---------- **Administrivia**

Introduce myself.

Class goals.

Class processes.

Grading.

---------- **Textbooks, cheat sheets, etc.**

pdf files.

---------- **Configure student workstations**

There may be little or a lot to do.

---------- **Introduction to Python**

History of Python.

Python 2 vs Python 3.

2to3 - Automated Python 2 to 3 code translation
(docs.python.org/2/library/2to3.html)

Python basic modes – script and interactive.

Python basic data types.

Demo. Student hands on activities.

dir(abc)

entries in all caps are constants
capitalized items are classes
items that start with lowercase letters are methods
entries starting with 1 or 2 underscores are private entries

Everything in Python is an object.

```
$ python3
Python 3.5.3 (default, Sep 14 2017, 22:58:41)
[GCC 6.3.0 20170406] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> type(123)
>>> dir(123)
```

```
>>> type(456.7)
>>> dir(456.7)
>>> type('abc')
<class 'str'>
>>> dir('abc')
>>> x = 123
>>> type(x)
>>> x

$ python2
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170118] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> type(123)
<type 'int'>
>>> x = 123
>>> x
>>> y = long(123)
>>> y
>>> type(y)

>>> print "hello World!"
>>> print 'Hello World!'
```

---------- Integer Max?

```
python 2.7
>>> import sys
>>> n = sys.maxint
>>> type(n)
.....
>>> n += 1
>>> type(n)
......

python3
>>> import sys
>>> sys.int_info
.....
```

---------- Float Max

```
import sys
sys.float_info
import numpy as np
np.finfo(np.float128)
np.finfo(np.float64)
```

---------- **sort vs sorted**

All we do is call "sort" on the list, for in-place sorting, or the built in function "sorted" for not modifying the original list and returning a new sorted list. Both functions take in the same arguments and can be treated as "the same" for our purposes here, except for the reason above.

https://www.pythoncentral.io/how-to-sort-a-list-tuple-or-object-with-sorted-in-python/

---------- **First Script**

Create first script - "hello world"

---------- **Major data types**

http://www.diveintopython3.net/native-datatypes.html

1. Booleans are either True or False.
2. Numbers can be
   integers (1 and 2)
   floats (1.1 and 1.2)
   fractions (1/2 and 2/3)
   complex numbers
3. Strings are sequences of Unicode characters, e.g. an HTML document.
4. Bytes and byte arrays, e.g. a JPEG image file.
5. Lists are ordered sequences of values.
6. Tuples are ordered, immutable sequences of values.
7. Sets are un-ordered bags of values.
8. Dictionaries are un-ordered bags of key-value pairs.

---------- **iterate**

Iterator objects in python conform to the **iterator protocol**, which basically means they provide two methods: __iter__() and next() .

```
for element in [1, 2, 3]:
    print(element)
for element in (1, 2, 3):
    print(element)
for key in {'one':1, 'two':2}:
    print(key)
for char in "123":
    print(char)
for line in open("myfile.txt"):
    print(line, end='')
```

---------- **variables**

---------- **underscore**

“Private” instance variables that cannot be accessed except from inside an object don’t exist in Python. However, there is a convention that is followed by most Python code: a name prefixed with an underscore (e.g. `_spam`) should be treated as a non-public part of the API (whether it is a function, a method or a data member). It should be considered an implementation detail and subject to change without notice.

Since there is a valid use-case for class-private members (namely to avoid name clashes of names with names defined by subclasses), there is limited support for such a mechanism, called *name mangling*. Any identifier of the form `__spam` (at least two leading underscores, at most one trailing underscore) is textually replaced with `_classname__spam`, where `classname` is the current class name with leading underscore(s) stripped. This mangling is done without regard to the syntactic position of the identifier, as long as it occurs within the definition of a class.

---------- **Integers**

int vs long

range vs xrange

In Python 3.x, the `xrange` function does not exist anymore. The `range` function now does what `xrange` does in Python 2.x, so to keep your code portable, you might want to stick to using `range` instead.

---------- **Float**

---------- **Strings**

" ", vs ' ' vs """ """

---------- **Fractions**

---------- **List and Tuples**

list comprehension

[*exp* for *val* in *collection*]

[*exp* for *val* in *collection* if <*test*>]

[*exp* for *val* in *collection* if <*test*> and <*test*>]

[*exp* for *val1* in *collection1* and *val2* in *collection2*]

squares = [i**2 for i in range(1,101)]

remainders = [x**2 % 5 for x in range(1,101)]
remainders = [x**2 % 11 for x in range(1,101)]

gmovies = [title for title in movies if title.startswith("G")]

suppose movies is a list of tuples (title, release year)
all movies releases before 2000

per2k = [title for (title,year) in movies if year < 2000)]

movies = [("Citizen Kane", 1944), ("Spirited Away", 2001),
("Its A Wonderful Life", 1946), ("Gattaca", 1997),
("No Country for Old Men", 2007), ("Rear Window", 1954),
("The Lord of the Rings: The Fellowship of the Ring", 2001),
("Groundhog Day", 1993), ("Close Encounters of the Third Kind", 1977),
("The Royal Tenenbaums", 2001), ("The Aviator", 2004),
("Radiers of the Lost Ark", 1981)]

from: www.youtube.com/watch?v=AhSvKGTh28Q

Use '*del list[]*' to remove an element by index, '*list.pop()*' to remove it by index if you need the returned value, and '*list.remove*()' to delete an element by value. The latter requires searching the list, and raises '*ValueError*' if no such value occurs in the list.

'*list.Append()*' adds to the end of a list.

'*list.insert(index,obj)*' inserts object *obj* at offset *index*.

---------- **Dictionaries**

---------- **Scope of Variables**

LEGB Rules

(L) Local - Names assigned in any way within a function (`def` or `lambda`)), and not declared global in that function.

(E) Enclosing-function locals — Name in the local scope of any and all statically enclosing functions (*`def`* or *`lambda`*), from inner to outer.

(G) Global (module) — Names assigned at the top-level of a module file, or by executing a *`global`* statement in a *`def`* within the file.

(B) Built-in (Python) — Names preassigned in the built-in names module :
`open`, *`range`*, *`SyntaxError`*, ...

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

---------- Class Variables

Variables declared inside the class definition, but not inside a method are class or static variables

```
>>> class MyClass:
...     i = 3
...
>>> MyClass.i
3
```

`https://stackoverflow.com/questions/68645/static-class-variables-in-python`

---------- Static Variables

--------- Python Scripts

---------- main

if __name__ == "__main__":

---------- Bubble Sort

`https://en.wikipedia.org/wiki/Bubble_sort`

---------- Functions vs Methods

Methods are associated with objects/classes. for example, `.lower` is a method of the string object. It will do something to that object, typically.

Functions/procedures stand alone, so you must provide them with any arguments they need.

Another way to look at it is both are chunks of code. One stand alone and must be passed data, and the other uses the data from the object it is associated with.