Introduction to Python

A HMS Research Computing and DevOps Production

rchelp@hms.harvard.edu

HMS Research Computing Training

- HMS-RC hosts workshops on other subjects like Matlab, Basic/Intermediate O2, etc.

  Current Training Class Schedule and Signup Links

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

- 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 vs. Compiled Languages

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

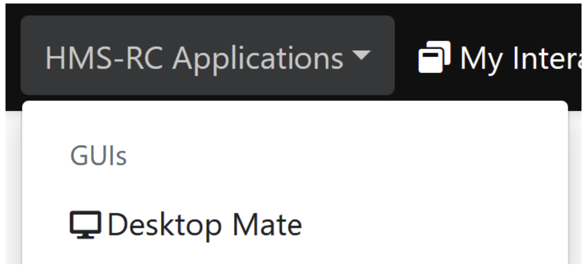
Accessing Python On O2

Accessing Python on O2 – O2Portal Desktop Mate

- Today's class will be held on the O2 cluster via the O2Portal, a GUI interface for the environment. Documentation for the O2 Desktop Mate App can be found on the O2 wiki.
- Navigate to https://o2portal.rc.hms.harvard.edu/pun/sys/dashboard and log in with your HMS ID or training account ID, then click on:

"HMS-RC Applications" -> "Desktop Mate"

At the top of the screen.

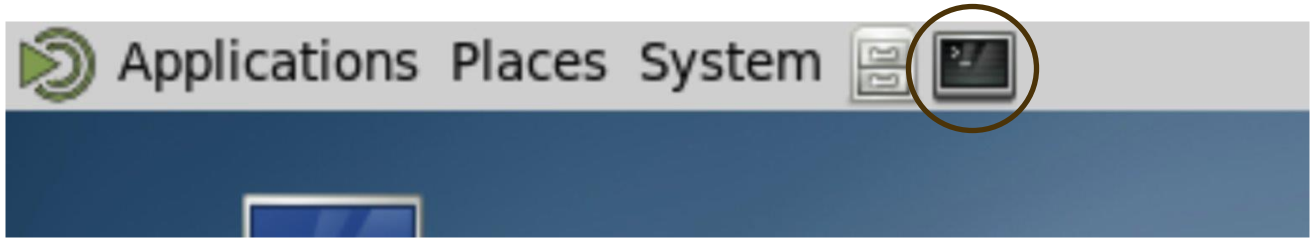


O2Portal Desktop Mate, continued

- Configure your Desktop Mate job with the following settings (if not mentioned, then do not modify):
  - Partition: **priority**
  - Wall Time (requested in hours): change to **2**
- When ready, press the Launch button at the bottom of the form.
- Wait for the job to dispatch, and when it's ready, press the "Launch Desktop Mate" button.

O2Portal Desktop Mate, continued 2

- Once the desktop finishes loading, open a terminal window by clicking here in the top left:



Accessing Python on O2 – Finding Python

```
$ module spider python
       Versions:
          python/2.7.12
          python/3.6.0
          python/3.7.4
          python/3.8.12
          python/3.9.14
          python/3.10.11
```

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

```
$ module spider python/3.9.14
    You will need to load all module(s) on any one of the lines below before the
"python/3.9.14" module is available to load.

      gcc/9.2.0
    (snip)

$ module load gcc/9.2.0 python/3.9.14
$ which python3
/n/app/python/3.9.14/bin/python3
$ python3
Python 3.9.14 (main, Sep 16 2022, 09:43:18)
[GCC 9.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Today's course will use 3.9.14. (To exit the interpreter, type ctrl+D, exit(), or quit().)

Alternatively, virtual environments

```
$ module load gcc/9.2.0 python/3.9.14
 $ which virtualenv
/n/app/python/3.9.14/bin/virtualenv
 $ virtualenv NAMEOFENV --system-site-packages
    (snip)
 $ 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. Refer to environment_setup.txt for how to replicate this environment for yourself.

