Introduction to Python

Goals:

- Introduction to python and programming in general terms
- Introduction to the python interpreter
- Introduction to conda
- Setting up jupter notebook on the HPC
- Survey of terms and data types in python
- Operators in python
- Lists, dictionaries, sets
- Libraries and modules

What is programming?

Programming is the processes of writing a set of instructions (**program**) to tell a computer to carry out a process. The writing of the program can occur in one of any number of different languages– however many of the key concepts are consistent.

General types of programs

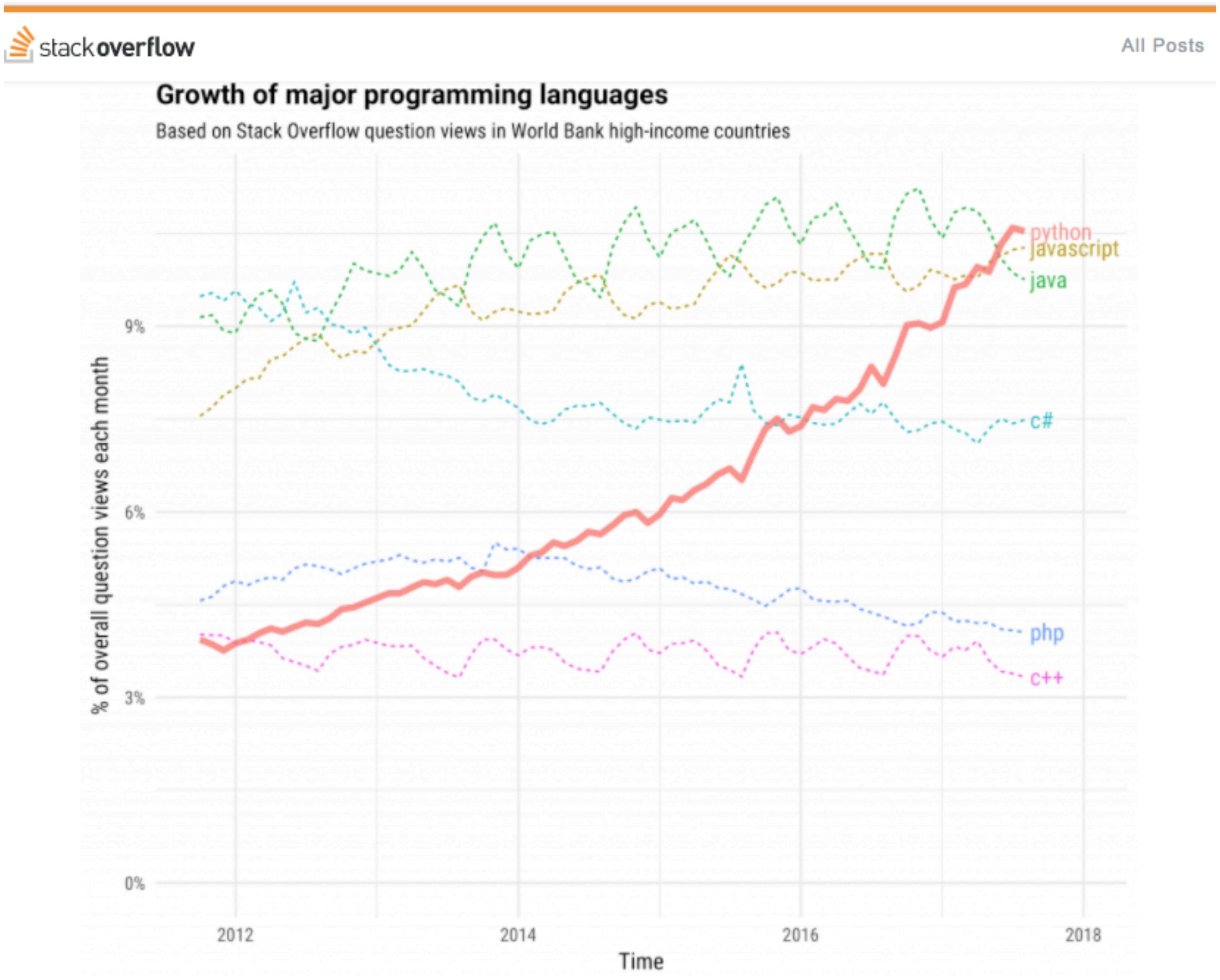
Some programming languages (e.g. C, C++) need to be **compiled**. This process takes a human readable code (source code) and turns it into less human readable (or not readable) set of instructions that can be carried out by a computer. Once compiled on a computer the code should be able to be re-run as much as you want. This is how *most* computer programs that you interactive with are run.

