File input and output
`if-then-else`

Genome 559: Introduction to Statistical and Computational Genomics

Prof. James H. Thomas

Opening files

- The `open()` command returns a file object:

```
<file_object> = open(<filename>, <access type>)
```

- Python will read, write or append to a file according to the access type requested:
  - `'r'` = read
  - `'w'` = write
  - `'a'` = append

- Open for reading a file called "hello.txt":

```
>>> myFile = open("hello.txt", "r")
```

Reading the whole file

- You can read the entire content of the file into a single string. If the file content was the text "Hello, world!\n":

```
>>> myString = myFile.read()
>>> print myString
Hello, world!

>>>
```

why is there a blank line here?

Reading the whole file

- Now add a second line to your file ("How ya doin'?\n") and try again.

```
>>> myFile = open("hello.txt", "r")
>>> myString = myFile.read()
>>> print myString
Hello, world!
How ya doin'?

>>>
```

Reading the whole file

- Alternatively, you can read the file into a list of strings:

```
>>> myFile = open("hello.txt", "r")
>>> myStringList = myFile.readlines()
>>> print myStringList
['Hello, world!\n', 'How ya doin'?\n']
>>> print myStringList[1]
How ya doin'?
```

this file method returns a list of strings

Reading one line at a time

- The `readlines()` method puts <u>all</u> the lines into a list of strings.
- The `readline()` method returns the next line:

```
>>> myFile = open("hello.txt", "r")
>>> myString = myFile.readline()
>>> print myString
Hello, world!

>>> myString = myFile.readline()
>>> print myString
How ya doin'?
```

notice that `readline()` automatically keeps track of where you are in the file

Writing to a file

- Open the file for writing or appending:

```
>>> myFile = open("new.txt", "w")
```

- Use the `<file>.write()` method:

```
>>> myFile.write("This is a new file\n")
>>> myFile.close()
>>> Ctl-D (exit the python interpreter)
> cat new.txt
This is a new file
```

always close a file after you are finished reading from or writing to it.

`<file>.write()` is a little different from `print()`

- `<file>.write()` does not automatically append a new-line character.
- `<file>.write()` requires a string as input.

```
>>> newFile.write("foo")
>>> newFile.write(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: argument 1 must be string or read-only
   character buffer, not int
```

(also of course `print()` goes to the screen and `<file>.write()` goes to a file)

if-then-else

The `if` statement

```
>>> if (seq.startswith("C")):
...   print "Starts with C"
...
Starts with C
>>>
```

- A **block** is a group of lines of code that belong together.

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

- In the Python interpreter, the ellipse indicates that you are inside a block (on my Win machine it is just a blank indentation).
- Python uses <u>indentation</u> to keep track of blocks.
- You can use any number of spaces to indicate blocks, but you must be consistent. Using <tab> is simplest.
- An unindented or blank line indicates the end of a block.

The `if` statement

- Try doing an `if` statement without indentation:

```
>>> if (seq.startswith("C")):
... print "Starts with C"
  File "<stdin>", line 2
    print "Starts with C"
        ^
IndentationError: expected an indented block
```

Multiline blocks

- Try doing an `if` statement with multiple lines in the block.

```
>>> if (seq.startswith("C")):
...     print "Starts with C"
...     print "All right by me!"
...
Starts with C
All right by me!
```

When the `if` statement is true, all of the lines in the block are executed.

Multiline blocks

- What happens if you don't use the same number of spaces to indent the block?

```
>>> if (seq.startswith("C")):
...   print "Starts with C"
...    print "All right by me!"
  File "<stdin>", line 4
    print "All right by me!"
    ^
SyntaxError: invalid syntax
```

This is why I prefer to use the <tab> character – it is always exactly correct.

Comparison operators

- 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
```

Examples

```
seq = 'CAGGT'
>>> if ('C' == seq[0]):
...   print 'C is first'
...
C is first
>>> if ('CA' in seq):
...   print 'CA in', seq
...
CA in CAGGT
>>> if (('CA' in seq) and ('CG' in seq)):
...   print "Both there!"
...
>>>
```

Beware!

= versus ==

- <u>Single</u> equal assigns a variable name.
- <u>Double</u> equal tests for equality.

Combining tests

```
x = 1
y = 2
z = 3
if ((x < y) and (y != z)):
     do something
if ((x > y) or (y == z)):
     do something else
```

Evaluation starts with the innermost parentheses and works out

```
if (((x <= y) and (x < z)) or ((x == y) and not (x == z)))
```

`if-else` statements

```
if <test1>:
  <statement>
else:
  <statement>
```

- The `else` block executes only if `<test1>` is false.

