Genome Sciences 373
Genome Informatics

Quiz Section 3
April 14, 2015
Reminder:
Office hours Monday 2-3pm
Foege S-110
Topics for today

Questions from lecture

Homework 2 due tomorrow 5pm
Homework 3 assigned tomorrow

Python overview: dictionaries, loops
How can I count the repeating characters in a string?

my_dictionary = {}
my_string = “ACGATA”

for my_base in my_string:
    if
    else
How can I count the repeating characters in a string?

my_dictionary = {}
my_string = “ACGATA”

for my_base in my_string:
    if
        else
How can I iterate through all entries in a dictionary?

# print out only the characters there more than once
for my_key, my_value in my_dictionary.items():
    if my_value > 1:
        print "saw key %s more than once" % my_key

# an alternative way
for my_key in my_dictionary.keys():
    if my_dictionary[my_key] > 1:
        print "saw key %s more than once" % my_key
Python if/elif/else: are these different?

if (<test evaluates to true>):
    <execute this>
    <block of code>
elif (<different test evaluates to true>):
    <execute that>
    <block of code>

VS

if (<test evaluates to true>):
    <execute this>
    <block of code>
if (<different test evaluates to true>):
    <execute that>
    <block of code>
# assign some numbers for testing
x=1
y=2
z=3

# test if two statements are BOTH true
if (z > x) and (z!=y):
    <do something>

# test if one or both statements is/are true
if (x*x + y == z) or (y<=z):
    <do something>
Python comparison operators

and       both conditions are true
or        either or both conditions are true
not       negation

X in Y    is X substring/sublist of Y?
X not in Y negation of above
<         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
if (<test evaluates to true>):
    <execute this>
    <block of code>
    <The program continues with>
    <this block of code>
Python if/elif/else

if (<test evaluates to true>):
    <execute this>
    <block of code>
elif (<different test evaluates to true>):
    <execute this different>
    <block of code>

<The program continues with>
<this block of code>
if (<test evaluates to true>):
    <execute this>
    <block of code>
elif (<different test evaluates to true>):
    <execute this different>
    <block of code>
elif (<third test evaluates to true>):
    <execute this third>
    <block of code>

<The program continues with>
<this block of code>
if (<test evaluates to true>):
    <execute this>
    <block of code>
elif (<different test evaluates to true>):
    <execute this different>
    <block of code>
elif (<third test evaluates to true>):
    <execute this third>
    <block of code>
else:
    <all tests failed, so execute this>
    <block of code>
Python if/elif/else: combining tests

# assign some numbers for testing
x=1
y=2
z=3

# test if two statements are BOTH true
if (z > x) and (z!=y):
    <do something>

# test if one or both statements is/are true
if (x*x + y == z) or (y<=z):
    <do something>

Evaluation goes from left to right following rules of precedence
Math > [In]Equality > and/or/not

Use () to group things for ease of reading/debugging
Python loops: for loops

*For* loops allow you to perform an operation on each element in a list (or character in a string)

for `<element>` in `<object>`:
  `<execute this>`
  `<block of code>`
`<The program continues> #loop ended`<with this block of code>`
For loops allow you to perform an operation on each element in a list (or character in a string)

```python
for <element> in <object>:
    <execute this>
    <block of code>
<The program continues> #loop ended
<with this block of code>
```

```
Total_A=0
for my_character in "ACTTG":
    if my_character == "A":
        Total_A = Total_A + 1

# now my loop is done
print "I saw %d A’s in my string" % Total_A
```
For loops allow you to perform an operation on each element in a list (or character in a string)

```
for <element> in <object>:
    <execute this>
    <block of code>
<The program continues> #loop ended
<with this block of code>
```

Total_A=0
for my_character in “ACTTG”:
    if my_character == “A”:
        Total_A = Total_A + 1

# now my loop is done
print “I saw %d A’s in my string” % Total_A
Python for loops: getting out of the loop

Example code:

```python
for my_character in "ACGAT":
    <execute this>
<The program continues> #loop ended
```

At the end, all characters will have been visited. What if I want to stop if I see a G?

```python
for my_character in "ACGAT":
    if my_character == "G":
        break
<The program continues> #loop ended
```
Python for loops: skipping in a loop

Example code:

```python
for my_character in "ACGAT":
    <execute this>
<The program continues> #loop ended
```

At the end, all characters will have been visited. What if I want to skip all G’s?

```python
for my_character in "ACGAT":
    if my_character == "G":
        continue
    <do something to all non-G characters>
<The program continues> #loop ended
```

“continue” means: keep going with the loop, just skip *this* particular element. “break” means: stop the loop.
Python for loops: looping on a list

Example code:

```python
>>> for animal in ['cat', 'human', 'spider']:
...     print animal
...     cat
...     human
...     spider
>>>```
Example code:

```python
>>> for animal in ['cat', 'human', 'spider']:
...    print animal
...

  cat
human
spider
```
Python for loops: looping on a list

Example code:
>>> for animal in ['cat', 'human', 'spider']:
...     print animal
...
cat
human
spider
>>>
Python for loops: looping on a list

Example code:

```python
>>> for animal in ['cat','human','spider']:
...     print animal
...
cat
human
spider
>>> 
```

Iteration 3 – and finished
>>> matrix = [[12, 25], [0.3, 2.1], [-3, -1.8]]

>>> for my_row in range(0, 3):  # [0,1,2]
    ...        print 'row=', my_row

... for my_column in range(0, 2):  # [0,1]
    ...        print matrix[my_row][my_column]
Python for loops: handle a “matrix”

