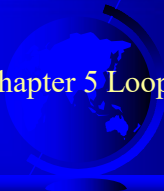


# Chapter 5 Loops




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 1

## 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?



© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 2

## Opening Problem

Problem:


```
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!");
```

100 times

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 3

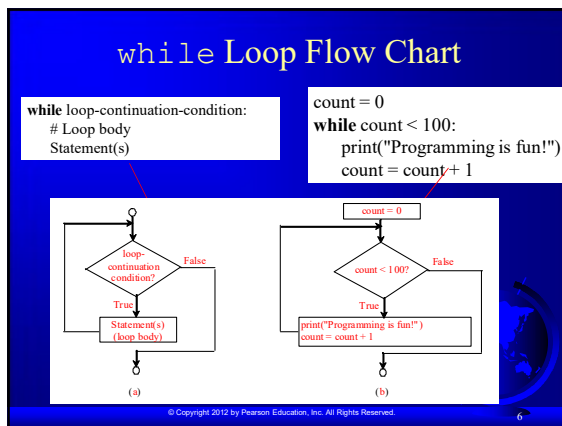
## Introducing while Loops

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



© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 4

- ## Objectives
- ☞ To write programs for executing statements repeatedly by using a **while** loop (§5.2).
  - ☞ To develop loops following the loop design strategy (§§5.2.1-5.2.3).
  - ☞ To control a loop with the user's confirmation (§5.2.4).
  - ☞ To control a loop with a sentinel value (§5.2.5).
  - ☞ To obtain a large amount of input from a file by using input redirection instead of typing from the keyboard (§5.2.6).
  - ☞ To use **for** loops to implement counter-controlled loops (§5.3).
  - ☞ To write nested loops (§5.4).
  - ☞ To learn the techniques for minimizing numerical errors (§5.5).
  - ☞ To learn loops from a variety of examples (**GCD**, **Future Tuition**, **MonteCarloSimulation**, **PrimeNumber**) (§§5.6, 5.8).
  - ☞ To implement program control with **break** and **continue** (§5.7).
  - ☞ To use a loop to control and simulate a random walk (§5.9).
- © Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 5




**animation**

### Trace while Loop

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

Initialize count




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

**animation**

### Trace while Loop, cont.

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

(count < 2) is true




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

**animation**

### Trace while Loop, cont.

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

Print Welcome to Python




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

**animation**

### Trace while Loop, cont.

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

Increase count by 1  
count is 1 now




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

**animation**

### 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




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

**animation**

### Trace while Loop, cont.

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

Print Welcome to Python




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

**animation**

### Trace while Loop, cont.

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

Increase count by 1  
count is 2 now




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 13

**animation**

### 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




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 14

**animation**

### 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.




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 15

### 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 Run




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 16

### Problem: Guessing Numbers

Write a program that randomly generates an integer between 0 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:

GuessNumberOneTime Run

GuessNumber Run




© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 17

### 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 Run



© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 18

## Caution

Don't use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing  $1 + 0.9 + 0.8 + \dots + 0.1$ :

```
item = 1
sum = 0
while item != 0: # No guarantee item will be 0
    sum += item
    item -= 0.1
print(sum)
```

Variable `item` starts with 1 and is reduced by 0.1 every time the loop body is executed. The loop should terminate when `item` becomes 0. However, there is no guarantee that `item` will be exactly 0, because the floating-point arithmetic is approximated. This loop seems OK on the surface, but it is actually an infinite loop.

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

19

## 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
```

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

20

## range(a, b)

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

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

21

## range(b)

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

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

22

## range(a, b, step)

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

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

23

## range(a, b, step)

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

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

24

## Nested Loops

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

MultiplicationTable

Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

25

## Minimizing Numerical Errors

Numeric errors involving floating-point numbers are inevitable. This section discusses how to minimize such errors through an example.

Here is an example that sums a series that starts with 0.01 and ends with 1.0. The numbers in the series will increment by 0.01, as follows: 0.01 + 0.02 + 0.03 and so on.

TestSum

Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

26

## 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 common 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

Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

27

## 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

Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

28

## Problem: Predicating the Future Tuition

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

FutureTuition

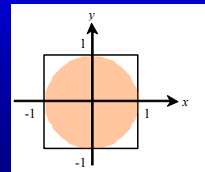
Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

29

## Problem: *Monte Carlo Simulation*

The Monte Carlo simulation refers to a technique that uses random numbers and probability to solve problems. This method has a wide range of applications in computational mathematics, physics, chemistry, and finance. This section gives an example of using the Monte Carlo simulation for estimating  $\pi$ .



$$\text{circleArea} / \text{squareArea} = \pi / 4.$$

$\pi$  can be approximated as  $4 * \text{numberOfHits} / 1000000$ .

MonteCarloSimulation

Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved.

30

## Using break and continue

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

- TestBreak.py
 

TestBreak
Run
- TestContinue.py
 

TestContinue
Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 31

## 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

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 32

## 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

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 33

## Guessing Number Problem Revisited

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

GuessNumberUsingBreak
Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 34

## Problem: Displaying Prime Numbers

**Problem:** Write a program that displays the first 50 prime numbers in five lines, each of which contains 10 numbers. An integer greater than 1 is *prime* if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

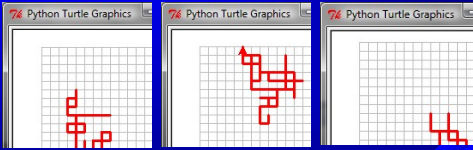
**Solution:** The problem can be broken into the following tasks:

- For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
- Determine whether a given number is prime.
- Count the prime numbers.
- Print each prime number, and print 10 numbers per line.

PrimeNumber
Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 35

## Turtle: Random Walk



RandomWalk
Run

© Copyright 2012 by Pearson Education, Inc. All Rights Reserved. 36