ISE 314X
Computer Programing for Engineers

Chapter 3
Computing with Numbers

Yong Wang
Assistant Professor
Systems Science & Industrial Engineering
Binghamton University

Objectives

- To understand the basic numeric data types
- To be able to use the Python math library
- To understand the accumulator program

Numeric Data Types

- The information that is stored and manipulated by computer programs is referred to as *data*
- The *data type* of an object determines what values it can have and what operations can be performed on it

Numeric Data Types

- There are two types of numbers
  - (−5, 0, 3) are whole numbers (no fractional parts)
  - (0.25, 3., −.5) are decimal numbers
  - Inside the computer, whole numbers and decimal numbers are represented quite differently

Numeric Data Types

- Whole numbers are represented using the *integer* (*int* for short) data type
- Decimal numbers are represented as *floating point* (or *float*) values

Numeric Data Types

- How can we tell which is which?
  - A number without a decimal point produces an int value
  - A number with a decimal point is represented by a float (even if the fractional part is 0)

Numeric Data Types

- Python has a function that tells the data type

```
>>> type(3)
<class 'int'>
>>> type(-.5)
<class 'float'>
>>> type(3.)
<class 'float'>
>>> myInt = 32
>>> type(myInt)
<class 'int'>
```

Numeric Data Types

- Why do we need two numerical data types?
  - Values that represent counts cannot be fractional
  - Most mathematical algorithms are very efficient with integers
  - The float type stores only an *approximation* to the real number being represented

Numeric Data Types

In general, operations on ints produce ints, operations on floats produce floats

```
>>> 3.0+4.0
7.0
>>> 3+4
7
>>> 3.0*4.0
12.0
>>> 3*4
12
```

```
>>> 10.0/3.0
3.3333333333333335 (why?)
>>> 10/3
3.3333333333333335 (why?)
```

Numeric Data Types

```
>>> 10 // 3 #floor division  (10 = 3 * 3 + 1)
3
>>> 10.0 // 3.0
3.0
>>> 10 % 3 #modulo/remainder  (10 = 3 * 3 + 1)
1
>>> 10.0 % 3.0
1.0
>>> 2.0 ** 3  #exponentiation
8.0
>>> abs(-3.5) #absolute value
3.5
```

Using the Math Library

- Besides (+, -, *, /, //, **, %, abs), we have lots of other math functions available in a *math library*
- A *library* is a module with some useful definitions/functions

Using the Math Library

- For a quadratic equation $ax^2 + bx + c = 0$, the roots are

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

Using the Math Library

- To use a library, we need to *import* it into our program first:

```
import math
```

Using the Math Library

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

- To calculate the square root of the discriminant

```
discRoot = math.sqrt(b*b – 4*a*c)
```

quadratic.py

$$ax^2 + bx + c = 0 \qquad x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
# quadratic.py
import math  # Makes the math library available.
def main():
    print("Find the real solutions to a quadratic.")
    a, b, c = eval(input("Enter the coef (a, b, c): "))
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print("The solutions are:", root1, root2 )

main()
```

quadratic.py

- Run the program in IDLE:

```
Find the real solutions to a quadratic
Please enter the coef (a, b, c): 3, 4, -1
The solutions are: 0.215250437022 -1.54858377035
```

quadratic.py

- **What do you think this means?**

```
Find the real solutions to a quadratic
Please enter the coef (a, b, c): 1, 2, 3
Traceback (most recent call last):
  File "quadratic.py", line 10, in <module>
    main()
  File "quadratic.py", line 5, in main
    discRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
```

quadratic_debug.py

$ax^2 + bx + c = 0$ $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
# quadratic_debug.py
import math  # Makes the math library available.
def main():
    print("Find the real solutions to a quadratic.")
    a, b, c = eval(input("Enter the coef (a, b, c): "))
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = -b + discRoot / (2 * a)
    root2 = -b - discRoot / (2 * a)
    print("The solutions are:", root1, root2 )

main()
```

quadratic_debug.py

- Run the program in IDLE:

```
Find the real solutions to a quadratic
Please enter the coef (a, b, c): 3, 4, -1
The solutions are: -3.1180828963118 -4.881917103688
```

Why?

Using the Math Library

Python	Mathematics	English
`pi`	π	An approximation of pi.
`e`	e	An approximation of e.
`sin(x)`	$\sin x$	The sine of x.
`cos(x)`	$\cos x$	The cosine of x.
`tan(x)`	$\tan x$	The tangent of x.
`asin(x)`	$\arcsin x$	The inverse of sine x.
`acos(x)`	$\arccos x$	The inverse of cosine x.
`atan(x)`	$\arctan x$	The inverse of tangent x.
`log(x)`	$\ln x$	The natural (base e) logarithm of x
`log10(x)`	$\log_{10} x$	The common (base 10) logarithm of x.
`exp(x)`	e^x	The exponential of x.
`ceil(x)`	$\lceil x \rceil$	The smallest whole number $>= x$
`floor(x)`	$\lfloor x \rfloor$	The largest whole number $<= x$

