Quiz Today

Covers:

- Concepts from lab 1
- Material from lecture 1
- Reading from:
  - Model Chap 1 & App A [449-451]
  - Lutz (Learning Python) Chap 2

Time:
Programming for Biologists and Biochemists

Lecture 2: Manipulating Data

Brian Kidd
April 5, 2010
Overview

1. Housekeeping Items and Questions
2. Name Assignments
3. Basic I/O, PyDocs and Debugging
4. Review of Primitives
5. Compound Data Types: “Collections”
6. Project Preview
Overview

1. Housekeeping Items and Questions
2. Name Assignments
3. Basic I/O, PyDocs and Debugging
4. Review of Primitives
5. Compound Data Types: “Collections”
6. Project Preview
Access Check

Get on the Class email List:

Currently 45 / 58 signed up
Email TA: eyliaw@soe.ucsc.edu to get on class list!

Reading:

Access to the textbook? [online or hardcopy]
Access to other online material? [website, lecture slides, articles or books]

eCommons:

Does everyone have an account now?
Lab 1 Feedback

Overall people did very well!

Test your program!!!

Follow instructions

Label submitted code files correctly

Comment, comment, comment

Be careful recycling my comments

Remember: overview, inputs, & outputs

Good habit to include assumptions
Overview

1. Housekeeping Items and Questions
2. Name Assignments
3. Basic I/O, PyDocs and Debugging
4. Review of Primitives
5. Compound Data Types: “Collections”
6. Project Preview
Names

A Python name consists of an arbitrary number of letters, underscores, and digits.*

Examples:

myVariable
myVariable
my_VaRiaBle123

Avoid:

1 # digits alone not allowed
1myValue # digits in pos 1 not allowed
__myValue # special meaning (later)
Assigning Names

An assignment statement binds a name to an object

In general: \( \text{name} = \text{value} \)

Examples:

- \( \text{myValue} = 20 \)
- \( \text{my_Value} = 0.25 \)
- \( \text{my_value1} = "Hello kind Sir" \)
- \( \text{s} = "ATGGGCCTGAUG" \)

Important: a name is not a string!
Assigning Names II

Think of an assignment like creating a variable that will hold the value of the right hand side of =

Better Examples:

```python
numberOfAminoAcids = 20
percent_GC_content = 0.25
Greeting = "Hello kind Sir"
DNA_sequence = "ATGGGCCTGAUG"
```

Pick descriptive names
Be consistent with your format
Overview

1. Housekeeping Items and Questions
2. Name Assignments
3. Basic I/O, PyDocs and Debugging
4. Review of Primitives
5. Compound Data Types: “Collections”
6. Project Preview
Basic Input

Multiple types of input in Python

**Input functions**

**Arguments**

**Files**

**Examples**: 

```python
score = raw_input("Enter score: ")
numBases = input("Enter number of bases: ")
```

*Chapter 1, page 13.

**Python 2.x uses `raw_input` and `input`. Python 3.x uses `input`.**
Basic Output

The most basic output in Python is “print”

Statement versus function*

Examples:

print “Hello World”
print “number of bases is: “ + numBases

*Chapter 1, page 13.
Formatting Your Output

While Python doesn’t care about the format of your output, other people do

<table>
<thead>
<tr>
<th>ATGCTGGCCTGGGCGCGC</th>
<th>1 1 1</th>
<th>1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A T G C T G G C C T G G G C G C G C</td>
<td>3 9 27</td>
<td>3 9 27</td>
</tr>
<tr>
<td>ATG CTG GCC TGG GCG CGC</td>
<td>5 25 125</td>
<td>5 25 125</td>
</tr>
<tr>
<td>ATGCTG GCCTGG GC GCGC</td>
<td>10 100 1000</td>
<td>10 100 1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Number</th>
<th>Number</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.25</td>
<td>-1.250</td>
<td>150604000</td>
<td>1.5e+08</td>
</tr>
<tr>
<td>2.00001</td>
<td>2.000</td>
<td>2.578e+07</td>
<td>2.6e+07</td>
</tr>
<tr>
<td>250</td>
<td>250.000</td>
<td>188932930</td>
<td>1.9e+08</td>
</tr>
<tr>
<td>-0.095</td>
<td>-0.095</td>
<td>98483489</td>
<td>9.8e+07</td>
</tr>
<tr>
<td>.956</td>
<td>0.956</td>
<td>58.37e+06</td>
<td>5.8e+07</td>
</tr>
</tbody>
</table>
Formatting Your Output

