Python is not compiled - it is interpreted.

To run:
python (or python3 or python2.7) nameofprogram.py <— file extension must be .py

No pointers!

Do not need a main function!

Comments - pound sign (#)

Literals:
integers
real numbers
string - can be either single or double quotes
boolean - True or False (note capitalization)
arrays

Expressions:

x = 2
x = “hello” <———- no types!! Python is dynamically typed

You can use semicolons, but they are not required.

Functions:

```
def name_of_func(parameters):    <—- note colon
        x = 4
        y = 2
        return x + y, x                    <——- can return multiple items
```

Indentation matters - “soft” rule of thumb - wherever you put curly brackets in C, indent in Python

Python Lists

Act like a dynamic array
myList = [1, “roll”, “tide”, True]

Can combine all types into one list.

Built-In Functions

print()
x = input(“text to be displayed”)

len(“rolltide”) <- length of string or list
len(myList)

int(x) <— converts x to int
str(x) <— converts to string
eval(x) <—- converts to whatever Python think x is

Conditionals

if x == 10: <— — Can use parentheses but don't have to
if x != 0:
if x == 7 and y != 12: <—- note colon
if not(x < 5):
if x != True or y < 4:

Loops:

while x < 10: <— — note colon, plus optional use of parentheses

```
for i in range(10):
        print(i)

x = [1, 2, 3, 4, 5]
for num in x:
        print(num)
```

Command Line Args

Do not include argc and argv as parameters in main

Use the same way:
argc == number of arguments (including .py file as argument)
argv == array holding all command line arguments (.py file is at index 0)

python hello.py 2 True "yes"

argc equals 4
argv equals ["hello.py", "2", "True", "yes"] <— all arguments are strings, must convert

File I/O

To read a file:

```
f = open("file.txt", "r")

x = f.readline()

f.close()
```

To write to a file:

```
f = open("file.txt", "w")
```

```
f.write(“roll tide!”)

f.close()
```