Lab 1: Introduction to Python Programming

Adapted from
Nicole Rockweiler
01/09/2019
Overview

- Logistics
- Getting Started
- Intro to Unix
- Intro to Python
- Assignment 1
Getting the most out of this course

1. Start the homework EARLY
2. Collaborate
3. Use your resources – TAs, professors, labmates, Piazza discussions, the internet
Logistics

- **Office Hours**: Wednesdays, 11:30 am - 12:30 pm (right after class)
- **Contact TAs**:
  - For assignment-related questions: Piazza
  - For other questions: bio5488wustl@gmail.com
- Register for 4 credits
- Course website: [http://genetics.wustl.edu/bio5488/](http://genetics.wustl.edu/bio5488/)
- Bring your laptop to every lab
- **NO extensions** on homeworks
- **Late penalty** is 50% per day
Assignments

• Assignments are posted on the course website Wednesdays
  • We will send out emails when assignments are posted
• Assignments are due the following Friday at 10am (before lab)

• Assignment format
  • Given a bioinformatics problem
  • Write/complete a Python script
  • Analyze data with your script
  • Answer biological questions about your results

• Turn in format
  • More on this later 😊
Assignment policies

• See the Course Information → Assignment policies document in the course website

• There are 13 assignments
  • You must turn in all assignments
  • All assignments are weighted equally

• Collaboration
  • Group work is encouraged, but plagiarism is unacceptable
  • Try to “Google it” first
  • Cite your sources

• Read the assignment before coming to lab
Grading

• Each assignment is out of 10 points

• Graded on
  • Does the code work?
    • It doesn’t have to be the “fastest” or “most efficient” to get full credit
    • If doesn’t work, describe where you had problems
    • Is the code well commented and readable? (more on commenting later 😊)
  • Are the answers correct?

• Grades will be returned in a file called grades.txt on the class server
  • Only you and the TAs will be able to read this file
Getting started
Remote computers

• We will be doing all of our work on a remote computer, a server
• This is a Unix-based computer that we can securely connect to through a protocol called secure shell (SSH).
• The shell is a program that takes commands from the keyboard and gives them to the operating system to execute
How do I access the server?

• The way we are using here is **command-line interfaces (CLI)**
• A **terminal emulator** is a program that allows you to interact with the shell through a CLI
  • There are different terminal programs that vary across operating systems
  • We’ll be using **PuTTY** (Windows) or **Terminal** (Mac, Ubuntu)
1. Launch PuTTY
2. In the host name field, enter `<username>@genomic.wustl.edu`
3. In the port field, enter 22
4. Enter a session nickname, e.g., bio5488 (whatever name you want!)
5. Click Save
6. Click Open
How to log onto the remote computer (Mac/Ubuntu users)

1. Open Terminal (found in /Applications/Utilities)
How to log onto the remote computer (Mac/Ubuntu users)

2. SSH to the remote computer. Type:

```
ssh <username>@genomic.wustl.edu
```

where `<username>` is replaced with your username

3. A security message may be printed. Type `yes` and hit enter.
How to log onto the remote computer
(Mac users)

4. Enter your password - it will not show that you are typing! Hit enter.
A couple of notes

• When you log onto the class server you will be located in YOUR home directory.

• Every command that you run after logging onto a remote computer will be run on that computer.
Exercise: changing your password (passwd)

• To change your password, type the command

  $ passwd

• This will launch the interactive password changer
  • It will ask you for your current password, then your new password twice
  • When typing your password, **it will not show that you are typing!**

• Example

  $ passwd
  Changing password for xinxin.wang.
  (current) UNIX password:
  Enter new UNIX password:
  Retype new UNIX password:
  passwd: password updated successfully
• Sublime Text is a **text editor** for writing and editing scripts
• We’ll use Sublime to edit both local and remote files
• Installation: [https://www.sublimetext.com/3](https://www.sublimetext.com/3)
• Documentation: [http://www.sublimetext.com/support](http://www.sublimetext.com/support)
Cyberduck

• Cyberduck is a secure file transfer client and will allow you to transfer files from your local computer to a remote computer.
Exercise: setting up Cyberduck