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 YOU installed via `pip3`. **`pip freeze`** shows those same packages in a requirements.txt format.

```
$ python3 -m pip freeze
distlib==0.3.6
filelock==3.8.0
platformdirs==2.5.2
virtualenv==20.16.5
$ python3 -m pip list
Package                   Version
------------------------- --------
aiohttp                   3.8.4
aiosignal                 1.3.1
anndata                   0.9.1
(snip)
```

Viewing modules continued

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

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

and

```
$ python3
Python 3.9.14 (main, Sep 16 2022, 09:43:18)
[GCC 9.2.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> help('modules')
```

Viewing modules continued, 2

- And it will output something like:

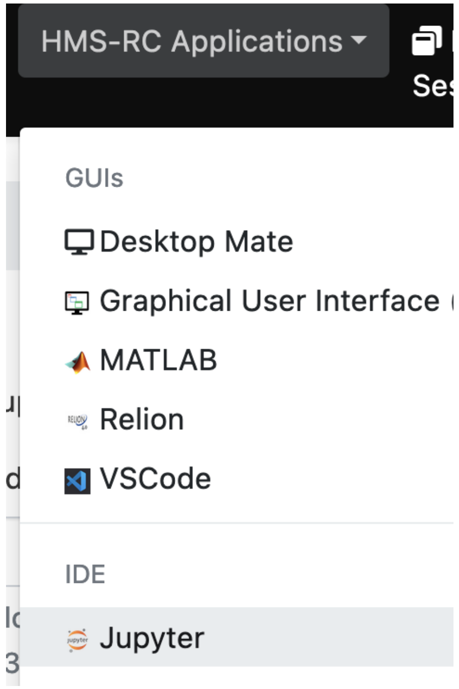
```
Please wait a moment while I gather a list of all available modules...

(there might be some warnings here...)

ANSI                cPickle             ipaddress           rpm
BaseHTTPServer      cProfile            itertools           rpmUtils
Bastion             cStringIO           json                runpy
.
.
.
```

Setup for hands-on portion

- Copy the Jupyter Notebook:

```
$ cp /n/groups/rc-training/python/2023/IntroToPython.ipynb .
```

- Next, open up the O2 Portal Jupyter App via:
    - "HMS-RC Applications" -> "Jupyter"



Accessing Python via Jupyter Notebook

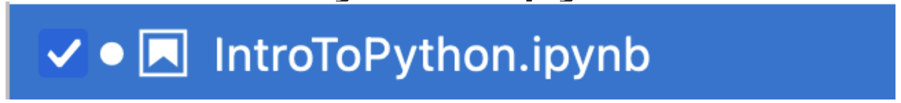
- Today we will be using a Jupyter Notebook for our hands-on portion, though you don't need to use Jupyter to run Python code.
- There are many advantages to using a Jupyter notebook:
  - All your code is saved in one place
  - You can share your notebook with others
  - Can create interactive plots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

O2 Portal Jupyter App

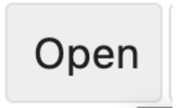
- Configure your Jupyter job with the following settings (if not mentioned, then do not modify):
  - Modules to be preloaded: **gcc/9.2.0 python/3.9.14**
  - Partition: **priority**
  - Wall Time (requested in hours): change to **2**
  - Jupyter environment: **source /n/groups/rc-training/python/env/trainingvenv/bin/activate**
  - Check the box next to "Enable JupyterLab"
- When ready, press the Launch button at the bottom of the form.
- Wait for the job to dispatch, and when it's ready, press the "Connect to Jupyter" button.

O2 Portal Jupyter App, cont.

