Some Lessons

• Often many ways to solve a problem
  – Don’t rush to code first idea that pops into your head. Think about the design and ask if there’s a better way to approach the problem
  – Your first task is to find a correct algorithm. After that, strive for clarity, simplicity, efficiency, scalability, and elegance
Some Lessons

• Be the computer
  – Formulate an algorithm by thinking about how you’d solve the problem
  – This straightforward approach is often simple, clear, and efficient (enough)

Some Lessons

• Generality is good
  – Consideration of a more general problem can lead to a better solution for a special case
  – If the max of $n$ program is just as easy to write as the max of three, write the more general program because it’s more likely to be useful in other situations
Some Lessons

• Don’t reinvent the wheel
  – If the problem you’re trying to solve is one that lots of other people have encountered, find out if there’s already a solution for it
  – As you learn to program, designing programs from scratch is a great experience
  – Expert programmers know when to borrow
    • Search through Python functions
    • Search the Internet
• Chapter 7 DONE
• Suggested exercises on the website.
  – Do them over the weekend
  – If people have questions, we’ll go over them in class next Tuesday
Quick Review

• **for** statement allows us to iterate through a sequence of values

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

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

• Consider a program to compute 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 --- only need to know the running sum and how many numbers
Quick Review

• Similar to what we’ve seen before
  – A series of numbers could be handled by some sort of loop. With $n$ numbers, the loop should execute $n$ times
  – Can store running sum in an accumulator
Quick Review

# averagel1.py
# A program to average a set of numbers
# Illustrates counted loop with accumulator

def main():
    n = eval(input("How many numbers do you have? "))
    sum = 0.0
    for i in range(n):
        x = eval(input("Enter a number >> "))
        sum = sum + x
    print("\nThe average of the numbers is", sum / n)
Indefinite Loops

• That last program got the job done, but you need to know **ahead of time** how many numbers you’ll have
• The **for** loop is a definite loop, meaning that the number of iterations is determined when loop starts
• 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.

• The **indefinite** or **conditional loop** keeps iterating until certain conditions are met
• While looping, have computer count the number of #s
Indefinite Loops

while <condition>:
  <body>
• condition is a Boolean expression, just like in if statements. Body is a sequence of 1+ statements
• Body of the loop executes (repeatedly) until condition becomes false
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

- Example of a while loop that counts from 0 to 10:

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

- The code has same output as this for loop:

```python
for i in range(11):
    print(i)
```
Indefinite Loop

• \texttt{while} loop requires us to manage the loop variable $i$ by initializing it to 0 before loop and incrementing at the bottom of body

• In \texttt{for} loop this is handled automatically
Indefinite Loop

• The `while` statement is simple, but can be a common source of program errors

```python
i = 0
while i <= 10:
    print(i)
```

• What happens with this code?
I need to use the computer.

Hold on. Let me see if anyone wrote on my Facebook wall... nope.

Now let me see if anyone replied to me on Twitter... nope.

Now let me see if anyone started following my Tumblr... nope.

Done? Let me check Facebook again. Someone could have written something in the last 30 seconds... nope.

Now let me check Twitter again... now Tumblr...

Our sister has gone infinitely loopy.

Now Facebook... now Twitter... now Tumblr... now Facebook... now Twitter... now Tumblr... now Facebook... now Twitter... now Tumblr...
Indefinite Loop

• Initially, \( i \) is equal to 0, which is less than 10. Body of the loop is executed, printing 0.
• Control returns to the condition, and since \( i \) is still 0, the loop repeats, etc.
• Example of an infinite loop
Indefinite Loop

• What if you’re caught in an infinite loop?
  – First, try pressing control-c
  – Windows: control-alt-delete, kill process
  – Mac
    • Force Quit Application (Cmd-Alt-Esc)
    • Quit Process (Activity Monitor)
  – Linux: open a terminal and xkill
  – All else fails, push the reset button!
Convert the following to *while* loops

```python
for i in range(5, -2, 1):
    print(i)
```

```python
for i in [2, 4, 8, 16, 32]:
    print(i)
```
Interactive Loops

- Interactive loops allow user to repeat certain portions of a program on demand
- Let’s keep track of how many numbers had been entered by using a second accumulator, count
- At each iteration of the loop, ask the user if there is more data. We need to preset it to “yes” to go through the loop the first time
Interactive Loops

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

Design to get average of indefinite # of numbers from user?
Interactive Loops

• Combining the interactive loop pattern with accumulators for sum and count:

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
def main():
    moredata = "yes"
    sum = 0.0
    count = 0
    while moredata[0] == 'y':
        x = eval(input("Enter a number >> "))
        sum = sum + x
        count = count + 1
        moredata = input("Do you have more numbers (yes or no)? ")
    print("\nThe average of the numbers is", sum / count)

• Using string indexing (moredata[0]) allows us to accept “y”, “yes”, “yeah” to continue loop
Interactive Loops

Enter a number >> 32
Do you have more numbers (yes or no)? y
Enter a number >> 45
Do you have more numbers (yes or no)? yes
Enter a number >> 34
Do you have more numbers (yes or no)? yup
Enter a number >> 76
Do you have more numbers (yes or no)? y
Enter a number >> 45
Do you have more numbers (yes or no)? nah

The average of the numbers is 46.4
More Efficient?