• Create a bookmark
  • Launch the Cyberduck application
  • Click Bookmark → New Bookmark
  • Select SFTP (SSH File Transfer Protocol) from the drop down menu
  • Enter a nickname for the bookmark, e.g., bio5488
  • Enter genomic.wustl.edu as the server name
  • Click the X

• Set the default text editor
  • Click Edit → Preferences → Editor
  • Select sublime text from the drop down menu. (You may need browse your computer for the editor)
  • Check Always use this application
  • Restart Cyberduck
Exercise: transferring files with Cyberduck

• To *download* a file to your local computer
  • Drag and drop a file from Cyberduck to your Finder/File Explorer window
  • Or, double-click

• To *upload* a file to the remote computer
  • Drag and drop a file from Finder/File Explorer to Cyberduck
Exercise: editing remote files with Sublime Text and Cyberduck

• New files
  • Click File → New file
  • Enter a filename
  • Click edit
  • Sublime Text should now launch
  • Add some text to the file
  • Click File → Save or ctrl+S

• Existing files
  • Select the file by clicking the filename 1X
  • Click the Edit button in the navigation bar
  • Edit the file
  • Click File → Save or ctrl+S
Attention about using Cyberduck:

• When clicking on
  
  o Make sure you see this

• When saving the file, make sure you see the following to make sure the upload is complete before you close the editor

• Before closing the editor, check the time stamp of file
FileZilla

- FileZilla is an alternative approach for Cyberduck
- Can be downloaded for free here: https://filezilla-project.org/
Follow the instructions.

Finally, we should see this.
Basic Unix
A few preliminary words…

A lot of Unix skills revolve around the file system

• This concept is similar to using Apple Finder or the Windows File Explorer GUIs, only this time, we can’t use a mouse or see any fancy graphics
The file system

• The **file system** is the part of the operating system (OS) responsible for managing files and folders
  • In Unix, folders are called **directories**.

• Unix keeps files arranged in a hierarchical structure
  • The topmost directory is called the **root directory**
  • Each directory can contain
    • Files
    • Subdirectories

• You will always be “in” a directory
  • When you open a terminal you will be in your own **home directory**.
  • Only you can modify things in your home directory
Determining where you are (pwd)

• If you get lost in the file system, you can determine where you are by typing:
  
  $ pwd
  
  /home/user

• `pwd` stands for **print working directory**

• `pwd` prints the full **path** of the **current working directory**
Listing directory contents (\texttt{ls})

• To list the contents of a directory:
  
  \texttt{\$ \texttt{ls}}
  
  
  \texttt{assignment1 foo}

• \texttt{ls} stands for \texttt{list directory contents}
Changing directories

(cd)

• To change to different directory
  
  \$ cd <directory_name>

  where

  <directory_name> = the path you want to move to

  • A path is a location in the file system

• cd stands for change directory

• To get back to your home directory

  \$ cd ~

  • ~ is shorthand for your home directory
Changing directories (cont.)

- To move *one* directory above the current directory
  
  ```bash
  $ cd ..
  ```

- To move *two* directories above the current directory
  
  ```bash
  $ cd ../../../
  ```

- You can string as many `../` as you need to
Making directories

(mkdir)

• To make a directory

$ mkdir <new_directory_name>

where

<new_directory_name> = name of the directory to create

• mkdir stands for make directory

• Do not use spaces or “/” in directory or file names
Exercise: create some directories

Try to create this directory structure:

Hints

• Use `pwd` to determine where you are in the directory structure
• Use `cd` to navigate through the directory structure.
• Use `mkdir` to create new directories
Last login: Thu Jan 16 21:25:01 2020 from 10.224.6.245
wwliao@genomics:~$ pwd
/home/wwliao
wwliao@genomics:~$ ls
wwliao@genomics:~$ mkdir assignment1
wwliao@genomics:~$ ls
assignment1
wwliao@genomics:~$ cd assignment1
wwliao@genomics:~/assignment1$ mkdir work submission test
wwliao@genomics:~/assignment1$ ls
submission  test  work
wwliao@genomics:~/assignment1$
Copying things (\texttt{cp})

• To create a copy of a file

\begin{verbatim}
$ \texttt{cp -i <filename> <copy_of_filename>}
\end{verbatim}

where

\begin{itemize}
  \item \texttt{<filename>} = file you want to copy
  \item \texttt{<copy_of_filename>} = name of copied file
\end{itemize}

The -i flag is a safety feature to make sure you do not overwrite a file that already exists.