> Benefits: Specific to computer,
> Disadvantage: sometimes slower, less optimized

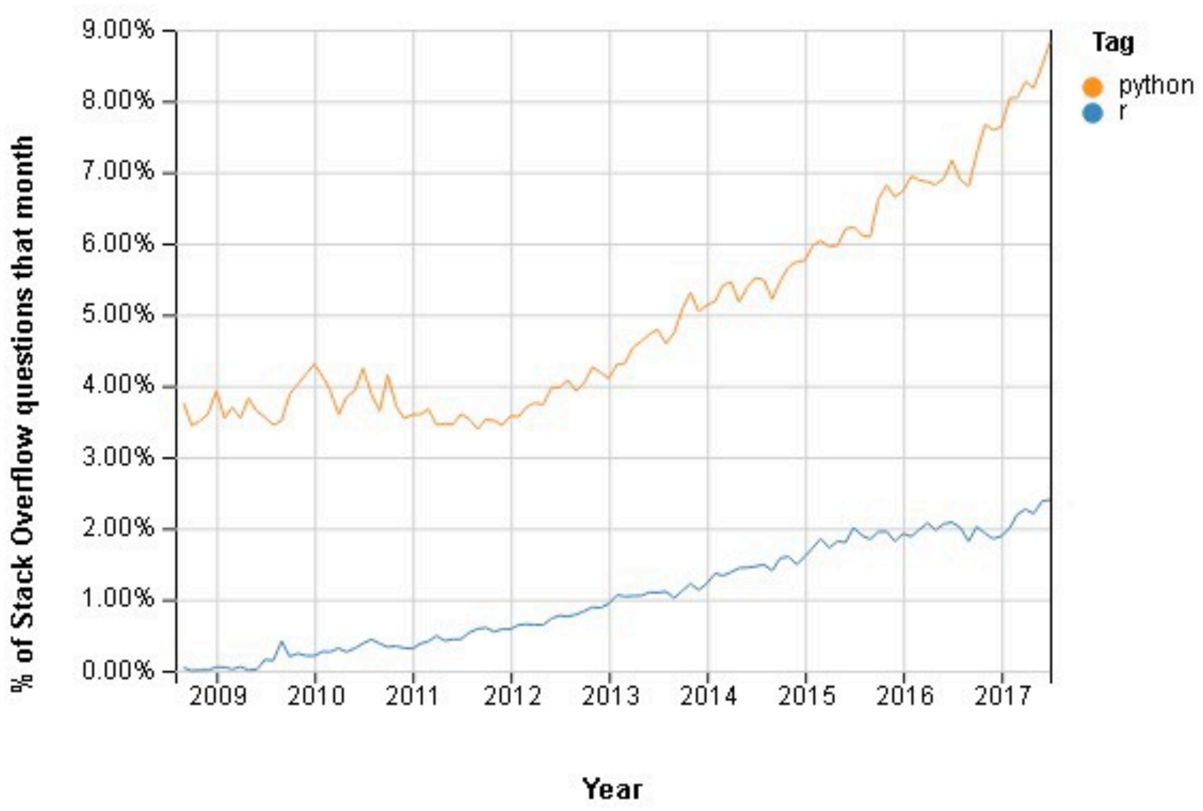
Other programming languages do not need to be compiled but rather are processed with an **interpreter**. These languages are called scripting languages (e.g. bash, Python, Matlab, R, etc.). Scripts or programs written in these languages require a program to run.

> Benefits: Modifiable, oprtable, no need for compiling
> Disadvantage: sometimes slower, less optimized

Why Python?



From https://hackernoon.com/top-3-most-popular-programming-languages-in-2018-and-their-annual-salaries-51b4a7354e06



We have selected to work with **Python** for this course. Honestly, for science you can use whatever suits your work. Popular choices include:

- Python
- Perl
- R
- Matlab
- C/C++

Each has its merits and draw backs, but the nice thing is that once you master one language it is much easier to learn the next – as the same general principles tend to apply. Many like teaching Python as a first language as it has a simple syntax and is fairly easy to read. What is more, Python is a fully functional language that is able to run complex tasks.

Python is:

- Interactive
- Interpreted
- Portable
- Modular
- Object-oriented
- Open access
- Free
- Widely used

Python has:

- An extensive (scientific) user base
- A large and varied set of support libraries
- Good system of scalable tools (e.g. Pangeo)
- Good scientific notebook style interactive interface (Jupyter)

Running Python

First things first, let's all *clone* a copy of the Repo we will be using for this part of the class. Go here: https://github.com/2019-MIT-Environmental-Bioinformatics/Lab-Python.