While Python doesn’t care about the format of your output, other people do

Two ways to format your output

1. Simple old school*
2. Use `str.format()`**

*http://docs.python.org/library/stdtypes.html#string-formatting

**Chapter 3, pages 58-60.

**http://docs.python.org/library/stdtypes.html#str.format
### Simple old school

1. reformatting methods [strip, rjust, center] *

2. print "format" % (values)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.25</td>
<td>-1.250</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2.00001</td>
<td>2.000</td>
<td>3</td>
<td>9</td>
<td>27</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>250</td>
<td>250.000</td>
<td>5</td>
<td>25</td>
<td>125</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>-0.095</td>
<td>-0.095</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>.956</td>
<td>.956</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Python Documentation

http://docs.python.org

http://docs.python.org/tutorial/

http://docs.python.org/howto/

help()
Debugging

Easy: Remove syntax errors from the code

Hard: Ensure program executes correctly
Debugging in Python

Debuggers are programs for:

- Finding errors
- Checking the status of your code
- Verifying what a piece of code does

Debugging tools available in Python

- interpreter
- print statements*
- IDLE debugger
- Other integrated development environments

*quick and dirty approach, not recommended for projects
The IDLE Debugger

Provides several useful features for interactive debugging

- Single stepping through programs
- Setting *breakpoints* to pause execution at specific instructions
- View *local* and *global* variables
- View the stack
Graphical Debuggers

GUI*’s for the standard Python debugger

- Provide quick view into what’s going on in a complex program
- IDLE includes is a free graphical Python debugger
- Eclipse in another integrated development environment that includes many additional features for debugging**

*GUI = graphical user interface

**Check out the FAQ for instructions to get this
Debugging Example
Overview

1. Housekeeping Items and Questions
2. Name Assignments
3. Basic I/O, PyDocs and Debugging
4. Review of Primitives
5. Compound Data Types: “Collections”
6. Project Preview
Different Types of Data

Data comes in different *types*

- Simple types like numbers and strings
  - Python infers data type
    - *Boolean, Integer, Float, String*

- Collections of simples types
  - Python calls these *lists or tuples*
    - Encapsulated in [] or ()

- Associations between data
  - Python calls these *dictionaries*
    - Encapsulated in {}
Primitives

Simple values describe numbers and strings

7          # integer value
1.5        # float value
2e4        # float (scientific notation)
True       # boolean
“Hello”    # a string value can be
            # single ‘Hello’ or
            # double quotes “Hello”
Overview

1. Housekeeping Items and Questions
2. Name Assignments
3. Basic I/O, PyDocs and Debugging
4. Review of Primitives
5. Compound Data Types: “Collections”
6. Project Preview
Compound Data Types

While many data can be represented by individual values (“primitives”), we often need represent more complex data types.

**Examples:**
- DNA bases \{ A C G T \}
- amino acids \{ A C D E ... W Y \}
- genes in a genome \[ gene1, gene2, ... etc.\]
- protein-protein binding partners
- codon table
- restriction enzyme mapping
- multiple sequences

*Chapter 3, pages 47-72.*
Collections

Objects in a collection are called **items** or **elements**

Collections can:

- hold any number of items, including mixed items
- be ordered or unordered
- be mutable or immutable
Types of Collections

Sequences – ordered collection

- **strings** e.g. “VLSPADKTN”, **lists** e.g. [“EcoR1”, “BamHI”, “Xbal”],
- **ranges** e.g. `range(0, 10)`, **tuples** e.g. `(x, y, z)`

Sets – unordered, unique collection

bases of DNA or RNA, collection of IDs, etc.

Mappings – unordered collection of key/value pairs

codon tables, 3to1 letter tables, restriction enzyme maps

*range is a type in Python 3.x and a function in Python 2.x.*
Sequence versus Sequence

In Biology:

Part of a biomolecule (DNA, RNA, protein); more often refers to the abstraction thereof, as represented with letters

In Python:

A linear, and therefore numerically indexable, collection of values
Representing Sequences

Biological molecules can be represented symbolically* (e.g. DNA ➔ A, T, G, C)

Symbols can be “strung” together in sequences that are easy to manipulate** with a computer

*Symbols are models for molecules
**Sort, match patterns, change, etc.
Biological Sequence Primitives*

DNA is composed of 4 bases:

A – adenine  
C – cytosine  
G – guanine  
T – thymine*

Proteins are composed of 20 amino acids:

A – alanine  
C – cytosine  
D – aspartic acid  
E – glutamic acid  
F – phenyalanine  
G – glycine  
H – histidine  
I – isoleucine  
K – lysine  
L – leucine  
M – methionine  
N – asparagine  
P – proline  
Q – glutamine  
R – arginine  
S – serine  
T – threonine  
V – valine  
W – tryptophan  
Y – tyrosine

