PARAMETERS, INDEFINITE LOOPS, AND LOOP PATTERNS

CSSE 120—Rose Hulman Institute of Technology

A better fix for the Eclipse PyDev input bug (we think)

- Download the win_in.py file from ANGEL (Modules to Download → Session 10), and follow the directions there.
- Once you have done that, and renamed your input and raw_input invocations as described there, you should be able to use **Run as…** in Eclipse instead of having to remember to use **External tools**.

Modifying Parameters

- How do functions send information back?
  - Return statements
  - *Mutating* parameters
- Consider:

```
def awardEC(score, extra):
    newScore = score + extra
    score = newScore

earned = 87
bonus = 4
awardEC(earned, bonus)
print earned
```

Parameter Passing in Python

- Formal parameters only receive the **values** of the actual parameters
- Assigning a new value to a formal parameter does not affect the actual parameter
- Python passes parameters *by value*
- How should we fix **awardEC**?

Full Credit for Getting It Right

```
def awardEC(score, extra):
    newScore = score + extra
    return newScore

earned = 87
bonus = 4
earned = awardEC(earned, bonus)
print earned
```

Extra-Credit for Everyone!

```
def awardEC(scores, extras):
    for i in range(len(scores)):
        scores[i] = scores[i] + extras[i]
earned = [87, 63, 94]
bonuses = [3, 5, 0]
awardEC(earned, bonuses)
print earned
```

- Did the value of earned change?
  - No, it refers to the same list
  - The list's *contents* changed
  - Functions can change state of *mutable* objects

A Central Parameter Example

- Can we write a function that exchanges the values of its two parameters?

```
# attempt to exchange two integers
def swapInts(x, y):
  x,y = y,x

# attempt to exchange two elements of a list
def swapListElements(list, i, j):
    list[i], list[j] = list[j], list[i]

x,y = 2, 5
print 'Before "swapInts": x=%d, y=%d' %(x, y)
swapInts (x, y)
print 'After "swapInts": x=%d, y=%d' %(x, y)

aList = [3, 4, 5, 6]
print "Original list:",aList
swapListElements(aList, 1, 3)
print "New list after swapListItems:",aList
```

Review: Definite Loops

- Review: For loop
  - Definite loop: knows *a priori* the number of iterations of loop body
  - Counted loop: sequence can be generated by range()
  - Example for loop in slideshow.py
- Syntax:
  - for <var> in <sequence>:
        <body>

Is This Loop a Definite Loop?

```
# open the file
inputFile = open(inputFileName, 'r')

# process each line of file
for line in inputFile:
    image = Image(imageCenter, line.rstrip())
    image.draw(win)
    time.sleep(delay)

win.getMouse()
inputFile.close()
win.close()
```

Indefinite Loops

- Number of iterations is not known when loop starts
- Is a conditional loop
  - Keeps iterating as long as a certain condition remains true
  - Conditions are Boolean expressions
- Implemented using while statement
- Syntax:

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

While Loop

- A *pre-test loop*
  - Condition is tested at the top of the loop
- Example use of while loops

Nadia deposits $100 in a savings account each month. Each month the account earns 0.25% interest on the previous balance. How many months will it take her to accumulate $10.000?

While Loops & Exception Handling

```
from win_in import *
while True:
    try:
        fileName = win_raw_input("Enter input file name: ")
        myfile = open(fileName.strip())
        print "myfile =" , myfile
        break
    except IOError:
        print "could not find file", fileName +
              ".   Try again!"

# do something with open file…

myfile.close()
```

Hint: You'll need code like this for your homework!

Summary: Two main types of loops

- Definite Loop
  - We know at the beginning of the loop how many times its body will execute
  - Implemented in Python as a **for** loop.
- Indefinite loop
  - The body executes as long as some condition is true.
  - Implemented in Python as a **while** loop.
  - Can be an infinite loop if the condition never becomes False.
- Python's `for line in file:` construct
  - indefinite loop that looks syntactically like a definite loop!

Some indefinite loop patterns

- Interactive loops
- Sentinel loops
- File loops
- post-test loops
- "loop and a half"

Interactive: Make the user count

```
# average1.py
#    A program to average a set of numbers
#    Illustrates counted loop with accumulator

from win_in import *

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

main()
```

- Not the best idea! (Why not?)

Interactive: Ask user if there is more

```
# average2.py
#    A program to average a set of numbers
#    Illustrates interactive loop with two accumulators
from win_in import *

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

This initial value makes the loop execute the first time through

- User no longer has to count, but still has a big burden.

Sentinel loop

- User signals end of data by a special "sentinel"value.

```
# average3.py
#    A program to average a set of numbers
#    Illustrates sentinel loop using negative input as sentinel
from win_in import *

def main():
    sum = 0.0
    count = 0
    x = win_input("Enter a number (negative to quit) >> ")
    while x >= 0:
        sum = sum + x
        count = count + 1
        x = win_input("Enter a number (negative to quit) >> ")
    print "\nThe average of the numbers is", sum / count
```

For this program, a negative input number is the sentinel that signals "no more data"

- Note that the sentinel value is not used in calculations.

Non-numeric Sentinel

- What if negative numbers are legitimate values?

```
# average4.py
#    A program to average a set of numbers
#    Illustrates sentinel loop using the empty string as sentinel
from win_in import *

def main():
    sum = 0.0
    count = 0
    xStr = win_raw_input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = win_raw_input("Enter a number (<Enter> to quit) >> ")
    print "\nThe average of the numbers is", sum / count
```

- **Again note:** sentinel value is not used in calculations.

Interactive loop with graphics

- Display a window that contains a circle and a message saying "Click inside Circle".
- Whenever the user clicks outside the circle, display "You missed!"
- If the user clicks inside the circle, display "Bull's eye!". Then pause and close the window.

File loop 1: using **for**

- Use a **for** loop as we have seen before:

```
# average5.py
#     Computes the average of numbers listed in a file.
from win_in import *

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

if __name__ == '__main__':
    main()
```

- Also note the conditional execution of **main()**

File loop 2: using **while**

- Use a **while** loop, as many other languages require:

```
# average6.py
#     Computes the average of numbers listed in a file.
from win_in import *

def main():
    fileName = win_raw_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
```

- What kind of **while** loop is this?

Start on Speed Reading HW exercise

- It is in HW 10 folder on ANGEL Begin it now.
- Ask questions if you get stuck
- Finish before Session 11