Intro to python™

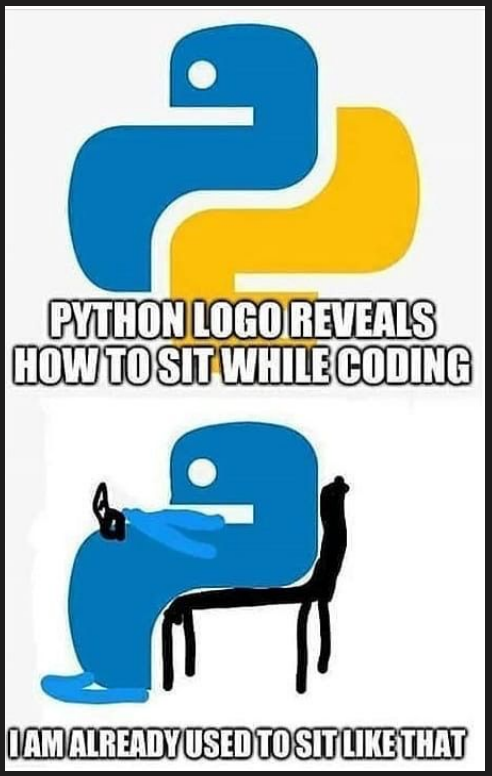
Summer Particle Astrophysics Workshop

May 9th, 2022

Hannah Fronenberg

hannah.fronenberg@mail.mcgill.ca

Arthur B. McDonald
Canadian Astroparticle Physics Research Institute

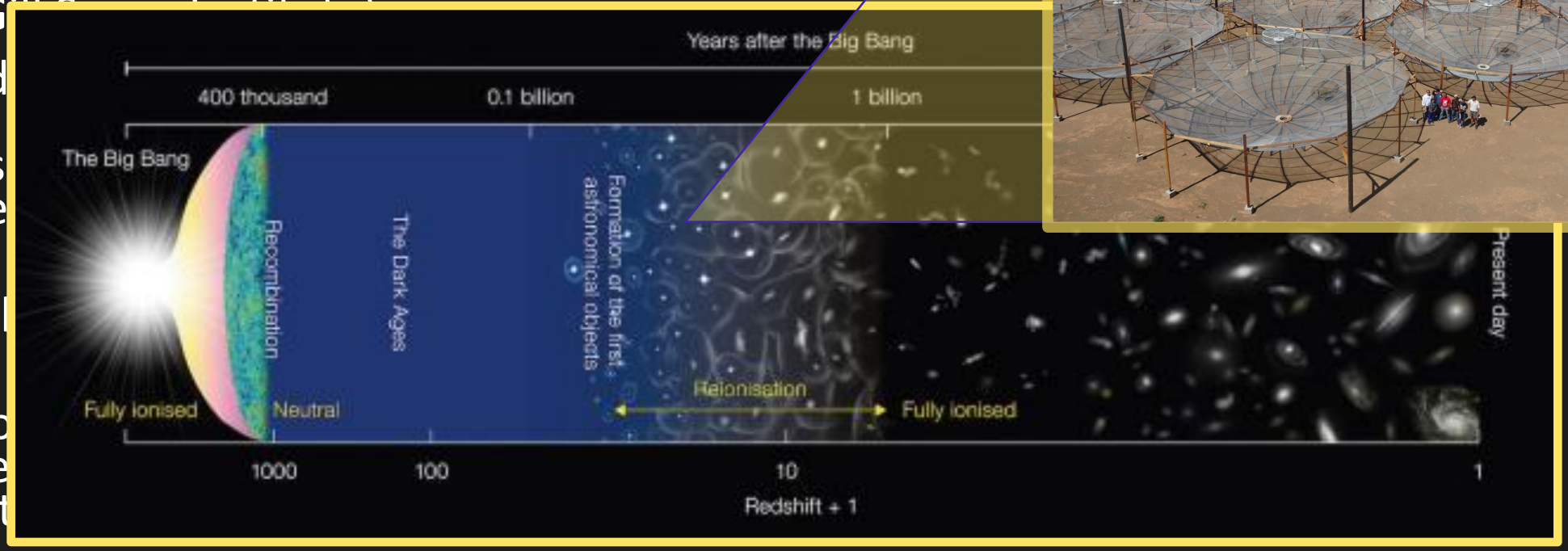


A bit about me

- Undergrad at U of T and MSc at McGill
- Currently a PhD student at McGill University (McGill Space Institute)
- I study cosmology where I try to learn about large structure formation, to precisely measure the parameters that govern our universe's evolution, and sometimes I like to think about weirder things like cosmic strings!
- I work with 21 cm, line intensity mapping, CMB observations
- I also love teaching and interacting with students so I'm really excited to spend the next two hours with you all.
- Apart from physics I love all things music and over the pandemic I also learned to knit.
- Fun fact: I have never taken a computer science course!

A bit about me

- Undergrad at U of T and MSc at McGill
- Currently a PhD student at McGill University (McG
- I stud
  large
  meas
  unive
  think
- I worl
  CMB
- I also
  stude
  next t
- Apart from physics I love all things music and over the pandemic I also learned to knit.
- Fun fact: I have never taken a computer science course!



Plan For Today

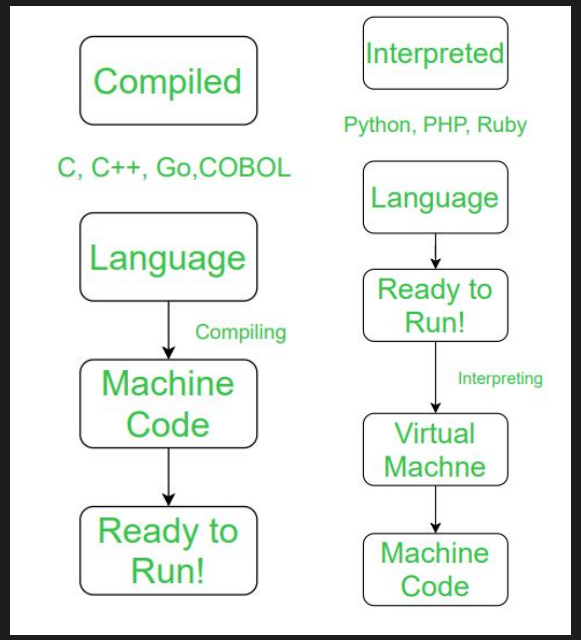
- What is python
- Installation and creating environments (you don't need to install it to participate in the activities today)
- Basic math and arithmetic with python
- Python objects and their functions
  - Strings, floats, lists, array, dictionaries
- Flow control
  - for, if, else, while
- Functions, Modules, and Libraries
- If time: debugging and test code

Plan For Today

- What is python
- Installation and creating environments
- Basic math and arithmetic with python
- Python objects and their functions
  - Strings, floats, lists, array, dictionaries
- Flow control
  - for, if, else, while
- Functions, Modules and Libraries
- If time: debugging and test code

What is python™

- Python is a programming language
- It is an interpreted language meaning it is not directly compiled into machine instructions. Python code is instead read and executed by some other program (ex. CPython, Jython, PyPy…)
- It is an object oriented language which menas that it organizes software design around data, or objects, rather than functions and logic.
  - Def object: a data field that has unique attributes and behavior.



What is 

Pros	Cons
Versatile and easy to use	Not as fast as other languages
Fast to develop	Memory can be an issue
Lots of libraries to simplify your life	Not easy to parallelize and do threaded computation
Easy to add non-python extension (ex. Numpy is a library for really efficient FORTRAN routines)	Not good for mobile development
Comes built in with Linux and macOS	
Open source with a big community of users	
It writes “nice” code with not as much syntax as other languages and is easy to read	

