

while loops

Genome 559: Introduction to Statistical
and Computational Genomics

Prof. James H. Thomas

Hints on variable names

- Pick names that are descriptive
- Change a name if you decide there's a better choice (use search and replace to be sure you don't miss any).
- Use the name to describe the type of object
- Very locally used names can be short and arbitrary

```
listOfFileLines = myFile.readlines()  
seqString = "GATCTCTATCT"  
myDPMatrix = [[0,0,0],[0,0,0],[0,0,0]]
```

```
intSum = 0  
for i in range(len(listOfInts)):  
    intSum = intSum + listOfInts[i]  
(more code)
```

Comment your code!

- Any place a # sign appears, the rest of the line is a comment (ignored by program).
- Blank lines are also ignored - use them to visually group code.

```
import sys
query = sys.argv[1]
myFile = open(sys.argv[2], "r")
lineList = myFile.readlines()    # put all the lines from a file into a list

# now process each file line to remove the \n character, then
# search the line for query and record each result in a list of ints
intList = []
for line in lineList:
    position = line.find(query)
    intList.append(position)
etc.
```

for loop review

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

} block of code

don't forget
the colon!

- `<element>` is a newly created variable name. You can access the variable only inside the loop.
- `<object>` is a container of 1 or more `<element>`s and it must already exist.
- `range()` will make a list of ints "on the fly"

```
for index in range(0,100):  
    <statement>
```

while loop

Similar to `for` loop

```
while <conditional test>:  
    <statement1>  
    <statement2>  
    . . .  
    <last statement>
```

While some test is `True` keep running the loop, exit when `False`.

The conditional test syntax is the same as for `if` and `elif` statements, and can be arbitrarily complex.

What does this program do?

```
sum = 0
count = 1
while count < 10:
    sum += count          # adjust sum
    count += 1           # increment count
print count              # should be 10
print sum                # should be 45
```

for vs. while

- you will probably use `for` loops more.
- `for` is natural to loop through a list, characters in a string, etc. (anything of determinate size).
- `while` is natural to loop an indeterminate number of times until some condition is met.

Examples of `for` loops

```
for base in sequence:
```

```
    <do something with each base>
```

```
for sequence in database:
```

```
    <do something with each sequence>
```

```
for index in range(5,200):
```

```
    <do something with each index>
```


Examples of `while` loops

```
while error > 0.05:
```

```
    <do something that will reduce error>
```

```
while score > 0:
```

```
    <traceback through a DP matrix, each  
    time setting the current score>
```

Warning: if you write a while loop and the conditional test stays True, the loop will run forever (infinite loop).

Reminder - comparison operators

Comparisons evaluate to True or False

- 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

Terminating a loop

`while` loops use `continue` and `break` in the same way as `for` loops:

- `continue` : jumps to the top of the enclosing loop
- `break` : breaks completely out of the enclosing loop

One of the arts of programming is seeing how to write elegant loops that do complex things (one simple step at a time).

It takes time and practice.

Shorthand reminder

```
x += 1
```

is the same as

```
x = x + 1
```

A common idiom in Python (and other languages). It's never necessary, but people use it frequently. Also works with other math operators:

```
x += y      # adds y to the value of x
x *= y      # multiplies x by the value y
x -= y      # subtracts y from x
x /= y      # divides x by y
```

program exit

In addition to accessing command-line arguments, the `sys` module has many other useful functions (look them up in the Python docs).

```
sys.exit()    # exit program immediately
```

In use:

```
import sys
# Make sure we got one argument on the command line.
if len(sys.argv) != 2:
    print "USAGE: one argument expected"
    sys.exit()
<argument count correct, continue 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 using a while loop.

```
> 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
i = 1
while i < len(sys.argv):
    total += int(sys.argv[i])
    print total
    i += 1
```


Sample problem #2

Write a program `count-fasta.py` that counts the number of fasta sequences in a file specified on the command line. Use either a `while` loop or a `for` loop.

Fasta format:

```
>identifier1 [optional comments]
AAOSIUBOASIUETOAISOBUAOSIDUGOAIBUOABOIUAS
AOSIU DTOAISUETOIGLKB JLZXC OITLJLBIULEI JLIJ } sequence on any number
>identifier2 [optional comments]                of lines until next line
TXDIGSIDJOIJEOITJOSIJOIGJSOIEJT SOE           that starts with ">"
>identifier3
Etc.
```

Two files are linked on the course web page - run your program on both: `small.fasta` and `large.fasta`

Solution #2

```
import sys
```

Not required, but a
good habit to get into

```
# Make sure we got an argument on the command line.  
if (len(sys.argv) != 2):  
    print "USAGE: one file argument required"  
    sys.exit()
```

```
# Open the file for reading.  
fasta_file = open(sys.argv[1], "r")  
lineList = fastaFile.readlines()  
fasta_file.close()  
num_seqs = 0  
for line in lineList:  
    # Increment if this is the start of a sequence  
    if (line[0] == ">"): # parentheses are optional  
        num_seqs += 1  
  
print num_seqs
```

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
```

Challenge problem solution 1

```
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 # mark that we are done with the first sequence
            continue
        else:                # we are past the first 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     # write the last values
print "Total length", total_len
```

Another solution (slightly more compact but has the disadvantage that it assumes the first line is a fasta name)

```
import sys

filename = sys.argv[1]
myFile = open(filename, "r")
myLines = myFile.readlines()
myFile.close()           # we read the file, now close it

cur_name = myLines[0]    # initialize required variables
cur_len = 0
total_len = 0
for index in range(1, len(myLines)):
    line = myLines[index]
    if (line.startswith(">")):    # we reached a new fasta 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())
    index += 1
print cur_name, cur_len      # write the last values

print "Total length", total_len
```

Here is a simpler solution, but it won't work if there are internal '>' characters, which are technically allowed in fasta names.

```
import sys
filename = sys.argv[1]
myFile = open(filename, "r")
whole_string = myFile.read()
myFile.close()
seqBlockList = whole_string.split(">") # split into blocks on ">"
total_len = 0
for seq in seqBlockList:
    lineList = seq.split("\n") # split the block on new lines
    length = len("".join(lineList[1:])) # get the length of the sequence
    total_len += length
    print lineList[0], length
print "Total length", total_len
```

What this does is split the text of the entire file on ">", which gives a list of strings (each containing the sequence with its name). Each of these strings is split at "\n" characters, which gives a list of lines. The 0th line in this list is the name, and the rest of the lines are sequence.

By the way, here is the challenge problem solution done using BioPython (which you may learn about later)

```
import sys
from Bio import Seq
from Bio import SeqIO

filename = sys.argv[1]
myFile = open(filename, "r")
seqRecords = SeqIO.parse(myFile, "fasta")
total_len = 0
for record in seqRecords:
    print record.name, len(record.seq)
    total_len += len(record.seq)
print "Total length", total_len
myFile.close()
```

shorter and much easier to write and understand