```python
>>> matrix = [[12, 25], [0.3, 2.1], [-3, -1.8]]
>>> for my_row in range(0, 3):  # [0,1,2]
...     print 'row=', my_row
...     for my_column in range(0, 2):  # [0,1]
...         print matrix[my_row][my_column]
...
row= 0
12
25
row= 1
0.3
2.1
row= 2
-3
-1.8
```
Python for loops: handle a “matrix”

```python
>>> matrix = [[12, 25], [0.3, 2.1], [-3, -1.8]]
>>> for my_row in range(0, 3):  # [0,1,2]
...     print 'row=', my_row
...     for my_column in range(0, 2):  # [0,1]
...         print matrix[row][column]
...
row= 0
12
25
row= 1
0.3
2.1
row= 2
-3
-1.8
```
```python
>>> matrix = [[12, 25], [0.3, 2.1], [-3, -1.8]]
>>> for my_row in range(0, 3):  # [0,1,2]
...    print 'row=', my_row
...    for my_column in range(0, 2):  # [0,1]
...        print matrix[my_row][my_column]

row= 0
12
25
row= 1
0.3
2.1
row= 2
-3
-1.8
```
Python for loops: handle a “matrix”

```python
>>> matrix = [[12, 25], [0.3, 2.1], [-3, -1.8]]
>>> for my_row in range(0, 3):  # [0,1,2]
...     print 'row=', my_row
...     for my_column in range(0, 2):  # [0,1]
...         print matrix[my_row][my_column]
```

```
row= 0
12
25
row= 1
0.3
2.1
row= 2
-3
-1.8
```
```python
>>> matrix = [[12, 25], [0.3, 2.1], [-3, -1.8]]

>>> for my_row in range(0, 3):  # [0,1,2]
...     print 'row=', my_row
...     for my_column in range(0, 2):  # [0,1]
...         print matrix[my_row][my_column]
... ```

```
row= 0
12
25
row= 1
0.3
2.1
row= 2
-3
-1.8
```
Python while loops

Example code:

```python
i = 0
while i < 5:
    print "i is still less than 5!"
    i += 1
<The program continues> #loop ended
```

Note that I did not have to explicitly exit the loop.
Example code:

```python
my_gene_file = open("my_genes.txt", "r")

total = 0
startsWithA = 0

while total < 100:
    total = total + 1
    line = my_gene_file.readline()
    if line.startswith("A"):
        startsWithA = startsWithA + 1

print "I have %d of 100 genes starting with A" % startsWithA
<The program continues> #loop ended
```
Example code:

```python
my_gene_file = open("my_genes.txt", "r")

total = 0
startsWithA = 0
while total < 100:
    total = total + 1
    line = my_gene_file.readline()
    if line.startswith("A"): 
        startsWithA = startsWithA + 1
    if startsWithA >= 10:
        print "Breaking out of the loop!"
        break

print "I have %d of 100 genes starting with A" % startsWithA
```

I can break out of while loops too!
Python while loops vs for loops: summary

**for** loops visit every element in a collection (list, string, etc)
- they exit automatically
- skip with “continue”, break out with “break”

**while** loops keep going forever until the test is false
- make sure your loop will eventually end before you run!
- break out with “break”
How to make a sequential list of numbers

**Common problem:**
You want a list of numbers from \( a \) to \( b \)

**Solution: range()**
range(start, stop, step)

**Example:**
>>> range(4, 10, 1)
[4, 5, 6, 7, 8, 9]

>>> range(4, 10, 2)
[4, 6, 8]

>>> range(4, 10)
[4, 5, 6, 7, 8, 9]

*start*: first number (inclusive)
*stop*: last number (exclusive)
*step*: how big is the jump?

The last argument (step) is optional! It defaults to 1.
How to make a sequential list of numbers

Example:

```python
>>> for i in range(1, 4, 1):
    ...     print i
    ...     print i * i
... 1
... 1
... 2
... 4
... 3
... 9
```
How to make a sequential list of numbers

Example:

```python
>>> for i in range(1, 4, 1):
...     print i
...     print i * i

1
1
1
2
2
4
4
3
3
9
9
```
How to make a sequential list of numbers

Example:

```python
>>> for i in range(1, 4, 1):
    ...     print i
    ...     print i * i
... 1
... 1
... 1
... 2
... 2
... 4
... 3
... 3
... 9
... 9
>>>  
```
How to make a sequential list of numbers

Example:

```python
>>> for i in range(1, 4, 1):
...     print i
...     print i * i
...
1
1
1
2
2

3
3

9
9

>>>```

Nested loops: loops inside loops

Example:

```python
>>> for i in range(1, 4, 1):
...     for j in range(5, 7, 1):
...         print "i is", i, "and j is", j
...
 i is 1 and j is 5
 i is 1 and j is 6
 i is 2 and j is 5
 i is 2 and j is 6
 i is 3 and j is 5
 i is 3 and j is 6
>>>```

Nested loops: loops inside loops

Example:

```python
>>> my_ex = [[1,2], [3,4], [5,6]]
>>> for my_small_list in my_ex:
...    print my_small_list
...    for my_number in my_small_list:
...        print my_number
```

What output should we see here?
In-class example: reading a matrix