• To create a copy of a directory

\begin{verbatim}
$ \texttt{cp -r <directory> <copy_of_directory>}
\end{verbatim}

where

\begin{itemize}
  \item \texttt{<directory>} = directory you want to copy
  \item \texttt{<copy_of_directory>} = name of copied directory
\end{itemize}

The -r flag is required to copy all of the directory’s files and subdirectories.
Copying things (cont.)

(cp)

• cp stands for copy files/directories

• To create a copy of file and keep the name the same

  $ cp -i <filename> .

  where

  $<filename> = file you want to copy

• The shortcut is the same for directories, just remember to include the -r flag
Exercise: copying things

Copy /home/assignments/assignment1/README.txt to your work directory. Keep the name the same.
ssh wwliao@genomic.wustl.edu

wwliao@genomic.wustl.edu's password:
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-52-generic x86_64)

Last login: Thu Jan 16 21:25:01 2020 from 10.224.6.245

wwliao@genomics:~$ pwd
/home/wwliao
wwliao@genomics:~$ ls
wwliao@genomics:~$ mkdir assignment1
wwliao@genomics:~$ ls
assignment1
wwliao@genomics:~$ cd assignment1
wwliao@genomics:~/assignment1$ mkdir work submission test
wwliao@genomics:~/assignment1$ ls
submission  test  work
wwliao@genomics:~/assignment1$ cd work/
wwliao@genomics:~/assignment1/work$ cp /home/assignments/assignment1/README.txt .
wwliao@genomics:~/assignment1/work$ ls
README.txt
wwliao@genomics:~/assignment1/work$ 
Renaming/moving things
(mv)

• To rename/move a file/directory

  $ mv -i <original_filename> <new_filename>

  where

  <original_filename> = name of file/dir you want to rename
  <new_filename> = name you want to rename it to

• mv stands for move files/directories
Printing contents of files (cat)

• To print a file

$ cat <filename>

where

  <filename> = name of file you want to print

• cat stands for concatenate file and print to the screen

• Other useful commands for printing parts of files:
  • more
  • less
  • head
  • tail
Deleting Things

\((\text{rm})\)

• To delete a file

\[
\text{\$ rm <file\_to\_delete>}
\]

where

\(<\text{file\_to\_delete}> = \text{name of the file you want to delete}\)

• To delete a directory

\[
\text{\$ rm -r -i <directory\_to\_delete>}
\]

where

\(<\text{directory\_to\_delete}> = \text{name of the directory you want to delete}\)

• \text{rm} stands for \text{remove} files/directories

\[\text{TIP: Check that you’re going to delete the correct files by first testing with 'ls' and then committing to 'rm'}\]

\[\text{IMPORTANT: there is no recycle bin/trash folder on Unix!! Once you delete something, it is gone forever. Be very careful when you use rm!!}\]
Exercise: deleting things

Delete the test directory that you created in a previous exercise.
wwliao@genomics:~/assignment1$ ls
submission test work
wwliao@genomics:~/assignment1$ rm -r -i test
rm: remove directory 'test'? y
wwliao@genomics:~/assignment1$ ls
submission work
wwliao@genomics:~/assignment1$
Saving output to files

• *Save* the output to a file
  
  $ \langle \text{cmd} \rangle \ > \ <\text{output\_file}>$

  where

  $\langle \text{cmd} \rangle = \text{command}$
  
  $\langle \text{output\_file} \rangle = \text{name of output file}$

  • **WARNING:** this will overwrite the output file if it already exists!

