Info Session: Virtual Environments
By Nadia Blostein

" Every program is a set of instructions, whether it's to add two numbers or send a request over the internet. Compilers and interpreters take human-readable code and convert it to computer-readable machine code. "

About Python...

- "Python is an **interpreted**, **high-level** and **general-purpose** programming language" (Wiki)
  - **Interpreted**: executed line-by-line
  - **High-level**: human readable (as opposed to machine readable)
  - **General purpose**: designed for variety of applications
- Python interpreter:
  - "The Python interpreter is a *virtual machine*, meaning that it is software that emulates a physical computer." (https://www.aosabook.org/en/500L/a-python-interpreter-written-in-python.html)
  - Thus, when you install a version of Python on your computer, you are installing a version of a Python interpreter
  - The Python interpreter can read and execute Python code line-by-line
  - Practise (from your command line):
    - *$ python test_script.py*
    - you are asking your Python interpreter to read and execute each line in test_script.py

NOTATION. $ some_text_here: some_text_here is a command-line argument (you can just copy-paste it into your terminal!)

About Python...

- "Python is an **interpreted**, **high-level** and **general-purpose** programming language" (Wiki)
  - **Interpreted**: executed line-by-line
  - **High-level**: human readable (as opposed to machine readable)
  - **General purpose**: designed for variety of applications
- Python interpreter:
  - "The Python interpreter is a *virtual machine*, meaning that it is software that emulates a physical computer." (https://www.aosabook.org/en/500L/a-python-interpreter-written-in-python.html)
  - Thus, when you install a version of Python on your computer, you are installing a version of a Python interpreter
  - The Python interpreter can read and execute Python code line-by-line
  - Practise (from your command line):
    - *$ python test_script.py*
    - you are asking your Python interpreter to read and execute each line in test_script.py
- Python installation: https://wiki.python.org/moin/BeginnersGuide/Download
- **Great! Now that your computer has a Python interpreter, all you're just missing is...**

MODULES, LIBRARIES AND PACKAGES

module < library < packages.
(*https://stackoverflow.com/questions/16997950/whats-the-difference-between-module-package-and-library-in-haskell*)

- "A **module** is a set of functions, types, classes, ... put together in a common namespace.
- A **library** is a set of modules which makes sense to put together and that can be used in a program or another library.
- A **package** is a unit of distribution that can contain a library or an executable or both. It's a way to share your code with the community."

Cool… But how does Python download, store and resolve packages?

How does Python download, store and resolve packages?

1. **<u>System packages</u>**: packages that are part of the standard Python library (automatically downloaded when you download your interpreter)
2. **<u>Site Packages:</u>** 3rd party packages, usually installed using pip or easy_install commands
   - **<u>PyPI (Python Package Index)</u>**: 3rd party software repository for Python (Wiki)
   - **<u>Pip</u>**: standard package-management system used to install and manage software packages written in Python. Many packages can be found in the default source for packages and their dependencies — Python Package Index (Wiki)

How does Python download, store and resolve packages?

```
[welcome to ~ §          Command line prompt (usually takes commands in a language called Bash)
[welcome to ~ § python --version
Python 3.6.1
[welcome to ~ § python -V
Python 3.6.1
[welcome to ~ § python
Python 3.6.1 | packaged by conda-forge | (default, May 23 2017, 14:31:56)
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.53)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.prefix
'/anaconda3'
>>> import site
>>> site.getsitepackages()
['/anaconda3/lib/python3.6/site-packages']
>>> exit()
welcome to ~ § █
```

How does Python download, store and resolve packages?

```
[welcome to ~ §
[welcome to ~ §   python --version
Python 3.6.1
[welcome to ~ §   python -V
Python 3.6.1
[welcome to ~ § python
Python 3.6.1 | packaged by conda-forge | (default, May 23 2017, 14:31:56)
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.53)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.prefix
'/anaconda3'
>>> import site
>>> site.getsitepackages()
['/anaconda3/lib/python3.6/site-packages']
>>> exit()
welcome to ~ § █
```

Check what Python interpreter you are using

How does Python download, store and resolve packages?

```
[welcome to ~ § 
[welcome to ~ § python --version
Python 3.6.1
[welcome to ~ § python -V
Python 3.6.1
[welcome to ~ § python
Python 3.6.1 | packaged by conda-forge | (default, May 23 2017, 14:31:56)
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.53)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.prefix
'/anaconda3'
>>> import site
>>> site.getsitepackages()
['/anaconda3/lib/python3.6/site-packages']
>>> exit()
welcome to ~ § 
```

