CS177 Python Programming

Recitation 6:
Loop Structures and Booleans
Before starting, let’s review

- Variables: “mutable” vs “non-mutable”
- Garbage Collection
- How we pass arguments to a function
Mutable vs Non-Mutable

• Mutable Variables:
  – lists
• Non-Mutable Variables:
  – int, float, string

What does it mean?
Non-Mutable Variables: int, float

```python
def main():
    x = 5.5
    y = 10
    x = x + y
```

What happens with the first 5.5?
The Garbage Collection process will eventually remove it when there is no any variable referencing to it any more.
Non-Mutable Variables: Strings

def main():
    s1 = "he"
    s2 = "lllo"
    s1 = s1 + s2

What happens with the first “he”?

The Garbage Collection process will eventually remove it when there is no any variable referencing to it any more.
Mutable Variables: Lists

def main():
    x = [10,20,30]
    x[1] = x[1] + 10

Memory

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>
Mutable Variables: Lists

```python
def main():
    x = [10, 20, 30]
    x[1] = x[1] + 10
```

Memory:

```
[0] 10
[1] 30
[2] 30
```
Mutable Variables: Lists

```python
def main():
    x = [10, 20, 30]
    x[1] = x[1] + 10
```

See the difference

- Values of a list are continuously stored in memory
- Changes are stored in the same memory location
- **Garbage Collection** mechanism does not operate here
How does it affect passing arguments to functions

• Mutable Variables:
  – Changes done to the variable in the called function will be reflected in the caller function

• Non-Mutable
  – Changes done to the variable in the called function will NOT be reflected in the caller function

Let’s see why
Passing Non-Mutable Variables

```python
def test(y):
    y = y + 5

def main():
    x = 5
    test(x)
    print(x)
```

Notice:
- In `test()` changes to the `y` variable will be stored in a new position. The `x` variable in `main` will continue referencing to the number 5.
- The `print (x)` statement in `main` will generate 5.
Passing Non-Mutable Variables

Would it be different if the parameter of test() is called \( x \) instead of \( y \)?

```python
def test(x):
    x = x + 5
def main():
    x = 5
    test(x)
    print(x)
```

**Memory**

- X of test()
- X of main()
- 10
- 5

**NO...** `print(x)` in `main()` will generate 5
Would it be different if the parameter of test() is called x instead of y?

```python
def test(x):
    for i in range(len(x)):
        x[i] = x[i] + 10

def main():
    x = [10, 20, 30]
    test(x)
    print(x)
```

Memory

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>[0]</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>[1]</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>[2]</td>
</tr>
</tbody>
</table>
Passing Mutable Variables

Would it be different if the parameter of test() is called \texttt{x} instead of \texttt{y}?

```
def test(x):
    for i in range (len(x)):
        x[i] = x[i] + 10

def main():
    x = [10, 20, 30]
    test(x)
    print(x)
```

Memory

\begin{array}{c}
\text{X of main()} \\
20 \\
30 \\
40 \\
\text{X of Test()} \\
\end{array}

print(x) in main will generate [20, 30, 40]
Today’s topics

• Exception Handling (Chapter 7)
• Chapter 8:
  – Loop Structures
  – Booleans
Exception Handling

• Let’s consider a program that solve quadratic equations? Let’s call it quadratic solver.

• What can go wrong at first glance?

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

This can be negative and produce an `ValueError`

What can you do then?
Exception Handling

- You might use decision structures and check that the value is non-negative before using `math.sqrt()`
Exception Handling

```python
# quadratic4.py
import math

def main():
    print "This program finds the real solutions to a quadratic\n"

    a, b, c = input("Please enter the coefficients (a, b, c): ")

    discrim = b * b - 4 * a * c
    if discrim < 0:
        print "\nThe equation has no real roots!"
    elif discrim == 0:
        root = -b / (2 * a)
        print "\nThere is a double root at", root
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print "\nThe solutions are:\", root1, root2
```
Exception Handling

What is wrong with this mechanism?

• This is “old-fashion” manner. Programming languages such as C require this.

• You can have a program that will need checking too many special cases… you will code a lot just to avoid errors

• There can be many causes of errors… How to know what to check?
Exception Handling

• Python includes in its design an Exception Handling mechanism to solve this limitation

First:
Do what is specified in <body>

```python
try:
    <body>
except <ErrorType>:
    <handler>
```

Second:
If any problem crops up, handle it as specified in <handler>

Let’s see how to use try...except with the same example
Exception Handling

```python
# quadratic5.py
import math

def main():
    print("This program finds the real solutions to a quadratic")
    try:
        a, b, c = input("Please enter the coefficients (a, b, c): ")
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("The solutions are:", root1, root2)
    except OverflowError:
        print("No real roots")
```

The Magic Words: Here we first attempt and in case of error we print a message
Exception Handling

What is nice with try…except?