• *Append* the output to the end of a file
  
  $\langle \text{cmd} \rangle \ >> \ <\text{output\_file}>$

There are 2 “>”
wwliao@genomics:~$ pwd
/home/wwliao
wwliao@genomics:~$ ls -l ../assignments/assignment1/
total 63936
-rw-rw-r-- 1 sharon.freshour genomic 65455484 Mar 30 2018 hs_ref_GRCh38.p12Chr20.fa
-rw-rw-r-- 1 sharon.freshour genomic 1630 Jan 16 12:40 make_seq.py
-rw-rw-r-- 1 sharon.freshour genomic 1608 Jan 16 12:40 nuc_count.py
-rw-rw-r-- 1 sharon.freshour genomic 850 Jan 16 12:40 README.txt
wwliao@genomics:~$ ls -l ../assignments/assignment1/ > file_longlist.txt
wwliao@genomics:~$ cat file_longlist.txt
total 63936
-rw-rw-r-- 1 sharon.freshour genomic 65455484 Mar 30 2018 hs_ref_GRCh38.p12_chr20.fa
-rw-rw-r-- 1 sharon.freshour genomic 1630 Jan 16 12:40 make_seq.py
-rw-rw-r-- 1 sharon.freshour genomic 1608 Jan 16 12:40 nuc_count.py
-rw-rw-r-- 1 sharon.freshour genomic 850 Jan 16 12:40 README.txt
Learning more about a command (man)

- To view a command’s documentation
  
  $ man <cmd>

  where

  $ cmd = command

- man stands for manual page

- Use the ↑ and ↓ arrow keys to scroll through the manual page

- Type “q” to exit the manual page
NAME

ls - list directory contents

SYNOPSIS

ls [OPTION]... [FILE]...

DESCRIPTION

List information about the FILES (the current directory by default). Sort entries alphabetically if none of `--sort` is specified.

Mandatory arguments to long options are mandatory for short options too.

-a, --all
   do not ignore entries starting with .

-A, --almost-all
   do not list implied . and ..

--author
   with -l, print the author of each file

-b, --escape
   print C-style escapes for nongraphic characters
-k, --kibibytes
default to 1024-byte blocks for disk usage

-l       use a long listing format

-L, --dereference
when showing file information for a symbolic link, show information for the file
the link references rather than for the link itself

-m       fill width with a comma separated list of entries

-n, --numeric-uid-gid
like -l, but list numeric user and group IDs

-N, --literal
print entry names without quoting

-o       like -l, but do not list group information

-p, --indicator-style=slash
append / indicator to directories

-q, --hide-control-chars
print ? instead of non-graphic characters

Manual page ls(1) line 105 (press h for help or q to quit)
Getting yourself out of trouble

• Abort a command

  \[\text{ctrl} + \text{C}\]

• Temporarily stop a command

  \[\text{ctrl} + \text{Z}\]

To bring the job back just run \texttt{fg}
Unix commands cheatsheet--your new bestie

<table>
<thead>
<tr>
<th>File Commands</th>
<th>File Permissions</th>
<th>Process Management</th>
<th>Shortcuts</th>
</tr>
</thead>
</table>
| `ls` - directory listing      | `chmod octal file` - change the permissions of file to octal, which can be found separately for user, group, and world by adding:  
  * 4 - read (r)  
  * 2 - write (w)  
  * 1 - execute (x)  
| `ls -al` - formatted listing with hidden files | Examples:  
  * `chmod 777` - read, write, execute for all  
  * `chmod 755` - rwx for owner, rx for group and world  
| `cd` - change directory to dir | `kill pid` - kill process id    | `Ctrl+C` - halts the current command  
| `cd ~` - change home          | `killall proc` - kill all processes named proc *    | `Ctrl+Z` - stops the current command, resume with `fg` in the foreground or `bg` in the background  
| `pwd` - show current directory | `bg` - lists stopped or background jobs; resume a stopped job in the background  
| `mkdir dir` - create a directory dir | `fg` - brings the most recent job to foreground  
| `rm file` - delete file       | `fg n` - brings job n to the foreground  
| `rm -r dir` - delete directory dir | **SSH**  
| `rm -rf dir` - force remove directory dir * |  
| `cp file1 file2` - copy file1 to file2 |     
| `cp -r dir1 dir2` - copy dir1 to dir2; create dir2 if it doesn't exist |  
| `ln -s file link` - create symbolic link link to file | `ssh user@host` - connect to host as user  
| `touch file` - create or update file | `ssh -p port user@host` - connect to host on port port as user  
| `cat > file` - places standard input into file | `ssh-copy-id user@host` - add your key to host for user to enable a keyed or passwordless login  
| `more file` - output the contents of file | **Searching**  
| `head file` - output the first 10 lines of file |  
| `tail file` - output the last 10 lines of file |  
| `tail -f file` - output the contents of file as it grows, starting with the last 10 lines | `grep pattern files` - search for pattern in files  
| `grep -r pattern dir` - search recursively for pattern in dir | `command | grep pattern` - search for pattern in the output of command  
| `locate file` - find all instances of file | `exit` - log out of current session  