>

Running the Python interpreter

Python code can be run in many ways. You can run code directly within the python interpreter. To directly open the python interpreter you can type `python`.

> You should see that your `prompt` has changed. Try typing `ls`. What happens? Why?

In `python` you can run basic commands such as basic math:

```
>> 2 + 3

>> 4**12
```

or printing:

```
>> print('Hello World.')
```

Running the Python interpreter

Now, let's exit the `python` environment and try running a `python` script. To do this (just as with our bash script) we must specify the **interpreter**:

```
python factorial.py
```

Now, let's just take a quick look at the contents of this program. Note that the *shebang* at the top of this script is different from the one used by bash. Here we are specifying that this program should be run with python 3.

> As an aside, python recently went through a major transition changing from `python 2.7` to `python 3`. While you may run into some program written in `2.7` you should try to do all of your analyses in `3`. To figure out which version you are running type `python --version` or `which python`.

Jupyter notebooks

For the rest of this class we will be running all of our code in `jupyter notebooks`. The main reason that I like working within in `jupyter notebooks` is that they are *interactive* and *easily readable*. Notably, code and output graphics or answers are tightly linked together rather like a laboratory notebook.

First, let's open up a new `tmux` session. You can name `tmux` sessions with the command:

```
tmux new -s lab
```

Recall that the magic key for `tmux` is `ctrl+B`. Here are some useful tmux commands .

Anaconda

To run and manage `jupyter notebooks` we will be using `anaconda`. Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing. Through the function `conda` Anaconda facilitates the installation of python and R programming languages as well as packages. It is a great tool for ensuring that computational environment is associated with code and really streamlines program installation and use.

The first thing we are going to do is load anaconda on to our compute environment on the HPC. To do this we will use the command:

```
module load anaconda
```

> The `module load` is a common aspect of HPC environments. Try typing `module avail`. This prints all the available programs that you can load on the HPC. Many of the programming languages we already mentioned are listed. Type `module list` to see what programs are loaded in your environment.

Now that we have loaded anaconda the first thing we are going to do is create a new `conda environment`. A `conda environment` is a type of virtual environment. Virtual environments helps to keep dependencies required by different projects separate by creating isolated spaces for them that contain per-project dependencies for them. Basically, you can think of it as having a special room (environment) within your house (computer) where you do one specific activity. For example, in the kitchen you cook, in the bed room you sleep. Only with virtual environments you can be even more specific. It is good practice to associate environments with computational projects as you are able to specify things like program versions within them. Hypothetically, with conda you should be able to hand a friend an `environment.yaml` file and all your code and data and they should be able to run everything. It is pretty transformative, really.

You can read more about `conda` environments here.

For now, let's make our first conda environment. I have provided a yaml file that contains the packages we might available during our lab. Let's take a quick look at it. `less lab.yaml`. Now type or copy:

```
conda env create -f lab.yaml
```

This is going to create a conda environment called `python-lab`. Whenever we activate `python-lab` it will take us into that special room on our computer where all these programs are installed with specific versions. To enter it type:

```
conda activate python_lab
```

> What happened to your prompt?

Starting Jupyter Notebook

Now, we are reading to start a `jupyter notebook`! Jupyter notebooks run within a web browser and act as a GUI interface of sorts to run python code. Yet, they are saved as convenient chunks of code (ending in `.ipynb`) that can be opened again, re-run, shared, and modified.

If you were working on your local computer you could simply type `jupyter notebook` and a notebook would open up. However, as we are trying to run this on a remote computer we will need to specify a bit more. First, we are going to set a password for our `jupyter notebooks`. Jupyter notebooks basically create a port into whatever computer you are using (especially if it is a remote machine). You only have to do this once (or whenever you want to change you password).

```
jupyter notebook password
```

Now, let's open jupyter notebook!

```
ssh -N -f -L localhost:8888:localhost:8888 USERNAME@poseidon-[l1 or l2].whoi.edu
```

This should prompt you for your password. You can enter it and then hit enter. Once you do that, you are ready to open your browser of choice and type `localhost:8888` into the prompt. Now, enter your jupter notebook password. Hit enter– and you should be ready to go!

Let's take a quick tour of Jupyter!

