CS A109
While Loops, File I/O, and Random Numbers

In this lecture we'll cover an assortment of programming constructs. They are all covered to some degree in chapter 10 of the textbook.

While Loop

The while loop takes a Boolean expression and if it is true, executes the intended block of code. It then retests the expression and if it is still true, the whole block is executed again. Eventually, if the code is designed properly, the expression will become false and the loop will exit and the next line after the while block is executed.

Syntax:

```
while (Boolean expression):
        statement1
        statement2
        …etc…
```

Here are some simple examples. The following prints the numbers from 1 to 4

```
x = 1
while (x < 5):
  print x
  x = x + 1
```

There are two types of while loops that we can construct. The first is a *count-based* loop, like the one we just used above. The loop continues incrementing a counter each time, until the counter reaches some maximum number. The second is an event-based loop, where the loop continues indefinitely until some event happens that makes the loop stop. Here is an example of an event-based loop:

```
def getInput(message):
  return swing.JOptionPane.showInputDialog(message)
```

…
in some other function:

```
i = 0
sum = 0
while (i <> -9999):
 i = int(getInput("Enter an integer, -9999 to stop"))
 if (i <> -9999):
     sum = sum + i
print "The total is " , sum
```

This loop will input a number and add it to sum as long as the number entered is not -9999. Once –9999 is entered, the loop will exit and the sum will be printed. This is an event-based loop because the loop does not terminate until some event happens – in this case, the special value of –9999 is entered. This value is called a *sentinel* because it signals the end of input. Note that it becomes possible to enter the sentinel value as data, so we have to make sure we check for this if we don't want it to be added to the sum.

What is wrong with the following code? Hint: It results in what is called an infinite loop.

```
x = 1
y = 1

while (x<=10):
     print y
     y=y+1
```

What would be the output of the following code?

```
def test():
   i = 1
   while i < 5 :
     s = ""
     j = 0
     while j < i :
        s = s + "*"
        j = j + 1
     print s
     i = i + 1
```

Nested loops are quite common, especially for processing tables of data.

File I/O

If you have stored data in a text file, using a program such as Notepad, you can also read it with pyhon. Reading from a text file is useful to load the program with large amounts of data that would otherwise be too tedious to type.

Here is how to open and read a text file:

1. Execute a statement of the form

```
fileVar = open(filepath, "rt")
```

where filepath identifies the file to be read, e.g. r"c:\myfile.txt". This statement establishes a communications link between the computer and the disk drive for reading

data from the disk. Data then can be input from the specified file and assigned to variables in the program. This assignment statement is said to "open the file for input."

We will reference the file on the disk using the variable "fileVar".

The "rt" means that we should read the file as text .

2. We can now read the file. Here are two general options for text file:

   a) Read the whole file in as a giant string. To do this:

      ```
      giantString = fileVar.read()
      ```

      giantString now contains the content of the entire file and could be output or processed in some way.

   b) Read the file one line at a time:

      ```
      oneline  = fileVar.readline()
      ```

      This reads in one line of text and stores it in the variable oneline. Each time readline is called, the next line of text is read from the file. If the end of the file is reached then the function returns an empty string.

      readline appends a newline (carriage return) at the end of each string. To get rid of it, you can use the strip command which removes all leading or trailing whitespace:

      ```
      oneline = oneline.strip()
           or
      oneline  = fileVar.readline().strip()
      ```

3. After the desired items have been read from the file, terminate the communications link set in Step 2 with the statement

   ```
   fileVar.Close()
   ```

As an example, suppose the following file is stored in C:\TEST.TXT:

```
hello
1
2
3
```

Here is a program that reads in everything into one string and outputs it:

```
def readAll():
   filevar = open(r"c:\test.txt","rt")
   s = filevar.read()
   print s
   filevar.close()
```

This will read in the whole file and output

```
hello
1
2
3
```

Here is a program that only reads in the first and second lines and outputs them:

```
def fileTest():
   filevar = open(r"c:\test.txt","rt")
   line1 = filevar.readline()
   print line1
   line2 = filevar.readline()
   print line2
   filevar.close()
```

