Introduction to Computer Programming

Lecture #3 - Looping Around Loops
I: Counting Loops

Why loops?

• Computers offer several advantages over calculators.
• If it is necessary, they can perform the same steps over and over again, simply by rerunning the program.
• But is this the only way to get a computer to perform the same action repeatedly? And is this the only reason for getting a computer to repeat itself?
Example: Average of three numbers

• Let’s take another look at our program that finds the average of three numbers:

```python
value1 = input("What is the first value?")
value1 = int(value1)

value2 = input("What is the second value?")
value2 = int(value2)

value3 = input("What is the third value?")
value3 = int(value3)

sum = value1 + value2 + value3
average = sum / 3
print("The average is ", average)
```

Example: Average of three numbers (continued)

• What would we do if we wanted the program to average 5 values instead of 3? or 10? or 100?
• This is clearly not the best way to write this!
Loops

• We need the ability to perform the same set of instructions repeatedly so we don’t have to write them over and over again.

• This is why Python includes a few ways of using repetition in a program.

• Each case where we repeat a set of statement is called a loop.

Counting Loops

• The first type of loop is a counting loop.

• Counting loops are repeated a specific number of times.

• If you read the loop, you can easily figure out how many times its statements will be performed.
Example: Hello Again

- **Example** - Write a program that greets the user with "Hi there!" five times.
- We could write the program like this:
  
  ```python
  # Hello again - this program writes
  # "Hello, again" five times
  print("Hello, again")
  print("Hello, again")
  print("Hello, again")
  print("Hello, again")
  print("Hello, again")
  ```

Counting Loops

- We use a for loop to write basic counting loops
- In Python, it looks like this:
  ```python
  for count in range(size) :
    statements
  ```
  
  or
  ```python
  for count in range(start, size) :
    statements
  ```
  
  or
  ```python
  for count in range(start, size, increment) :
    statements
  ```
  
  or
  ```python
  for count in range(size, increment) :
    statements
  ```
Counting Loops (continued)

```
for count in range(start, size, increment) :
    statement(s)
```

- `count` : variable used to count times through the loop
- `start` : initial value of the counter
- `size` : number of loops
- `increment` : increment of the counter

**for Loops - Examples**

```
for i in range(3) :
    print(i, " ", end="")
print()

for i in range(1, 3) :
    print(i, " ", end="")
print()

for i in range(1, 6, 2) :
    print(i, " ", end="")
print()
```

Output

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Example: Rewriting HelloAgain

- Let's write the steps that our new version of that program must perform:

1. Write "Hi, there!" on the screen 5 times.

```
1. FOR i goes from 1 TO 5
1.1 Write “Hi, there!”
```

Refining HelloAgain

```
for i in range(5) :
```

Refining *HelloAgain*

for i in range(5) :
    1.1 Write “Hi, there!”
    print("Hello, again")

The New *HelloAgain*

# HelloAgain2 - This is a better way to write
# "Hello, again" five times

for i in range(5) :
    print("Hello, again")
Generalizing *HelloAgain*

- This program is also flawed; it gives us no choices as to how many times we can print “Hi, there!”
- We can let the user select how many times to print the message and making this version of the program more general is fairly easy:
- Our algorithm will start as:
  1. Find out how many times to print the message.
  2. Print "Hi, there!" that many times.

Generalizing *HelloAgain* (continued)

```python
totalTimes = int(input("How many times do you want to say "hello"?"))
```
Generalizing HelloAgain (continued)

totalTimes = int(input
  ("How many times do you want to say \"hello\"?"))

2. Print "Hi, there!" that many times.

print("Hello, again")

The Revised HelloAgain

# HelloAgain3 - Write "Hello, again" as many times
#               as the user wants
totalTimes = int(input
  ("How many times do you want to say \"hello\"?"))

for count in range(totalTimes) :
  print("Hello, again")]
Example: Averaging \( n \) Numbers

- Let's get back to our original problem. We want to able to average any number of values.
- Let's start by outlining our algorithm:
  1. Find out how many values there are.
  2. Add up all the values.
  3. Divide by the number of values
  4. Print the result

Refining Avg\( n \)

```
numValues = int(input
       ("How many values are you going to enter?"))
```
Refining Avg\(n\)

\[\text{numValues} = \text{int}(\text{input})\]
\[\text{"How many values are you going to enter?"})\]

2. Add up all the values.

3. Divide by the number of values

4. Print the result

   2.1 For CurrentValue goes from 1 to NumValues:
   2.1.1 Get the next value
   2.1.2 Add it to the total

2.0 Set the total to zero (initially there are no values)
Refining Avg

numValues = int(input
    ("How many values are you going to enter?"))

