Introduction to Python

A HMS Research Computing and DevOps Production

rchelp@hms.harvard.edu

HMS-RC Training

- HMS-RC hosts workshops on other subjects like R/Bioconductor, Matlab, Basic/Intermediate O2

  https://harvardmed.atlassian.net/wiki/spaces/O2/pages/1588666299/User+Training

- HMS-RC Training Github – Links to notes for all workshops (including this one)

  https://github.com/hmsrc/user-training

- Today's materials are: IntroToPython.pdf & IntroToPython.ipynb

Introduction

Why python?

Python Is...

simple - resembles plain English

easy - no need to declare (in most cases), memory management

clean - whitespace-formatted for visibility

interpreted - good for developing, bad for performance (more on this later)

Interpreted	Compiled
Rapid prototyping	Faster Performance
Requires interpreter	Requires compiler (GCC, etc.)
Dynamic typing	Static typing
Code-level optimization	Code- and Compiler-level optimization

Data and Script Management

Data Management

1. **Planning: Plan ahead**
2. **Active Research: Document**
3. **Dissemination: Share confidently**

Note: be sure to ask your PI and your department about standard practices in your field!

Harvard Biomedical Data Management Website: https://datamanagement.hms.harvard.edu

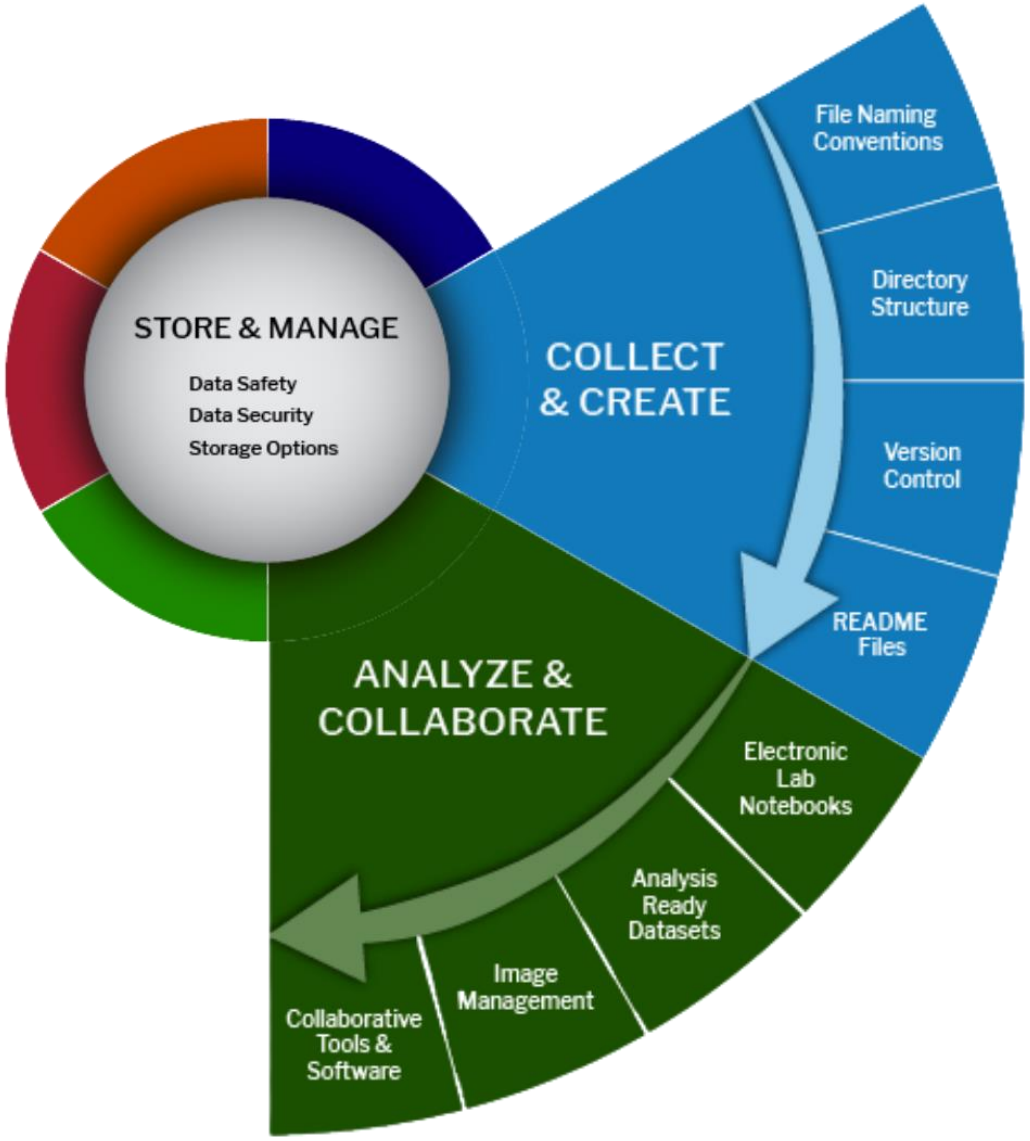
Resources: https://datamanagement.hms.harvard.edu/about/what-research-data-management/rdm-resources

