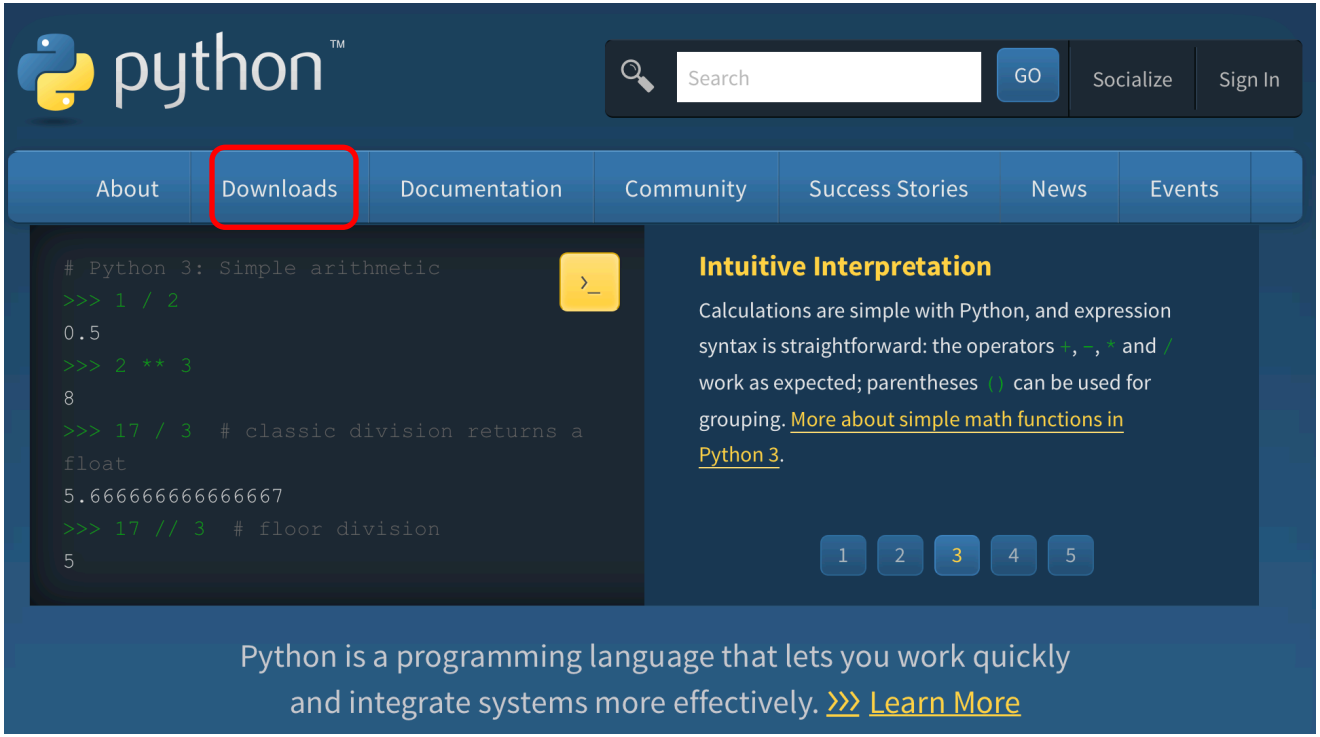
The Python IDLE: Introduction

This tutorial assumes you have a PYTHON 3 version of the programming language installed. (Current version at of this writing is Python 3.6)

Python is available at: https://www.python.org/
Why Python is awesome: https://www.python.org/about/success/



On a Linux machine or a Mac you can check to see if Python 3 is installed by opening a terminal window and typing *python* at the prompt. If Python is not installed, you can download it at the python.org website.