https://ubuntudanmark.dk/filer/fwunixref.pdf
Python in minutes*

*not really
There are 2 widely used versions of Python: Python2.7 and Python3.x

- We’ll use Python3

- Many help forums still refer to Python2, so make sure you’re aware which version is being referenced

NOTE
How do I program in python?

• Two Main Ways:
  • Normal mode
    • Write all your code in a file and save it with a .py extension
    • Execute it using `python3 <file name>` on the terminal.

  • Interactive mode
    • Start Interactive mode by typing `python3` on the terminal and pressing enter/return.
    • Start writing your python code
Python Variables

• The most basic component of any programming language are "things," also called **variables**

• Variables can be integers, decimal numbers (floats), words and sentences (string), lists etc. etc.

• **Int** : -5, 0, 1000000
• **Float** : -2.0, 3.14159, 453.234
• **Boolean** : True, False
• **String** : "Hello world!", "K3WL", “AGCTGCTAGTAGCT”
• **List**: [1, 2, 3, 4], ["Hello", "world!"], [1, "Hello", True, 0.2], [“A”, “T”, “C”, “G”]
How do I create a variable and assign it a value?

• \( x = 2 \)
  • This creates a variable named \( x \) with value 2
  • \( 2 = x \) is not a valid command; variable name needs to be on the left.

• `print(x)`
  • This prints the value stored in \( x \) (2 in this case) on the terminal.

```
a = 3
b = 4
c = a + b
print(c)
```
Prints 7 on the terminal

```
a = "Hello"
b = " 
```
```
c = "World"
print(a+b+c)
```
Prints Hello World on the terminal
Variables naming rules

• Must start with a letter

• Can contain letters, numbers, and underscores ← no spaces!

• Python is case-sensitive: x ≠ X

• Variable names should be descriptive and have reasonable length (more of a styling advice)

• Use ALL CAPS for constants, e.g., PI

• Do not use names already reserved for other purposes (min, max, int)

Want to learn more tips? Check out http://www.makinggoodsoftware.com/2009/05/04/71-tips-for-naming-variables/
Cool, what else can I do in python?

• Conditionals
  • If a condition is TRUE do something, if it is FALSE do something else

```python
if(boolean-expression-1):
    code-block-1
else:
    code-block-2
```

**CODE BLOCKS ARE INDENTED, USE 4 SPACES**
Cool, what else can I do in python?

• Conditionals
  • If a condition is TRUE do something, if it is FALSE do something else

```python
x = 2
if(x == 2):
    print("x is 2")
else:
    print("x is not 2")

Prints **x is 2** on the terminal

x = 3
if(x == 2):
    print("x is 2")
else:
    print("x is not 2")

Prints **x is not 2** on the terminal
```
• Conditionals with multiple conditions

```python
grade = 89.2
if grade >= 80:
    print("A")
elif grade >= 65:
    print("B")
elif grade >= 55:
    print("C")
else:
    print("E")
```

Prints A on the terminal

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
<td>&gt;&gt;&gt;&gt; 2 &lt; 3</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
<td>&gt;&gt;&gt;&gt; 2 &lt;= 3</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
<td>&gt;&gt;&gt;&gt; 2 &gt; 3</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
<td>&gt;&gt;&gt;&gt; 2 &gt;= 3</td>
</tr>
<tr>
<td><code>==</code></td>
<td>Equal to</td>
<td>&gt;&gt;&gt;&gt; 2 == 3</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Not equal to</td>
<td>&gt;&gt;&gt;&gt; 2 != 3</td>
</tr>
</tbody>
</table>
Loops
For loop

• Useful for repeating code!

Start with a list of items

Have we reached the last item?

Do stuff

for <counter> in <collection_of_stuff>:
    code-block-1
For loop

• Useful for repeating code!