Tell your OS that you now want to use the Python interpreter (*ie write commands in Python instead of in Bash*)

How does Python download, store and resolve packages?

```
[welcome to ~ §
[welcome to ~ § python --version
Python 3.6.1
[welcome to ~ § python -V
Python 3.6.1
[welcome to ~ § python
Python 3.6.1 | packaged by conda-forge | (default, May 23 2017, 14:31:56)
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.53)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys        You are now using your Python interpreter
>>> sys.prefix
'/anaconda3'
>>> import site
>>> site.getsitepackages()
['/anaconda3/lib/python3.6/site-packages']
>>> exit()
welcome to ~ § 
```

RECALL:
1. **System packages:** standard Python library
2. **Site Packages:** 3rd party packages (usually from PyPI)

```
>>> import sys
>>> sys.prefix
'/anaconda3'
>>> import site
>>> site.getsitepackages()
['/anaconda3/lib/python3.6/site-packages']
>>> exit()
welcome to ~ §
```

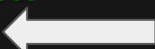**Find out where your system packages are stored!**

RECALL:
1. **<u>System packages:</u>** standard Python library
2. **<u>Site Packages:</u>** 3rd party packages (usually from PyPI)

```
>>> import sys
>>> sys.prefix
'/anaconda3'        ⬅
>>> import site
>>> site.getsitepackages()
['/anaconda3/lib/python3.6/site-packages']
>>> exit()
welcome to ~ §
```

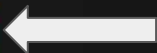**Find out where your system packages are stored!**

RECALL:
1. **<u>System packages:</u>** standard Python library
2. **<u>Site Packages:</u>** 3rd party packages (usually from PyPI)

```
>>> import sys
>>> sys.prefix
'/anaconda3'
>>> import site
>>> site.getsitepackages()
['/anaconda3/lib/python3.6/site-packages']
>>> exit()
welcome to ~ §
```

Find out where your site packages are stored!

RECALL:
 1. **System packages:** standard Python library
 2. **Site Packages:** 3rd party packages (usually from PyPI)

```
>>> import sys
>>> sys.prefix
'/anaconda3'
>>> import site
>>> site.getsitepackages()
['/anaconda3/lib/python3.6/site-packages']
>>> exit()
welcome to ~ §
```

Find out where your site packages are stored!

How does Python download, store and resolve packages?

```
[welcome to ~ §
[welcome to ~ § python --version
Python 3.6.1
[welcome to ~ § python -V
Python 3.6.1
[welcome to ~ § python
Python 3.6.1 | packaged by conda-forge | (default, May 23 2017, 14:31:56)
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.53)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.prefix
'/anaconda3'
>>> import site
>>> site.getsitepackages()
['/anaconda3/lib/python3.6/site-packages']
>>> exit()
welcome to ~ §
```

or: CTRL + D   stop using your Python interpreter

How does Python download, store and resolve packages?

1. **<u>System packages</u>**: packages that are part of the standard Python library
2. **<u>Site Packages:</u>** 3rd party packages
3. **<u>Virtual Environments:</u>**
   - Python tool for package management and project isolation
   - Allow for site packages (3rd party packages) to be installed in a local directory (ex: specific project) instead of globally
     - ie create a "local" (project-specific) site-packages directory that can use its own version of Python!
   - Why? https://towardsdatascience.com/virtual-environments-104c62d48c54#ee81
     - **"Resolve dependency issues** by allowing you to use different versions of a package for different projects.
     - Make your project **self-contained** and **reproducible** by capturing all package dependencies in a requirements file.
     - Install packages on a host on which you do not have admin privileges.
     - Keep your global `site-packages/` system site-packages directory tidy by removing the need to install packages system-wide which you might only need for one project."

Virtual Environments: Motivation

- Project B examines the relationship between TBV and age in a *new* dataset
  - *project_B.py* (script) loads & analyzes *new_dataset.csv* (data file)
  - Dependencies:
    - **numpy v 1.19**
    - **pandas v 1.1.2**
    - **matplotlib 3.3.1**
- Project A examines the relationship between TBV and age in an *old* dataset
  - *project_A.py* (script) loads & analyzes *dataset.csv* (data file)
  - Dependencies
    - **numpy v 1.15**
    - **pandas v 1.1.2**
    - **matplotlib 3.3.1**,
    - **linearmodels**