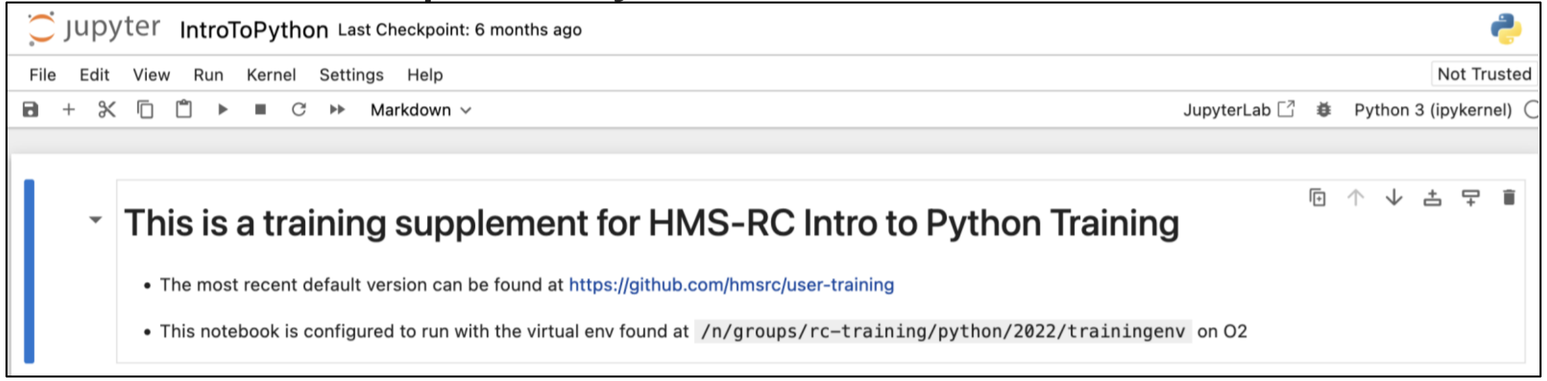
- You can open the Jupyter Notebook we copied in a previous step.
  - Select "IntroToPython.ipynb" in the file list.



  - Click the "Open" button in the top left corner.



  - A new tab will open in your browser for the notebook:


jupyter
IntroToPython Last Checkpoint: 6 months ago
File Edit View Run Kernel Settings Help
Not Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

This is a training supplement for HMS-RC Intro to Python Training

- The most recent default version can be found at https://github.com/hmsrc/user-training
- This notebook is configured to run with the virtual env found at `/n/groups/rc-training/python/2022/trainingenv` on O2

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 on Object Oriented Programming 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')
```

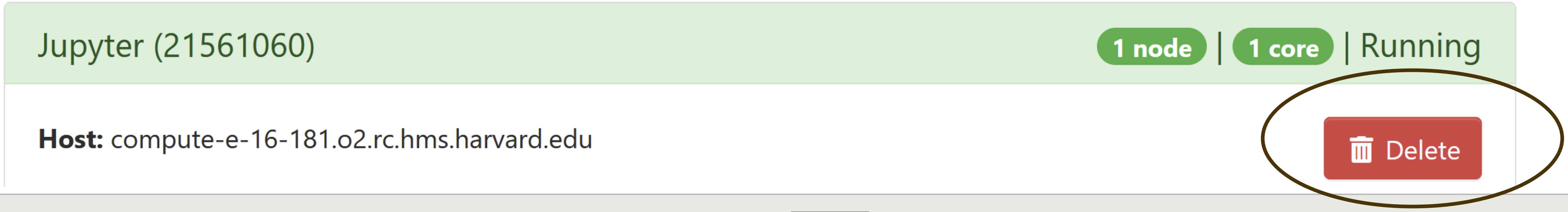
- Once you open a new session, you can import your history:

```
>>> 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.
- In O2Portal, you can also click the Delete button associated with the running server.


Jupyter (21561060)

Host: compute-e-16-181.o2.rc.hms.harvard.edu

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.

Putting it all together:

```python
#!/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()
```

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)
```

Compatibility

- Today's course was taught on 3.9.14.
- 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 attending!

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!

- Use the link:

https://hms.az1.qualtrics.com/jfe/form/SV_02LPnmjTaLKkFdY

- Or scan the QR code:

- We appreciate any feedback or comments!