*RNA contains uracil (U) instead of thymine.
Sequences Represent Molecules

Information differs for each representation

Sequences are a simple, yet powerful representation
Biological Sequences as Strings

We will work with a number of biological sequences as **strings** Python

e.g. DNA sequence of hemoglobin is:
    ATG GTG CTG TCT CCT GCC GAC AAG

    e.g. protein sequence of hemoglobin is:
    VLSPADK TNVKA AWWGKVGAH
Sequences

Strings*
Immutable collection of ordered characters** in quotes ‘ ‘ or “ “

Ranges
Series of integers specified by (start, stop, step)

Tuples
Immutable collection of any object enclosed in parentheses ()

Lists
Mutable collection of any object enclosed in brackets []

*Straddle the line between primitive and compound
**Unicode is the standard text encoding.
Manipulating Strings

Python has many **operators**, **assignments**, **methods** and **functions*** for strings

For example:

- Operator – concatenate: `+`
- Assignment – binding: `=`
- Method – `.count()`
- Function – `len()`

*We'll talk more about functions in Week 4.
Operating on Strings

Concatenate strings:

polyA = "AAAA"
polyT = "TTTT"
pAT = polyA + polyT  # stores string "AAAATTTT"

Repeat strings:

scaleFactor = 3
polyGC6 = 'GC' * 6  # stores "GCGCGCGCGCGGC"
polyGC = 'GC' * scaleFactor  # stores "GCGCGC"
Augmented Assignments*

**Concatenate Assignment:**

```python
polyA = "AAAA"
polyT = "TTTT"
polyA += polyT  # stores string "AAAATTTT"
```

**Repeat Assignment:**

```python
scaleFactor = 3
sGC = 'GC'
sGC *= 6             # stores "GCGCGCGCGCGC"
sGC = 'GC'
sGC *= scaleFactor   # stores "GCGCGC"
```

*Chapter 2, page 23.*
Functions and Methods

A function performs an operation on arguments in parentheses and returns the result of that operation.

Function name

Number and type of arguments expected

\[
\text{FXN}( \ arg1, \ arg2, \ etc \ )
\]

name function arguments
Example Function

Get the length of a object:

```python
cpolyA = "AAAA"
len_polyA = len(polyA)  # stores the value 4
    name = len, len(object) -> integer
```

Get unicode number of a character:

```python
testChar = "A"
unicodeNumber = ord(testChar)  # gets 65
    name = ord, ord(c) -> integer
```
Functions and Methods

A method is almost identical to a function, except the first argument goes before the function name.

Type associated with the method

Function name

Number and type of arguments expected

\[ \text{arg1.FXN( arg2, arg2, etc )} \]

type name function arguments
Example Methods

Get the count of something in a string:

```python
DNAseq = "AAAATGAATGGCTAACTTTTGAA"
numA = DNAseq.count("A")  # returns the value 10
```

Convert to lowercase letters:

```python
DNAseq = "AAAATGAATGGCTAACTTTTGAA"
DNAseq.lower() # returns "aaaatgaatggctaactttttgaa"
```
Other Functions and Methods

Creating: `str()`, `ord()`, `bytes()`

Testing: `.isalpha()`, `.isnumeric()`, `.islower()`

Searching: `.find()`, `.index()`, `.count()`

Replacing: `.replace()`, `.translate()`

Changing Case: `.lower()`, `.upper()`, `.title()`

Reformatting: `.strip()`, `.ljust()`, `.center()`
Simple Bioinformatics Problem

Problem:
Suppose you got a sequence with an error that all “A”s need to be “T”s. What do you do?

Solution:

```python
orig_seq = "AAATTTGGGCCC";
new_seq = orig_seq.replace("A", "T")
```
String Methods Documentation

http://docs.python.org/library/stdtypes.html#string-methods
Ranges and Tuples

Ranges - flow control through iteration

Series of integers specified by \((\text{start}, \text{stop}, \text{step})\)

Tuples - points and print outs

Immutable collection of any object enclosed in parentheses \((\ )\)

Examples:

\[
\begin{align*}
\text{range}(5) & \quad \# \ 0,1,2,3,4 \\
\text{range}(1, 5) & \quad \# \ 1,2,3,4 \\
\text{range}(2, 12, 2) & \quad \# \ 2,4,6,8,10 \\
\text{tuple}(1.2, 3.0, 2.2) & \quad \# \ \text{e.g. coordinates} \\
\text{print} \quad \text{"%s %s" \% (var1, var2)}
\end{align*}
\]
Lists

The workhorse of datatypes
very common and useful type (superset of tuples)