Virtual Environments: Motivation

- Project B examines the relationship between TBV and age in a *new* dataset
  - *project_B.py* (script) loads & analyzes *new_dataset.csv* (data file)
  - Dependencies:
    - **numpy v 1.19**
    - **pandas v 1.1.2**
    - **matplotlib 3.3.1**
- Project A examines the relationship between TBV and age in an *old* dataset
  - *project_A.py* (script) loads & analyzes *dataset.csv* (data file)
  - Dependencies
    - **numpy v 1.15**
    - **pandas v 1.1.2**
    - **matplotlib 3.3.1**,
    - **linearmodels**
- **Alert!** Different package dependencies → store in separate virtual environments

Build Your Own Venv

Note: this example assumes that you are using Python 3

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Make a directory and cd into it

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Example from:

Create a virtual environment (can name it anything you want)

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt())))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Create a virtual environment (can name it anything you want)

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
```

Build Your Own Venv

Example from: [Python Virtual Environments: A Primer – Real Python](#)

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
```

```
/Users/nadiablostein/Downloads/python_test_env
└── test_env
    ├── bin
    │   ├── activate
    │   ├── activate.fish
    │   ├── easy_install
    │   ├── easy_install-3.7
    │   ├── pip
    │   ├── pip3
    │   ├── pip3.7
    │   ├── python -> python3
    │   └── python3 -> /Library/Frameworks/Python.framework/Versions/3.7/bin/python3
    ├── include
    ├── lib
    │   └── python3.7
    │       └── site-packages
```

contains script that allows you to switch from global venv to local venv

Build Your Own Venv

Example from: [Python Virtual Environments: A Primer – Real Python](#)


```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
```

```
/Users/nadiablostein/Downloads/python_test_env
└── test_env
    ├── bin
    │   ├── activate
    │   ├── activate.csh
    │   ├── activate.fish
    │   ├── easy_install
    │   ├── easy_install-3.7
    │   ├── pip
    │   ├── pip3
    │   ├── pip3.7
    │   ├── python -> python3
    │   └── python3 -> /Library/Frameworks/Python.framework/Versions/3.7/bin/python3
    ├── include
    └── lib
        └── python3.7
            └── site-packages
```

Copy of your local env's version of Python

Build Your Own Venv

Example from: [Python Virtual Environments: A Primer – Real Python](#)

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
```

```
/Users/nadiablostein/Downloads/python_test_env
└── test_env
    ├── bin
    │   ├── activate
    │   ├── activate.csh
    │   ├── activate.fish
    │   ├── easy_install
    │   ├── easy_install-3.7
    │   ├── pip
    │   ├── pip3
    │   ├── pip3.7
    │   ├── python -> python3
    │   └── python3 -> /Library/Frameworks/Python.framework/Versions/3.7/bin/python3
    ├── include
    ├── lib
        └── python3.7
            └── site-packages
```

Where the 3rd party packages of the local venv are stored

Build Your Own Venv

Switch from global to local venv

```
welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
welcome to ~/Downloads/python_test_env § python3 -m venv test_env
welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env § pip -q install bcrypt
welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Example from:

Switch from global to local venv

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Example from:

Switch from local to global venv

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Example from:

Switch from local to global venv


```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env § pip -q install bcrypt
welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Install 3rd party package in global site-packages directory

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
[b'$2b$12$u/6wZBvKRoo0cJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Use bcrypt to hash a password

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pa
ssword'.encode('utf-8'), bcrypt.gensalt()))"
b'$2b$12$u7UW2DVKROOUeJJEIeToDeWhioYr.ueYr.JxNVqFJFLGNVyDbLAbG
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Use bcrypt to hash a password

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Example from:

Switch back to your local venv

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt())))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF_3xNVgFJfLGNvvDhFAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt())))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Switch back to your local venv

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Use bcrypt to hash a password

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
[ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Build Your Own Venv

Example from:

ModuleNotFoundError !

