Programming for Biologists

Python command line interface

Learning Objectives

- Running python script from command line
- Using argparse

Python Command Line Interface (CLI)

- Just like linux commands are executed on the terminal, we can also execute python scripts.
- Executing python scripts essentially runs the commands in the order it has been written in the .py script

```
[msk8@log-0 ~]$ echo "print('hello world')" > hello.py
[msk8@log-0 ~]$ cat hello.py
print('hello world')
[msk8@log-0 ~]$ python3 hello.py
hello world
```

Useful commands for command line

Command	Function
ls	List of files in director
pwd	Path of the current directory
mkdir <dir_name>	Make a directory
cd <dir_name>	Change into the directory
rmdir <dir_name>	Remove a directory
cat <file_name>	Reads through contents in the file
echo “string”	Repeat everything
command > file_name	> can be used to save the output of the command to a new file. >> is to append to the file.

Using a python module as a script

- We have seen python scripts used as python modules.
    - Simply import the file and start using its functions
- We can also have the same script be used as a separate command AND a module.
    - The `if __name__ == "__main__":` statement can be used to provide specific sequence of events that use the functions in the script to do the work.
    - This is a great way to build functions that test the module.

Using __name__ == “__main__”

```
[msk8@log-0 ~]$ cat hello.py
def say_hello(name):
    print("hello " + str(name) + "!")

if __name__ == "__main__":
    say_hello("Manny")

[msk8@log-0 ~]$ python3 hello.py
hello Manny!
```

Importing the module

- Importing the module skips give the __name__ variable the value of the name of the script.
    - So in the previous case, __name__ == ‘hello’ but we don’t need to check this.

```
[msk8@log-0 ~]$ nano get_name.py
[msk8@log-0 ~]$ cat get_name.py
import hello

hello.say_hello("Reena")

[msk8@log-0 ~]$ python3 get_name.py
hello Reena!
```

Argparse module to parse command line arguments.

- Argparse allows you to parse command line options and arguments.
- With Argparse you can provide the arguments your script will need to run.

Running Python Programs

```
[msk8@log-0 ~]$ cat get_name.py
import argparse
import hello

parser = argparse.ArgumentParser()

parser.add_argument("-n", "--name", help="Please provide \
the name of the person you are planning to welcome")
args = parser.parse_args()

hello.say_hello(args.name)

[msk8@log-0 ~]$ python get_name.py -n katari
hello katari!
```