Chapter 5 Loops

(and File I/O)

Motivations

Suppose that you need to print a string (e.g., "Programming is fun!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
print("Programming is fun!");
```

So, how do you solve this problem?

Opening Problem

Problem:

100 times

```
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");

...

...

...
print("Programming is fun!");
print("Programming is fun!");
print("Programming is fun!");
```

Introducing while Loops

```
count = 0
while count < 100:
    print("Programming is fun!")
    count = count + 1
```

Objectives

- To write programs executing statements repeatedly using a **while** loop
- To control a loop with the user's confirmation
- To control a loop with a sentinel value
- To use **for** loops to implement counter-controlled loops
- To write nested loops
- program control with **break** and **continue**
- To obtain a large amount of input from a file (Chapter 13)

Trace while Loop

```
count = 0
while count < 2:
    print("Programming is fun!")
    count = count + 1
```

Initialize count

Trace while Loop, cont.

```
count = 0
while count < 2:
    print("Programming is fun!")
    count = count + 1
```

(count < 2) is true

Trace while Loop, cont.

```
count = 0
while count < 2:
    print("Programming is fun!")
    count = count + 1
```

Print Programming is fun

Trace while Loop, cont.

```
count = 0
while count < 2:
    print("Programming is fun!")
    count = count + 1
```

Increase count by 1
count is 1 now

Trace while Loop, cont.

```
count = 0
while count < 2:
    print("Programming is fun!")
    count = count + 1
```

(count < 2) is still true since count is 1

Trace while Loop, cont.

```
count = 0
while count < 2:
    print("Programming is fun!")
    count = count + 1
```

Print Programming is fun

Trace while Loop, cont.

```
count = 0
while count < 2:
    print("Programming is fun!")
    count = count + 1
```

Increase count by 1
count is 2 now

Trace while Loop, cont.

```
count = 0
while count < 2:
    print("Programming is fun!")
    count = count + 1
```

(count < 2) is false since count is 2 now

Trace while Loop

```
count = 0
while count < 2:
    print("Programming is fun!")
    count = count + 1
```

The loop exits. Execute the next statement after the loop.

Problem: An Advanced Math Learning Tool

The Math subtraction learning tool program generates just one question for each run. You can use a loop to generate questions repeatedly. This example gives a program that generates five questions and reports the number of the correct answers after a student answers all five questions.

SubtractionQuizLoop

Problem: Guessing Numbers

Write a program that randomly generates an integer between 1 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently. Here is a sample run:

GuessNumber

Ending a Loop with a Sentinel Value

Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*.

Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

SentinelValue

for Loops

```
i = initialValue  # Initialize loop-control variable
while i < endValue:
    # Loop body
    ...
    i++ # Adjust loop-control variable
```

```
for i in range(initialValue, endValue):
    # Loop body
```

range(a, b)

```
>>> for v in range(4, 8):
...     print(v)
...
4
5
6
7
>>>
```

range(b)

```
>>> for i in range(4):
...     print(i)
...
0
1
2
3
>>>
```

range(a, b, step)

```
>>> for v in range(3, 9, 2):
...     print(v)
...
3
5
7
>>>
```

range(a, b, step)

```
>>> for v in range(5, 1, -1):
...     print(v)
...
5
4
3
2
>>>
```

Nested Loops

Problem: Write a program that uses nested for loops to print a multiplication table.

MultiplicationTable

Problem: Finding the Greatest Common Divisor

Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor.

Solution: Suppose you enter two integers 4 and 2, their greatest common divisor is 2. Suppose you enter two integers 16 and 24, their greatest common divisor is 8. So, how do you find the greatest common divisor? Let the two input integers be n1 and n2. You know number 1 is a common divisor, but it may not be the greatest commons divisor. So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2.

GreatestCommonDivisor

Problem: Predicting the Future Tuition

Problem: Suppose that the tuition for a university is $10,000 this year and tuition increases 7% every year. In how many years will the tuition be doubled?

FutureTuition

Problem: Predicating the Future Tuition

```
year = 0                          # Year 0
tuition = 10000
year = year + 1                   # Year 1
tuition = tuition * 1.07
year = year + 1                   # Year 2
tuition = tuition * 1.07
year = year + 1                   # Year 3
tuition = tuition * 1.07
```

FutureTuition

Using `break` and `continue`

Examples for using the `break` and `continue` keywords:

- TestBreak.py

TestBreak

- TestContinue.py

TestContinue

break

```
sum = 0
number = 0

while number < 20:
    number += 1
    sum += number
    if sum >= 100:
        break

print("The number is ", number)
print("The sum is ", sum)
```

Break out of the loop

continue

```
sum = 0
number = 0

while (number < 20):
    number += 1
    if (number == 10 or number == 11):
        continue
    sum += number

print("The sum is ", sum)
```

Jump to the end of the iteration

Guessing Number Problem Revisited

Here is a program for guessing a number. You can rewrite it using a **break** statement.

GuessNumberUsingBreak

File Input (Chapter 13)

Open the file for reading (r)

infile = open(“inputSystemFileName”,’r’)

File Input (Chapter 13)

- Read data from the file (readline)

  variable = infile.readline()

File Input (Chapter 13)

- Close the file when done.

- NOTE: You must create the data file.

  (we will do this in class)