```
[welcome to ~/Downloads § mkdir python_test_env && cd python_test_env
[welcome to ~/Downloads/python_test_env § python3 -m venv test_env
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
[(test_env) welcome to ~/Downloads/python_test_env § deactivate
[welcome to ~/Downloads/python_test_env § pip -q install bcrypt
[welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.hashpw('pas
sword'.encode('utf-8'), bcrypt.gensalt()))"
[b'$2b$12$u/6wZBvKRooOcJ5LICtobeWh18YF.deYF.3xNVqFJfLGNvyDbEAbG'
[welcome to ~/Downloads/python_test_env § source test_env/bin/activate
(test_env) welcome to ~/Downloads/python_test_env § python -c "import bcrypt; print(bcrypt.
hashpw('password'.encode('utf-8'), bcrypt.gensalt()))"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ModuleNotFoundError: No module named 'bcrypt'
(test_env) welcome to ~/Downloads/python_test_env § deactivate
welcome to ~/Downloads/python_test_env §
```

Anaconda and Miniconda

- **Pip**: package manager
- **Virtualenv**: environment manager
- **Conda:** both!
  - Packaging tool and installer (like pip!) → packages from Anaconda Repository and Anaconda Cloud (instead of PyPI)
  - Creates a virtual environment (like virtualenv!)
  - **Command line tool**: does not require a Python interpreter
  - Conda Cheat Sheet
- **Anaconda:** Python and R distribution, "aims to simplify package management and deployment" (Wiki)
  - **Python interpreter** (*any version you chose*) + **Conda** + **"meta-packages anaconda"** (predetermined set of Python packages)
- **Miniconda:**
  - **Python interpreter** (*any version you chose*) + **Conda**
  - manually install any packages/dependencies you need
  - ⊕: less space, quicker to install, customizable
  - ⊖: requires a better understanding of venvs as well as what your project requires
- **READ MORE:**
  - **Anaconda vs. Miniconda vs. Virtualenv**
  - **Understanding Conda and Pip**

Build your own Conda Virtual Environment!

(1) Download the correct files from the CobraLab Google Drive: CobraLab → PRESENTATIONS → LAB MEETINGS → Info_Session_2020 → 2020-09-16-virtual-environments → test_conda_env.tar.gz: https://drive.google.com/drive/folders/1Ngtmwn-ePr-8Kt5ai8CluKVG93E3qupI?usp=sharing

(2) Unzip the .tar file and cd into through your terminal!

NOTATION
$ some_text_here: some_text_here is a command-line argument (you can just copy-paste it into your terminal!)

Build your own Conda Virtual Environment!

OPTION 1 (2nd option later):

(3) Create your conda environment:
```
$ conda create --name my_conda_env python=3.8.3
```

(2) Activate your condal environment:
```
$ source activate my_conda_env
```

(5) Download all of the right packages and dependencies to be able to run HCP_interactive_plot.ipynb:
```
$ conda install numpy
$ conda install pandas
$ conda install plotly
$ conda install dash
$ conda install -c conda-forge -c plotly jupyter-dash
$ conda install jupyter
$ conda install nb_conda_kernels
```

(6) Launch Jupyter Notebook!
```
$ jupyter notebook
```

Build your own Conda Virtual Environment!

(8) Open the Jupyter Notebook:

Build your own Conda Virtual Environment!

(9) Make sure you are using the right kernel

Build your own Conda Virtual Environment!

(10) Run all of the cells

Your **final goa**l is for anybody (including yourself) to be able **reproduce** your results **whenever** and as **seamlessly** as possible, whilst minimizing the steps required to do so.
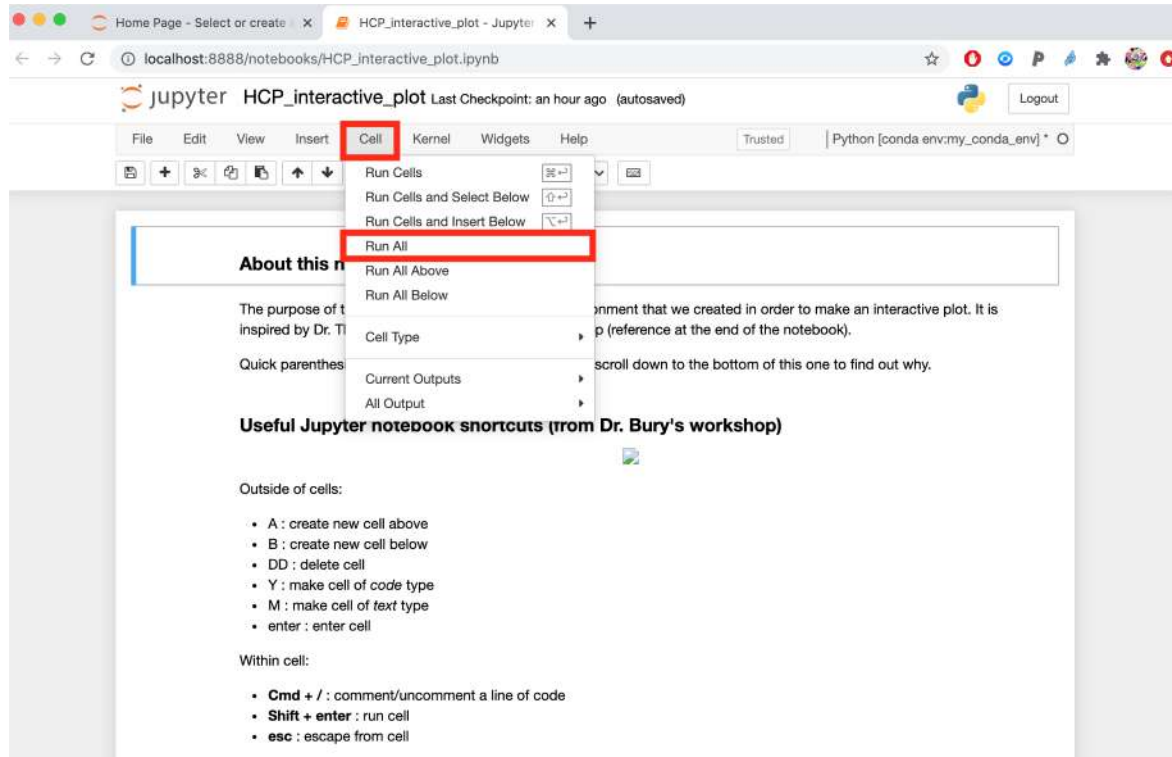
Maintaining good habits… save dependencies

(1) <u>pip freeze</u>: virtual environments (venv)
- List all packages in the current environment:
  - $ pip freeze -l
- Save packages for future use:
  - $ PATH_TO_ENV_1/bin/pip pip freeze > requirements.txt
- Reinstall packages from an export file:
  - $ PATH_TO_ENV_2/bin/pip install -r requirements.txt

(2) <u>conda list</u>: conda environments
- List all packages in the current environment:
  - $ conda list
- List all packages installed into the environment 'myenv':
  - $ conda list -n myenv
- Save packages for future use:
  - $ conda list --explicit > package-list.txt
- Reinstall packages from an export file:
  - $ conda create -n myenv --file package-list.txt

(3) <u>Makefiles</u>
- Deploying software, sharing projects

OPTION 2: **package list**

(3) Create your conda environment:
    $ conda create --name my_conda_env python=3.8.3 --file=package-list.txt

(4) Activate your condal environment:
        $ source activate my_conda_env

(5) Launch Jupyter Notebook!
        $ jupyter notebook or $ python HCP_interactive_plot.py

OPTION 3: makefile