2.0 Set the total to zero (initially there are no values)
2.1 ForCurrentValue goes from 1 to NumValues:
   2.1.1 Get the next value
   2.1.2 Add it to the total

3. Divide by the number of values
4. Print the result

    sum = 0.0;
    for currentValue in range(numValues):
        value = float(input
            ("What is the next value?"))
        sum = sum + value

    average = sum / numValues
    print("The average is ", average)
The *AverageN* Program

```python
# AverageN - Find the average of N values

# Find out how many values there are
numValues = int(input
    ("How many values are you going to enter?"))

# Read in each value and add it to the sum
sum = 0.0;
for  currentValue in range(numValues) :
    value = float(input("What is the next value?"))
    sum = sum + value

# Calculate and print out the average
average = sum / numValues
print("The average is ", average)
```

Formatting *float* output in C++

- `cout` and `cin` are examples of what are called stream input/output.

- Stream I/O uses a set of built-in values called *format flags* to determine how data is printed. We can changes these values by using `setf()`. For now, we will use it only to reformat float values.
Example: Interest Program

- Example - Write a program that calculates the interest that the Canarsie Indians would have accumulated if they had put the $24 that they had received for Manhattan Island in the bank at 5% interest.

Input - none; all the values are fixed
Output - Year and Principle
Other Information -
    Principle is initially 24
    Interest = Interest Rate * Principle
    New Principle = Old Principle + Interest

Example: Interest Program

- Our initial algorithm is:
  1. Set the principle to 24
  2. For every year since 1625, add 5% interest to the principle and print out the principle.
Refining The Interest Algorithm

1. Set the principle to 24

2. For every year since 1625, add 5% interest to the principle and print out the principle.

2.1 FOR Year goes from 1625 TO Present:
  2.1.1 Add 5% interest to the principle
  2.1.2 Print the current principle
Refining The Interest Algorithm

1. Set the principle to 24

2.1 FOR Year goes from 1625 TO Present:
   2.1.1.1 Calculate 5% Interest
   2.1.1.2 Add the interest to the principle
   2.1.2 Print the current principle

```
principle = 24
```

Refining The Interest Algorithm

```
principle = 24;

2.1 FOR Year goes from 1625 TO Present:
   2.1.1.1 Calculate 5% Interest
   2.1.1.2 Add the interest to the principle
   2.1.2 Print the current principle

for year in range(1625, present) :
```
Refining The Interest Algorithm

```python
principle = 24;
for year in range(1625, present):
    interest = rate * principle
    principle = principle + interest
    print("year = ", year, ", principle = ", principle)
```
The Interest Program

# Calculate the interest that the Canarsie Indians could have accrued if they had deposited the $24 in a bank account at 5% interest.
present = 2015
rate = 0.05;

# Set the initial principle at $24 principle = 24

# For every year since 1625, add 5% interest to the principle and print out the principle
for year in range(1625, present):
    interest = rate * principle
    principle = principle + interest
    print("year = ", year, "\tprinciple = ", principle)
Output from the Compound Interest Program

• What will our output look like?
  year = 1625  principle = 25.2
  year = 1626  principle = 26.46
  year = 1627  principle = 27.783
  year = 1628  principle = 29.172150000000002
  ...          ...
  year =  2010 principle = 3624771902.2233915
  year =  2011 principle = 3806010497.3345613
  year =  2012 principle = 3996311022.201289
  year =  2013 principle = 4196126573.3113537
  year =  2014 principle = 4405932901.976921

• This does not look the way we expect monetary amounts to be written!

