

for loops

Genome 559: Introduction to Statistical
and Computational Genomics

Prof. James H. Thomas

Reminders

- use `if - elif - else` statements for conditional code blocks
- code blocks share the same indentation
- indexing and slicing always use `[]` - e.g. `myString[0]`
- functions always use `()` - e.g. `len(myString)`

for loop

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

New variable name
available inside loop

Must already be
defined

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

Must be
indented

```
→ <statement>
```

```
→ <statement>
```

```
→ ...
```

} block of code

```
<statement> # unindented - loop end
```

Try it ...

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



Here we loop on a list

Another example

```
>>> 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  
>>>
```

think of the string as
a list of characters

Indexing inside loop

- If needed, use an integer variable to keep track of a numeric index during looping.

```
>>> index = 0    # initialize index
>>> for base in DNA:
...     index = index + 1    # increment index
...     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 increment operation is so common there is a shorthand: `index += 1`

The range () function

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

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

[optional arguments],
default to 0 and 1

```
>>>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]
```

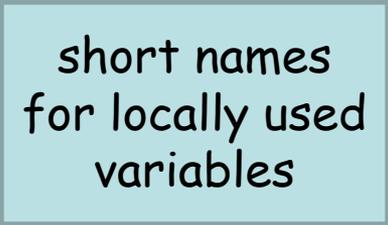
Using `range()` in a `for` loop

```
>>> 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
```

`range()` produces a list of integers (so this is really just like looping over a list)

Nested loops

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



short names
for locally used
variables

Nested loops

```
>>> index = 0
>>> for i in [1, 3, 5]:
    index += 1
    print 'outer loop ' + str(index) + ':'
...     for j in [4, 5]:
...         print i * j
...
outer loop 1:
4
5
outer loop 2:
12
15
outer loop 3:
20
25
```

Terminating a loop

- **break** jumps out of the closest enclosing loop

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

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
```

Summary

```
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, default to 0 and 1. Increment can be negative (go backwards with start > stop)

break – jump out of a loop

continue – jump to the top of the loop

You now know everything you need to know to write quite complex programs.

There's lots more to learn, but you could now (for example) write a sequence alignment program.

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 -3
```

```
1
```

```
5
```

```
4
```

```
1
```

Tip - remember that `sys.argv` is a list of command line strings.

Solution #1

```
import sys
total = 0 # initialize total
# for each argument, increment
# the total and print it
for argVal in sys.argv[1:]:
    total = total + int(argVal)
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
```

Don't worry about punctuation -
just assume white-space
separated strings are words

Solution #2

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

# alternative for loop
for i in range(0, len(fileLines)):
    words = fileLines[i].split()
    print len(words)
```

Sample problem #3 (harder)

Write a program `variance.py` that reads a specified BLOSUM score matrix file and computes the variance of scores for each amino acid. Assume the matrix file has tab-delimited text with the data as shown on the next page. You can download the example "matrix.txt" from the course web page.

```
> python variance.py matrix.txt
```

```
A    2.17
```

```
R    4.05
```

```
N    5.25
```

```
D    5.59
```

```
etc.
```

$$\text{var} = \frac{\sum (x - \mu)^2}{N - 1}$$

where x is each value, μ is the mean of values, and N is the number of values

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

(each line has 21 text fields separated by 20 tabs)

Solution #3

```
import sys
openFile = open(sys.argv[1], "r")
fileLines = openFile.readlines()
openFile.close()
varianceList = [] # make list for variances
aaList = [] # make list for amino acid names
for i in range(1, len(fileLines)): # skip the 0th line - single letter code
    fields = fileLines[i].strip().split('\t') # strip is precautionary
    scoreList = [] # list of scores for this line
    for j in range(1, len(fields)): # skip the 0th field
        scoreList.append(int(fields[j])) # convert to int and append
    scoreSum = 0
    for score in scoreList: # add all the scores to compute the mean
        scoreSum += score
    mean = float(scoreSum) / len(scoreList) # compute mean using float math
    squareSum = 0
    for score in scoreList: # compute the numerator of variance
        squareSum += (score - mean) * (score - mean)
    variance = float(squareSum) / (len(scoreList) - 1) # compute variance
    aaList.append(fields[0]) # append the aa code to list
    varianceList.append(variance) # append the variance to list
# now print the two lists out in parallel
for i in range(0, len(aaList)):
    print aaList[i] + '\t' + "%.2f" % varianceList[i]
```

This may seem complex, but each part of it is very simple. We will soon learn how to write functions, which would make this code much easier to read.

Solution #3 variant