```
install: # create your virtual environment
        @echo "Creating virtual environment"
        conda create --prefix ./my_temp_conda_env python=3.8.3 --file=package-list.txt

plot: # generate interactive plot
        echo "generating interactive plot (.html file)"
        ./my_temp_conda_env/bin/python HCP_interactive_plot.py

clean: # delete your virtual environment
        @echo "Cleaning up"
        rm -rf ./my_temp_conda_env
```

makefile anatomy

OPTION 3: makefile

(3) $ make install


```
install: # create your virtual environment
        @echo "Creating virtual environment
        conda create --prefix ./my_temp_conda_env python=3.8.3 --file=package-list.txt

plot: # generate interactive plot
        echo "generating interactive plot (.html file)"
        ./my_temp_conda_env/bin/python HCP_interactive_plot.py

clean: # delete your virtual environment
        @echo "Cleaning up"
        rm -rf ./my_temp_conda_env
```

makefile anatomy

OPTION 3: makefile

(3) $ make install
(4) $ make plot


```
install: # create your virtual environment
        @echo "Creating virtual environment"
        conda create --prefix ./my_temp_conda_env python=3.8.3 --file=package-list.txt

plot: # generate interactive plot
        echo "generating interactive plot ( html file)"
        ./my_temp_conda_env/bin/python HCP_interactive_plot.py

clean: # delete your virtual environment
        @echo "Cleaning up"
        rm -rf ./my_temp_conda_env
```

makefile anatomy

OPTION 3: makefile

(3) $ make install
(4) $ make plot
(5) $ make clean

```
install: # create your virtual environment
        @echo "Creating virtual environment"
        conda create --prefix ./my_temp_conda_env python=3.8.3 --file=package-list.txt

plot: # generate interactive plot
        echo "generating interactive plot (.html file)"
        ./my_temp_conda_env/bin/python HCP_interactive_plot.py

clean: # delete your virtual environment
        @echo "Cleaning up"
        rm -rf ./my_temp_conda_env
```

makefile anatomy

Thank you!