Introduction to Computer Programming for Non-Majors
CSC 2301, Fall 2015
Chapter 3

Guoliang Liu

The Department of Computer Science
Review of Chapter 2

• Stages of creating a program:
  – Analyze the problem
  – Determine specifications
  – Create a design
  – Implement the design
  – Test/debug the program
  – Maintain the program
• Example of the temperature converting
Review of Chapter 2

- Elements of programs
  - Names (identifier)
    - Starts with underscore or letter
    - Case-sensitive
    - Can not use reserved words
- Expressions
  - Variables
  - Operations
  - Literals
Review of Chapter 2

• Output statements
  – Print (<expr>,<expr>,…,<expr>)
  – Print (<expr>,<expr>,…,<expr>, end = ‘\n’)
  – Print (<expr>,<expr>,…,<expr>, end = ‘ ’)
  – Separated by commas
  – String are quoted by “ ” (double quotation marks)
Review of Chapter 2

- Assignment Statement
  - Simple assignment
    - `<variable> = <expr>`
    - Eg. Fahrenheit = 9/5 * celsius + 32
  - Assigning input
    - `<variable> = input(<prompt>)`
    - `<variable> = eval(input<prompt>)`
  - Simultaneous assignment
    - `<var>,<var>,...,<var> = <expr><expr>,...,<expr>`

Python Programming, 2/e
Review of Chapter 2

• Definite Loops
  – General form
    • for <var> in <sequence>:
      – <body>

  – Counted loop
    • for <var> in range(<expr>)
      – <body>
Chapter 3 -- Objectives

• To understand the concept of data types.
• To be familiar with the basic numeric data types in Python.
• To understand the fundamental principles of how numbers are represented on a computer.
Objectives (cont.)

• To be able to use the Python math library.
• To understand the accumulator program pattern.
• To be able to read and write programs that process numerical data.
Outline

• Data Types
• Math Library
  ✓ Roots of a quadratic equation
  ✓ Factorial of a number
• Limits of numeric data types
• Type conversion
Outline

- **Data Types**
- **Math Library**
  - Roots of a quadratic equation
  - Factorial of a number
- Limits of numeric data types
- Type conversion
Example of calculating the value of some change in dollars

- quarters = eval(input("Quarters: "))
- dimes = eval(input("Dimes: "))
- nickels = eval(input("Nickels: "))
- pennies = eval(input("Pennies: "))
- total = quarters * .25 + dimes * .10 + nickels * .05 + pennies * .01
Numeric Data Types

- The information that is stored and manipulated by computers programs is referred to as data.
- There are two different kinds of numbers!
  - (5, 4, 3, 6) are whole numbers – they don’t have a fractional part
  - (.25, .10, .05, .01) are decimal fractions
Numeric Data Types

- Inside the computer, whole numbers and decimal fractions are represented quite differently!
- We say that decimal fractions and whole numbers are two different data types.

• The data type of an object determines what values it can have and what operations can be performed on it.
Numeric Data Types

• Whole numbers are represented using the integer (int for short) data type.
• These values can be positive or negative whole numbers.
Numeric Data Types

- Numbers that can have fractional parts are represented as *floating point* (or *float*) values.
- How can we tell *which is which*?
  - A numeric literal without a decimal point produces an *int* value
  - A literal that has a decimal point is represented by a *float* (*even if the fractional part is 0*)
Numeric Data Types

- Python has a special function `type()` to tell us the data type of any value.

```python
>>> type(3)
<class 'int'>

>>> type(3.1)
<class 'float'>

>>> type(3.0)
<class 'float'>

>>> myInt = 32
>>> type(myInt)
<class 'int'>
```
Numeric Data Types

• Why do we need two number types?
  – Values that represent counts can’t be fractional (you can’t have 3 ½ quarters)
  – Most mathematical algorithms are very efficient with integers
  – The float type stores only an approximation to the real number being represented!
  – Since floats aren’t exact, use an int whenever possible!
Numeric Data Types

- **Operations on ints produce ints, operations on floats produce floats (except for `/`).**
  - Look up table 3.1
    - // integer division
    - % remainder
    - ** exponentiation

```python
>>> 3.0+4.0
7.0
>>> 3+4
7
>>> 3.0*4.0
12.0
>>> 3*4
12
>>> 10.0/3.0
3.3333333333333335
>>> 10/3
3.3333333333333335
>>> 10 // 3
3
>>> 10.0 // 3.0
3.0
```
Outline