Formatted Output With \texttt{print()}\footnote{The method \texttt{print()} gives us a way to write output that is formatted, i.e., we can control its appearance.}

• We write:
  \texttt{print(ControlString, \%( Arg1, Arg2, ... ))}

• The control string is a template for our output, complete with the text that will appear along with whatever values we are printing.
Special Characters

• There are a number of special characters that all begin with a backslash:
  – \n        new line
  – \b        backspace
  – \t        tab
• These can appear anywhere with a string of characters:
  \n  print("This is a test\nIt is!!\n")

%\d and %\f

• The specifiers %\d and %\f allow a programmer to specify how many spaces a number will occupy and (in the case of float values) how many decimal places will be used.
• %\n\d will use at least \n  \d spaces to display the integer value in decimal (base 10) format.
• %w.\d\f will use at least \w \d spaces to display the value and will have exactly \d decimal places.
## Changing the width

<table>
<thead>
<tr>
<th>Number</th>
<th>Formatting</th>
<th>Print as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>182</td>
<td>%2d</td>
<td>182</td>
</tr>
<tr>
<td>182</td>
<td>%3d</td>
<td>182</td>
</tr>
<tr>
<td>182</td>
<td>%5d</td>
<td><code>\</code>182</td>
</tr>
<tr>
<td>182</td>
<td>%7d</td>
<td><code>\</code>182</td>
</tr>
<tr>
<td>-182</td>
<td>%4d</td>
<td>-182</td>
</tr>
<tr>
<td>-182</td>
<td>%5d</td>
<td>`-182</td>
</tr>
<tr>
<td>-182</td>
<td>%7d</td>
<td>```-182</td>
</tr>
</tbody>
</table>

## Changing the width (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Formatting</th>
<th>Print as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>%1d</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>%2d</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>%6d</td>
<td>....23</td>
</tr>
<tr>
<td>23</td>
<td>%8d</td>
<td>......23</td>
</tr>
<tr>
<td>11023</td>
<td>%4d</td>
<td>11023</td>
</tr>
<tr>
<td>11023</td>
<td>%6d</td>
<td>.11023</td>
</tr>
<tr>
<td>-11023</td>
<td>%6d</td>
<td>-11023</td>
</tr>
<tr>
<td>-11023</td>
<td>%10d</td>
<td>.....11023</td>
</tr>
</tbody>
</table>
Changing The Precision

<table>
<thead>
<tr>
<th>Number</th>
<th>Formatting</th>
<th>Prints as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.718281828</td>
<td>%8.5f</td>
<td><code>2.71828</code></td>
</tr>
<tr>
<td>2.718281828</td>
<td>%8.3f</td>
<td><code>2.718</code></td>
</tr>
<tr>
<td>2.718281828</td>
<td>%8.2f</td>
<td><code>2.72</code></td>
</tr>
<tr>
<td>2.718281828</td>
<td>%8.0f</td>
<td><code>3</code></td>
</tr>
<tr>
<td>2.718281828</td>
<td>%13.11f</td>
<td>2.71828182800</td>
</tr>
<tr>
<td>2.718281828</td>
<td>%13.12f</td>
<td>2.718281828000</td>
</tr>
</tbody>
</table>

The revised *Compound* program

```plaintext
# Calculate the interest that the Canarsie Indians could have accrued if they had deposited the $24 in an bank account at 5% interest.
present = 2015
rate = 0.05;

# Set the initial principle at $24
principle = 24;

# For every year since 1625, add 5% interest to the principle and print out the principle
```
for year in range(1625, present):
    interest = rate * principle;
    principle = principle + interest;
    print("year = %4d
tprinciple = $%13.2f"
         % (year, principle))

The output from the Revised Compound Program

Our output now looks like this:

year = 1625 principle = $ 25.20
year = 1626 principle = $ 26.46
year = 1627 principle = $ 27.78
year = 1628 principle = $ 29.17
... ... ... ...
year = 2010 principle = $3624771902.22
year = 2011 principle = $3806010497.33
year = 2012 principle = $3996311022.20
year = 2013 principle = $4196126573.31
year = 2014 principle = $4405932901.98
Integer Division

• Our compound interest program prints the values for every year where every ten or twenty years would be good enough.

