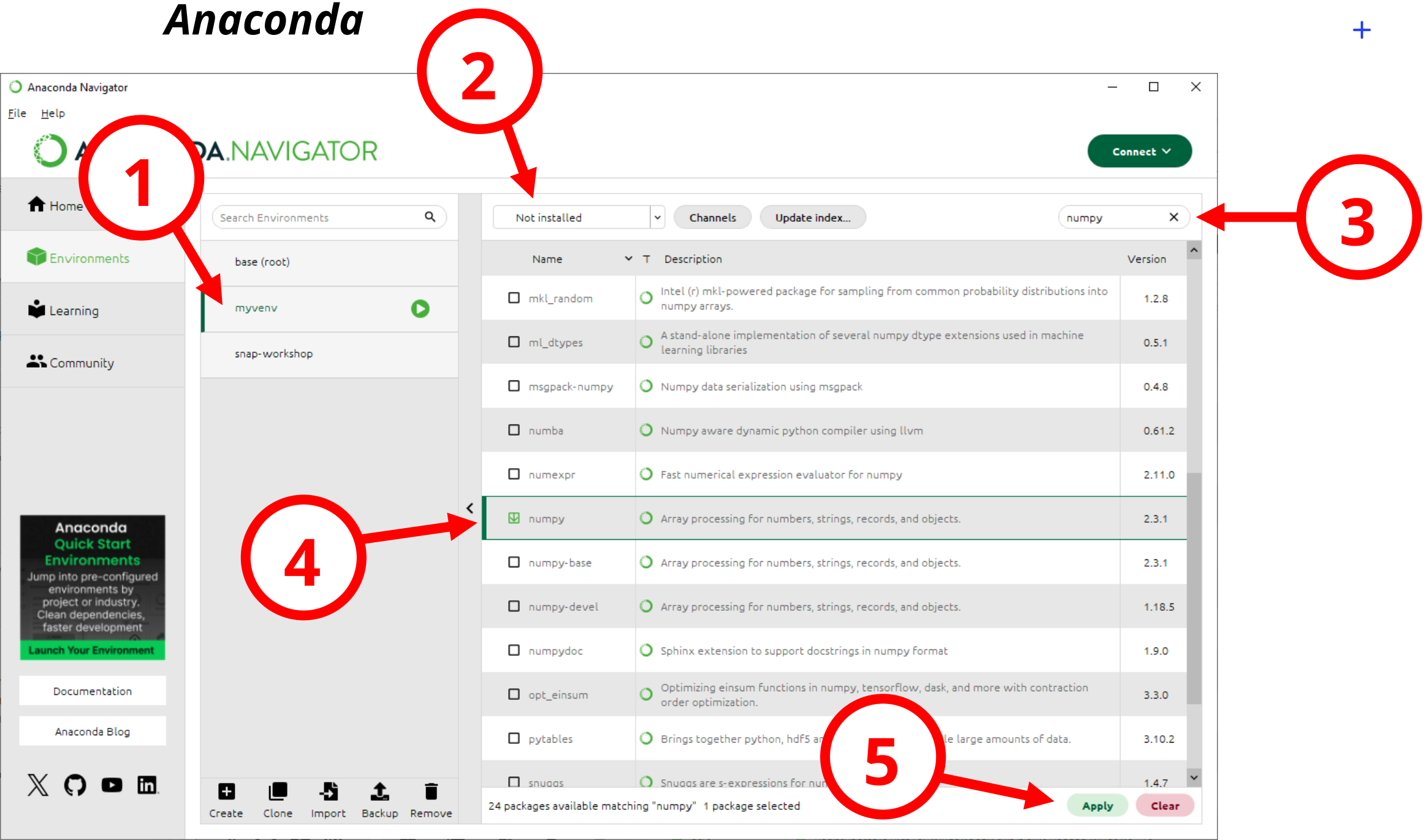
```
(my-venv)$ pip install numpy
(my-venv)$ conda install numpy
```

+ Installs latest available version by default OR specify which version to install.
+ Dependencies are installed automatically.

