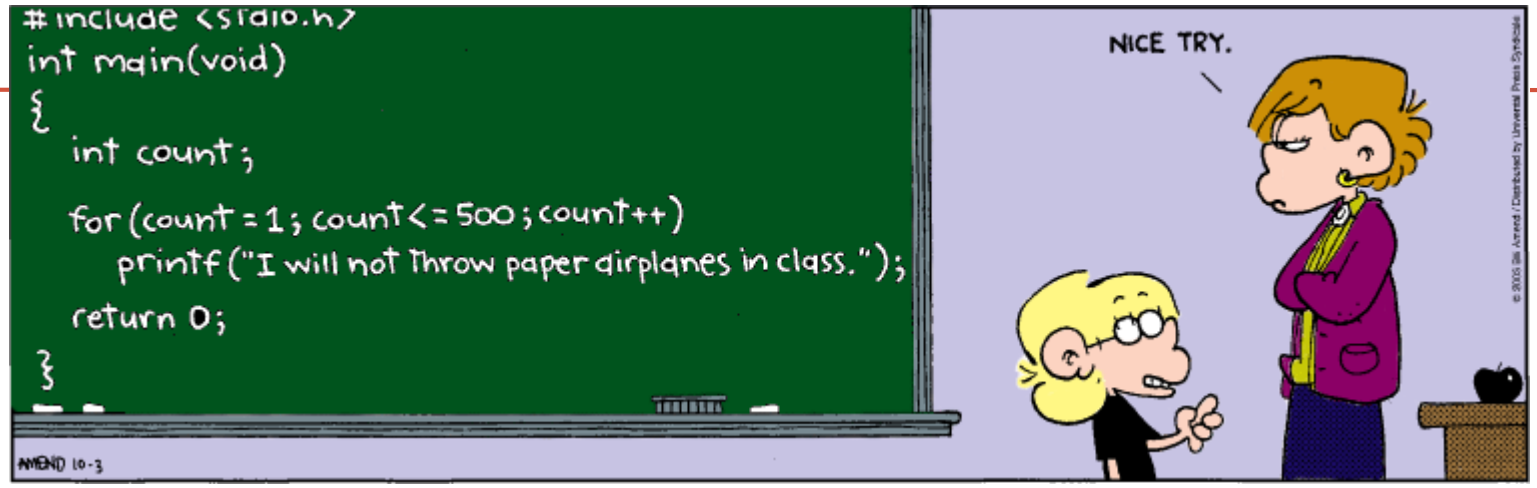
ITERATION (REPETITION OF CODE, OR LOOPING)



Outline

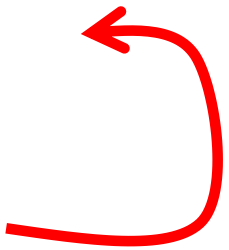
- Loop Statements
- Types of Loops
  - `while`
  - `for`
- Programming with Loops

Python Loop Statements

- A portion of a program that repeats a statement or a group of statements is called a *loop.*
- The statement or group of statements to be repeated is called the *body* of the loop.
  - For example, a loop could be used to compute grades for each student in a class.
- There must be a means of exiting the loop.

while Loop

- while loop: common way to repeat code
  - Evaluate a `boolean` expression
  - If `true`, do a block a code
    - Go back to start of while loop
  - If `false`, skip over block



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

while loop with multiple statements in a block

```
while (expression):
    statement1
    statement2
    …
else:
    statement
```

while loop with else clause

while Loop Example 1

- Print out summations, 0 + 1 + 2 + … + N

```
import sys

limit = int(sys.argv[1])
i     = 1
sum   = 0

while i <= limit:
    sum += i
    print("sum 0..." + str(i) + " = " + str(sum))
    i += 1
```

```
% python Summation.py 4
sum 0...1 = 1
sum 0...2 = 3
sum 0...3 = 6
sum 0...4 = 10
```

while Loop Example 2

- Print powers of 2 up to but not including limit

```
import sys

limit = int(sys.argv[1])
total = 1

while total < limit:
    print(total)
    total = total * 2
```

```
% python Powers2.py 16
1
2
4
8
```

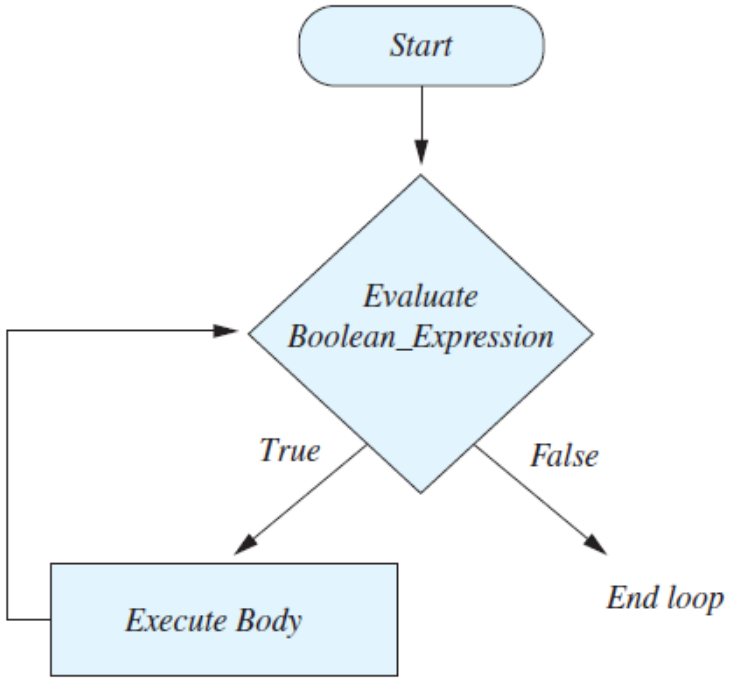
The `while` Statement

- Syntax

```
while Boolean_Expression:
    Body_Statement
```

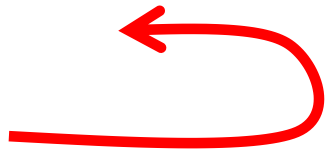
```
while (Boolean_Expression)
    Body
```