Examples:
```python
all_natural = list('ACDEFGHIKLMNPQRSTVWY')
amino_acids = ['A', 'D', 'K', 'R']
codons = ['ATG', 'AUG', 'CTG']
empty_list = []
```
Assignment Modifications

Lists can be assigned new values based on slicing

Replacing:

amino_acid[0] = ‘E’

Insertion:

amino_acid[1:1] = ‘W’
Modification Methods

Methods can operate on lists, changing the list in the process

**Append:** `amino_acid.append(x*)` # adds to list

**Extend:** `amino_acid.extend(x)` # adds elements of x

**Insert:** `amino_acid.insert(i, x)` # insert x before i

**Remove:** `amino_acid.remove(x)` # first occurrence of x

**Pop:** `amino_acid.pop([i])` # remove last, can specify i

**Reverse:** `amino_acid.reverse()` # reverse list

**Sort:** `amino_acid.sort()` # sort by comparing elements

*x is a dummy object*
String and List Methods

Two important methods provide useful conversions between strings and lists

Split and Join

Examples:

"Hello World".split()

[“Hello”, “World”]

“DNA, RNA, Protein”.split(‘,’)

[“DNA”, “RNA”, “Protein”]


‘ACGT’

*x is a dummy object*
Sets

Unordered collection of objects that contains no duplicates

Examples:

\[
\text{dnaBases} = \{\text{‘A’, ‘C’, ‘G’, ‘T’}\}
\]
\[
\text{rnaBases} = \{\text{‘A’, ‘C’, ‘G’, ‘U’}\}
\]
\[
\text{IDs} = \{\text{‘CP0140.1’, ‘EF3613.1’, ‘EF3616.1’}\}
\]
\[
\text{emptySet} = ()
\]
\[
\text{nucleotides} = \text{set(‘TCAGTTAT’)}
\]
Set Operations I

Modifications

- .add()
- .remove()
- .discard()

Modifications

- .issubset()
- .issuperset()
- .isdisjoint()
Set Operations II

Intersections

Union

Difference

Symmetric Difference
Mappings

Unordered collection of key/value pairs, think dictionary*

*associative arrays, lookup tables, hash tables
Mappings

Unordered collection of key/value pairs, think dictionary*

Examples:
empty_dict = {}
ThreeToOne = {'Ala':'A', 'Cyt':'C'}

*associative arrays, lookup tables, hash tables

Chapter 3, pages 66-72.
Dictionary Operators

In Python, we’re looking at the keys when we consider operating

Operations:

- `d[key]`  # get value of key
- `d[key] = value`  # associate value to a key
- `del d[key]`  # delete key

Examples:

- `ThreeToOne[‘Ala’]`
- `ThreeToOne[‘Tyr’] = ‘Y’`
Dictionary Methods

We have methods for dealing with keys, values, and pairs

Methods:
- `d.get(key)`  # like `d[key]`, but no error
- `d.pop(key)`  # like `del d[key]`, but no error

- `d.keys()`
- `d.values()`
- `d.items()`
Overview

1. Housekeeping Items and Questions
2. Name Assignments
3. Basic I/O, PyDocs and Debugging
4. Review of Primitives
5. Compound Data Types: “Collections”
6. Project Preview
Final Project

All students enrolled in BIOL 180 or BME 160

Write a non-trivial Python program that can be used to investigate a biological research problem (think of developing a research tool)

Project chosen in collaboration with me and/or an outside faculty advisor

Deliverables

5–10 page report
5 minute oral presentation and demo
Deliverables

Report:

Introduce problem, why important and how programming will help

Description of programming concepts implemented and algorithms used to complete program

Write-up of results and discussion/interpretation
  - how well does the code perform
  - how could it be extended to other problems
Deliverables

Presentation:

Provide quick overview of problem

Describe one or two main challenges to solving this problem and how you overcame them

Demonstrate how your program works
Program to identify sequence motifs associated with human disease in alternative genes

Program to perform various statistical tests on a large dataset (microarray, facs populations, etc)

Program to perform various sequence manipulations and statistical gathering for individual and multiple sequences
Homework

Reading:

Model:

Chapter 4, pages 99-141
Appendix A, tables A3-A8, A10, A12-A13

Quiz:

Quiz 2 next Tues (4/12)
Covers readings, lectures, and lab

Pre Lab 2:

Email TA: eyliaw@soe.ucsc.edu to get on class list
Lab 2 Preview

Practice manipulating data types

1. in silico restriction digest
2. parse information from a FASTA header
3. compute geometry between atomic coordinates
4. create mapping tables for codons and amino acid letter conversion

Username/password: bme60/bme60