CSC 110: Week 6, Lecture 2

Dr. Bodine

Last time…

- Loops
- Indefinite vs definite loops
- Interactive loops

This time…

- Sentinel loops
- File loops
- Nested loops
- Boolean algebra
- Other structures…

Indefinite vs definite loops

- Definite loops have automatic updating of the loop variable
- Indefinite loops require you to initialize and update variables yourself

- Definite loops occur a set number of times defined at the beginning
- Indefinite loops occur until the condition changes, i.e. the number of times through the loop is not predetermined

Interactive loops

- Indefinite loop that repeats part of the program on demand
- User determines when to end the loop
- Uses an input statement to update the value of a variable that is checked at the start of the loop

Sentinel loop:

- Processes until a pre-set value signals the end

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

Sentinel loop:

- Processes until a pre-set value signals the end

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

Sentinel loop:

- Processes until a pre-set value signals the end

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

Sentinel loop:

- More commonly a blank line signals the end

```
sum = 0
count = 0
xStr = input("Enter a number (<Enter> to quit) "
while xStr!='':
    x = eval(xStr)
    sum = sum + x
    count = count + 1
    xStr = input("Enter a number (<Enter> to quit)")
print("\nThe average of the numbers is",sum/count)
```

Sentinel loop:

- More commonly a blank line signals the end

```
sum = 0
count = 0
xStr = input("Enter a number (<Enter> to quit) ")
while xStr!='':
    x = eval(xStr)
    sum = sum + x
    count = count + 1
    xStr = input("Enter a number (<Enter> to quit) ")
print("\nThe average of the numbers is",sum/count)
```

Let’s write a sentinel loop:

- Let’s update our averaging programming from assignment 2.

Let's write a sentinel loop:

- Let's update our averaging programming from assignment 2.

```
#average test program with sentinel loop
def main():
    sumtotal = 0 # accumulator variable for sum
    count = 0 #counting variable

    numstring = input("Enter a number:")
    while numstring!='':
        number = eval(numstring)
        sumtotal = sumtotal + number
        count = count + 1
        numstring = input("Enter a number:")

    print("\nAverage = ",sumtotal/count)
```

File loops

- What about reading from a file that we've opened as infile?
- We've done this before with `for line in infile`
  - This is a special feature of python
- In general, should know how to loop over file

```
line = infile.readline()
while line !="":
    <some stuff we want to do with the line>
    line = infile.readline()
```

Let’s add this to our averaging program!

- Update our A2.py to take the numbers from the file using a while loop

Let's add this to our averaging program!

```
#average test program with a file loop
def main():

    filename = input("Please enter the filename:")
    infile = open(filename,"r")

    #initialize counts and sum
    sumtotal = 0
    ncounts = 0

    #read lines using while loop
    line = infile.readline()
    while line != "":
        number = eval(line)
        sumtotal = sumtotal + number
        ncounts = ncounts + 1
        line = infile.readline()

    print("Average = ",sumtotal/ncounts)

    infile.close()
```

Nested Loops

- Can use to process complex information
- Example: when different lines of a file have a different number entries on them -> not a single way process all lines
- Book example: Averaging program where some lines have multiple numbers separated by a comma
- Can use any combination of for and while loops

Post-test loop

- So far we talked about pre-test loops
  - Loop only ran if condition was satisfied based on initial value
- Now, what about testing after the loop?
  - Want the loop to run first then test the condition
- Directly implemented in some languages, not in others

- Not directly implemented in python ☹
  - But we can brute force it!!!

Post-test loop in python

```
x = -1
while x < 0:
    <do something>
    <update x>
```

- This runs through the loop at least once and tests at the beginning of each iteration
  - But flow chat can be twisted to consider it checking at the end…

`break` statements

- So far we talked about loops that naturally had an end based on a single condition we were checking for
- Can also implement `break` statements to end the loop based on other criteria.
- A `break` ends the loop immediately
- Typically placed in `if` statements for condition checking

Another post-test loop implementation

```
while True:
    <do some stuff>
    if <something>: break
```

- This is a more direct implementation of the post-test loop as it explicitly checks the condition at the bottom of the loop code

- Question: is there a difference in how these two implementations work? Are there advantages/disadvantages to either?

Loop Examples

- Write a program that uses a while loop to determine how long it takes for an investment to double at a given interest rate. The input will be an annualized interest rate and the output is the number of years it takes an investment to double.

- Write a program to find all divisors of a number entered as a parameter. How would we modify this to determine if a number is prime?

Boolean Operators with Truth Tables

And

P	Q	P and Q
T	T	T
T	F	F
F	T	F
F	F	F

Or

P	Q	P or Q
T	T	T
T	F	T
F	T	T
F	F	F

Not

P	Not P
T	F
F	T

Building more complicated conditions

- Using operators we can string together conditions
- Order of operations: Not, And, Or
  - Similar to PEMDAS in arithmetic
- Can build truth tables to clarify

Example: Not A and B or C

- Use order of operations to rewrite

Example: Not A and B or C

- Use order of operations to rewrite as:
  - ((not A) and B) or C
- Break it down into parts:
  - ((not A) and B)
    - Requires A is False, B is True
  - C
    - Requires C is True
- Put it back together
  - (A is False and B is True) or C is True
  - If C is True, then the result is True
  - If C is False, then A must be False and B must be True for the result to be T

Example: Not A and B or C

- Use order of operations to rewrite as:
  - ((not A) and B) or C
- Break it down into parts:
  - ((not A) and B)
    - Requires A is False, B is True
  - C
    - Requires C is True
- Put it back together
  - (A is False and B is True) or C is True
  - If C is True, then the result is True
  - If C is False, then A must be False and B must be True for the result to be T

A	B	C	not A and B or C
T	T	T	T
F	T	T	T
T	T	F	F
T	F	T	T
T	F	F	F
F	T	F	T
F	F	T	T
F	F	F	F

Boolean algebra

- All decisions in a computer essentially rely on Boolean expressions

Description	Algebra	Boolean algebra
Multiplication property of zero	a * 0 = 0	a and false == false
Multiplicative identity property	a*1 = a	a and true == a
Additive identity property	a + 0 = a	a or false == a

- Can logically think through the effects of each operation
- This is for **and**. What about **or**?

How do we use this?

- Can write conditions using this logic
- while (a==0 and b<100):
- while not (a==0 or b<100):
- There are often multiple ways to write conditions!
  - Example: `a!=0` vs `not a==0`

What did we talk about this week?

- Loops, loops and more loops! Definite loops and indefinite loops
  - for vs while
  - Condition checking
- Common loop structures
  - Interactive loops, Sentinel loops, File loops,
  - Tools to put in our toolkit!
  - `break` statements
- Boolean logic and complex conditions