TCSS 142 – Introduction to Programming

Autumn 2014
Day 06

Day 7 Overview

- while loop
  - loop structure
  - count-control
  - validating user input
  - Sentinels

- nested loops

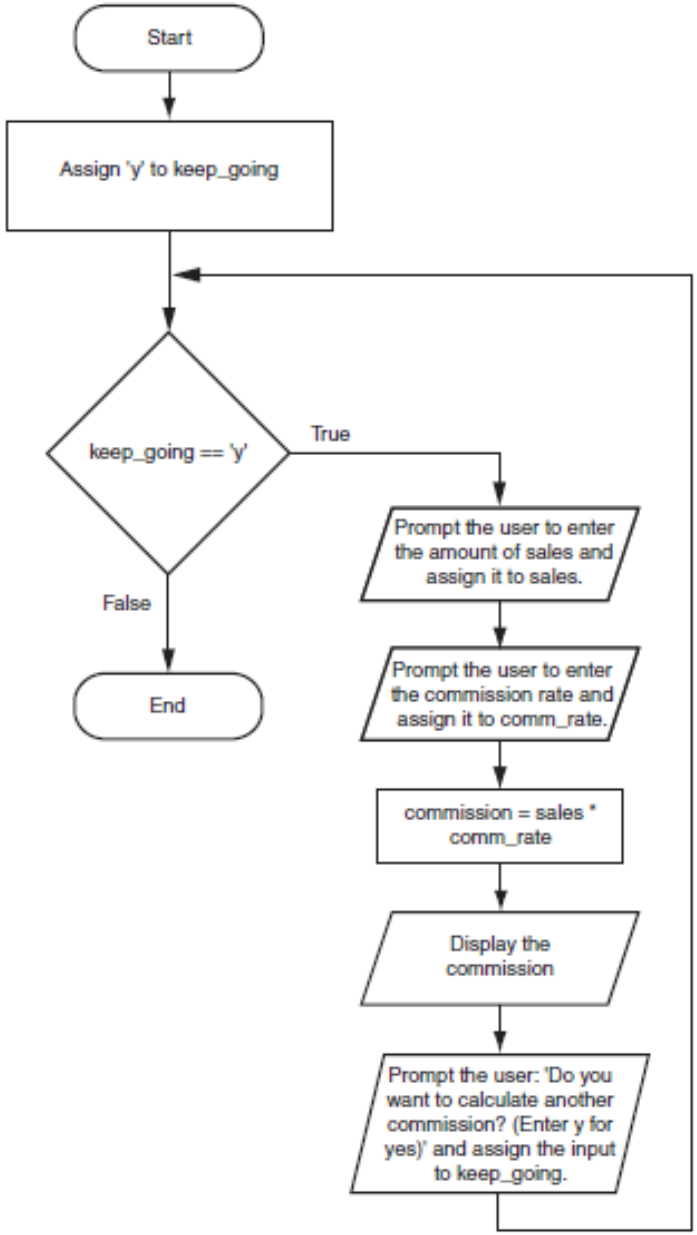
`while` structure

- A while loop: while condition is true, do something
  - it is a type of a pretest loop because a condition needs to be tested before ***every*** iteration
  - general format:

```
while condition:
        statements
```

- A while loop has to be structured according to the following guidelines:
  - initialization (before the loop body)
  - test (within the loop statement)
  - update (inside the loop body) that brings the loop closer to its termination

Figure 4-3 Flowchart for Program 4-1



Count-controlled

- In fact, a while loop can be controlled with a counter

```
counter = 0                          # initialize
while counter < 10:                  # test
     print(counter)
     counter = counter + 1           # update

-----------------------------

for counter in range (10):
     print(counter)
```

Exercise

- Download `whileTests.py` and rewrite the `for` loops as `while` loops.

```
print('a. ')
max = 5
for n in range(1, max):
    print(n)
-----------------------------------
print('b. ')
total = 25
for i in range(1, total // 2 , 1):
     total = total - i
     print(total, '  ', i)
```

Input validation

```
score = int(input("Enter a test score: "))

while score < 0 or score > 100:
   print("Error: grade must be between 0 and 100)
   print("Try again")
   score = int(input("Enter a test score: "))
```

Input validation

- What if a user does not enter something translatable to an int?
  - Python provides a string method called `isdigit()` that returns true if a string consists of digits only

```
score = input("Enter a test score: ")
while not score.isdigit():


while score < 0 or score > 100:
   print("Error: grade must be between 0 and 100)
   print("Try again")
   score = int(input("Enter a test score: "))
```

Exercise

- Find one of the `ifExample.py` programs from day04 and save it as `whileExample.py` add two input validation loops:
  - One to handle negative radius
  - One to handle any other choice but 1, 2, or 3

Other `while` loop uses

- Write a program called `digitSum.py` that asks a user for an integer and prints the sum of its digits
  - e.g. if a user enters `29107`, your program prints `19` since 2+9+1+0+7 is `19`

- Divide and conquer:
  - How can we extract digits from a number? hint: modulus
  - How can we use a loop to help with processing?
  - How do we sum the digits?