• Data Types
• Math Library
  ✓ Roots of a quadratic equation
  ✓ Factorial of a number
• Limits of numeric data types
• Type conversion
Using the Math Library

• Besides (+, -, *, /, //, **, %, abs), we have lots of other math functions available in a *math library*.
  • [http://docs.python.org/library/math.html](http://docs.python.org/library/math.html)

• A *library* is a *module* with some useful definitions/functions.
  • [http://docs.python.org/library/](http://docs.python.org/library/)
Outline

• Data Types
• Math Library
  ✓ Roots of a quadratic equation
  ✓ Factorial of a number
• Limits of numeric data types
• Type conversion
Using the Math Library

- Let’s write a program to compute the roots of a quadratic equation!

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

- The only part of this we don’t know is how to find a square root ........
Using the Math Library

- To use a library, we need to make sure this line is in our program:

  import math

- Importing a library makes whatever functions are defined within it available to the program.
Using the Math Library

• To access the `sqrt library routine`, we need to access it as `math.sqrt(x)`.
• Using this `dot notation` tells Python to use the `sqrt` function found in the math library module.
• To calculate the root, you can do:

```
discRoot = math.sqrt(b*b - 4*a*c)
```
Using the Math Library

# quadratic.py
#    A program that computes the real roots of a quadratic equation.
#    Illustrates use of the math library.
#    Note: This program crashes if the equation has no real roots.
import math  # Makes the math library available.

def main():
    print("This program finds the real solutions to a quadratic")
    print()

    a, b, c = eval(input("Please enter the coefficients (a, b, c): "))
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)

    print()
    print("The solutions are:", root1, root2 )

main()
Using the Math Library

This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 3, 4, -1

The solutions are: 0.215250437022 -1.54858377035

• What do you suppose this means?

This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 1, 2, 3

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

Traceback (most recent call last):
  File "<pyshell#26>", line 1, in -toplevel-
    main()
  File "C:\Documents and Settings\Terry\My Documents\Teaching\W04\CS 120\Textbook\code\chapter3\quadratic.py", line 14, in main
    discRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error

>>>
Math Library

• If $a = 1$, $b = 2$, $c = 3$, then we are trying to take the square root of a negative number!

• Using the `sqrt` function is more efficient than using `**`. How could you use `**` to calculate a square root?
Outline

• Data Types
• Math Library
✓ Roots of a quadratic equation
✓ Factorial of a number
• Limits of numeric data types
• Type conversion

Python Programming, 2/e
Accumulating Results: Factorial

• Say you are waiting in a line with five other people. How many ways are there to arrange the six people?

• 720 -- 720 is the factorial of 6 (abbreviated 6!)

• Factorial is defined as:
  \[ n! = n(n-1)(n-2)\ldots(1) \]

• So, 6! = 6*5*4*3*2*1 = 720
Accumulating Results: Factorial

• How could we write a program to do this?

• IPO (Input-Process-Output)
  ✓ Input: Input number to take factorial of, \( n \)
  ✓ Process: Compute factorial of \( n, \text{fact} \)
  ✓ Output: Output \( \text{fact} \)
Accumulating Results: Factorial

- The general form of an accumulator algorithm looks like this:
  - Initialize the accumulator variable
  - Loop until final result is reached
    - update the value of accumulator variable
Accumulating Results: Factorial

• It looks like we’ll need a **loop**!

```python
fact = 1
for factor in [6, 5, 4, 3, 2, 1]:
    fact = fact * factor
```

• Let’s trace through it to verify that this works!
Accumulating Results: Factorial

• Why did we need to initialize fact to 1? There are a couple reasons...

  – Each time through the loop, the previous value of fact is used to calculate the next value of fact. By doing the initialization, you know fact will have a value the first time through.

  – If you use fact without assigning it a value, what does Python do?
Accumulating Results: Factorial

- Since multiplication is **associative** and **commutative**, we can rewrite our program as:

  ```python
  fact = 1
  for factor in [2, 3, 4, 5, 6]:
    fact = fact * factor
  ```

- Great! But what if we want to find the factorial of some other number??
Accumulating Results: Factorial

- What does `range(n)` return? 0, 1, 2, 3, …, n-1
- `range` has another optional parameter! `range(start, n)` returns start, start + 1, …, n-1
- But wait! There’s more! `range(start, n, step)` returns start, start+step, …, n-1
- `list(<sequence>)` to make a list
Accumulating Results: Factorial

• Let’s try some examples!

```python
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```python
>>> list(range(5,10))
[5, 6, 7, 8, 9]
```

```python
>>> list(range(5,10,2))
[5, 7, 9]
```
Accumulating Results: Factorial

Using this souped-up *range statement*, we can do the range for our loop a couple different ways.