If you have a list of required packages, run:

```
(my-venv)$ pip install -r requirements.txt
(my-venv)$ conda install --file requirements.txt
```

Anaconda


Anaconda Navigator
File Help
ANACONDA.NAVIGATOR
Connect
Home
Environments
Learning
Community
Search Environments
base (root)
myvenv
snap-workshop
Not installed
Channels
Update index...
numpy
Name
T
Description
Version
mkl_random
Intel (r) mkl-powered package for sampling from common probability distributions into numpy arrays.
1.2.8
ml_dtypes
A stand-alone implementation of several numpy dtype extensions used in machine learning libraries
0.5.1
msgpack-numpy
Numpy data serialization using msgpack
0.4.8
numba
Numpy aware dynamic python compiler using llvm
0.61.2
numexpr
Fast numerical expression evaluator for numpy
2.11.0
numpy
Array processing for numbers, strings, records, and objects.
2.3.1
numpy-base
Array processing for numbers, strings, records, and objects.
2.3.1
numpy-devel
Array processing for numbers, strings, records, and objects.
1.18.5
numpydoc
Sphinx extension to support docstrings in numpy format
1.9.0
opt_einsum
Optimizing einsum functions in numpy, tensorflow, dask, and more with contraction order optimization.
3.3.0
pytables
Brings together python, hdf5 an
le large amounts of data.
3.10.2
snugas
Snugas are s-expressions for nu
1.4.7
Anaconda Quick Start Environments
Jump into pre-configured environments by project or industry. Clean dependencies, faster development
Launch Your Environment
Documentation
Anaconda Blog
Create
Clone
Import
Backup
Remove
24 packages available matching "numpy" 1 package selected
Apply
Clear
1
2
3
4
5

It’s good practice to make
a new virtual environment for each project.

RUNNING SCRIPTS

TYPES OF PYTHON FILES

(This is a guide and not strictly enforced by Python)

Scripts (.py)	Modules (.py)	Notebooks (.ipynb)
Executes all at once	Generally not executed.	Cells are executed individually.
Runs unattended.	Stores functions, classes, global variables.	Interactive. You can log your work.

Creating a script

Use any text editor: vim, nano, Notepad++, Visual Studio Code

Then save file with a ".py" extension and run.

```
(my-venv)$ python /path/to/file/my-script.py
(my-venv)$ python3 /path/to/file/my-script.py
```

IDEs

Integrated development environments (IDEs) provide a comprehensive environment to write, run, interact, debug.

Examples include Spyder, PyCharm.

VUW_SNAP

Welcome | my-script.py

my-script.py

```python
print("Hello world!")
```

yangelaxue@yange: ~/Documents/VUW_SNAP

```
(snap-venv) yangelaxue@yange:~/Documents/VUW_SNAP$ python3 my-script.py
Hello world!
(snap-venv) yangelaxue@yange:~/Documents/VUW_SNAP$
```

Python reads top to bottom, left to right.

VUW_SNAP

my-script.py | analyse.py | data_util.py

analyse.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt

data_name = 'data/inflammation-01.csv'
data = np.loadtxt(data_name,delimiter=',')

n_patients, n_days = data.shape
days = np.arange(n_days)

fig, axes = plt.subplots(1,3,figsize=(8,3),sharey=True,tight_layout=True)

axes[0].plot(days,data.mean(axis=0))
axes[0].set_title('Mean')

axes[1].plot(days,data.max(axis=0))
axes[1].set_title('Max')

axes[2].plot(days,data.min(axis=0))
axes[2].set_title('Min')

for ax in axes:
    ax.set_xlabel('Days')
axes[0].set_ylabel('Frequency')

plt.savefig('figures/data-01.png')
```

Live Share | Spaces: 4 | UTF-8 | LF | {} Python | Finish Setup | snap-venv (3.12.3) | Go Live | Prettier

View Go Run ··· VUW_SNAP

my-script.py | analyse-1.py | analyse-2.py | analyse-3.py | util.py

util.py > plot_mean_max_min

```python
"""
Contains functions that can help analyse and plot data.

