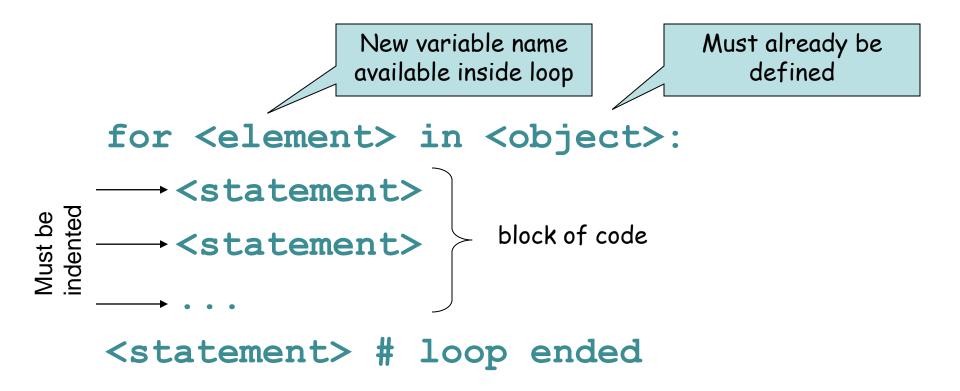
### for loops

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).



### Try it ...

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

### Reminder - multiline blocks

Each line in block must have the same indentation.

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

# 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

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

```
>>> index = 0 # initialize index
>>> for base in DNA:
\dots 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 <u>list of</u> integers covering a specified range.

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

[optional arguments], default to 0 and 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 i in [1, 2, 3]:
         for j in [4, 5]:
             print i * j
                                    short names
                                     for locally
                                    used indexes
10
12
15
```

### 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
>>>
```

### 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
```

### Summary

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

Define a list of numbers.
<start> and <increment> are optional, default to 0 and 1.
```

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
 python add-arguments.py 1 4 -1 -3
```

#### 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 (don't worry about punctuation).

```
> 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")
fileLines = myFile.readlines()
for line in fileLines:
   words = line.split()
   print len(words)
myFile.close()
# alternative loop form
for i in range(0, len(sys.argv)):
   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.

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

where x is each value,  $\mu$  is the mean of values, and N is the number of values

|   | Α  | R  | N  | D  | С  | Q  | Ε  | G  | Н  | I  | L  | K  | M  | F  | P  | S  | T  | W  | Υ  | V  |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Α | 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 |
| С | 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 |
| Ε | -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 |
| Н | -2 | 0  | 1  | -1 | -3 | 0  | 0  | -2 | 8  | -3 | -3 | -1 | -2 | -1 | -2 | -1 | -2 | -2 | 2  | -3 |
| l | -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 |
| Р | -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
fileLines = open(sys.argv[1], "r").readlines()
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
   scoreList = [] # 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
   aaList.append(fields[0]) # append the aa code to list
   varianceList.append(variance) # 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]
```

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
fileLines = open(sys.argv[1], "r").readlines()
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')
   scoreList = [] # 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)): # scores start in field 1
        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
fileLines = open(sys.argv[1], "r").readlines()
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
                             Here's what fasta sequences look like:
seq2 237
                             >foo
                             gatactgactacagttt
seq3 231
                             ggatatcg
Total length 900
                             >bar
                             agctcacggtatcttag
                             agctcacaataccatcc
                             ggatac
                             >etc...
                           ('>' followed by name, newline, sequence
```

on any number of lines until next '>')

```
import sys
                                Challenge problem solution
filename = sys.arqv[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
                                # we are past the previous sequence
     else:
        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
fileLines = open(sys.argv[1], "r").readlines() # read file
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).