Planning

- Document the activities for the entire lifecycle in a Data Management Plan
- Determine if you need a Data Use Agreement to acquire or share data
- Adopt a community-based metadata standard if applicable
- Consider how the data will be stored and protected over the duration of the project and beyond
- Assign roles & responsibilities for managing data

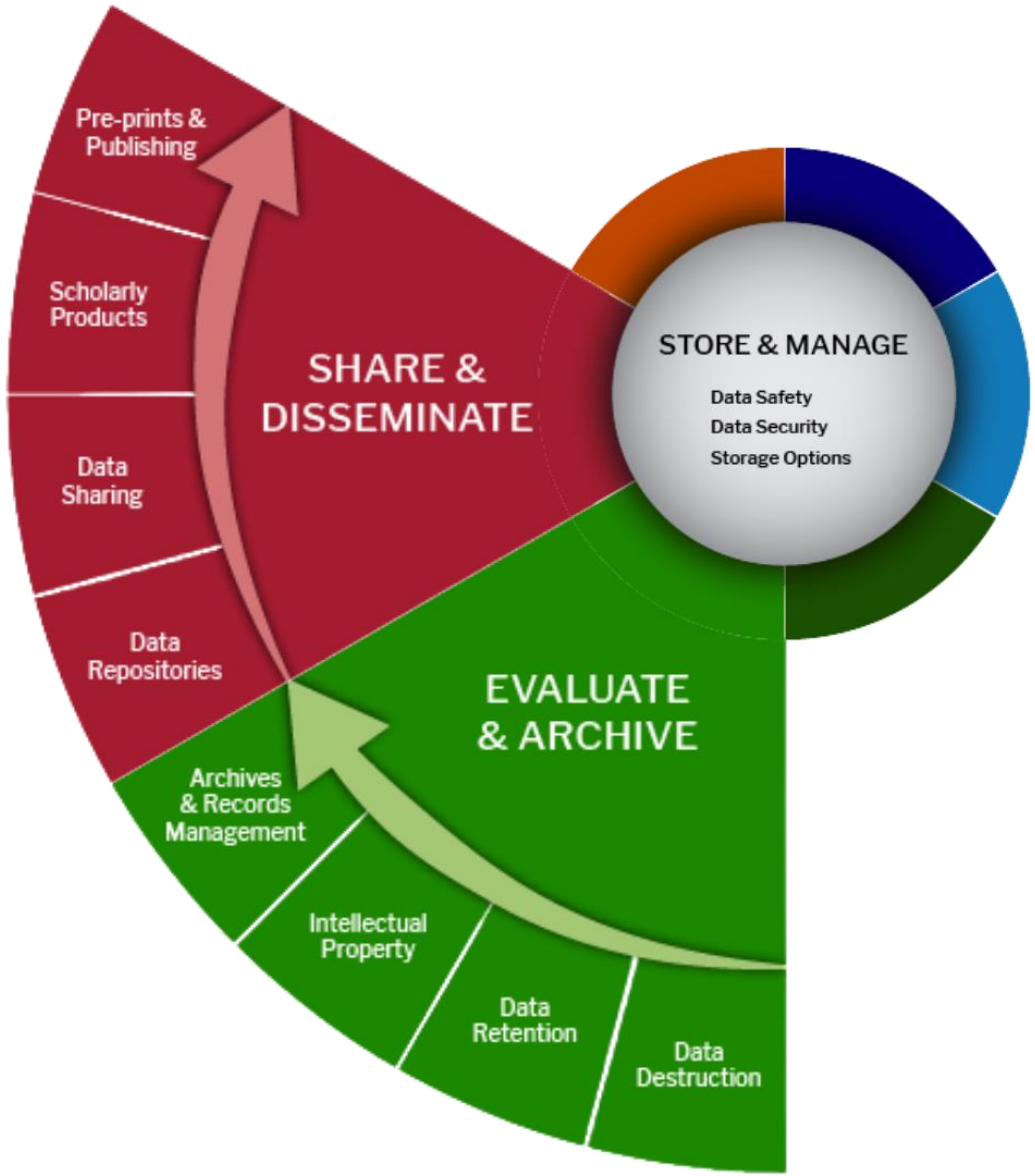
Active Research

- Determine how the data will be organized including folder structure & file naming
- Consider versioning control for changes for both software and data products
- Create a README file to record the metadata that will be associated with data
- Link related code and algorithms
- Use tools & software to work with collaborators during the project



Dissemination

- Determine what data will be disseminated, to who, when, and where
- Publish data in an open repository to receive a DOI and citation for your work
- Use standard, nonproprietary approaches and provide accompanying metadata
- Think about your preservation strategy and adhere to your lab's standard practices
- Research records should generally be retained no fewer than seven (7) years after the end of a research project or activity



Accessing Python On O2

Accessing Python on O2 – Cluster Access

- Connect using terminal applications
  - Linux or Mac – Native Terminal, Putty
  - Windows – MobaXterm (Recommended), Putty
- For detailed instructions on how to login to O2, visit our wiki.
- Please Note: **All** O2 cluster logins originating from outside the HMS network will require two-factor authentication

Accessing Python on O2 – Logging In

If on Mac, open up XQuartz prior to logging into O2.
If on Windows, use MobaXterm.

Open a terminal and ssh into o2.hms.harvard.edu:

```
$ ssh -Y -L PORT:127.0.0.1:PORT rc_training01@o2.hms.harvard.edu
(rc_training01@o2.hms.harvard.edu) password:

rc_training01@login01:~$