Start with a list of items

Have we reached the last item?

Yes → Exit loop

No → Do stuff

```python
genes = ['GATA4', 'GFP', 'FOXA1', 'UNC-21']

for i in genes:
    print(i)

print("printed all genes")
```

GATA4
GFP
FOXA1
UNC-21

printed all genes
More examples

```python
my_string = "Hello"
for i in my_string:
    print(i)
```

```
my_number = 2500
for i in my_number:
    print(i)
```

Functions

```python
def <function name>(<input variables>):
    do some stuff
    return <output>

def celsius_to_fahrenheit(celsius):
    fahrenheit = celsius * 1.8 + 32.0
    return fahrenheit
```
def celsius_to_fahrenheit(celsius):
    fahrenheit = celsius * 1.8 + 32.0
    return fahrenheit

temp1 = celsius_to_fahrenheit(37)  # sets temp1 to 98.6
 temp2 = celsius_to_fahrenheit(100)  # sets temp2 to 212
 temp3 = celsius_to_fahrenheit(0)   # sets temp3 to 32
But how do I use a function?

def addition(num1, num2):
    num3 = num1 + num2
    return num3

sum = addition(4, 5)  # sets sum to 9
A = 2
B = 3
sum2 = addition(A, B)  # sets sum2 to 5
sum3 = addition(5)    # throws an error
Python functions: where can I learn more?

• Python.org tutorial
  • User-defined functions: https://docs.python.org/3/tutorial/controlflow.html#defining-functions

• Python.org documentation
  • Built-in functions: https://docs.python.org/3/library/functions.html
Commenting your code

• Why is this concept useful?
  • Makes it easier for--you, your future self, TAs 😊, anyone unfamiliar with your code--to understand what your script is doing

• Comments are human readable text. They are ignored by Python.

• Add comments for
  The how
  • What the script does
  • How to run the script
  • What a function does
  • What a block of code does
  The why
  • Biological relevance
  • Rationale for design and methods
  • Alternatives

TREAT YOUR CODE LIKE A LAB NOTEBOOK
Commenting rule of thumb

Always code [and comment] as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. Code for readability.

-- John Woods

- Points will be deducted if you do not comment your code
- If you use code from a resource, e.g., a website, cite it
## Comment syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
</table>
| **Block comment**    | # <your_comment>  
# <your_comment>  
# Part 5  
# TODO Use overlapping windows to count the  
# dinucleotides in alphabetical order. See the  
# assignment for more information on overlapping  
# windows. |
| **In-line comment**  | <code> # <your_comment>  
num_genes = 42 # number of diff. expressed genes |
Python modules

- A module is a file containing Python definitions and statements for a particular purpose, e.g.,
  - Generating random numbers
  - Plotting
- Modules must be imported at the beginning of the script
  - This loads the variables and functions from the module into your script, e.g.,
    ```python
    import sys
    import random
    ```
- To access a module's features, type `<module>.<feature>`, e.g.,
```python
sys.exit()
```
Random module

• Contains functions for generating random numbers for various distributions

  • TIP: will be useful for assignment 1

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>random.choice</td>
<td>Return a random element from a list</td>
</tr>
<tr>
<td>random.randint</td>
<td>Return a random integer in a given range</td>
</tr>
<tr>
<td>random.random</td>
<td>Return a random float in the range [0, 1)</td>
</tr>
<tr>
<td>Random.seed</td>
<td>Initialize the (pseudo) random number generator</td>
</tr>
</tbody>
</table>

https://docs.python.org/3.4/library/random.html
Example

```python
import random

numberList = [111, 222, 333, 444, 555]

# assigns a value from numberList to x at random
x = random.choice(numberList)
```
Strings

- String is a sequence of characters, like "Python is cool"
- Each character has an index

<table>
<thead>
<tr>
<th>Python</th>
<th>i</th>
<th>s</th>
<th>cool</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
<td>5</td>
<td>6</td>
<td>7 8 9</td>
</tr>
</tbody>
</table>

- Accessing a character: string[index]
  
x = "Python is cool"
  
  print(x[10])

- Accessing a substring via slicing: string[start:finish]
  
  print(x[2:5])