Since readline adds a newline to the end of each line, the output is:

```
hello

1
```

To remove the blank line, use the strip function:

```
def fileTest():
   filevar = open(r"c:\test.txt","rt")
   line1 = filevar.readline().strip()
   print line1
   line2 = filevar.readline().strip()
   print line2
   filevar.close()
```

Here is another example that uses a while loop to read each line in and output it until the end of file is reached:

```
def fileOutputAll():
   filevar = open(r"c:\test.txt","rt")
   line = filevar.readline().strip()
   while (line <> ""):        # blank when end of file
     print line
     line = filevar.readline().strip()
   filevar.close()
```

Finally, here is an example that skips the first line, then adds together all remaining lines:

```
def fileSumNumbers():
   filevar = open(r"c:\test.txt","rt")
   line = filevar.readline().strip()  # Skip first line
   sum = 0
   while (line <> ""):
     line = filevar.readline().strip()  # Read in number
     if (line <> ""):
        val = int(line)
        sum = sum + val
   filevar.close()
   print "The sum of the numbers is " , sum
```

Try the examples out and make sure you understand how they are working!

Here is one more example to play MIDI notes stored in a file. Let's say the format for storing the notes is as follows:

# of notes to play, or N
<Code for note 1>
<Duration for note 1>
<Velocity for note 1>
<Code for note 2>
<Duration for note 2>
<Velocity for note 2>
…
<Code for note N>
<Duration for note N>
<Velocity for note N>

For example here is a file to play three notes, (60,500,127) (62,250,127) (63,250,127):

```
3
60
500
127
62
250
127
64
250
127
```

Here is a program to read the notes file in and play it:

```
def playSong(songFile):
   filevar = open(songFile,"rt")
   line = filevar.readline().strip()  # Number of notes
   numNotes = int(line)
   for i in range(1, numNotes+1):  # Read in exactly num+1 notes
     note = int(filevar.readline())   # Read and convert to int
     duration = int(filevar.readline())
     velocity = int(filevar.readline())
     playNote(note, duration, velocity)
   filevar.close()
```

Now if we want to change our song, we can just change the text file and don't have to mess with changing any code.

Random Numbers

Random number generation is briefly described on page 232 of the book. It is often useful to generate random numbers to produce simulations or games (or homework :)

The random object generates pseudo-random numbers. What is a pseudo-random number? It is a number that is not truly random, but appears random. That is, every number between 0 and 1 has an equal chance (or probability) of being chosen each time random() is called. (In reality, this is not the case, but it is close).

Here is a very simple pseudorandom number generator to compute the ith random #:

$R_i = (R_{i-1} * 7) \bmod 11$

Initially, we set the "seed", $R_0 = 1$. Then our first "random" number is 7 mod 11 or 7.
Our second "random" number is then (7*7) mod 11 or 5.
Our third "random" number is then (5*7) mod 11 or 3.
Our fourth "random" number is then (3*7) mod 11 or 10.
..etc.

As you can see, the values we get seem random, but are really not. This is why they are called pseudorandom. We can get slightly more random results by making the initial seed some variable number, for example, derived from the time of day. The particular function shown above would not be a very good pseudorandom number generator because it would repeat numbers rather quickly.

Here is an example of using python's random number generator.

1. At the top of your code, import the random module:

   ```
   import random
   ```

   This gives you access to the random number generator.

2. To generate a random integer *x*, where $min \leq x \leq max$, use:

   ```
   x = random.randint(min,max)
   ```

3. To generate a random float *f* where $0 \leq f < 1$, use:

   ```
   f = random.random()
   ```

Here is a short demonstration program that simulates rolling two six-sided dice:

```
def rolldice():
   die1 = random.randint(1,6)
   die2 = random.randint(1,6)
   return die1+die2
```

If we invoke the function we might get something like:

```
>>> rolldice()
7
>>> rolldice()
4
>>> rolldice()
11
```

And the numbers should be different each time.