```

Replace PORT with a number between 50000 and 60000

Accessing Python on O2 –Interactive job

Once on O2, start an interactive job:

```
# make sure you use the same PORT as when logging in
rc_training01@login01:~$ srun --pty -p interactive -t 0-2 --x11 --tunnel
PORT:PORT bash

# will get a message about your job waiting for resources
# once resources are allocated, you will be placed on a compute node and your
prompt will change:

rc_training01@compute-a:~$

# now can enter any commands you want
```

Accessing Python on O2 – Finding Python

```
$ module avail python
    No modules found!
    Use "module spider" to find all possible modules.
    Use "module keyword key1 key2 ..." to search for all possible modules matching
    any of the "keys".

$ module spider python
       Versions:
          python/2.7.12
          python/3.6.0
          python/3.7.4
          Python/3.8.12
```

Accessing Python on O2 – Finding Python, cont'd.

```
$ module spider python/3.8.12
    You will need to load all module(s) on any one of the lines below before the
    "python/3.8.12" module is available to load.

          gcc/9.2.0
    (snip)

$ module load gcc/9.2.0 python/3.8.12
$ which python3
/n/app/python/3.8.12/bin/python3
$ python3
Python 3.8.12 (default, Sep 13 2021, 17:05:27)
[GCC 9.2.0] on linux
 Type "help", "copyright", "credits" or "license" for more information.
 >>>
```

Today's course will use 3.8.12. (To exit the interpreter, type ctrl+D, exit(), or quit().)

Alternatively, virtual environments

```
$ module load gcc/9.2.0 python/3.8.12
 $ which virtualenv
 /n/app/python/3.8.12/bin/virtualenv
 $ virtualenv nameofenv --system-site-packages
# truncated
 $ source nameofenv/bin/activate
 (nameofenv)$ which python3
 ~/nameofenv/bin/python
```

To deactivate:

```
(nameofenv)$ deactivate
$
```

To follow the practical portion of the class, we will be sourcing a pre-made virtual environment.

