Loops in Visual Basic I

IMPORTANT: While writing programs involving loops, sometimes you may accidentally write code that contains an *infinite loop*. An infinite loop will cause your display to freeze up or output continually to a text box, depending on what your loop contains. If this happens you can break out of the infinite loop and stop the run of the program by clicking the Stop icon (a red square to the right of the Play icon).

Topics

- loop using `while` and `for` loop statements
- count-controlled loops based on predefined value
- count-controlled loops based on user-input value
- using loops to count
- program testing

Preliminaries

If you haven't already done so, start by making a copy of today's lab folder (`Lab05_Loops`) and saving it in your `M:\CS130\Labs` folder. Right-click on the folder you just copied and rename it `Lab05_YourLastName_YourFirstName` (but use your actual last and first names). You may now work locally by opening the write-up from within the copied folder.

Flipping a coin with loops

1. Use **File Explorer** to navigate to your lab folder from **Lab04**. Copy the project folder `FlippingACoin_Part2` (NOTICE **Part2**) and paste it into your lab folder for today's lab (**Lab05**). Right-click on the folder that you pasted rename it as `FlippingACoinWithLoops_Part1`. Launch VS Express and use it to open the project `FlippingACoinWithLoops_Part1`.

   Recall this program uses random numbers to simulate flipping a fair coin. But instead of flipping a coin only once, make the necessary changes to the project `FlippingACoinWithLoops_Part1` so that it now uses a *count-controlled* `while`*-loop* to flip a coin 100 times. The project should report the generated number and whether it is *Heads* or *Tails* for each iteration of the loop in the following format:

   ```
   0.7055475 Tails
   0.533424 Tails
   0.5795186 Tails
   0.2895625 Heads
   ...
   ```

Your project should behave identically to the solution which can be run by double-clicking the file `FlippingACoinWithLoops_Part1.exe` found in the `Executables` folder inside of your lab folder.

When done, save and close project `FlippinACoinWithLoops_Part1`.

Checkpoint 1 (30/100): A successful `FlippingACoinWithLoops_Part1` project:

- uses a count-controlled `while`-loop to flip a fair coin 100 times, each time reporting the generated number and whether it is *Heads* or *Tails*
- produces correctly formatted output according to the example above

2. Use **File Explorer** to navigate to your lab folder. Copy and paste the folder `FlippingACoinWithLoops_Part1`, and then right-click on the copy to rename it as `FlippingACoinWithLoops_Part2`. Launch VS Express and use it to open the project `FlippingACoinWithLoops_Part2`.

   Make the necessary changes to the project `FlippingACoinWithLoops_Part2` to count and report on the number of generated *Heads* and *Tails* out of 100. You SHOULD NOT change the loop structure for this part, only what happens in the *loop body*. The total count for each side of the coin should be reported after the output from Part1 in the following format:

   ```
   0.7055475 Tails
   0.533424 Tails
   0.5795186 Tails
   0.2895625 Heads
   ...
   Heads: 52
   Tails: 48
   ```

   Does this still look like a fair coin (in the sense that the coin is just as likely to generate *Heads* as *Tails*)?

Your project should behave identically to the solution which can be run by double-clicking the file `FlippingACoinWithLoops_Part2.exe` found in the `Executables` folder inside of your lab folder.

When done, save and close project `FlippinACoinWithLoops_Part2`.

Checkpoint 2 (65/100): A successful `FlippingACoinWithLoops_Part2` project:

- uses a *count-controlled* `while`*-loop* statement to flip a fair coin 100 times, each time reporting the generated number and whether it is *Heads* or *Tails*
- reports the counts of *Heads* and *Tails* in the 100 flips
- produces correctly formatted output according to the example above

Interest calculator

1. Launch the RAPTOR software and use it to create and save a new program called `InterestCalculator_Part1.rap` inside of your lab folder.

   Complete `InterestCalculator_Part1.rap` so that it computes the total amount of interest earned on a savings account after a user specified number of years. Assuming that the interest is compounded annually, then the interest earned for a single year can be expressed as, $P \times R$, where P is the principal amount, i.e., the amount in the savings account at the beginning of the year (note: this will change each year as interest is compounded), and R is the nominal interest rate.

   What equation can you use to express then new principle after one year of compounding?

Have your lab instructor or one of the TAs check your equation before moving on.

   Your program should get *three inputs* from the users: the starting principle, the nominal interest rate (between 0 and 1), and the period (number of years). Your program should produce one output which is a statement containing the total amount of interest earned in a statement in the following format:

```
After 10 years of compounding, your savings account with an initial balance
of $500 and APR of 3%, will have earned a total of $171.9582 in interest.
```

   After you get your program working, fill out `Test Table 1` in the file `InterestCalculator_TestTables.docx`, making sure that your expected output and actual output are consistent.

When done, save and close file `InterestCalculator_Part1.rap`.

2. Launch VS Express 2013 and open the project `InterestCalculator_Part2` which can be found inside of your lab folder.

   Complete the action for `btnComputeTotalInterest` so that it is equivalent to your RAPTOR program. You MUST use a *while*-loop.

   Input for your program should be done using a combination of *text boxes* and *input boxes*. In other words, you should NOT change the design of the form. You may use the two existing text boxes, but for other necessary input you should use an `InputBox`.

   Output for the Visual Basic version of this program should use the `FormatCurrency` and `FormatPercent` functions to produce output that has the following format:

   ```
   After 10 years of compounding, your savings account with an initial balance
   of $500.00 and APR of 3.00%, will have earned a total of $171.96 in interest.
   ```

   After you get your project working, fill out `Test Table 2` in the file `InterestCalculator_TestTables.docx`, making sure that your expected output and actual output are consistent.

Your project should behave identically to the solution which can be run by double-clicking the file `InterestCalculator_Part2.exe` found in the `Executables` folder inside of your lab folder.

When done, save and close project `InterestCalculator_Part2`.

Checkpoint 3 (70/100): A successful `InterestCalculator_Part2` project:

- is based on a correct `InterestCalculator_Part1.rap` RAPTOR program which produces correct output for all test cases in Test Table 1
- uses a `while`-loop statement
- produces correct output for all test cases in Test Table 2
- produces correctly formatted output according to the example above
- must have successfully completed Checkpoint 2

3. Use **File Explorer** to navigate to your lab folder. Copy and paste the folder `InterestCalculator_Part2`, and then right-click on the copy to rename it as `InterestCalculator_Part3`. Launch VS Express and use it to open the project `InterestCalculator_Part3`.

   Change the project `InterestCalculator_Part3` so that it uses a `for`-loop instead of a `while`-loop. The format of the output of your program should not change, only the loop statement that is used. After you get your project working, fill out `Test Table 3` in the file `InterestCalculator_TestTables.docx`, making sure that your expected output and actual output are consistent.

Your project should behave identically to the solution which can be run by double-clicking the file `InterestCalculator_Part3.exe` found in the `Executables` folder inside of your lab folder.

When done, save and close project `InterestCalculator_Part3`.

Checkpoint 4 (75/100): A successful `InterestCalculator_Part3` project:

- uses a `for`-loop statement
- produces correct output for all test cases in Test Table 3
- produces correctly formatted output according to the example from Part 2
- must have successfully completed Checkpoint 2

Submission Instructions

During our next lab meeting, you will be asked to expand your solutions to Part 2. Prior to the meeting, you are expected to

- finish ALL parts of this lab,
- study (again) any material you struggled with in this lab, and
- study new material needed for the next lab

You will submit your work for this lab and the next one together at the end of our next lab meeting.