- We can count up from 2 to n:
  ```python
  range(2, n+1)
  ```
  (Why did we have to use n+1?)

- We can count down from n to 2:
  ```python
  range(n, 1, -1)
  ```
Accumulating Results: Factorial

• Our completed factorial program:

```python
# factorial.py
# Program to compute the factorial of a number
# Illustrates for loop with an accumulator

def main():
    n = eval(input("Please enter a whole number: "))
    fact = 1
    for factor in range(n,1,-1):
        fact = fact * factor
    print("The factorial of", n, "is", fact)

main()
```

Python Programming, 2/e
Outline

• Data Types
• Math Library
  ✓ Roots of a quadratic equation
  ✓ Factorial of a number
• Limits of numeric data types
• Type conversion
Number system review

Understand the numbers in computer system.

How numbers in different system coverts!
• What is 100!?

```python
>>> main()
Please enter a whole number: 100
The factorial of 100 is
93326215443944152681699238856266700490715968264381621
46859296389521759999322991560894146397615651828625369
79208272237582511852109168640000000000000000000000
```

• Wow! That’s a pretty big number!
The Limits of Int

- Newer versions of Python can handle it, but…

Python 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)] on win32
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam

>>> import fact
>>> fact.main()
Please enter a whole number: 13
13
12
11
10
9
8
7
6
5
4
Traceback (innermost last):
  File "<pyshell#1>", line 1, in ?
    fact.main()
  File "C:\PROGRA~1\PYTHON~1.2\fact.py", line 5, in main
    fact=fact*factor
OverflowError: integer multiplication
The Limits of Int

- What’s going on?
  - While there are an infinite number of integers, there is a finite range of ints that can be represented.
  - This range depends on the number of bits a particular CPU uses to represent an integer value.
  - Typical PCs use 32 bits.
The Limits of Int

• Typical PCs use 32 bits
• That means there are $2^{32}$ possible values, centered at 0.
• This range then is $-2^{31}$ to $2^{31}-1$. We need to subtract one from the top end to account for 0.
• But our 100! is much larger than this. How does it work?
Handling Large Numbers

• Does switching to float data types get us around the limitations of ints?

• If we initialize the accumulator to 1.0, we get

>>> main()
Please enter a whole number: 15
The factorial of 15 is 1.307674368e+012

• We no longer get an exact answer!
• Very large and very small numbers are expressed in *scientific or exponential notation*. 

• $1.307674368e+012$ means $1.307674368 \times 10^{12}$

• Here the decimal needs to be moved right 12 decimal places to get the original number, but there are only 9 digits, so 3 digits of precision have been lost.
Handling Large Numbers

- **Floats are approximations**
- Floats allow us to represent a larger range of values, but with lower precision.

- Python has a solution, **expanding ints**!

- Python Ints are not a fixed size and expand to handle whatever value it holds.
Handling Large Numbers

• Newer versions of Python automatically convert your ints to expanded form when they grow so large as to overflow.

• We get indefinitely large values (e.g. 100!) at the cost of speed and memory
Outline

• Data Types
• Math Library
  ✓ Roots of a quadratic equation
  ✓ Factorial of a number
• Limits of numeric data types
• Type conversion
Type Conversions

• We know that combining an int with an int produces an int, and combining a float with a float produces a float.
• What happens when you mix an int and float in an expression?
  \[ x = 5.0 + 2 \]
• What do you think should happen?
Type Conversions

• For Python to evaluate this expression, it must either convert 5.0 to 5 and do an integer addition, or convert 2 to 2.0 and do a floating point addition.

• Converting a float to an int will lose information

• Ints can be converted to floats by adding “.0”
Type Conversion

• In *mixed-typed expressions* Python will convert ints to floats.

• Sometimes we want to control the type conversion. This is called *explicit typing*. 
Type Conversions and round

>>> float(22//5)
4.0

>>> int(4.5)
4

>>> int(3.9)
3

>>> round(3.9)
4

>>> round(3)
3
Exercise and homework

• Read Through chapter 3.
• Test the largest integer you can store in python in your machine.
  – Note that use $2^{**N}$ to generate a number, N means the number of bits you could use.
  – Use `sys.getsizeof()` function to measure how many bytes are used to store the number
• Homework
  – Check course website schedule to download it.