"""

import numpy as np
import matplotlib.pyplot as plt

def plot_mean_max_min(data,save_name:str):
    """
    Creates a plot of the mean, max and min number of events per day
    for a single dataset.

    Parameters
    ----------
        data : np.nparray
            Has shape==(n_patients,n_days)
        save_name : str
            The name of the figure to be saved
    """

    n_patients, n_days = data.shape
    days = np.arange(n_days)

    fig, axes = plt.subplots(1,3,figsize=(8,3),sharey=True,tight_layout=True)

    axes[0].plot(days,data.mean(axis=0))
    axes[0].set_title('Mean')

    axes[1].plot(days,data.max(axis=0))
    axes[1].set_title('Max')

    axes[2].plot(days,data.min(axis=0))
    axes[2].set_title('Min')

    for ax in axes:
        ax.set_xlabel('Days')
    axes[0].set_ylabel('Frequency')

    fig.savefig(f'figures/{save_name}.png')
```

Live Share | Ln 19, Col 47 | Spaces: 4 | UTF-8 | LF | {} Python | Finish Setup | snap-venv (3.12.3) | Go Live | Prettier

You can move this function to another file. *"util" for utility*

Brief description of the files contents.

Describe the function, its parameter(s) and return(s)

VUW_SNAP

my-script.py | analyse-1.py | analyse-2.py | analyse-3.py | util.py

analyse-3.py > ...

```python

import numpy as np
import glob

from util import plot_mean_max_min

# Load all the data
data_names = [
    'data/inflammation-01.csv',
    'data/inflammation-05.csv',
    'data/inflammation-10.csv',
]
# The data size is too large - use a generator
all_data = (np.loadtxt(name,delimiter=',') for name in data_names)

idxes = [int(name.split('-')[-1].split('.')[0]) for name in data_names]

# Begin analysing
for i, data in zip(idxes,all_data):

    plot_mean_max_min(data,save_name=f'data-{i}')
```

Import your function

Replace mess with a single line

0 Live Share Spaces: 4 UTF-8 LF {} Python Finish Setup snap-venv (3.12.3) Go Live Prettier

COMPLEX FUNCTIONS

*args and **kwargs

```python
def my_function(x,y,a=10,*args,**kwargs):
    """
    This is my function.

    Parameters
    ----------
        x, y : float
            Required input parameters.
        a : float
            Optional constant. Default is 10.
        *args
            Optional unnamed arguments.
        **kwargs
            Optional named arguments, stands for 'keyword arguments'.
    Returns
    -------
        ret : float
            Combination of x and y.
    """
    ret = a*x + y
    return ret
```

“args” is a tuple
“kwargs” is a dictionary

LOADING LARGE DATA FILES

GENERATORS

Problem: The data file(s) size is too large, so we are unable to load it all into RAM :(

Solution: Load only a portion of data into memory at a time, so that we are not storing information that we are not using.

```python
def get_data(f_names):
    for name in f_names:
        yield np.loadtxt(name,delimiter=',')

data_names = [
    'data/inflammation-01.csv',
    'data/inflammation-05.csv',
    'data/inflammation-10.csv',
]

data_generator = get_data(data_names)

for data in data_generator:
    print("Perform operations here.")
```

This is in functional form:

the keyword “`yield`” is used instead of “`return`”.

`data_generator` is a generator type. The data is only loaded or calculated as needed.

```python
data_names = [
    'data/inflammation-01.csv',
    'data/inflammation-05.csv',
    'data/inflammation-10.csv',
]

data_generator = (np.loadtxt(name,delimiter=',') for name in data_names)

for data in data_generator:
    print("Perform operations here.")
```

This is in list comprehension form:

It does the same thing shown in the previous slide.