Genome Sciences 373
Genome Informatics

Quiz Section #1
March 28, 2017
About me

Email: cnoecker@uw.edu
Office hours: Mondays, 4:30-5:30 PM
Foege S-040
Or by appointment!

My research focuses on modeling to integrate different kinds of data from microbial communities
Quiz section goals

• solidify in-class material
• develop understanding of programming concepts
• learn basic Python to write bioinformatics programs

attendance is not required, but the material covered in section is required
Homework policy

No late homework accepted without prior arrangements.

Group work, Internet searching: You can (and should) use them, but don’t copy exactly! We can tell!

The point is to learn.

Grading is equally about your effort and your execution.
Homeworks continued

- First assignment will be assigned tomorrow via Catalyst

- Due Wednesday 4/5 before class (1:30PM)
  Start early!

Questions:
Catalyst discussion board:
https://catalyst.uw.edu/gopost/board/cnoecker/43929/
Email, office hours
Questions about course logistics?
Today’s goals

• Quick review on alignments
• Algorithms and programs: what and why
• Getting started programming in Python
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.

```
G   A   A   T   T   C   A   G   T   T   A
|     |     |     |     |     |     |
G G   A   T   C   G   -   -   A
```
What are some reasons to align sequences?
One big reason: compare *homologous* sequences

Sequences with shared ancestry

**Orthologs**

- Common Ancestor
  - Hemoglobin: G – A A T T C A G T T A
  - Hemoglobin: G G – A – T C – G – - A

**Paralogs**

- Common Ancestor
  - Hemoglobin: G – A A T T C A G T T A
  - Myoglobin: G G – A – T C – G – - A
Alignment example

Write down 2 possible alignments of the following two sequences:

ACCTTGT
TCTGTCC

Which one is “better”? Does it depend what you’re trying to do?
Algorithms
What is an algorithm?

A step by step list of instructions that, if followed exactly, will solve the problem under consideration

The instructions can be carried out or implemented in different ways:
- Programmed to be run by a computer
- Carried out yourself

Like a recipe!
Properties of algorithms

- Is an unambiguously defined series of steps
- Works for all inputs in a defined set
- Always produces a defined set of outputs, for those inputs
- Is guaranteed to produce a correct result, for those inputs

Often written in “pseudocode”
Example algorithm: Find the smallest number

Input: three numbers A, B, and C
Output: the largest number

current_smallest <- A
if B < current_smallest:
    current_smallest <- B
else:
    [do nothing]
if C < current_smallest:
    current_smallest <- C
else:
    [do nothing]
return current_smallest

What set of inputs is this algorithm defined for?
Which of these is an algorithm?

- Instructions for how to find the reverse complement of a DNA sequence
- A program that finds the reverse complement for any DNA sequence
Programming with Python
Why are we learning to program?

This class is designed for you to understand and use bioinformatics algorithms.

You won’t learn to implement all of them, but understanding them requires programmatic thinking.

Plus, if you do want to implement an algorithm or otherwise code anything, you will be off to a good start!
What is a program?

A series of instructions that performs a specific task when executed by a computer

Why are programs useful?
A note for those with programming experience

• Some of this will be review

• It’s fine to use Python tricks and modules beyond what I show in quiz section
  • But please don’t, for ex, use a BioPython function to do an entire homework problem in one command
What is a program?

A series of instructions that performs a specific task when executed by a computer.

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

- A variable is a container for data.
- An operator is the verb that assigns values to variables.
- The operator "=" assigns values to variables.
- A variable can be thought of as a box that now exists in memory.

$x = 4$

Now exists in memory!
Variables and operators

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

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

\[ x \]
\[ y \]
\[ 8 \]
\[ + \]
\[ = \]
\[ 12 \]

\[ z \]
\[ 12 \]
Let’s use Python!

1. Open a new text file 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

\[ x = 4 \]  # This is a line of code
\[ y = 8 \]  # This is another
\[ z = x + y \]  # z is the sum of x and y
# print(z)
Why are comments useful?

• For when you look back later

• If other people are trying to read, use, or understand your code
  • E.g. your grader!

• To help make sure your thinking is clear
You can also use Python interactively

• Open a terminal and type “python”
  • OR: Install Jupyter and open a notebook

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

>>> 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]
?```
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 = 'moo!'
```

```plaintext
>>> print x[0]
>>> print x[2]
>>> print x[-1]
```
Variables have *types*

- **Boolean**
  - True or False
- **Int**
  - 1, 12, -46, 0
- **Float**
  - 1.24, 12.0, -0.5
Variables have types

• List
  – [True, False, 1, 12]

• String
  – ‘hello how are you?’

• Hash/Dictionary
  – [True:12, False:1]
Common Boolean operators

```
x = 4  # not boolean! (assignment)
```

```
x  ==  4
x  !=  4
x  >  4
x  <=  3
x  >  2  and  x  <  5
x  ==  4  or  x  !=  4
```
We can use Boolean operators in If/else statements

```python
x = 4

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

Only things that evaluate to a Boolean go here

```python
if x == 5:
    print 'x is 5!'
else:
    print 'x is not 5!'
```
Review and practice problems

http://interactivepython.org/runestone/static/thinkcspy/index.html Sections 1, 2, 4

Homework on Catalyst tomorrow...