  Prints *tho* and not *thon*
More string stuff

```python
>>> x = "Python is cool"

>>> "cool" in x  # membership

>>> len(x)  # length of string

>>> x + "?"  # concatenation

>>> x.upper()  # to upper case

>>> x.replace("c", "k")  # replace characters in a string
```
Lists

• If a string is a sequence of characters, then a list is a sequence of items!
• List is usually enclosed by square brackets [ ]
• As opposed to strings where the object is fixed (= immutable), we are free to modify lists (that is, lists are mutable).

```python
x = [1, 2, 3, 4]
x[0] = 4
x.append(5)
print(x)  # [4, 2, 3, 4, 5]
```
More lists stuff

```python
>>> x = [ "Python", "is", "cool" ]
>>> x.sort()  # sort elements in x
>>> x[0:2]    # slicing
>>> len(x)    # length of string
>>> x + ['!'] # concatenation
>>> x[2] = "hot"  # replace element at index 2 with "hot"
>>> x.remove("Python")  # remove the first occurrence of "Python"
>>> x.pop(0)  # remove the element at index 0
```
Lists: where can I learn more?

• Python.org tutorial: 
  https://docs.python.org/3.4/tutorial/datastructures.html#more-on-lists

• Python.org documentation: 
  https://docs.python.org/3.4/library/stdtypes.html#list
Command-line arguments

• Why are they useful?
  • Passing command-line arguments to a Python script allows a script to be customized

• Example
  • `make_nuc.py` can create a random sequence of `any length`
  • If the length wasn’t a command-line argument, the length would be hard-coded
    • To make a 10bp sequence, we would have to 1) edit the script, 2) save the script, and 3) run the script.
    • To make a 100bp sequence, we’d have to 1) edit the script, 2) save the script, and 3) run the script.
  • This is tedious & error-prone
  • Remember: be a lazy programmer!
CANT YOU PASS THE SALT?

I SAID-
I KNOW! IM DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!
Command-line arguments

- Python stores the command-line arguments as a list called `sys.argv`
  - `sys.argv[0]` # script name
  - `sys.argv[1]` # 1st command-line argument
  - ...

**IMPORTANT**: arguments are passed as strings!
- If the argument is not a string, convert it, e.g., `int()`, `float()`

- `sys.argv` is a list of variables
  - The values of the variables, are not “plugged in” until the script is run
Reading (and writing) to files in Python

Why is this concept useful?

• Often your data is much larger than just a few numbers:
  • Billions of base pairs
  • Millions of sequencing reads
  • Thousands of genes

• It’s may not feasible to write all of this data in your Python script
  • Memory
  • Maintenance

How do we solve this problem?
Reading (and writing) to files in Python

The solution:

• Store the data in a separate file

• Then, in your Python script
  • **Read** in the data (line by line)
  • Analyze the data
  • **Write** the results to a new output file or print them to the terminal

• When the results are written to a file, other scripts can read in the results file to do more analysis
Reading a file syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>with open(&lt;file&gt;) as &lt;file_handle&gt;: for &lt;current_line&gt; in open(&lt;file&gt;) , ‘r’): &lt;current_line&gt; = &lt;current_line&gt;.rstrip() # Do something</code></td>
<td><code>with open(fasta) as f: for line in f: line = line.rstrip() print(line)</code></td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td><code>&gt;chr1 ACGTTGAT ACGTA</code></td>
</tr>
</tbody>
</table>
The anatomy of a (simple) script

- The first line should always be `#!/usr/bin/env python3`
- This special line is called a shebang
- The shebang tells the computer how to run the script
- It is NOT a comment
The anatomy of a (simple) script

• This is a special type of comment called a **doc string**, or documentation string
• Doc strings are used to explain 1) what script does and 2) how to run it
• ALWAYS include a doc string
• Doc strings are enclosed in triple quotes, """

```python
#!/usr/bin/env python3

"""
hello_world.py prints a greeting

Usage: python3 hello_world.py <name>

  <name> = Name of person you want to say hello to

"""

# Import modules
import sys

ame = sys.argv[1]
print("Hello ", name, ",!", sep="")
```
The anatomy of a (simple) script

- This is a comment
- Comments help the reader better understand the code
- Always comment your code!
The anatomy of a (simple) script

