About me, course logistics, etc.

Alex’s contact info

Email: alexhu@uw.edu

Office hours: Fridays at 1:30 to 2:30 PM
Foege South S110

…or by appointment

• I work on some genomics but mostly mass spectrometry-based proteomics in the Noble and Wolf-Yadlin labs
About me, course logistics, etc.

Homework policy:

No late homework accepted without PRIOR arrangements

Grading is equally about your effort and your execution

First homework assigned tomorrow
What is the quiz section all about?

not a “how-to” homework session

mostly we will learn review in-class material and learn Python

attendance is not required, but the material covered in section is required
Questions about course logistics?
What is an alignment?

Arrangement of nucleotide (or amino acid) sequences to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences.
Evolutionary relationship implies shared ancestor

Homologous
-same gene from different species

Common Ancestor

G – A A T T C A G T T A
|   |   |   |   |   |   |
G G – A – T C – G - - A
Evolutionary relationship implies shared ancestor

Paralogous
-different genes from same family (duplication event)

Common Ancestor

Hemoglobin

Myoglobin
A series of instructions, compiled or interpreted, intended to accomplish a task.

```python
x = 4
y = 8
z = x + y
print(z)
```
What is a program?

A series of instructions, compiled or interpreted, intended to accomplish a task.

```python
x = 4  # A line of code...
y = 8
z = x + y
print(z)
```
What is program?

A series of instructions, compiled or interpreted, intended to accomplish a task.

subject verb object

x = 4  #A line of code...
y = 8  #is like a sentence
z = x + y
print(z)
Variables and operators

subject  verb  object

\[ x = 4 \]
Variables and operators

• An operator is the verb
• “=” assigns values to variables
• A variable can be thought of as a box

\[ x = 4 \]

Now exists in memory!
Variables and operators

\[ x = 4 \]
\[ y = 8 \]
Variables and operators

\[ x = 4 \]
\[ y = 8 \]
\[ z = x + y \]

\[ x \]
\[ y \]
\[ 8 \]
\[ 4 + 8 = 12 \]
\[ z \]
\[ 12 \]
Let’s use python!

1. Open a file in Textwrangler and save it as “myfirstprogram.py”

2. Type the text below and save.
   
   ```python
   x = 4
   y = 8
   z = x + y
   print(z)
   ```

3. Open terminal and type “python myfirstprogram.py”
Comments!

• Any text followed by a “#” in the same line is not read by the computer.

```python
x = 4  # This is a line of code
y = 8  # This is another
z = x + y  # z is the sum of x and y
# print(z)

# Use comments to liberally annotate your code
```
Python also has an interactive mode

• Just type “python” in the terminal

• Now you can type lines of code, one at a time, and view the result in real time

```python
>>> x = 1
>>> print x
1
>>> x
1
```
A list is like a bookshelf of variables accessible by position in the sequence

\[
x = [1.2, 2.5, 3.3, 4.1]
\]

```python
>>> print x[0]
1.2
>>> print x[2]
3.3
>>> print x[-1]
4.1
```
You can “slice” a list into a smaller piece with notation below

\[ x = [1.2, 2.5, 3.3, 4.1] \]

```python
>>> print x[0:2]
[1.2, 2.5]
>>> print x[1:3]
[2.5, 3.3]
>>> print x[1:]
[2.5, 3.3, 4.1]
```
A string is like a list of characters

\[
x = \text{''moo!''}
\]

```
>>> print x[0]
'm'
>>> print x[2]
'o'
>>> print x[-1]
'!'`
Data types

“Simple”
1) Boolean
   a) True or False
2) Int
   a) 1, 12, -46, 0
3) Float
   1. 1.24, 12.0, -0.5

“Complex”
1) List
   1) [True, False, 1, 12]
2) Hash
   a) [True:12, False:1]
3) String
   1. ‘hello how are you?’
Common Boolean operators

```python
x = 4  # not boolean
x == 4          True
x != 4          False
x > 4           False
x < 5           True
x >= 4          True
x <= 3          False
x > 2 and x < 5 True
x == 4 or x != 4 True
```
For loops let you repeat the same commands for each element in a list

```python
x = [1, 2, 3]
for i in x:
    print i
print ‘done!’
```

- `i` takes on the value of each element in the list for each repetition of the code within the for loop.
For loops also work for strings!