Python on the command line

After installing Python, you should be able to invoke Python on the command line in a terminal window by typing the name of the program. This opens the *Python Interpreter*, where you can run Python code directly in the terminal by typing 'python' and hitting the *Enter* key:

```
~ $ python
Python 3.6.1 |Anaconda 4.4.0 (x86_64)| (default, May 11 2017, 13:04:09)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You can type in and run your very first Python program in the Python Interpreter. By Law of the Coders, it must print "Hello world!"

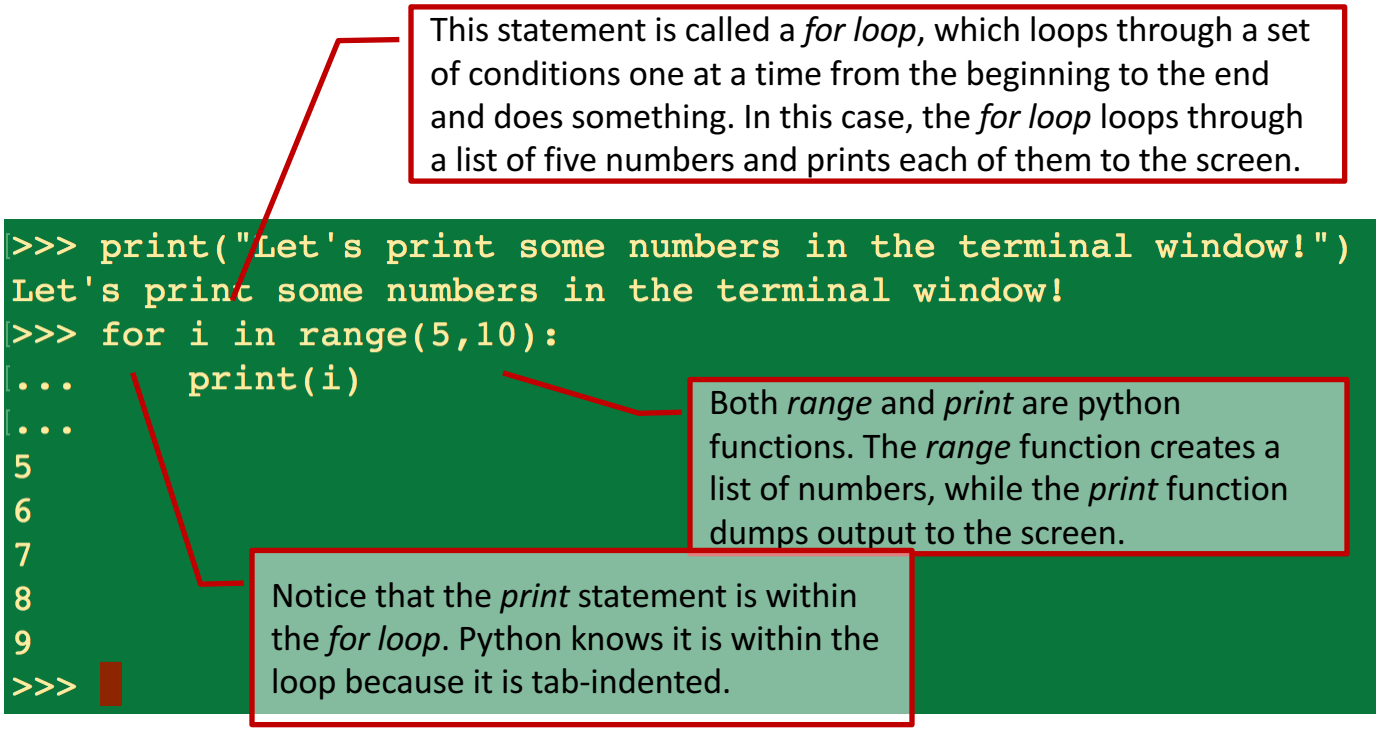
```
>>> print("Hello world!")
Hello world!
>>>
```

```
>>> print("Hello world!")
Hello world!
>>> 0.5*(2017)
1008.5
>>>
```

Look, it's a calculator too!

The Python Interpreter: Instant feedback

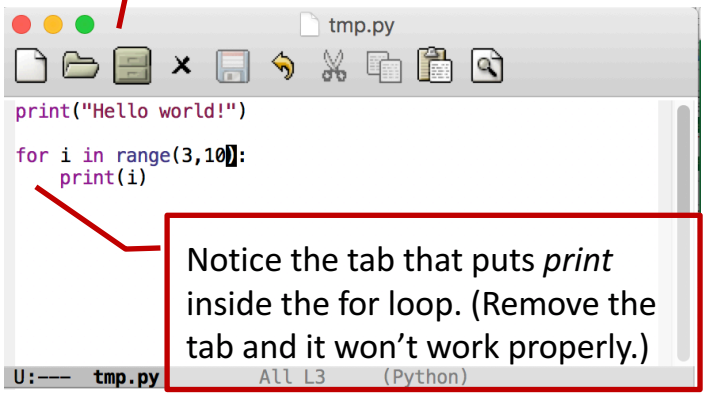
Let's do something a little more interesting. This "code snippet" prints the numbers from 5 to 9. How would you change the loop it to print numbers from 3 to 101? Try it yourself!



Python: Saving your programming code

Once you quit the Python Interpreter (type *Control-d* to quit) all your work disappears. The interpreter is nice for testing out simple functions or for doing calculations, but your work is lost after you quit. To save your, program you need to write it in a separate file. Below we save our work with the Emacs text editor, a commonly used text editor for programming. (The *nano* program is an editor that often comes with linux. Type *nano* at the prompt.)

Write the program in the text editor and save it. I called this python program 'tmp.py'.



Notice the tab that puts *print* inside the for loop. (Remove the tab and it won't work properly.)

To run the program type *python* and the name of the program like this:

```
~ $ python tmp.py
Hello world!
3
4
5
6
7
8
9
~ $
```

Python: Writing data to a file

One of the most important aspects of programming especially in bioinformatics is *writing to* and *reading from* files. Data is stored in files and biological data are primarily saved and stored in text files. Here is an example showing how one can use Python to write some tab-delimited text data to a file. Trying making this file in a text editor then running it (below right).

writing_a_file.py

```
#Fake data to be written to a text file
# Note: the \t meand a tab character and the \n is a return character

column_header="Sample\tData1\tData2\n"
data1="Mud\t4.5\t98.7\n"
data2="Water\t0.02\t45.1\n"

#Open a file for writing.

fout=open("my_data.txt","w")

#Use the write function to write data to a file

fout.write(column_header)
fout.write(data1)
fout.write(data2)

#Remember to close the file using the close function

fout.close()
```

Here is a Python script for writing data to a file called "my_data.txt"

The stuff in RED after the # sign are comments that are ignored by Python, but help us know what is happening.

Three variables with string data to be written to the file

The **fout** variable is a file object for writing data. The 'w' indicates we can write data to the file.

The **write** function writes the data to file.

```
~/Desktop $ python writing_a_file.py
~/Desktop $
~/Desktop $ ls my*
my_data.txt
~/Desktop $
~/Desktop $ cat my_data.txt
Sample  Data1   Data2
Mud     4.5     98.7
Water   0.02    45.1
~/Desktop $
```

Python: Reading data from a file

This program give an example of how to read a file from the computer. The program also has something called a 'loop' that loops through every line of the file. Note: For this to work, you must already have a file named 'my_data.txt' and it needs to be in the same folder/directory as the python file.

print_lines.py

Specify a new file's name, to edit the file

```
#Open a file for reading in data.

# By default the file path is the same directory as the python file.
# The 'U' option stands for Universal which recognizes all return characters.

fin=open('my_data.txt','U')

#Loop through every line of the file.
# Each the variable line is set to each line of the file one at a time
# and then the next line prints the line

for line in fin:
    print(line)

#Close the file
fin.close()
```

```
~/Desktop $ python print_lines.py
Sample   Data1    Data2

Mud      4.5      98.7

Water    0.02     45.1
```

Python Functions: Instant feedback

Functions are the key to programming. They are like little machines: You put in some data, then you get something different on the other side. Python has a lot of ones already made called built-in functions, but you can also make your own. Call they interpreter, then try the code below:

```
>>> x=9.8
>>> type(x)
<class 'float'>
>>> pet="llama"
>>> len(pet)
5
>>> #Make your own function
... def error_msg():
...     print("You made a horrible mistake. Try again!")
...
>>> #Calling your new function
... error_msg()
You made a horrible mistake. Try again!
>>>
>>> #A more interesting function with arguments
... def add_2_numbers(x,y):
...     answer=x+y
...     return answer
...
>>> add_2_numbers(5,3.4)
8.4
>>>
```

The functions **type** and **len** are functions already in Python. What do you think their purposes are?

Here is a simple function that takes no data.

Here we use the function. (Call the function).

Python Functions: Saving and running

Save the function to a file called 'run_functions.py'. Then call the functions.

run_functions.py

```
def error_msf():
    print("You made a horrible mistake. Try again!")


def add_2_numbers(x,y):
    answer=x+y
    return answer

error_msf()

add_2_numbers(7.3, 5.7)
```

Wait, why did add_2_numbers not print out the answer?

```
~/Desktop $ python run_functions.py
You made a horrible mistake. Try again!
~/Desktop $
```

Python Functions: Saving and running

Save the function to a file called 'run_functions.py'. Then call the functions.

run_functions.py

```
def error_msf():
    print("You made a horrible mistake. Try again!")


def add_2_numbers(x,y):
    answer=x+y
    return answer

error_msf()

add_2_numbers(7.3, 5.7)

output=add_2_numbers(7.3,5.7)

print(output)
```

Using return, you can assign the outcome to a variable!

Then you can print the variable separately.

```
~/Desktop $ python run_functions.py
You made a horrible mistake. Try again!
13.0
~/Desktop $
```

Now we see the answer on the screen.