• This is an import statement
• An import statement loads variables and functions from an external Python module
• The sys module contains system-specific parameters and functions
This grabs the command line argument using `sys.argv` and stores it in a variable called `name`. 

```python
#!/usr/bin/env python3

"""
hello_world.py prints a greeting
Usage: python3 hello_world.py <name>

<name> = Name of person you want to say hello to

"""

# Import modules
import sys

name = sys.argv[1]
print("Hello ", name, ",!", sep="")
```
The anatomy of a (simple) script

• This prints a statement to the terminal using the print function
• The first list of arguments are the items to print
• The argument sep="" says do not print a delimiter (i.e., a separator) between the items
• The default separator is a space.
Python resources

• Documentation
  • https://docs.python.org/3/

• Tutorials
  • https://www.learnpython.org/
  • https://www.w3schools.com/python/
  • https://www.codecademy.com/learn/learn-python-3
Assignment 1
How to complete & “turn in” assignments

1. Create a separate directory for each assignment
2. Create “submission” and “work” subdirectories
   • work = scratch work
   • submission = final version
   • *The TAs will only grade content that is in your submission directory*
3. Copy the starter scripts and README to your work directory
4. Copy the final version of the files to your submission directory
   • *Do not edit your submission files after 10 am on the due date (always Friday)*
# README files

- README.txt file contains information on how to run your code and answers to any of the questions in the assignment
- A template will be provided for each assignment
- Copy the template to your work folder
- Replace the text in {} with your answers
- Leave all other lines alone 😊

## README.txt template

<table>
<thead>
<tr>
<th>Question 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{nuc_count.py nucleotide count output}</code></td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>Comments:</td>
</tr>
<tr>
<td><code>{Things that went wrong or you can not figure out}</code></td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

## Completed README.txt

<table>
<thead>
<tr>
<th>Question 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 10</td>
</tr>
<tr>
<td>C: 15</td>
</tr>
<tr>
<td>G: 20</td>
</tr>
<tr>
<td>T: 12</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>Comments:</td>
</tr>
<tr>
<td>The wording for part 2 was confusing.</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
Usage statements in README and scripts

• Purpose
  • Tells a user (you, TA, anyone unfamiliar with the script) how to run the script
  • Documents how you created your results
• In your README
  • Write out exactly how you ran the script:
    python3 foo.py 10 bar
• In your scripts
  • Write out how to run the script in general with placeholders for command-line arguments
    python3 foo.py <#_of_genes> <gene_of_interest>
• TIP: copy and paste your commands into your README
• TIP: use the command history to view previous commands
Assignment 1 Set Up

• Create assignment1 directory

• Create work, submission subdirectories

• Copy assignment material (README, starter scripts) to work directory

• Download human chromosome 20 with **wget** or **FTP**
Fasta file format

• Standard text-based file format used to define sequences
• .fa, .fasta, .fna, …, extensions
• Each sequence is defined by multiple lines
  • Line 1: Description of sequence. Starts with “>”
  • Lines 2-N: Sequence
• A fasta can contain ≥ 1 sequence

Example fasta file

```
>chr22
ACGGTACGTACCGTAGATNAGTAN

>chr23
ACCGATGTGTGTAGGTACGTNACG
TAGTGATGTAT
```
Assignment 1 To-Do’s

• Given a starter script (nuc_count.py) that counts the total number of A, C, G, T nucleotides
  • Modify the script to calculate the nucleotide **frequencies**
  • Modify the script to calculate the dinucleotide frequencies
• Complete a starter script (make_seq.py) to generate a random sequence given nucleotide frequencies
• Use make_seq.py to generate random sequence with the same nucleotide frequencies as chr20
• Compare the chr20 di/nucleotide frequencies (observed) with the random model (expected)
• Answer conceptual questions in README
Requirements

• Due next Friday (1/24) at 10am

• Your submission folder should contain:
  □ A Python script to count nucleotides (nuc_count.py)
  □ A Python script to make a random sequence file (make_seq.py)
  □ An output file with a random sequence (random_seq_1M.txt)
  □ A README.txt file with instructions on how to run your programs and answers to the questions.

• Remember to comment your script!
CONGRATULATIONS

ON BEING FREAKING AWESOME