```python
x = 'hi!'
for i in x:
    print i
print 'done!'
```

```
h
i
!
done!
```
Compute the sum of the numbers in \( x \! \) !

\[
x = [1, 2, 4, 5]
\]

sum = 0
for v in x:
    sum = sum + v
print 'The sum is:', sum

The sum is: 12
Problem: output the product of the numbers in $x$

$x = [1, 2, 4, 5]$

def product(v):
    product = 1
    for v in x:
        product = product * v
    print('The product is:', sum x)

product = 40
• Homework will be assigned tomorrow via Catalyst

• Due Wednesday before class starts
Taking input strings from the terminal command line

pythonmyscript.pyapplebanana

#Thisismyscript.py
importsys
printsys.argv
printsys.argv[1]

[‘myscript.py’, ‘apple’, ‘banana’]
‘apple’
sys.argv will only contain strings, so to get numbers, you use float() or int()

```python
import sys
print sys.argv
print sys.argv[1]
['myscript.py', '1.0', '2']
'0.1'
print float(sys.argv[1])
1.0
```
Functions are sub-programs that you can call in one line

```python
>>> x = [1, 2, 3]
>>> print len(x)
3
```

- Used as a single word (no spaces) followed by “()”, where the input to the function goes within the parentheses
- The function will run, and it will be replaced by the output of the function

```python
>>> print len( ‘hello!’ )
6
```
Example problem 1!

\[
x = 4
\]

Write a program that takes as input

```python
if x == 5:
    print 'x is 5!'
else:
    print 'x is not 5!'
```

'x is not 5!'
If/else statements

\[
x = 4
\]

Only things that evaluate to a Boolean go here

```python
if x == 5:
    print 'x is 5!'
else:
    print 'x is not 5!'
```

'x is not 5!'
x = 4
if x == 5:
    print 'x is 5!'
elif x == 6:
    print 'x is 6!'
else:
    print 'x is neither 5 nor 6!'

'x is neither 5 nor 6!'
Example problem 1

\[ x = [ 1, 3, 2] \]

From a list \( x \) containing 3 numbers (looks like below but perhaps with different numbers), print the number with the highest value

\[
\text{highest_value} = x[0]
\]

if \( x[1] > \text{highest_value} \):
    \[
    \text{highest_value} = x[1]
    \]

if \( x[2] > \text{highest_value} \):
    \[
    \text{highest_value} = x[2]
    \]

print \( \text{highest_value} \)
Example problem 1

```python
x = [1, 3, 2]

From a list x containing 3 numbers (looks like below but perhaps with different numbers), print the number with the highest value

highest_value = x[0]
if x[1] > highest_value:
    highest_value = x[1]
if x[2] > highest_value:
    highest_value = x[2]
print highest_value
```
Example problem 2!

\[
x = 'atggataaccagg'
\]

print ‘Yes!’ if the first codon in the DNA sequence contained in x is a start codon, and ‘No!’ if it doesn’t

start_codon = ‘atg’
if start_codon == x[0:3]:
    print ‘Yes!’
Else:
    print ‘No!’
Example problem 3!

print ‘Yes!’ if the first codon in the DNA sequence given in the command line argument contained is a start codon, and ‘No!’ if it doesn’t

```python
start_codon = ‘atg’
if sys.argv[1] == start_codon:
    print ‘Yes!’
else:
    print ‘No!’
```
Example problem 4!

```
x = [12, 3, 4.4, 6]

Output how many numbers in the list x with values greater than 5

count = 0
for v in x:
    if v > 5:
        count = count + 1
print count
```

2
Converting between some data types

```
import sys
print sys.argv
```

```
['minus.py','3.3', '1.1']
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```
Converting between some data types

```python
import sys
print sys.argv
print float(sys.argv[1]) - float(sys.argv[2])
```

```
[‘minus.py’, ‘3.3’, ‘1.1’]
2.2
```
Summary

- **Variables** are objects that hold data
  - numbers, strings, lists, etc.

- **Operators** take objects, do something, and give you something else
  - ==, and, or, +, =

- **If/else statements** control the flow of the program based on Boolean objects derived from the data

- **For loops** repeat the same code for each value in a list