Introduction to Programming

Functions

- **For Loop: A Quick Review**
- **Indefinite Loops**
- **Common Loop Patterns**
  - **Interactive Loops**
  - **Senteniel Loops**
  - File Loops
  - Nested Loops
- Computing with Boolean
- Other Common Structures

For Loops: A Quick Review

- The for statement allows us to iterate through a sequence of values.

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

- The loop index variable `var` takes on each successive value in the sequence, and the statements in the body of the loop are executed once for each value.

For Loops: A Quick Review

- Suppose we want to write a program that can compute the average of a series of numbers entered by the user.
- To make the program general, it should work with any size set of numbers.
- We don’t need to keep track of each number entered, we only need know the running sum and how many numbers have been added.

For Loops: A Quick Review

- We've run into some of these things before!
- A series of numbers could be handled by some sort of loop. If there are n numbers, the loop should execute n times.
- We need a running sum. This will use an accumulator.

For Loops: A Quick Review

```
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 sum/n
```

Indefinite Loops

- That last program got the job done
  - but you need to know ahead of time how many numbers you'll be dealing with.
  - User will have to count the numbers that they have
- What we need is a way for the computer to take care of counting how many numbers there are.
- The `for` loop is a definite loop, meaning that the number of iterations is determined when the loop starts.

Indefinite Loops

- We can’t use a definite loop unless
  - We know the number of iterations ahead of time
  - We can’t know how many iterations we need until all the numbers have been entered.
- We need another tool!
- The indefinite or conditional loop keeps iterating until certain conditions are met.

Indefinite Loops

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

- `condition` is a Boolean expression
  - Just like in `if` statements
  - The body is a sequence of one or more statements.
- Semantically, the body of the loop executes repeatedly as long as the condition remains true
  - When the condition is false, the loop terminates.

Indefinite Loops

- The condition is tested at the top of the loop.
- This is known as a pre-test loop.
- If the condition is initially false, the loop body will not execute at all.

Indefinite Loop

- Here’s an example of a while loop that counts from 0 to 10:

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

- The code has the same output as this for loop:

```
for i in range(11):
    print(i)
```

Indefinite Loop

- The `while` loop requires us to manage the loop variable `i`:
  - by initializing it to 0 before the loop and
  - incrementing it at the bottom of the body.
- In the `for` loop this is handled automatically.

Indefinite Loop

- The while statement is simple, but yet powerful and dangerous – they are a common source of program errors.

```
while i <= 10:
    print(i)
```

- What happens with this code?

Indefinite Loop

- What should you do if you’re caught in an infinite loop?
  - Press control-c

Interactive Loops

- One good use of the indefinite loop is to write interactive loops.
- Interactive loops allow a user to repeat certain portions of a program on demand.
- Remember how we said we needed a way for the computer to keep track of how many numbers had been entered?
- Let’s use another accumulator, called count.

Interactive Loops

- At each iteration of the loop, ask the user if there is more data to process. We need to preset it to “yes” to go through the loop the first time.

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

Interactive Loops

```
initialize sum to 0.0
initialize count to 0
set moredata to “yes”
while moredata is “yes”
   input a number, x
   add x to sum
   add 1 to count
   ask user if there is moredata
output sum/count
```

Sentinel Loops

- A *sentinel loop* continues to process data until reaching a special value that signals the end.
- This special value is called the sentinel.
- The sentinel must be distinguishable from the data since it is not processed as part of the data.

Sentinel Loops

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

- The first item is retrieved before the loop starts.
  - This is sometimes called the priming read, since it gets the process started.
- If the first item is the sentinel, the loop terminates and no data is processed.
- Otherwise, the item is processed and the next one is read.

Sentinel Loops

- In our averaging example, assume we are averaging test scores.
- We can assume that there will be no score below 0, so a negative number will be the sentinel.

Sentinel Loops

- This version provides the ease of use of the interactive loop without the hassle of typing ‘y’ all the time.
- There’s still a shortcoming
  - using this method we can’t average a set of positive and negative numbers.
- If we do this, our sentinel can no longer be a number.

Sentinel Loops

- We could input all the information as strings.
- Valid input would be converted into numeric form.
- Use a character-based sentinel.
- We could use the empty string (“”)!

Class Work

1) What output is produced by the following code fragment?

```
num = 0
max = 20
while num < max:
    print (num)
    num = num + 4
```

2)Write a `while` loop that verifies that user enters a positive number. The loop will keep asking for number as long as user does not enter a positive number.