Semantics of the `while` statement



for Loop

- for loop: another common type of loop
  - Execute an initialization statement
    - target takes on each value in turn in the list of objects
  - If there are still items in the object list, do code block
  - If no more items, done with loop



```
for target in object:
    statement1
    statement2
    …
```

for Loop Example

- Print out summations, 0 + 1 + 2 + … + N

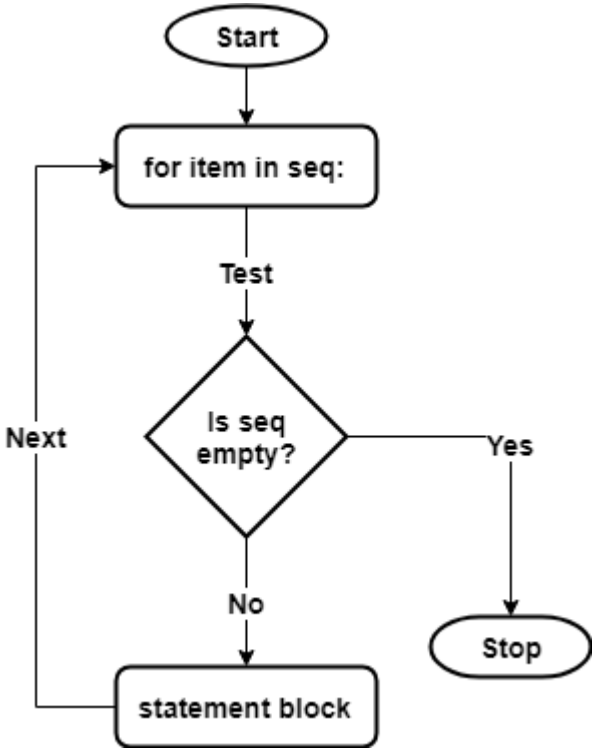
```
import sys

limit = int(sys.argv[1])
sum   = 0

for i in range (1, limit):
    sum += i
    print("sum 0..." + str(i) + " = " + str(sum))
```

The `for` Statement

- A `for` statement executes the body of a loop a fixed number of times.
  - That number is the number of “things” in the data you give it
  - If you use the range(start, end), it will execute the body once for each number from start to end-1
    - Why is this handy?
      - If you have a list, recall that indices go from 0 to the list length - 1
      - Makes it very handy to process according to list length

The `for` Statement

- The semantics of the `for` statement



Nested Loops

- A loop inside another loop

```
import sys

limit = int(sys.argv[1])
for i in range(0, limit+1):
    for j in range(0, i):
        print("*", end = "")
    print()
```

```
% python StarTriangle.py 4
*
**
***
****
```

The Loop Body

- To design the loop body, write out the actions the code must accomplish.
- Then look for a repeated pattern.
  - The repeated pattern will form the body of the loop.
  - Some actions may need to be done after the pattern stops repeating.

Loop Choice

- Does your loop need a counter variable?
  - e.g. Going from 0 to N or N to 0 in fixed steps
  - Use a for loop
- Does your loop need to execute on a sequence of items?
  - Use a for loop
- Do you need an unknown number of loops?
  - Use a while loop
- Do you need to perform some actions until a condition is met?
  - Use a while loop

Initializing Statements

- Some variables need to have a value before the loop begins.
  - Sometimes this is determined by what is supposed to happen after one loop iteration.
  - Often variables have an initial value of zero or one, but not always.
- Other variables get values only while the loop is iterating.

The `break` Statement in Loops

- A `break` statement can be used to end a loop immediately.
- The `break` statement ends only the **innermost** loop statement that contains the `break` statement.
- `break` statements make loops more difficult to understand.
- Use `break` statements sparingly (if ever).

The `break` Statement in Loops

- Program fragment, ending a loop with a `break` statement

```
while itemNumber <= MAX_ITEMS:
    if itemCost <= leftToSpend:
        if leftToSpend > 0:
            itemNumber += 1
        else:
            print("You are out of money.")
            break
    else:
        …

print(…)
```

The `continue` Statement in Loops

- A `continue` statement
  - Ends current loop iteration
  - Begins the next one
- Like a `break` statement, avoid using this
  - Introduce unneeded complications

Loop Bugs

- Common loop bugs
  - Unintended infinite loops
  - Off-by-one errors
  - Testing equality of floating-point numbers
- Subtle infinite loops
  - The loop may terminate for some input values, but not for others.
  - For example, you can’t get out of debt when the monthly penalty exceeds the monthly payment.

Tracing Variables

- *Tracing variables* means watching the variables change while the program is running.
  - Simply insert temporary output statements in your program to print of the values of variables of interest
  - Or, learn to use the debugging facility that may be provided by your system.

Infinite Loops

- A loop which repeats without ever ending is called an *infinite loop.*
- If the controlling boolean expression never becomes false, a `while` loop will repeat without ending.

Summary

- Loop Statements
- Types of Loops
  - `while`
  - `for`
- Programming with Loops

Your Turn

- Write a while loop that generates a random number between 0.0 and 100.0 as a test score and prints out that number on each iteration. The loop ends when a random number is generated that is a passing grade or better (70.0). After the loop completes, print out the score to the screen.

- Name your program RandomGrade.py and submit it to the Activity03 dropbox on Moodle. 1 point for turning something in, 2 points for turning in something that is correct.