Is the dice rolling function equivalent to the following?

```
def rolldice():
   return random.randint(2,12)
```

In-Class Exercise: The Monty Hall problem

You are a contestant on a game show and have won a shot at the grand prize. Before you are three doors. Behind one door is a new Mustang convertible and $1,000,000 in cash. Behind the other two doors are the booby prizes of dishwasher detergent. The location of the prizes is randomly selected. You want the car and the cash. The game show host asks you to select a door, and you randomly pick one. However, before revealing the contents behind your door, the game show host reveals one of the other doors that contains the booby prize. At this point, the game show host asks if you would like to stick with your original choice or switch your choice to the remaining door. What choice should you make to optimize your chances of winning the grand prize, or does it matter?

Write a computer program to simulate one run of the Monty Hall problem. Pick a random number from 1-3 to represent the door that holds the grand prize. Let the user choose one of the doors. The program should then display one of the doors that has the booby prize. Let the user pick another door (which could either be the same or the remaining door) and this time display what is behind the door the user selected. Run the program several times – is it better to switch, or does it matter?

Pseudocode:

1. Create a variable, **prize**, to store the door with the prize and initialize it to a random number from 1-3
2. Input the first door from the user
3. If the door is 1:
    a. See if door 2 is a booby prize. If so, display it. Otherwise see if door 3 is a booby prize. If so, display it.
4. If the door is 2:
    a. See if door 1 is a booby prize. If so, print it. Otherwise see if door 3 is a booby prize. If so, display it.
5. If the door is 3:
    a. See if door 1 is a booby prize. If so, print it. Otherwise see if door 2 is a booby prize. If so, display it.
6. Input a new choice from the user
7. If the **prize** variable is the same as the user's choice display "You win the car" otherwise display "You win detergent"

You might note that steps 3-5 could be better implemented in a function!
Here is working code. The printNow function is used to output the data immediately to the console (the print statement doesn't output until the function exits).

```
def montyhall():
  prize = random.randint(1,3)
  choice = int(getInput("Pick a door from 1-3."))
  if (choice == 1):
    if (prize <> 2):
       printNow("I reveal door 2 to contain detergent.")
    else:
       printNow("I reveal door 3 to contain detergent.")
  if (choice == 2):
    if (prize <> 1):
       printNow("I reveal door 1 to contain detergent.")
    else:
       printNow("I reveal door 3 to contain detergent.")
  if (choice == 3):
    if (prize <> 1):
       printNow("I reveal door 1 to contain detergent.")
    else:
       printNow("I reveal door 2 to contain detergent.")
  choice = int(getInput("Which door now?"))
  if (choice == prize):
     print "You win the car!"
  else:
     print "Sorry, you win detergent."
```

As one final example, let's instrument our solution to be a simulation where we'll play the game 1000 times and see how often we win by switching and how often we lose by

switching. This type of simulation can tell us what the probability of winning will be for the two choices.

Since the door location for the prize will be random, we don't lose any generality by always picking door #1. Here is a code solution that automatically plays the game 1000 times. The print statements have been removed so the game will run faster:

```
def montyhall():
  winswitch = 0      # Num of wins if we switch doors
  winstay = 0        # Num of wins if we stay with first choice
  for i in range(1,1001):   # Play 1000 times
   prize = random.randint(1,3)
   choice = 1    # We initially pick door 1
   if (choice == 1):
     if (prize <> 2):
       switch = 3        # Door 2 would be revealed
                         # We would switch to the closed door
                         # which is door 3
     else:
       switch = 2        # Door 3 revealed
   if (choice == 2):
     if (prize <> 1):
        switch = 3
     else:
        switch = 1
   if (choice == 3):
     if (prize <> 1):
        switch = 2
     else:
        switch = 1
   if (choice == prize):
      winstay = winstay + 1         # Count number of wins
   if (switch == prize):
      winswitch = winswitch + 1
  print "If you switched, you would win " , winswitch , " times."
  print "If you stayed, you would win ", winstay , " times."
```