Why virtual environments?

- Python has modules. if you want to install your own, you need a virtual environment.
- The version on O2 has a certain number of built-in modules; the `--system-site-packages` flag allows your virtual environment to inherit those packages so you don't have to reinstall them yourself, if you're using python modules that are older than 3.7.4.
- For more information, you can visit our Personal Python Packages wiki page.

Installing Python Modules

- To install modules, generally use `pip3` or `easy-install` (we recommend `pip3`):

```
(nameofenv)$ pip3 install packagename
```

- If that doesn't work (e.g. you've downloaded the archive manually), follow the instructions in the provided README file, but it'll go something like

```
(nameofenv)$ python3 setup.py install
```

- (some will have you use `build` before `install`). Make sure you read the (hopefully provided) instructions when installing modules manually.

Viewing Modules

- **pip list** shows ALL packages. **pip freeze** shows packages YOU installed via **pip3** command in a requirements format.

```
$ python3 -m pip freeze
aiohttp==2.3.10
alembic==1.4.2
appdirs==1.4.4
argh==0.26.2

$ python3 -m pip list
Package                 Version
----------------------- ----------
aiohttp                 2.3.10
alembic                 1.4.2
appdirs                 1.4.4
argh                    0.26.2
```

Viewing modules cont.

- To see ALL modules available on your Python build, type either of:

```
$ python3 -c "help('modules')"
```

and

```
$ python3
Python 3.8.12 (default, Sep 13 2021, 17:05:27)
[GCC 9.2.0] on linux2
 Type "help", "copyright", "credits" or "license" for more
information.
>>> help('modules')
```

Viewing modules cont.

- And it will output something like:

```
Please wait a moment while I gather a list of all available modules...

(there might be some warnings here...)

__future__          aifc                imaplib             scipy
_ast                antigravity         imghdr              secrets
_asyncio            argparse            imp                 select
_bisect             array               importlib           selectors
.
.
.
```

Accessing Python via Jupyter Notebook

- Today we will be using a Jupyter Notebook for our hands-on portion, though you don't need to use Jupyter to run Python code.
- There are many advantages to using a Jupyter notebook:
  - All your code is saved in one place
  - You can share your notebook with others
  - Can create interactive plots
- Generic instructions for using Jupyter on O2 can be found here.

Setup for hands-on portion

- Copy the Jupyter Notebook:

```
$ cp /n/groups/rc-training/python/2022/IntroToPython.ipynb .
```

- Source the virtual environment, which has the required packages for running the code in the notebook:

Make sure you have gcc/9.2.0 and python/3.8.12 loaded first!

```
$ source /n/groups/rc-training/python/2022/trainingenv/bin/activate
(trainingenv) $
```

- Start Jupyter:

```
(trainingenv)$ jupyter notebook --port=PORT --browser='none'
IntroToPython.ipynb
```

You'll get a link after running the Jupyter command. Copy and paste this into your browser.

Let's Learn Python!

We can switch to the Jupyter notebook now.

Object Oriented Programming

- Python can also implement classes.
- Using classes over functions in Python is generally to the user's discretion, but standard common sense and logic applies as usual. If your program expands to the point where you think an object is more effective than functions, then feel free to implement it.
- No instruction on classes will be given here, as it is beyond the scope of the course. For more information, you can consult an in-depth article like this or look straight at the Python documentation.

If you're working in the Python interpreter (not a Jupyter notebook):

- When you exit the interpreter, you will lose your Python command history.
- If you want save a list of your previous commands, you can use the readline Python interface. For example:

```
>>> import readline
>>> readline.write_history_file('/home/mfk8/python_history.txt')
```

- Once you open a new session, you can import your history:

```
>>> import readline
>>> readline.read_history_file('/home/mfk8/python_history.txt')
```

- For more information, check out the Python docs.

To properly stop your Jupyter server:

- Make sure your notebook has been saved recently!
  - It does autosave, but it may miss some changes.
- Then, you can close the Jupyter notebook browser tab.
- Within your terminal, you can hit CTRL+C to stop the running Jupyter process.