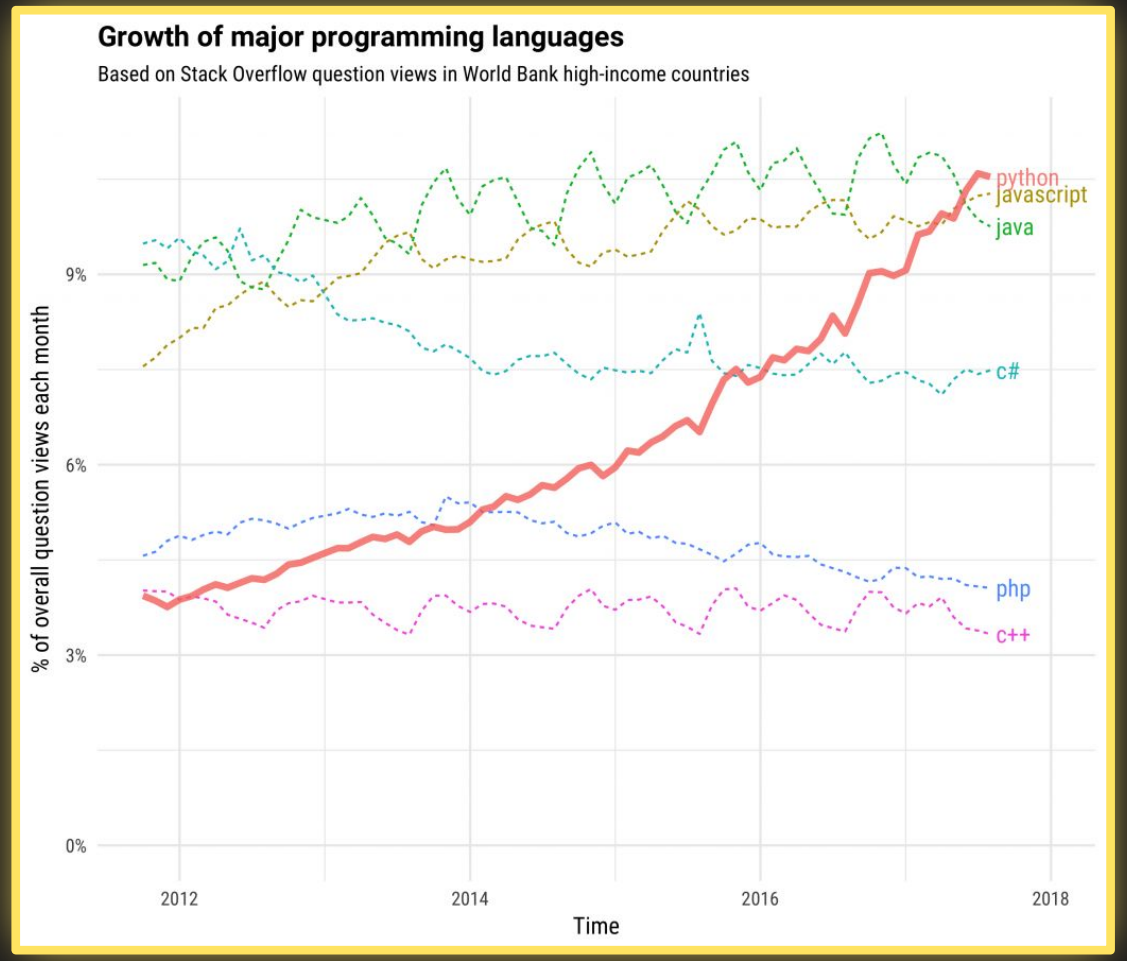
What do you use Python for?

> 100%

Main | Secondary | Combined

- 2020
- 2019

55%/59%	**Data analysis**
50%/51%	Web development
40%/40%	Machine learning
38%/39%	DevOps / System administration / Writing automation scripts
36%/37%	Programming of web parsers / scrapers / crawlers
29%/31%	Software testing / Writing automated tests
27%/26%	Educational purposes
23%/25%	Software prototyping
19%/18%	Desktop development
19%/21%	Network programming
13%/14%	Computer graphics
9%/7%	Game development
8%/8%	Embedded development
7%/6%	Mobile development
5%/4%	Multimedia applications development
7%/6%	Other



How can I use  ?