```
>>> if (seq.startswith('T')):
...   print 'T start'
... else:
...   print 'starts with', seq[0]
...
starts with C
>>>
```

evaluates to FALSE

if-elif-else

```
if <test1>:
  <block1>
elif <test2>:
  <block2>
else:
  <block3>
```

Can be read this way:

if test1 is true then run block1, else if test2 is true run block2, else run block3

- `elif` block executes if `<test1>` is false and then performs a second `<test2>`
- Only one of the blocks is ever executed.

Example

```
>>> base = 'C'
>>> if (base == 'A'):
...   print "adenine"
... elif (base == 'C'):
...   print "cytosine"
... elif (base == 'G'):
...   print "guanine"
... elif (base == 'T'):
...   print "thymine"
... else:
...   print "Invalid base!"
...
cytosine
```

```
<file> = open(<filename>, r|w|a>
<string> = <file>.read()
<string> = <file>.readline()
<string list> = <file>.readlines()
<file>.write(<string>)
<file>.close()
```

- Boolean: `and, or, not`
- Numeric: `< , > , ==, !=, <>, >=, <=`
- String: `in, not in`

```
if <test1>:
  <statement>
elif <test2>:
  <statement>
else:
  <statement>
```

Sample problem #1

- Write a program **`read-first-line.py`** that takes a file name from the command line, opens the file, reads the first line, and prints the result to the screen.

```
> python read-first-line.py hello.txt
Hello, world!

>
```

Solution #1

```
import sys
filename = sys.argv[1]
myFile = open(filename, "r")
firstLine = myFile.readline()
myFile.close()
print firstLine
```

Sample problem #2

- Modify your program to print the first line without an extra new line.

```
> python read-first-line.py hello.txt
Hello, world!
>
```

Solution #2

```
import sys
filename = sys.argv[1]
myFile = open(filename, "r")
firstLine = myFile.readline()
firstLine = firstLine[:-1]
myFile.close()
print firstLine
```

Sample problem #3

- Write a program **`add-two-numbers.py`** that reads one integer from the first line of one file and a second integer from the first line of a second file and then prints their sum.

```
> add-two-numbers.py nine.txt four.txt
9 + 4 = 13
>
```

Solution #3

```
import sys
fileOne = open(sys.argv[1], "r")
valueOne = int(fileOne.readline()[:-1])
fileTwo = open(sys.argv[2], "r")
valueTwo = int(fileTwo.readline()[:-1])
print valueOne, "+", valueTwo, "=", valueOne + valueTwo
```

Sample problem #4 (review)

- Write a program **`find-base.py`** that takes as input a DNA sequence and a nucleotide. The program should print the number of times the nucleotide occurs in the sequence, or a message saying it's not there.

```
> python find-base.py A GTAGCTA
A occurs twice
> python find-base.py A GTGCT
A does not occur at all
```

Hint: `S.find('G')` returns -1 if it can't find the requested string.

Solution #4

```
import sys
base = sys.argv[1]
sequence = sys.argv[2]
position = sequence.find(base)
if (position == -1):
  print base, "does not occur at all"
else:
  n = sequence.count(base)
  print base, "occurs " + n + "times"
```

Challenge problems

Write a program that reads a sequence file (seq1) and a sequence (seq2) from command line arguments and makes output to the screen that either:

1) says seq2 is entirely missing from seq1, or
2) counts the number of times seq2 appears in seq1, or
3) warns you that seq2 is longer than seq1

```
>python challenge.py seqfile.txt GATC
>GATC is absent
(or
>GATC is present 7 times)
(or
>GATC is longer than the sequence in seqfile.txt)
```

Make sure you can handle multiline sequence files.

Do the same thing but output a list of all the positions where seq2 appears in seq1 (tricky with your current knowledge).

TIP – **`file.read()`** includes all the newline characters from a multiline file

Reading

- Chapters 5 and 14 from Downey