Genome Sciences 373
Genome Informatics
Quiz Section 2
April 7, 2015
Topics for today

Questions from lecture

Homework 1 due tomorrow 5pm
Homework 2 assigned tomorrow

Python overview: more data types
Questions about material from lecture

Can python lists have strings and numbers mixed together?

What are some ways of writing a newline to my program’s output?

How do I decide what scores to put in my alignment scoring matrix?
More python for beginners: comments, sets, dictionaries
Commenting for beginners

Your homework MUST HAVE COMMENTS

It’s OK to “over-comment”

Usually we put comments just above the part of the program we’re referring to

In-class example
Today: data types, flow control

Dictionaries

Sets

If/elif/else statements

The importance of indenting!
Useful data type: sets

• Sets usually get introduced “later on” when learning to program

• But, they are VERY useful in bioinformatics! So we’re jumping ahead.

• A “set” in python implements the mathematical concept of a set [In-class example]

```python
>>> my_list = [1, 1, 2, 2, 3, 3]
>>> my_set = set(my_list)
>>> my_list
[1, 1, 2, 2, 3, 3]
>>> my_set
set([1, 2, 3])
```
Working with sets

- `len(s)` – cardinality or size of set `s`.
- `x in s` – test `x` for membership in `s`.
- `s.issubset(t)` – test whether every element in `s` is in `t`.
- `s.issuperset(t)` – test whether every element in `t` is in `s`.
- `s.update(t)` – Update set by adding all elements in `t`.
- `s.add(e)` – Add `e` to set.
- `s.remove(e)` – Remove `e` from set (or `KeyError`) compare:
- `s.discard(e)` – Remove `e` from set if it exists.
- `s.clear()` – Remove all items.
• $s \mid t$ – new set with elements from both $s$ and $t$. (a.k.a. “UNION”)

• $s \& t$ – new set with elements common to $s$ and $t$. (a.k.a. “INTERSECTION”)

• $s - t$ – new set with elements in $s$ but not in $t$

• $s \wedge t$ – new set with elements in either $s$ or $t$ but not both
```python
0406 16:18 s020:~% python
Python 2.7.2 (default, Feb 28 2012, 08:29:13)
[GCC 4.4.6 20110731 (Red Hat 4.4.6-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> odds = set([1,3,5,7,9])
>>> evens = set([2,4,6,8,10])
>>> primes = set([2,3,5,7])

>>> print odds & primes
set([3, 5, 7])
>>> print evens & primes
set([2])
>>> print odds & evens
set(
)  # empty set
>>> print odds + evens
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'set' and 'set'
>>> print odds | evens
set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```
In class example:

State names
Dictionaries: pretty much what it sounds like

Like a printed dictionary maps words to definitions,

Python dictionaries map keys to associated values

You can quickly “look up” the “value” associated with a “key”

```
>>> capitals = {}
>>> capitals[“WA”] = “Olympia”
>>> capitals[“ID”] = “Boise”
>>> capitals[“AK”] = “Juneau”
```
Working with dictionaries

```python
0406 21:32 iris:~% python
Python 2.7.6 (default, Sep 9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> capitals = {}
>>> capitals['WA'] = 'Olympia'
>>> capitals['ID'] = 'Boise'
>>> capitals['AK'] = 'Juneau'

 capitals
{AK: 'Juneau', ID: 'Boise', WA: 'Olympia'}

>>> capitals['ID']
'Boise'

Note: "random" order

>>> capitals['IE']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'IE'

>>> for state, city in capitals.items():
...    print "The capital of", state, "is", city
...
The capital of AK is Juneau
The capital of ID is Boise
The capital of WA is Olympia
```
my_dict.get(k, default) – returns the value associated with k, or default if key k does not exist

my_dict.items() – returns all key: value pairs as an iterator

my_dict.keys() – returns all keys in the dictionary (in “random” order)

my_dict.values() – returns all values in the dictionary (“random”)

Values can be anything, even other dictionaries!

<In class example>
num = int(sys.argv[1])

if num > 0:
    print "input is greater than zero"
elif num < 0:
    print "input is less than zero"
else:
    print "input must be zero!"

The order of these MUST be if $\rightarrow$ elif$_n$ $\rightarrow$ else

But you only need "if" – the others are optional
num = int(sys.argv[1])

if num == 1:
    print "input is exactly 1"
elif num == 2:
    print "input is exactly 2"
elif num == 3:
    print "input is exactly 3"
elif num == 4:
    [...] 
else:
    print "input didn’t match anything I wanted!"
Doing more than one thing

```
num = int(sys.argv[1])

if num == 1:
    print "input is exactly 1"
    prime = False
    even = False
elif num == 2:
    print "input is exactly 2"
    prime = True
    even = True
else:
    print "didn’t get a 1 or a 2"
```
Doing more than one thing

```python
num = int(sys.argv[1])

if num == 1:
    print "input is exactly 1"
    prime = False
    even = False
elif num == 2:
    print "input is exactly 2"
    prime = True
    even = True
else:
    print "didn’t get a 1 or a 2"
```

a “block” of code defined by having the same indenting

another block
For loops: iterating over groups of things

Often you want to do something to every element of a group:

• Check every number to see if it’s less than some value

• Read the second column in every line of input

• Look at every key: value pair in a dictionary
Lists

```python
>>> my_list = [1, 2, 3, 4]
>>> for temporary_name in my_list:
    ...    print temporary_name * 2
...
2
2
4
4
6
6
8
8
>>> temporary_name
4
>>> for temporary_name in my_list:
    ...    print temporary_name * 2
...
2
2
4
4
6
6
8
8
```
my_dict = {'geneA': 4500, 'geneB': 5000, 'geneC': 2000}
for g, v in my_dict.items():
    if v > current_max:
        current_max = v
        top_gene_name = g

Note: three layers of indentation!
What is the value of `current_max`?

What is the value of `top_gene_name`?
Comparison operators: comparing values

Boolean: **and**, **or**, **not**

Numeric: `<`, `>`, `==`, `!=`, `>=`, `<=`

String: **in**, **not in**

<  is less than
>  is greater than
== is equal to
!=  is NOT equal to
<= is less than or equal to
>= is greater than or equal to
>>> seq = 'CAGGT'

```python
>>> if ('C' == seq[0]):
    ...
    print 'C is first in', seq
C is first in CAGGT
```

```python
>>> if ('CA' in seq):
    ...
    print 'CA is found in', seq
CA is found in CAGGT
```

```python
>>> if (('CA' in seq) and ('CG' in seq)):
    ...
    print "Both there!"
```

```python
>>>```