Using the Math Library

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>> math.sin(1)
0.8414709848078965
>>> math.log(0.5)
-0.6931471805599453
```

```
>>> math.exp(4)  # math.e**4
54.598150033144236
>>> math.ceil(4.3) #round up
5
>>> math.floor(4.3) #rnd down
4
```

Using the Math Library

```
>>> from math import *
>>> pi
3.141592653589793
>>> e
2.718281828459045
>>> sin(1)
0.8414709848078965
>>> log(0.5)
-0.6931471805599453
```

```
>>> exp(4)
54.598150033144236
>>> ceil(4.3)
5
>>> floor(4.3)
4
```

Accumulating Results: Factorial

- How many ways are there to arrange six cats in line?
- 6! = 6 * 5 * 4 * 3 * 2 * 1 = 720

Accumulating Results: Factorial

- 6! = 6 * 5 * 4 * 3 * 2 * 1 = 720

- 6 * 5 = 30
- Then 30 * 4 = 120
- Then 120 * 3 = 360
- Then 360 * 2 = 720
- Then 720 * 1 = 720

Accumulating Results: Factorial

- We’re doing repeated multiplications, and we keep track of the *running product*
- This algorithm is known as an *accumulator*
- We’re building up the answer in a variable, known as the *accumulator variable*

Accumulating Results: Factorial

- The general form of an accumulator algorithm
  - Initialize the accumulator variable
  - Loop until final result is reached
  - Update the value of accumulator variable

Accumulating Results: Factorial

```
>>> fact = 1
>>> for i in [6, 5, 4, 3, 2, 1]:
...     fact = fact * i
...     print(fact)
>>>
6
30
120
360
720
720
```


BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Accumulating Results: Factorial

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

```
>>> fact = 1
>>> for i in [2, 3, 4, 5, 6]:
...     fact = fact * i
...     print(fact)
>>>
2
6
24
120
720
```

Accumulating Results: Factorial

- What if we want to find the factorial of some other number? *$n! = n(n-1)(n-2)...(1)$*
- Write a program to do this
  - Input number, n
  - Compute factorial of n, fact
  - Output fact

Accumulating Results: Factorial

- *range(n)* returns
  0, 1, 2, 3, …, n-1
- *range(start, n)* returns
  start, start+1, …, n-1
- *range(start, n, step)* returns
  start, start+step, …, n-1
- *list(<sequence>)* to make a list

Accumulating Results: Factorial

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

Accumulating Results: Factorial

- Using the *range* statement, we can simplify the factorial calculation
  - We can count up from 2 to n:
    `range(2, n+1)`
  - We can count down from n to 2:
    `range(n, 1, -1)`

Accumulating Results: Factorial

```
# factorial.py
# Compute the factorial of a number
# Illustrates the for loop with an accumulator
def main():
    n = eval(input("Enter a whole number: "))
    fact = 1
    for i in range(n,1,-1):
        fact = fact * i
    print("The factorial of", n, "is", fact)

main()
```

The Limits of Int

- Run the program in IDLE:

```
Please enter a whole number: 100
The factorial of 100 is
  93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000
```

Handling Large Numbers

- If we initialize the accumulator to a float number

```
fact = 1.0
```

- Run the program in IDLE:

```
Please enter a whole number: 100
The factorial of 15 is 9.332621544394418e+157
```

- We no longer get an exact answer

Handling Large Numbers

- Very large and very small numbers are expressed in *scientific* or *exponential notation*
- 1.3e+002 means 1.3 * 10^2

```
>>> 1.3E+2
130.0
>>> 1.3e-2
0.013
```

Handling Large Numbers

- Python ints are not a fixed size and expand to handle whatever value it holds
- Floats are approximations
- Floats allow us to represent a larger range of values, but with lower precision

Handling Large Numbers

- Python automatically convert ints to expanded form (when they grow very large) to avoid memory overflow
- We get very large values (e.g. 1000!, 10000!, 1000000!) at the cost of speed and memory

Handling Large Numbers

- If your command window program freezes, try:
  - CTRL+C
  - Close the window by clicking the button [X]
  - Restart the computer
  - Unplug and remove battery

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 = 5.4 + 2
>>> x
7.4
```

Type Conversions

- In *mixed-typed expressions* Python will convert ints to floats
- Python converts 2 to 2.0 and do a floating point addition
- Converting a float to an int will lose information (5.4 → 5)
- Ints can be converted to floats by adding ".0"

Type Conversions

- Sometimes we want to control the type conversion (*explicit typing*)

```
>>> float(22//5)      #22 = 4*5+2
4.0
>>> int(4.5)
4
>>> int(3.9)
3
>>> round(3.9)
4
>>> round(3)
3
```


BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Debugging Exercise

```
# debug1.py
def main():
    a = 1
    a
    b = 2
    print(b)

main()
```

Debugging Exercise

```
# debug2.py
def main():
        b = 2
        print(b)


main()
```

Silicon Valley - S03E06 (3)

Debugging Exercise

```
# debug3.py
def main():
   b = 2
      print(b)

main()
```

Debugging Exercise

```
# debug4.py
def main():
    print(“Hello, world!”)

main()
```