• What we really want to print the results only if the year ends in a 5. (The remainder from division by 10 is 5).

Integer Division (continued)

• There are two types of division where the dividend and divisor are both integers.

• Floor by an integer produces an integer quotient, which is the largest integer smaller than the quotient:

  \[ \frac{5}{3} = 1R2 \quad \frac{16}{3} = 5R1 \]
  \[ \frac{6}{2} = 3R0 \quad \frac{15}{4} = 3R3 \]
# A few examples of integer division using # // and %
print("8 / 3 = ", 8 / 3 )
print("8 // 3 = ", 8 // 3 )
print("8 % 3 = ", 8 % 3 )

print("2 / 3 = ", 2 / 3 )
print("2 // 3 = ", 2 // 3 )
print("2 % 3 = ", 2 % 3 )

print("49 // 3 = ", 49 // 3 )
print("49 % 3 = ", 49 % 3 )

print("49 // 7 = ", 49 // 7 )
print("49 % 7 = ", 49 % 7 )

print("-8 // 3 = ", -8 // 3 )
print("-8 % 3 = ", -8 % 3 )

print("-2 // 3 = ", -2 // 3 )
print("-2 % 3 = ", -2 % 3 )

print("-2 // -3 = ", -2 // -3 )
print("-2 % -3 = ", -2 % -3 )

print("2 // -3 = ", 2 // -3 )
print("2 % -3 = ", 2 % -3 )

print("-49 // 3 = ", -49 // 3 )
print("-49 % 3 = ", -49 % 3 )

print("-49 // -3 = ", -49 // -3 )
print("-49 % -3 = ", -49 % -3 )

print("49 // -3 = ", 49 // -3 )
print("49 % -3 = ", 49 % -3 )
print("-49 // 7 = ", -49 // 7 )
print("-49 % 7 = ", -49 % 7 )

print("-49 // -7 = ", -49 // -7 )
print("-49 % -7 = ", -49 % -7 )

print("49 // -7 = ", 49 // -7 )
print("49 % -7 = ", 49 % -7 )

Integer Division Results

<table>
<thead>
<tr>
<th>8 // 3 = 2</th>
<th>8 % 3 = 2</th>
<th>2 // 3 = 0</th>
<th>2 % 3 = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>49 // 3 = 16</td>
<td>49 % 3 = 1</td>
<td>49 // 7 = 7</td>
<td>49 % 7 = 0</td>
</tr>
<tr>
<td>-8 // 3 = -2</td>
<td>-8 % 3 = -2</td>
<td>-2 // 3 = 0</td>
<td>-2 % 3 = -2</td>
</tr>
<tr>
<td>-2 // -3 = 0</td>
<td>-2 % -3 = -2</td>
<td>2 // -3 = 0</td>
<td>2 % -3 = 2</td>
</tr>
<tr>
<td>-49 // 3 = -16</td>
<td>-49 % 3 = -1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Integer Division Results (continued)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{-49}{-3}$</td>
<td>$\frac{-49}{3}$</td>
</tr>
<tr>
<td>$= 16$</td>
<td>$= -1$</td>
</tr>
<tr>
<td>$\frac{49}{-3}$</td>
<td>$\frac{49}{3}$</td>
</tr>
<tr>
<td>$= -16$</td>
<td>$= 1$</td>
</tr>
<tr>
<td>$\frac{-49}{7}$</td>
<td>$\frac{-49}{7}$</td>
</tr>
<tr>
<td>$= -7$</td>
<td>$= 0$</td>
</tr>
</tbody>
</table>

Final Compound Interest Program

```
# Calculate the interest that the Canarsie Indians could have accrued if they had deposited the $24 in an bank account at 5% interest.
present = 2015;
rate = 0.05;

# Set the initial principle at $24
principle = 24;

# For every year since 1625, add 5% interest to the principle and print out the principle
for year in range(1625, present) :
    interest = rate * principle
    principle = principle + interest
```
# Print the principle for every 20th year
if year % 20 == 5 :
    print("year = %4d\tprinciple = $%13.2f"
          %(year, principle))

# Print the values for the last year
print("year = %4d\tprinciple = $%13.2f"
      %(year, principle))