Week 6

Review; file processing

raw_input

`raw_input` : Reads a string from the user's keyboard.

– reads and returns an entire line of input

```
>>> name = raw_input("Howdy. What's yer name? ")
Howdy. What's yer name? Paris Hilton

>>> name
'Paris Hilton'
```

- to read a number, cast the result of `raw_input` to an `int`

```
>>> age = int(raw_input("How old are you? "))
How old are you? 53
>>> print("Your age is", age)
Your age is 53
```

if/else

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```

- Example:

```
gpa = input("What is your GPA? ")
if gpa > 3.5:
    print("You have qualified for the honor roll.”)
elif gpa > 2.0:
    print("Welcome to Mars University!”)
else:
    print("Your application is denied.”)
```

if … in

`if` **value** `in` **sequence**`:`
    statements

- The sequence can be a range, string, tuple, or list
- Examples:

```
x = 3
if x in range(0, 10):
    print("x is between 0 and 9”)

name = raw_input("What is your name? ")
name = name.lower()
if name[0] in "aeiou":
    print("Your name starts with a vowel!”)
```

Logical Operators

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

Operator	Example	Result
`and`	`(2 == 3) and (-1 < 5)`	`False`
`or`	`(2 == 3) or (-1 < 5)`	`True`
`not`	`not (2 == 3)`	`True`

python™

while Loops

```
while test:
    statements
```

```
>>> n = 91
>>> factor = 2          # find first factor of n

>>> while n % factor != 0:
...     factor += 1
...

>>> factor
7
```

python™

bool

- Python's logic type, equivalent to `boolean` in Java
  - `True` and `False` start with capital letters

```
>>> 5 < 10
True

>>> b = 5 < 10
>>> b
True

>>> if b:
...     print("The bool value is true”)
...
The bool value is true

>>> b = not b
>>> b
False
```

Random Numbers

`from random import *`

`randint(`**`min`**`, `**`max`**`)`

- returns a random integer in range [**min**, **max**] inclusive

`choice(`**`sequence`**`)`

- returns a randomly chosen value from the given sequence
  - the sequence can be a range, a string, ...

```
>>> from random import *
>>> randint(1, 5)
2
>>> randint(1, 5)
5
>>> choice(range(4, 20, 2))
16
>>> choice("hello")
'e'
```

Tuple

tuple_name = (**value**, **value**, **...**, **value**)

- A way of "packing" multiple values into one variable

```
>>> x = 3
>>> y = -5
>>> p = (x, y, 42)
>>> p
(3, -5, 42)
```

name, **name**, **...**, **name** = **tuple_name**

- "unpacking" a tuple's contents into multiple variables

```
>>> a, b, c = p
>>> a
3
>>> b
-5
>>> c
42
```

Tuple as Parameter/Return

```
def name( (name, name, ..., name), ... ):
    statements
```

– Declares tuple as a parameter by naming each of its pieces

```
>>> def slope((x1, y1), (x2, y2)):
...     return (y2 - y1) / (x2 - x1)

>>> p1 = (2, 5)
>>> p2 = (4, 11)
>>> slope(p1, p2)
3
```

```
return (name, name, ..., name)
```

```
>>> def roll2():
...     die1 = randint(1, 6)
...     die2 = randint(1, 6)
...     return (die1, die2)

>>> d1, d2 = roll2()
```

python™

File Processing

File Processing

A text file can be thought of as a sequence of lines

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/

Details: 
http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

Opening a file

- Before we can read the contents of the file, we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the open() function
- open() returns a “file object” - a variable used to perform operations on the file
- Similar to “File -> Open” in a Word Processor

Using open()

```
fhand = open('mbox.txt', 'r')
```

- handle = open(filename, mode)
- returns a file object use to manipulate the file
- filename is a string
- mode is optional and should be 'r' if we are planning to read the file and 'w' if we are going to write to the file

When Files are Missing

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
 FileNotFoundError: [Errno 2] No such
 file or directory: 'stuff.txt'
```

File processing

A text file has newlines at the end of each line

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008\n
Return-Path: <postmaster@collab.sakaiproject.org>\n
Date: Sat, 5 Jan 2008 09:12:18 -0500\n
To: source@collab.sakaiproject.org\n
From: stephen.marquard@uct.ac.za\n
Subject: [sakai] svn commit: r39772 - content/branches/\n
\n
Details:
http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772\
n
```