A Brief Introduction to Scripting

Up until now, we've mostly been playing inside the Jupyter notebook. Here, we'll briefly go over what is required to write a proper Python program.

To start:

- Strictly speaking, all you need for a Python program is a text file with the shebang line on top. Recall:

```
#!/usr/bin/env python3
```

- This line indicates to the computer that this is a python program, and it should look in this location to execute. Similar shebangs may look like:

```
#!/usr/bin/python
#!/bin/bash
 #!/usr/bin/perl
 etc.
```

- The shebang is telling the computer to look in the specified directory for the proper method of execution.

Why use env?

- `env` is used for portability. On *nix machines, there is a path `/usr/bin` where most system programs/binaries are installed. If you need to install multiple versions, this can be an issue. Which version of Python do you want to use?
- This is what `env` is for. It tells the computer to look at the current environment, and choose the Python that is currently in use. This is especially helpful on O2, where we have multiple versions installed at the same time.

A basic program:

- Once you've included the shebang, you can get right to it. Start typing lines just as you would in the interpreter, and once you execute your program, each line will resolve itself in order.

```
#!/usr/bin/env python3

print("hello world!")
```

- Type this into a text file, and save it as whatever, and include `.py` at the end for your own convenience.
- To execute this program, just type at the terminal:

```
$ python3 file.py
```

- You've just written your first python program!

Basic structure:

- The previous program is not likely going to be your typical use case. More complex programs may use modules, functions, classes, etc.
- A typical program will have the following structure:

```
#!/usr/bin/env python3

# IMPORT EVERYTHING HERE

# CREATE FUNCTIONS AND/OR CLASSES HERE

 def main():
    # CODE THAT DOES NOT BELONG IN FUNCTIONS GOES HERE

if __name__ == '__main__':
    main()
```

Extending the previous example:

```
#!/usr/bin/env python3

# no modules were required, so none were imported

# no functions were needed, so none were created

def main():
    print("hello world")

if __name__ == '__main__':
    main()
```

What are functions?

- Functions are typically used when you have repetitive code. Instead of pasting the code over and over, just put it in a function, and use ("call") it when needed. For example:

```
def do_thing():
    print("hello world")
    return
```

- Then to reference it, just type elsewhere in your program:

```
...
do_thing()
...
```

- and the program will execute the code within the `do_thing` function.

Functions, cont'd:

- You can also make functions take arguments:

```
def do_thing(phrase):
    print(phrase)
    return

phrase = "hello world"
do_thing(phrase)
```

- You can also assign values to variables with functions. To do this, make use of the `return` keyword. Note that previously, it has been naked, which means nothing is returned. Python is smart enough to know what you mean if you omit `return`, but it's always nice to include it for consistency.

Extending the previous example again:

```
def do_thing(phrase):
    print(phrase)

    new_phrase = "goodbye world"

    return new_phrase

 ...

phrase = "hello world"

phrase2 = do_thing(phrase)
print(phrase2)
```

Putting it all together:

```
#!/usr/bin/env python3

 def do_thing(phrase):
    """Transform the input phrase into a new phrase."""
    print(phrase)
    new_phrase = "goodbye world"

    return new_phrase

 def main():
    phrase = "hello world"
    phrase2 = do_thing(phrase)

    print(phrase2)

if __name__ == '__main__':
    main()
```

Compatibility

- Today's course was taught on 3.8.12.
- There are many versions of Python available, and not all are created equal. Most distinct is the difference between 2.x and 3.x; there are several syntax changes that will be required to use version 3.x.
- 2.x has reached the end of its support life, it is recommended that 3.x is used. More information on 2.x Sunset here.
- For more information, look at this trusty 2.x to 3.x compatibility cheat sheet.

Thanks for coming!

If you have any questions, feel free to email us at rchelp@hms.harvard.edu or visit our website to submit a ticket. We also do consulting!

Please take the course survey!

- Accessible through the Harvard Training Portal
  - https://trainingportal.harvard.edu/

- Click on “Me” then “Intro to Python”
- Scroll to “Evaluations” and click on the survey
- We appreciate any feedback or comments!