```
import sys
openFile = open(sys.argv[1], "r")
fileLines = openFile.readlines()
openFile.close()
varianceList = [] # make list for variances
aaList = [] # make list for aa names
for i in range(1, len(fileLines)): # skip the 0th line - it has aa names
    fields = fileLines[i].strip().split('\t') # split line on tabs
    scoreList = [] # make a list of scores for this line
    for j in range(1, len(fields)): # scores start in field 1
        scoreList.append(int(fields[j]))
    scoreSum = 0
    for score in scoreList:
        scoreSum += score
    mean = float(scoreSum) / len(scoreList) # compute mean using float math
    squareSum = 0
    for score in scoreList: # compute the numerator of variance
        squareSum += (score - mean) * (score - mean)
    variance = float(squareSum) / (len(scoreList) - 1) # compute variance
    print fields[0] + '\t' + "%.2f" % variance
```

This is simpler because you print the values at the end of each loop iteration, rather than storing the values and printing them afterwards. HOWEVER, the previous version is more likely to be a useful part of a more complex program because the values get stored in an organized data structure (two parallel lists, one with the aa codes, one with the variances).

FYI - the first version written with a function

```
def variance(fields): # write once and forget
    scoreList = [] # list of scores for these fields
    for i in range(0, len(fields)):
        scoreList.append(int(fields[i]))
    scoreSum = 0
    for score in scoreList:
        scoreSum += score
    mean = float(scoreSum) / len(scoreList) # compute mean using float math
    squareSum = 0
    for score in scoreList: # compute the numerator of variance
        squareSum += (score - mean) * (score - mean)
    return float(squareSum) / (len(scoreList) - 1) # compute variance

import sys
openFile = open(sys.argv[1], "r")
fileLines = openFile.readlines()
openFile.close()
varianceList = [] # make list for variances
aaList = [] # make list for aa names
for i in range(1, len(fileLines)): # skip the 0th line
    fields = fileLines[i].strip().split('\t') # strip is precautionary
    aaList.append(fields[0]) # append the aa code to list
    varianceList.append(variance(fields[1:])) # append the variance to list
# now print the lists out in parallel
for i in range(0, len(aaList)):
    print aaList[i] + '\t' + "%.2f" % varianceList[i]
```

the core of this program is just the four bracketed lines - easy to read

Challenge problem

Write a program `seq-len.py` that reads a file of fasta format 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
```

```
agctcacggatatcttag
```

```
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")
fileLines = 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 fileLines:
    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 previous 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 cur_name, cur_len        # print the values for the last sequence
print "Total length", total_len
```

challenge - write this more compactly (e.g. you don't really need the `first_seq` flag)

Compact version

```
import sys
openFile = open(sys.argv[1], "r")
fileLines = openFile.readlines() # read file
openFile.close()
cur_name = "" # initialize required variables
cur_len = 0
total_len = 0
for line in fileLines:
    if (line.startswith(">")): # we reached a new fasta sequence
        if (cur_name == ""): # if first sequence, record name and continue
            cur_name = line.strip()
            continue
        else: # we are past the previous sequence
            print cur_name, cur_len # write values for previous sequence
            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 += len(line.strip())
print cur_name, cur_len # print the values for the last sequence
print "Total length", total_len
```

If you don't understand the solutions to problem 3 and the challenge problem, go over them carefully until they are crystal clear. Notice that each part is simple - it their organization that builds them into a complex program.

Work a problem from the inside out - e.g. decide what values you want to extract, then figure out how to extract them.

Use `print` to show intermediate values as you go (then remove the print statements).