- Interpreter vs Engine
  - Python as an interpreter allows you to run python code on the command line just like bash. By typing the command:

```
$ python3
```

    you can write and run python code in a python shell. (I almost never use this except sometimes as a calculator)

  - Python as an engine allows you to write and run python scripts in files with the file extension .py (this is what I use every day!). To do this make sure you are in a shell that has python running and run your code by typing:

```
$ python my_python_code.py
```

Demo: Interpreter vs Engine

Follow along in terminal or on the colab notebook in the empty cells labelled `#pretend command line`

Plan For Today

- What is python
- Installation and creating environments
- Basic math and arithmetic with python
- Python objects and their functions
  - Strings, floats, lists, array, dictionaries
- Flow control
  - for, if, else, while
- Functions, Modules and Libraries
- If time: debugging and test code

What do you need to install?

- For today, nothing! But in general, you will need to install python 3 if your operating system doesn't have it pre-installed. You can do so here:

  Python download: https://www.python.org/downloads/

- You probably also want to use a package manager and installer. What this does is manage the installation, storage, and updating of all your python packages. Some also allow you to create virtual environments (which we will talk about soon). Using a package manager will help reduce issues that come with package dependencies and versions.
  - Download Anaconda: https://www.anaconda.com/
  - Download Miniconda: https://docs.conda.io/en/latest/miniconda.html

Versions

"Online I see some stuff in python2 and some stuff in python3, does it matter which one I use?"

YES

These two versions are mildly incompatible:

python2	python3
print a , print "hello world"	print(a), print("hello world")
7/2 = 3 (integer division)	7/2 = 3.5 (floating point division) 7//2 = 3 (integer division)

Rule #1: We always use python3

Rule#2: We never ever ever ever use python2 (unless you're using someone's old code that they rudely did not update for you, but it's okay because at least they optimized it and it runs really fast)

You can check which version you have by typing `$ python -V` into the command line

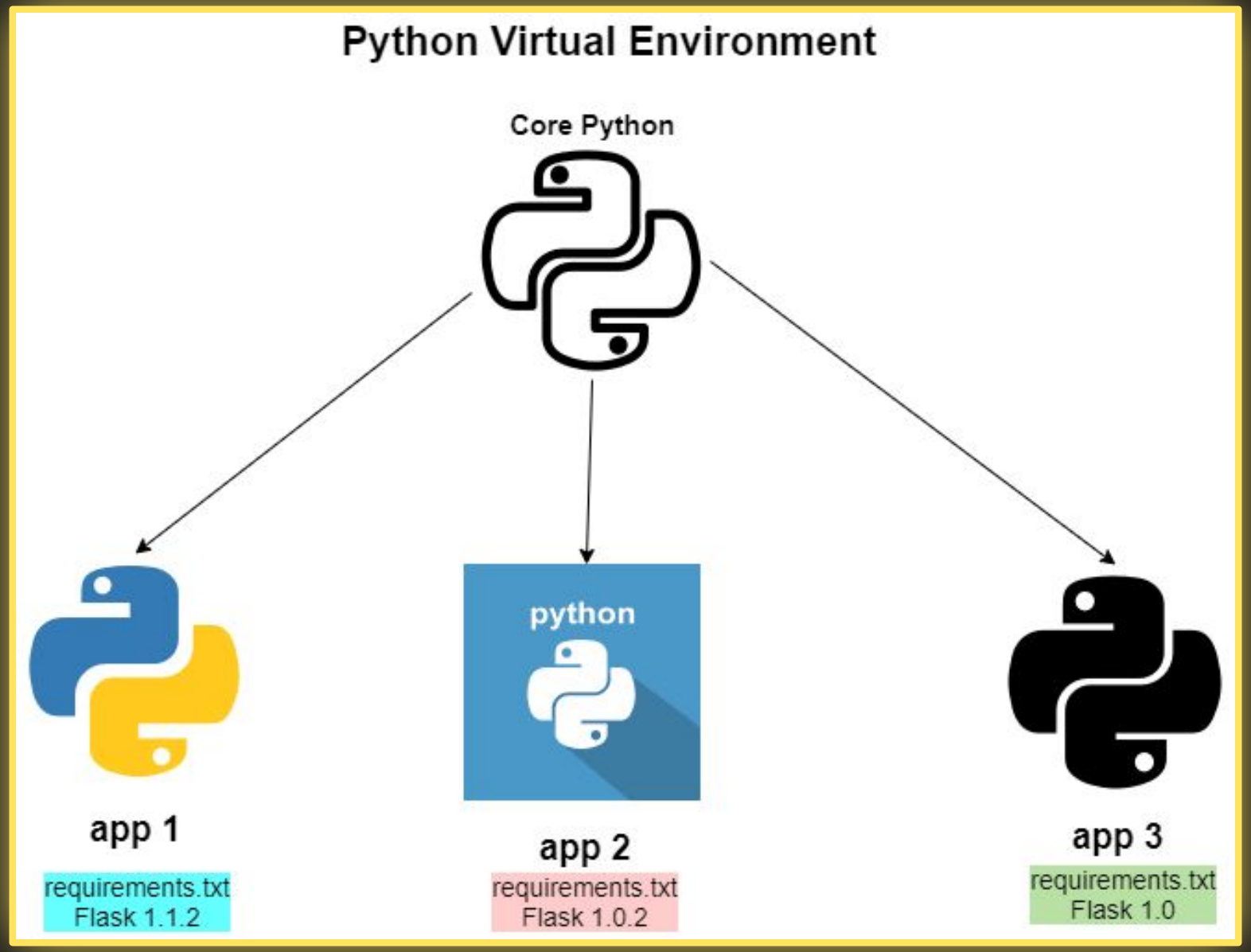
Virtual Environments: What are they and why we need them