Digit extraction

29107 % 10 = 7

29107 // 10 = 2910

2910 % 10 = 0

2910 // 10 = 291

291 % 10 = 1

291 // 10 = 29

29 % 10 = 9

29 // 10 = 2

2 % 10 = 2

2 // 10 = 0 stopping case

Another example

- Write a program that simulates rolling of two 6-sided dice until their combined result comes up as 7, e .g.

```
2 + 4 = 6
3 + 5 = 8
5 + 6 = 11
1 + 1 = 2
4 + 3 = 7
```

- Divide and conquer
  - How can we simulate dice rolling
  - How do we construct a loop

random

- There is a module in Python called random that defines a method `randint(a, b)`
  - The method returns a random integer N such that

    a <= N <= b

- In order to be able to use that module and its associated method, we need to first

```
import random
someNum = random.randint(1, 6)
```

- Create a new file called `dice.py` and first generate and print two random numbers within the range 1 - 6

Dice contd

- How do we construct a loop?
- Add code so that the message `You won after [n] tries!`

Sentinel

- **sentinel**: A value that signals the end of user input.
  - must be distinctive enough from a data set to be processed
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Exercise: Write a program `sentinel.py` that prompts the user for numbers until the user types 0, then outputs their sum.
  - (In this case, 0 is the sentinel value.)

```
Enter a number (0 to quit): 10
Enter a number (0 to quit): 20
Enter a number (0 to quit): 30
Enter a number (0 to quit): 0
The sum is 60
```

flag

- Signaling an error

```
systemOverheated = False                # initialize
while not systemOverheated:             # test
     temp = float(input("Enter temperature
                                        reading"))
     if temp > 1000:
          systemOverheated = True       # update
     # some temp processing
```

break

- If you need to immediately exit a loop, use `break`

```
systemOverheated = False                # initialize
while not error:                        # test
     temp = float(input("Enter temperature
                                        reading"))
     if temp > 1000:
          systemOverheated = True       # update
          break
     # some temp processing
if  systemOverheated:
     # do something about the problem
```

Exercise

- With a partner – do NOT use a computer – paper and pencil – 10 minutes

- Write a pseudocode for a program that asks the user to enter positive integers and determines whether or not the positive numbers entered by the user are sorted in <u>ascending</u> order. You do not know how many numbers will be entered but you know that a sentinel value -1 will be used to denote the end of the sequence. Print "sorted" if the sequence is sorted, print "not sorted" if it is not.

- Assume that at least two positive numbers are entered before the user enters -1
- Assume valid data entry, i.e. do NOT worry about handling invalid input, e.g. non-integers or negative values

Exercise Contd

- Combine into groups of 4 and compare your solutions – 10 minutes

- Let’s code it together as a class

Guidelines

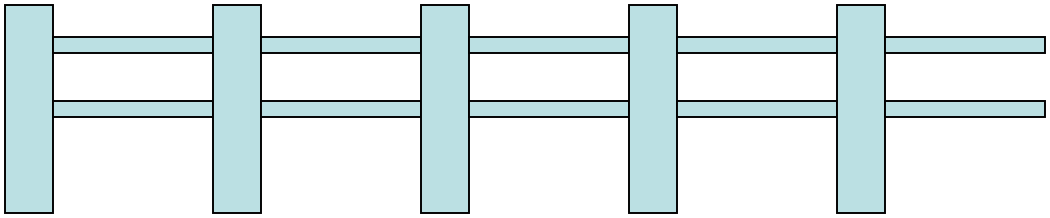
- Testing while
  - May not execute at all
  - May execute exactly once
  - May execute several times
- For a simple count-controlled loop, use the `for` statement
- For an event-controlled loop, use a `while` statement
- When in doubt, use a `while` statement

Fence post

- Suppose you were to print the sequences with comas in between, e.g.

  2 , 7, 12, 17, 22

- Similar to building a fence with wires separated by posts:
  - If we use a flawed algorithm that repeatedly places a post + wire, the last post will have an extra dangling wire.



Possible Solutions

for length of fence - 1:

place a post.

place some wire.

place a post.

---

place a post.

for length of fence ***- 1***

place some wire.

place a post.

for length of fence:

place a post.

if not the last post:

place some wire.

---

if length of fence > 0:

for loop solutions from the left

Nested Loops

- **nested loop**: A loop placed inside another loop.

- Imagine a situation in which you keep on repeating the entire program until a user types in q to quit
  - Let's add the code to `whileExample.py`

Nested Loops

- Example:
  - analog clock – for each iteration of the hours, 60 iterations of the minutes
  - sets and reps at the gym

```
for i in range(5):
    for j in range(3):
        print('*', end = '')
    print()
```

```
***
***
***
***
***
```

- The outer loop repeats 5 times; the inner one 3 times

- Output?

Nested `for`

- What is the output of the following code?

```
for i in range(5):
    for j in range(i):
        print('*', end = '')
    print()
```

```
*
**
***
****
```

Complex Lines

- What nested `for` loops produce the following output?

inner loop (repeated characters on each line)

```
....1
...2
..3
.4
5
```

outer loop (loops 5 times because there are 5 lines)

- We must build multiple complex lines of output using:
  - an *outer "vertical" loop* for each of the lines
  - *inner "horizontal" loop(s)* for the patterns within each line

Divide and Conquer

- Let's start by printing 5 lines with 4 dots each, followed by the line number:

 1

 2

 3

 4

 5

- How can we reduce the number of bullets on each line?
  - pattern?

Nested `for` loop exercise

- Make a table to represent any patterns on each line.

```
....1
...2
..3
.4
5
```

line	# of dots	-1 * line	-1 * line + 5
1	4	-1	4
2	3	-2	3
3	2	-3	2
4	1	-4	1
5	0	-5	0

Nested `for` loop solution

- Answer:

```
for line in range(5):
    for j in range(-1*(line+1) + 5):
        print('.', end = '')
    print(line + 1)
```

- Output:

```
....1
...2
..3
.4
5
```

Guidelines

- Key points:
  - For each iteration of the outer loop, the inner loop goes through all its iterations
  - Total number of iterations = outer iterations x inner iterations
  - If dealing with while loops, each loop has to have its own initialization, test, and update

Last Slide ☺

- Read chapter 5 and complete the quiz by the next class meeting on Tuesday.

- Class ends at 17:10