• Right now, we ask for y/n every time through the loop

• Make some assumptions about input to remove this step?
Sentinel Loops

- A **sentinel loop** continues to process data until reaching a special value that signals end
- Special end value is called the sentinel
- Sentinel must be distinguishable from the data since it’s not processed as data
Sentinel Loops

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

• First item is retrieved before the loop starts
• If the first item is the sentinel, the loop terminates and no data is processed
• Otherwise, item is processed and next one is read
Sentinel Loops

• Example: assume we are averaging test scores
• Then, can can assume no score below 0
• A negative number will be the sentinel
Sentinel Loops

# average3.py
# A program to average a set of numbers
# Illustrates sentinel loop using negative input as sentinel

def main():
    sum = 0.0
    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 Loops

Enter a number (negative to quit) >> 32
Enter a number (negative to quit) >> 45
Enter a number (negative to quit) >> 34
Enter a number (negative to quit) >> 76
Enter a number (negative to quit) >> 45
Enter a number (negative to quit) >> -1

The average of the numbers is 46.4
Sentinel Loops

• This version provides ease of use of an interactive loop without hassle of typing ‘y’ all the time

• Still a shortcoming – couldn’t average a set of positive and negative numbers

• What if sentinel weren’t a number?

• Is there another input option?
Sentinel Loops

• Input all information as strings
• Valid input would be converted into numeric form
• Use a character-based sentinel
• Could use empty string (""")
Sentinel Loops

initialize sum to 0.0
initialize count to 0
input data item as a string, xStr
while xStr is not empty
    convert xStr to a number, x
    add x to sum
    add 1 to count
input next data item as a string, xStr
Output sum / count
# average4.py
#    A program to average a set of numbers
#    Illustrates sentinel loop using empty string as sentinel

def main():
    sum = 0.0
    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 Loops

Enter a number (<Enter> to quit) >> 34
Enter a number (<Enter> to quit) >> 23
Enter a number (<Enter> to quit) >> 0
Enter a number (<Enter> to quit) >> -25
Enter a number (<Enter> to quit) >> -34.4
Enter a number (<Enter> to quit) >> 22.7
Enter a number (<Enter> to quit) >>

The average of the numbers is 3.38333333333
File Loops

• The biggest disadvantage of our program?
• What happens if you make a typo?
• A better solution: read data from a file
# average5.py

#     Computes the average of numbers listed in a file.

def main():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    for line in infile.readlines():
        sum = sum + eval(line)
        count = count + 1
    print("\nThe average of the numbers is", sum / count)
File Loops

• Many languages don’t have a mechanism for looping through a file like this. Rather, they use a sentinel!

• We could use `readline` in a loop to get the next line of the file.

• At the end of the file, `readline` returns an empty string, “”
File Loops

• Alternative:

```python
line = infile.readline()
while line != ""
    #process line
    line = infile.readline()
```

• Blank line OK?

• Empty line ends with the newline character, and `readline` includes the newline

• "\n" != ""
# average6.py
# Computes the average of numbers listed in a file.

def main():
    fileName = input("What file are the numbers in? ")
infile = open(fileName, 'r')
sum = 0.0
count = 0
line = infile.readline()
while line != ":
    sum = sum + eval(line)
count = count + 1
    line = infile.readline()
print("\nThe average of the numbers is", sum / count)
Nested Loops

• Before: nested if statements
• Now: can also nest loops
• Suppose we allow any number of numbers on a line in the file (separated by commas), rather than one per line
Nested Loops

- At the top level, we will use a file-processing loop that computes a running sum and count

```python
sum = 0.0
count = 0
line = infile.readline()
while line != "":
    #update sum and count for values in line
    line = infile.readline()
print("\nThe average of the numbers is", sum/count)
```
def main():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        for xStr in line.split("",""): 
            sum = sum + eval(xStr)
            count = count + 1
        line = infile.readline()
    print("\nThe average of the numbers is", sum / count)
In next level, need to update `sum` and `count` in loop body

Each line of file contains one or more numbers separated by commas

Can split the string into substrings, each of which represents a number

Then, loop through substrings, convert each to a number, and add to `sum`

Also, update `count`

def main():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        for xStr in line.split("",""):  
            sum = sum + eval(xStr)
            count = count + 1
        line = infile.readline()
        print("\nThe average of the numbers is", sum / count)
Nested Loops

for xStr in line.split("",""):  
    sum = sum + eval(xStr)  
    count = count + 1

• Notice that for statement uses line, which is also loop control variable for the outer loop
Nested Loops

# average7.py
#     Computes the average of numbers listed in a file.
#     Works with multiple numbers on a line.

import string

def main():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        for xStr in line.split(","):  
            sum = sum + eval(xStr)
            count = count + 1
        line = infile.readline()
    print("\nThe average of the numbers is", sum / count)
Nested Loops

• Loop that processes the numbers in each line is indented inside of the file processing loop
• Outer `while` loop iterates once for each line of file
• For each iteration of the outer loop, inner `for` loop iterates as many times as there are numbers on line
• When inner loop finishes, next line of the file is read, and process begins again
Nested Loops

Pro tips:

– Design outer loop without worrying about what goes inside
– Design what goes inside, ignoring outer loop
– Put pieces together, preserving nesting structure