A virtual environment a folder structure that allows you to run Python in a lightweight and isolated environment.

- Ex. You can have a different environment for each project with only the packages that you use for that specific project installed.

Why would you want to use one exactly?

- Avoid pollution: If you install packages to your operating system's global Python, these packages will mix with the system-relevant packages (this has happened to me…)
- Avoid system conflicts: If you update your operating system, then the packages you installed might get overwritten and lost (this has happened to me…)
- Avoid dependency conflicts: You might want different versions of things for different codes and you can't have multiple versions installed in the same place.
- Keep track of dependencies for a project: If you want to make your code publicly available then you need to be able to tell people what packages your code uses.
- No administrator lockouts during installation: Usually don't need full system access to install things in an environment.


Python Virtual Environment
Core Python
python
app 1
requirements.txt
Flask 1.1.2
app 2
requirements.txt
Flask 1.0.2
app 3
requirements.txt
Flask 1.0

My python™ Set-Up

- OS: MacOS
- Terminal: Bash
- Python Version: Python 3.9.7
- Package Manager/Installer: Miniconda
- Text editor: Sublime Text (locally), nano (remote server)
- Random tools: Jupyter notebook, nb-extensions, virtual environments

Demo: Installing Python Packages and making environments

Follow along in terminal (sadly not in colab because everything there is already installed)

Plan For Today

- What is python
- Installation and creating environments
- Basic math and arithmetic with python
- Python objects and their functions
  - Strings, floats, lists, array, dictionaries
- Flow control
  - for, if, else, while
- Functions, Modules, and Libraries
- If time: debugging and test code

Basic Operations in python™

Operation	Symbol	Example
Addition	+	5+5 = 10
Subtraction	-	6-5 = 1
Multiplication	*	3*7 = 21
Division	/	7/2 = 3.5
Integer Division	//	7//2 = 3
Exponentiation	**	2**2 = 4
Imaginary numbers	j	3+3j
Modulo (remainder)	%	7%2 = 1
Powers of 10 (orders of magnitude)	e"number"	$3x10^9$ = 3e9
Assigning a variable	=	a = 4

A few extra things

- You can add numbers to a variable by performing an “augmented assignment” like this

  a += 3 is the same as a = a+3

  - Note that the original a you defined has now been overwritten, a no longer has its original value. I use this a lot for unit conversions!
  - You can do the same thing with -= , *= , /=
- Python numbers are “BigNum” meaning that there is no maximum value or rollover. Numbers will just get as big as your computer’s memory will allow.
- Sometimes you also see numbers that should be 0 approximated as a really really small number.