• It can be used to catch any kind of error
• Besides error caused for using math.sqrt() with a negative number (ValueError) we have:
  – User fails to type correct inputs (different type of ValueError)
  – If user types an identifier instead of a number (NameError)
  – If input is not a valid Python expression (SyntaxError)
  – If user types a valid Python expression that produces non-numerical results (TypeError)
```python
# quadratic6.py
import math

def main():
    print "This program finds the real solutions to a quadratic\n"

    try:
        a, b, c = input("Please enter the coefficients (a, b, c): ")
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print "\nThe solutions are: ", root1, root2
    except OverflowError:
        print "\nNo real roots"
    except ValueError:
        print "\nYou didn’t give me three coefficients."
    except NameError:
        print "\nYou didn’t enter three numbers"
    except TypeError:
        print "\nYour inputs were not all numbers"
    except:
        print "\nSomething went wrong, sorry!"
```

The Magic Words dealing with several types of errors.
Loop Structures

Definite Loop:

```
for <var> in <sequence>:
  <body>
```

Example:
```
for i in range(10):
  print(i)
```

Here, it will iterate through a pre-defined sequence.

Indefinite Loop:

```
while <condition>:
  <body>
```

Example:
```
i = 0
While i < 10:
  print(i)
  i = i + 1
```

Here, it will iterate while the condition is True.

Note: if i is not incremented it will loop for ever.
Loop Structures

Definite Loop:

```python
for <var> in <sequence>:
    <body>
```

Results:

```python
>>> for i in range(10):
    print(i)
0
1
2
3
4
5
6
7
8
9
```

Indefinite Loop:

```python
while <condition>:
    <body>
```

Results:

```python
>>> i = 0
>>> while i < 10:
    print(i)
    i = i + 1
0
1
2
3
4
5
6
7
8
9
```
Loop Structures

Let’s define a problem to be able to compare:

Problem: Find the average of series of numbers entered by the user

\[
\overline{X} = \frac{X_1 + X_2 + X_3 + \ldots + X_n}{n}
\]
Loop Structures

Solution to the problem using **for**:

**Algorithm:**

Input the count of the numbers, \( n \)
Initialize sum to 0
Loop \( n \) times
  - Input a number, \( x \)
  - Add \( x \) to sum
Output average as \( \text{sum} / n \)

**Output**

How many numbers do you have? 5
Enter a number >> 32
Enter a number >> 45
Enter a number >> 34
Enter a number >> 76
Enter a number >> 45

The average of the numbers is 46.4

**Code:**

```python
# averagel.py

def main():
    n = input("How many numbers do you have? ")
    sum = 0.0
    for i in range(n):
        x = input("Enter a number >> ")
        sum = sum + x
    print \nThe average of the numbers is", sum / n
```

Here, the value of \( n \) is entered before looping
Loop Structures

Solution to the problem using **interactive loop with while**:

**Algorithm:**

```
set moredata to "yes"
while moredata is "yes"
    get the next data item
    process the item
    ask user if there is moredata
```

**Code:**

```python
# average2.py

def main():
    sum = 0.0
    count = 0
    moredata = "yes"
    while moredata[0] == "y":
        x = input("Enter a number >> ")
        sum = sum + x
        count = count + 1
        moredata = raw_input("Do you have more numbers (yes or no)? "")
    print "\nThe average of the numbers is", sum / count
```

**Output**

```
Enter a number >> 32
Do you have more numbers (yes or no)? yes
Enter a number >> 45
Do you have more numbers (yes or no)? y
Enter a number >> 34
Do you have more numbers (yes or no)? y
Enter a number >> 76
Do you have more numbers (yes or no)? y
Enter a number >> 45
Do you have more numbers (yes or no)? nope
The average of the numbers is 46.5
```
Loop Structures

Solution to the problem using sentinel loop with while:

Algorithm:

- get the first data item
- while item is not the sentinel
- process the item
- get the next data item

Output

Here one of the input values entered by the user is checked in the condition

```
# average3.py

def main():
    sum = 0.0
    count = 0
    x = input("Enter a number (negative to quit) >> ")
    while x >= 0:
        sum = sum + x
        count = count + 1
        x = input("Enter a number (negative to quit) >> ")
    print "\nThe average of the numbers is", sum / count
```

Value entered by the user (no question required)
Loop Structures

What can go wrong with interactive loop and sentinel loop?

