`for` loops

Genome 559: Introduction to Statistical and Computational Genomics

Prof. James H. Thomas

`for` loop

- Allows you to perform an operation on each element in a list (or character in a string).

Variable name available inside loop

```
for <element> in <object>:
    <statement>
    <statement>
    ...
<statement>
```

Must be indented

block of code

Try it ...

```
>>> for name in ["Andrew", "Teboho", "Xian"]:
...   print "Hello", name
...
Hello Andrew
Hello Teboho
Hello Xian
>>>
```

Multiline blocks

- Each line must have the same indentation.

```
>>> for integer in [0, 1, 2]:
...    print integer
...    print integer * integer
...
0
0
1
1
2
4
```

Looping on a string

```
>>> DNA = 'AGTCGA'
>>> for base in DNA:
...     print "base =", base
...
base = A
base = G
base = T
base = C
base = G
base = A
```

Indexing

- Use an integer variable to keep track of a numeric index during looping.

```
>>> index = 0
>>> for base in DNA:
...     index = index + 1
...     print "base", index, "is", base
...
base 1 is A
base 2 is G
base 3 is T
base 4 is C
base 5 is G
base 6 is A
>>> print "The sequence has", index, "bases"
The sequence has 6 bases
```

The `range()` function

- The range() function returns a list of integers covering a specified range.

```
range([start,] stop [,step])
```

```
range(5)
[0, 1, 2, 3, 4]
range(2,8)
[2, 3, 4, 5, 6, 7]
>>> range(-1, 2)
[-1, 0, 1]
```

```
>>> range(0, 8, 2)
[0, 2, 4, 6]
>>> range(0, 8, 3)
[0, 3, 6]
>>> range(6, 0, -1)
[6, 5, 4, 3, 2, 1]
```

Using range() in a for loop

```
>>> for index in range(0,4):
...    print index, "squared is", index * index
...
0 squared is 0
1 squared is 1
2 squared is 4
3 squared is 9
```

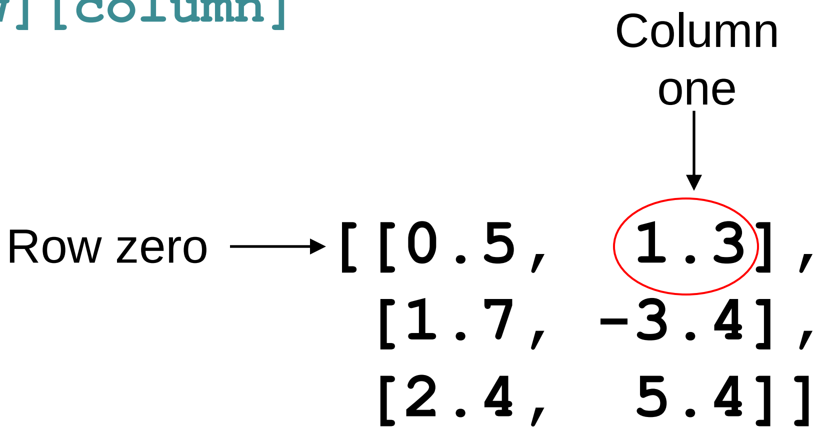
Nested loops

```
>>> for ix1 in [1, 2, 3]:
...     for ix2 in [4, 5]:
...         print ix1 * ix2
...
4
5
8
10
12
15
```

shorthand
for index2

Nested loops

```
>>> matrix = [[0.5, 1.3], [1.7, -3.4], [2.4, 5.4]]
>>> for row in range(0, 3):
...     print "row = ", row
...     for column in range(0, 2):
...         print matrix[row][column]
...
row =  0
0.5
1.3
row =  1
1.7
-3.4
row =  2
2.4
5.4
>>>
```



```
[[0.5,  1.3],
 [1.7, -3.4],
 [2.4,  5.4]]
```

Terminating a loop

- Break: Jumps out of the closest enclosing loop

```
>>> for index in range(0,3):
...     if (index == 1):
...         break
...     print index
...
0
```

Terminating a loop

- Continue: Jumps to the top of the closest enclosing loop

```
>>> for index in range(0, 3):
...     if (index == 1):
...         continue
...     print index
...
0
2
```

```
for <element> in <object>:
    <block>
```

Perform `<block>` for each element in `<object>`.

```
range(<start>, <stop>, <increment>)
```

Define a list of numbers. `<start>` and `<increment>` are optional.

`break` – Jump out of a loop

`continue` – Jump to the top of the loop

Sample problem #1

- Write a program `add-arguments.py` that reads any number of integers from the command line and prints the cumulative total for each successive argument.

```
> python add-arguments.py 1 2 3
1
3
6
> python add-arguments.py 1 4 -1
1
5
4
```

Solution #1

```
import sys
total = 0
for argument in sys.argv[1:]:
    integer = int(argument)
    total = total + integer
    print total
```

Sample problem #2

- Write a program `word-count.py` that prints the number of words on each line of a given file.

```
> cat hello.txt
Hello, world!
How ya doin'?
> python count-words.py
2
3
```

Solution #2

```
import sys
filename = sys.argv[1]
myFile = open(filename, "r")
myLines = myFile.readlines()
for line in myLines:
    words = line.split()
    print len(words)
myFile.close()
```

Sample problem #3

- Write a program `count-letters.py` that reads a file and prints a count of the number of letters in each word.

```
> python count-letters.py hello.txt
6
6
3
2
6
```

Solution #3

```
import sys
filename = sys.argv[1]
myFile = open(filename, "r")
myLines = myFile.readlines()
for line in myLines:
    for word in line.split():
        print len(word)
```

Challenge problem

Write a program `seq-len.py` that reads a file of fasta sequences and prints the name and length of each sequence and their total length.

```
>seq-len.py seqs.fasta
seq1 432
seq2 237
seq3 231
Total length 900
```

Here's what fasta sequences look like:

```
>foo
gatactgactacagttt
ggatatcg
>bar
agctcacggtatcttag
agctcacaataccatcc
ggatac
>etc...
```

('>' followed by name, newline, sequence on any number of lines until next '>')

Challenge problem solution

```
import sys
filename = sys.argv[1]
myFile = open(filename, "r")
myLines = myFile.readlines()
myFile.close()                # we read the file, now close it
cur_name = ""                 # initialize required variables
cur_len = 0
total_len = 0
first_seq = True              # special variable to handle the first sequence
for line in myLines:
   if (line.startswith(">")):  # we reached a new fasta sequence
      if (first_seq):          # if first sequence, record name and continue
         cur_name = line.strip()
         first_seq = False
         continue
      else:                       # we are past the first sequence
         print cur_name, cur_len  # write values for previous sequence
         total_len = total_len + cur_len     # increment total_len
         cur_name = line.strip()  # record the name of the new sequence
         cur_len = 0              # reset cur_len
   else:                 # still in the current sequence, increment length
      cur_len = cur_len + len(line.strip())
print “Total length", total_len
```