Reading the *whole* Files

name = `open("`**filename**`")`

– opens the given file for reading, and returns a file object

name`.read()` – file's entire contents as a string

```
>>> f = open("hours.txt")
>>> inp = f.read()
'123 Susan 12.5 8.1 7.6 3.2\n
456 Brad 4.0 11.6 6.5 2.7 12\n
789 Jenn 8.0 8.0 8.0 8.0 7.5\n'
```

```
print(len(inp))
85
```

```
print(inp[:20])
123 Susan 12.5 8.1 7
```

Line-based File Processing

name.`readline()` – next line from file as a string

- Returns an empty string if there are no more lines in the file

name.`readlines()` – file's contents as a list of lines

- (we will discuss lists in detail next week)

```
>>> f = open("hours.txt")
>>> f.readline()
'123 Susan 12.5 8.1 7.6 3.2\n'

>>> f = open("hours.txt")
>>> f.readlines()
['123 Susan 12.5 8.1 7.6 3.2\n',
'456 Brad 4.0 11.6 6.5 2.7 12\n',
'789 Jenn 8.0 8.0 8.0 8.0 7.5\n']
```

Line-based Input Template

- A file object can be the target of a `for` ... `in` loop
- A template for reading files in Python:

```
for line in open("filename"):
    statements
```

```
>>> for line in open("hours.txt"):
...     print(line.strip())    # strip() removes \n

123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

OOPS!

- What are all these blank lines doing here?
  - Each line from the file has a newline at the end
  - The print statement adds a newline to each line

```
From:
stephen.marquard@uct.ac.za\n
\n
From:
louis@media.berkeley.edu\n
\n
From: zqian@umich.edu\n
\n
From: rjlowe@iupui.edu\n
\n
...
```

Searching Through a File

- We can strip the whitespace from the right-hand side of the string using rstrip() from the string library

- The newline is considered “white space” and is stripped

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:') :
        print(line)
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
....
```

Searching Through a File

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:') :
        continue
    print(line)
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
....
```

Exercise

- Write a function `stats` that accepts a file name as a parameter and that reports the longest line in the file.
  - example input file, `vendetta.txt`:

```
Remember, remember the 5th of November.
The gunpowder, treason, and plot.
I know of no reason why the gunpowder treason
should ever be forgot.
```

  - expected output:

```
>>> stats(”vendetta.txt")
longest line = 46 characters
I know of no reason why the gunpowder treason
```

Exercise Solution

```
def stats(filename):
    longest = ""
    for line in open(filename):
        if len(line) > len(longest):
            longest = line

    print("Longest line =", len(longest))
    print(longest)
```

Writing Files

```
name = open("filename", "w")      # write
name = open("filename", "a")      # append
```

- opens file for write (deletes any previous contents) , or
- opens file for append (new data is placed after previous data)

`name.write(str)` – writes the given string to the file

`name.close()` – closes file once writing is done

```
>>> out = open("output.txt", "w")
>>> out.write("Hello, world!\n")
>>> out.write("How are you?")
>>> out.close()

>>> open("output.txt").read()
'Hello, world!\nHow are you?'
```

Exercise

- Write a function `remove_lowercase` that accepts two file names and copies the first file's contents into the second file, with any lines that start with lowercase letters removed.
  - example input file, `carroll.txt`:

```
Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,
Beware the JubJub bird and shun
the frumious bandersnatch.
```

  - expected behavior:

```
>>> remove_lowercase("carroll.txt", "out.txt")
>>> print(open("out.txt").read())
Beware the Jabberwock, my son,
Beware the JubJub bird and shun
```

Exercise Solution

```
def remove_lowercase(infile, outfile):
    output = open(outfile, "w")
    for line in open(infile):
        if not line[0] in "abcdefghijklmnopqrstuvwxyz":
            output.write(line)
    output.close()
```

python™

Exercise

- Write a function `statsCount` that accepts a file name as a parameter and that reports number of lines in the file.
  - example input file, `vendetta.txt`:

```
Remember, remember the 5th of November.
The gunpowder, treason, and plot.
I know of no reason why the gunpowder treason
should ever be forgot.
```

python™

Exercise

- Write a function `statsCount` that accepts a file name as a parameter and that reports number of lines that not contain 'the' in the file.
  - example input file, `vendetta.txt`:

```
Remember, remember the 5th of November.
The gunpowder, treason, and plot.
I know of no reason why the gunpowder treason
should ever be forgot.
```