- What if a user makes a mistake when entering the 98th number to be averaged?

- Does he have to start over?

- What if the user writes all the values in a file (one number per line) and just reads the file and computes the average. Let's see this implementation with both for and while
Loop Structures

Solution to the problem using **file loops**: 

File with the numbers to be averaged

```
43
71
95
44
14
22
62
99
11
4
```

List_Numbers.txt
Loop Structures

Solution to the problem using **file loop with for**:

```python
# average5.py

def main():
    fileName = raw_input("What file are the numbers in? ")
    infile = open(fileName, 'r')
    sum = 0.0
    count = 0
    for line in infile.readlines():
        sum = sum + eval(line)
        count = count + 1
    print "\nThe average of the numbers is", sum / count
```

- Read name of the file from user
- Remember `readlines()` reads a file as a list of strings where each element is a line of the file. In this case just one of the numbers to be averaged
Loop Structures

Solution to the problem using **file loop with while**:

```python
# average6.py

def main():
    fileName = raw_input("What file are the numbers in?")
    infile = open(fileName, 'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        sum = sum + eval(line)
        count = count + 1
        line = infile.readline()
    print "\nThe average of the numbers is", sum / count
```

- Read name of the file from user
- `readline()` reads next line of the file as a string. In this case just one of the numbers to be averaged
Loop Structures

What would happen if the file contains several numbers separated by comma in the same line?

• We could treat each line as a sub-file and apply any of the algorithms previously seen.
• How? Using Nested Loops
Loop Structures

Solution to the problem using **nested loops:**

File with the numbers to be averaged

43, 71, 67,
95, 44, 14, 22, 62, 99
11, 4, 15, 48, 29, 37
55, 45, 66
...
.
.
.

List_Numbers.txt
Loop Structures

Solution to the problem using nested loops:

```python
# average7.py
import string

def main():
    fileName = raw_input("What file are the numbers in? ")
infile = open(fileName, 'r')
sum = 0.0
count = 0
line = infile.readline()
while line != ":
    # update sum and count for values in line
    for xStr in string.split(line):
        sum = sum + eval(xStr)
        count = count + 1
    line = infile.readline()
print "\nThe average of the numbers is", sum / count
```
Loop Structures

Other common patterns

repeat
  get a number from the user
until number is >= 0

Two ways to implement it:

Way 1:

```python
number = -1  # Start with an illegal value to get into the loop.
while number < 0:
    number = input("Enter a positive number: ")
```

Way 2: Known as Loop and A Half

```python
while 1:
    number = input("Enter a positive number: ")
    if x >= 0: break  # Exit loop if number is valid.
```
Booleans

Let’s say we want to know if two points objects are in the same position?

Instead of this:
```python
if p1.getX() == p2.getX():
    if p1.getY() == p2.getY():
        # points are the same
    else:
        # points are different
else:
    # points are different
```

We do:
```python
if p1.getX() == p2.getX() and p2.getY() == p1.getY():
    # points are the same
else:
    # points are different
```

One of the boolean operators. In addition we will see OR and NOT
Booleans

Remember, in a previous class we saw the True Tables of the boolean operators

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
<td>P and Q</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
<td>P or Q</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>not P</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

**SUMMARY:**

AND: True when **both** P and Q are **True**

OR: True when **any or both** are **True**. Also you can remember that it is False when both P and Q are False.
Booleans

You can use algebra to remember some results

<table>
<thead>
<tr>
<th>Algebra</th>
<th>Boolean algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \times 0 = 0$</td>
<td>$a \text{ and false} \equiv \text{false}$</td>
</tr>
<tr>
<td>$a \times 1 = a$</td>
<td>$a \text{ and true} \equiv a$</td>
</tr>
<tr>
<td>$a + 0 = a$</td>
<td>$a \text{ or false} \equiv a$</td>
</tr>
</tbody>
</table>

**SUMMARY:**
False = 0
True = 1
AND = $\times$ (Multiplication)
OR = $+$ (Addition)
$a = \text{our boolean variable}$
Booleans

How the conversion of values is evaluated?

```python
>>> bool(0)
False
>>> bool(1)
True
>>> bool(32)
True
>>> bool("hello")
True
>>> bool(""")
False
>>> bool([1,2,3])
True
>>> bool([])
False
>>> |
>>> while 32:
    print(3)
```

The `while()` statement includes an implicit cast `boolean()` to any specified condition.
Thank you!