Terms and data types

General terminology

Term	Definition
Arguments	Values given to a program when it is run
Code	Program or or portion of program; Act of writing a program
Execute	To begin to run a program (see also: run)

Function	A subprogram that can be called to run the same task
Parameters	Values given to a function
Return	The act of sending back a value as part of a function
Variable	A name that holds a value

Basic data types

Type	**Example**
Integer	A whole number; e.g. 85, 0
Float	Any number (scientific, decimal); e.g 3.14, 4.2e-10
Boolean	Binary True/False
String	A collection of text characters (numbers, letters, etc.); e.g. “Homo sapiens”, “33”

Mathematical Operators

Symbol	**Example**
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`**`	Power/exponent

`%`	Modulo
`//`	Truncated division (without remainder)

Comparative and Logical Operators

Symbol	Example
`==`	Addition
`!=`	Subtraction
`>`, `>=`	Multiplication
`<`, `<=`	Division
`and` `&`, ``	Power/exponent
``	Modulo
`and`, `&`	And
`or`, `	`
`not`, `!`	NOT

Variables as containers of many things

Variables at their most basic

Unlike some other programming languages, you do not need to specify what a variable is going to be. It can honestly be anything and will take on anything. Any data type can be assigned to a variable with the `=`. For example:

```
my_name = 'Harriet'
blue = 'red'
apple = 5
```

You can `print` the value of a variable with the command `print()`:

```
print(my_name)
```

Variables must be created before they are used. If for example I wanted to print a variable called `elephant` I would need to initiate it first.

> What happens if you try to print a variable that hasn’t been set up yet?

What is more– variables persist between calls (until they are actively changed by assigning a new value). So, when you set a variable in one cell it is going to be the same further down.

Variables can also be used in any calculation you want. For example,

```
favorite_number = 24
favorite_number_squared = favorite_number ** 2
```

Also, variables that contain strings can be indexed and sliced to grab particular parts. Let’s make a long string:

```
bronte = 'Whatever our souls are made of, his and mine are the same.'
bronte[2]
bronte[3:20]
```

What is more, we can search in strings using our good, old regular expressions. To do this we will import a **library** or **package** into python. Python has a lot of utility on its own– however not everything is automatically available. Most of the coolest functionality are compartmentalized into packages.

The first one we will try out is called `re`. In a cell type: `import re`. You will now be able to use all the functionality within that function. You can call functions (tools) within this package using the dot. For example we can search for the regular expression `'\s[A-z]+a[rmd]e'` in `bronte`.

```
re.findall('\s[A-z]+a[rmd]e', bronte)
```

Try typing `re.` and hitting tab– you will be able to see the different tools in this package.

Variables with Arrays and Lists

Lists

An `array` is a collection of data that is called by a single variable name.

A `list` is a 1D array that contains a series of values. List variables are declared by using brackets `[ ]` following the variable name. Values do not need to be of the same type (i.e. you can have a mixture of strings, integers, floats, and booleans). You can also have `list` contained within a `list`.

Let's try making a `list` that contains your favorite number, letter, and fruit:

```
Favorite=[42, 'Q', 'kumquats']
```

`arrays` and `lists` can be indexed – meaning that you can ask `python` to return only the value at one particular location. As such, `lists` and `arrays` are necessarily ordered. Python is zero indexed (meaning that counting starts at zero rather than 1).

So, if I wanted to return the 2nd element of `Favorites`:

```
Favorites[1]
 >> 'Q'
```

Arrays

Generally, I do not recommend working with arrays for anything other than numeric data. One of the options for working with numeric data is the package `numpy` (Numerical Python… there is debate about pronunciation). Let's import `numpy`. In a cell type:

```
import numpy as np
```

Unlike `re` numpy is kind of long. For longer named packages it is common practice to import them to a shorthand name with the word `as`. Here we are importing `numpy` as a variable named `np`. If you then type `np` it will automatically pull up numpy.

Let's create a matrix!

```
x_lists = [[1,2],[3,4]]
arr = np.array(x)
```

```
x_lists = [[1,2],[3,4]]
arr = np.array(x)
```

I am not going to go into the matrix math application of python here– but if you are interested I recommend checking out `numpy` and `xarray`. More here.

You can also read in data from a file like a csv. For example:

```
data = np.loadtxt('data/random.csv',delimiter=',')
```

Useful customization for after class

There is a lot of typing involved in getting jupyter running on the HPC. I recommend that you add the following bash function to your `.bash_profile` to help speed things along.

```
 jpt(){
   # Fires-up a Jupyter notebook by supplying a specific port
   jupyter notebook --no-browser --port=$1
}
```

As you can see the command `jpt` takes one user input $1 which specifies the port number that you want to open your `jupyter notebook` in.