  Ex. $\sqrt{}(-2) = 0 + \sqrt{}2j$ but Python outputs (8.65956056e17+1.41421356237j)

Exploration: Try doing some math!

You can try this out in a python shell on the command line or on colab!

Plan For Today

- What is python
- Installation and creating environments
- Basic math and arithmetic with python
- Python objects and their functions
  - Strings, floats, lists, array, dictionaries
- Flow control
  - for, if, else, while
- Functions, Modules, and Libraries
- If time: debugging and test code

Objects

- In Python EVERYTHING is an object
- Since everything is an object, you can pass anything to a function. This is not true of other languages. This is sometimes referred to as **duck typing.**

Objects in python:

  - Strings
  - Floats
  - Lists
  - Arrays
  - Tuples
  - Functions (methods)

- There are also things called classes which are kind of like object constructors. They do take in arguments, like a function, but they themselves don’t produce any output per se.

Typing in 

- Python has **dynamic typing** as opposed to static typing. This means you do not need to specify what type of object something is. Python is smart. Python knows.

Python	C
a = 5.00123	`float a[= 5.00123];`

- Python has **strong typing** as opposed to weak typing. This means that if you want to change the type of object something is, you need to do so explicitly. Here, Python is not smart, it doesn't know.

Ex. `a = "3"  float(a) = 3.0`

Strings and Floats

Strings

- A string is a single character or set of characters (or glyphs)
- They are denoted using “ ” or ‘ ‘ (it doesn’t matter which you use)
- They are
  - Immutable
  - Homogeneous
  - Ordered

Ex. `a = “I love physics”`

Floats

- A set of numbers that represent the actual numerical value of the characters, not just the glyph.
- They are denoted just using regular numbers.
- They are not integers

Ex. `b = 3.5`

Lists, Arrays, and Tuples

- These are all data structures that can used to store multiple strings/floats.
- These use indices to label their elements:

Lists	Numpy arrays	Tuples
`my_list=['a',4,'hannah',32.0]`	`my_arr = np.array([1,2,3])`	my_tup = ('1',33,96)
Ordered	Ordered	Ordered
Mutable	Mutable	Immutable
Heterogeneous	Technically heterogeneous (but you need to communicate that)	Heterogeneous

Dictionaries

- Dictionaries also store data like lists/arrays/tuples but they store *pairs* of data using a *key:value* structure.
- These are
  - Unordered (the key is the index instead)
  - Heterogeneous
  - Mutable

```
my_dict = {“my name”: ”hannah”, “my age”: 24, ”my school”: “McGill”}
```

Activities: Let’s get comfortable with manipulating these objects!

You can try this out in a python shell on the command line or on colab!

Plan For Today

- What is python
- Installation and creating environments
- Basic math and arithmetic with python
- Python objects and their functions
  - Strings, floats, lists, array, dictionaries
- Flow control
  - for, if, else, while
- Function, Modules, and Libraries
- If time: debugging and test code

Flow Control

- You can define a set of computations using loops
  - If/elif/else : performs a computation if the condition is satisfied
  - While: repeats the computation until the condition is false
  - For: performs a computation for all the indices defined in the sequence
- With if and while statements, you quantify their conditions using these symbols:
-

Condition	Symbol	Example
equal to	==	`if x == 8:`
Greater than/ Less that	> / <	`while x > 5:`
Greater than or equal to	>=	`if x >= 10:`
Not equal to	!= or <>	`if x != 2:`

- Continue will skip to the next loop iteration.
- Break will break out of the innermost loop.

Flow control: Indentation

- It is important to organize your code in a block
- Blocks are typically indented with 4 spaces (1 tab in most setups).
- Most importantly, your indentation needs to be consistent across your whole script.
- You can have unlimited loops in loops and all you need to do is keep indenting.
- It is common to run into indentation errors! You can use your text editor to help you avoid these.

Activity: Using a loop

You can try this out in a python shell on the command line or on colab!

Plan For Today

- What is python
- Installation and creating environments
- Basic math and arithmetic with python
- Python objects and their functions
  - Strings, floats, lists, array, dictionaries
- Flow control
  - for, if, else, while
- Functions, Modules, and Libraries
- If time: debugging and test code

Functions (and methods) in 

- Functions allow us to nicely package tasks into something that is callable and interchangeable with different variables.
- They are defined like this:

```
def my_function(variables):
    stuff to compute goes here
    return result
```

- You can also save functions in .py files and import them into another script.
- Here are some MAGICAL lines for auto-importing into a jupyter notebook which I use all the time:

```
%load_ext autoreload
%autoreload 2
```

Libraries

- Python libraries are large sets of pre-defined functions (but here we all them modules) that are available for us to use.
- My favourites are:
  - Numpy
  - Scipy
  - Matplotlib
  - OS
  - Sys
  - Astropy

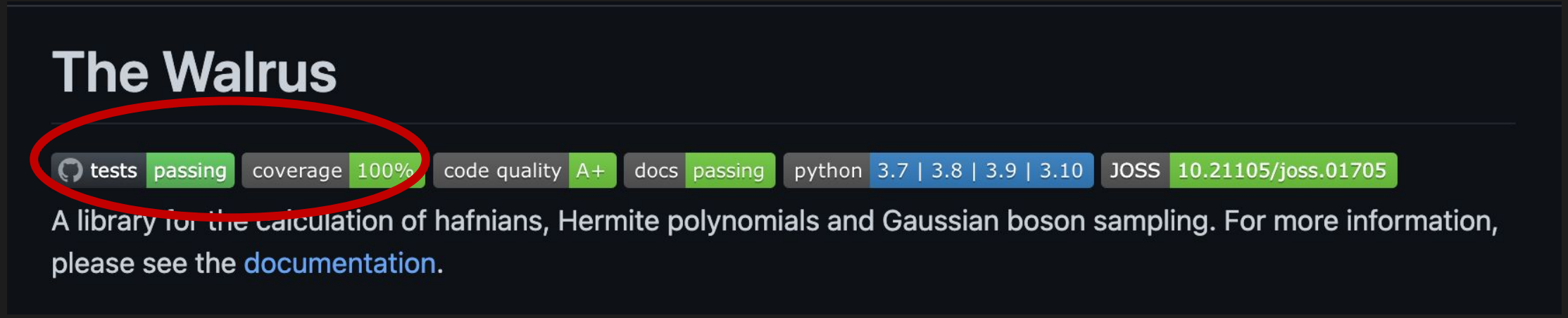
Activity: Building a module and using libraries

You can try this out in a jupyter notebook or on colab!

Plan For Today

- What is python
- Installation and creating environments
- Basic math and arithmetic with python
- Python objects and their functions
  - Strings, floats, lists, array, dictionaries
- Flow control
  - for, if, else, while
- Functions, Modules, and Libraries
- If time: debugging and test code

What does it mean that a code is covered?



If a project has X% coverage, that means that X% of the lines, functions and/or branches of the files included in your tests are covered by tests.

But this doesn't mean that your code is safe from all bugs…

Unit testing in 

- Unit testing is a method that allows you to test individual units of code with program specific tests.
- In ”real” world coding, every program has to be tested. Getting in the habit of writing test code will help you level-up your programming and help you spend less time debugging code.
- In python the module that I use to do this is called nosetests and here are some basic examples of things to test:
  - Check that arrays are the right shape
  - If something has to be a particular value, check that
  - Check the data type of something

Happy coding and thanks everyone!