Python Programming: An Introduction to Computer Science

Chapter 3
Computing with Numbers
Outline

- Numeric data type
- Using Math library
- The limits of int
- Handling large numbers
- Type conversion
Outline

- Numeric data type
- Using Math library
- The limits of int
- Handling large numbers
- Type conversion
Numeric Data Types

- Example of count the changes
- Example of convert the temperatures
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)

- Type statement
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!
Outline

- Numeric data type
- **Using Math library**
- The limits of int
- Handling large numbers
- Type conversion
# Operators in python

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Float division</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>abs()</td>
<td>Absolute value</td>
</tr>
<tr>
<td>//</td>
<td>Integer division</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
</tr>
</tbody>
</table>
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

- 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 how to do is find a square root… but it’s in the math library!
Using the Math Library

- To use a library, we need to make sure this line is in our program:
  \textit{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)`
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 we could we write a program to do this?
- Input number to take factorial of, n
- Compute factorial of n, fact
- Output fact
Accumulating Results: Factorial

- How did we calculate 6!?
- \(6 \times 5 = 30\)
- Take that 30, and \(30 \times 4 = 120\)
- Take that 120, and \(120 \times 3 = 360\)
- Take that 360, and \(360 \times 2 = 720\)
- Take that 720, and \(720 \times 1 = 720\)
Accumulating Results: Factorial

- What’s really going on?
- We’re doing repeated multiplications, and we’re keeping track of the running product.
- This algorithm is known as an accumulator, because we’re building up or accumulating the answer in a variable, known as the accumulator variable.
Accumulating Results: Factorial

- The general form of an accumulator algorithm looks like this:
  1. Initialize the accumulator variable
  2. 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 $\text{range}(n)$ return? $0, 1, 2, 3, \ldots, n-1$
- $\text{range}$ has another optional parameter! $\text{range}(\text{start}, n)$ returns start, start + 1, \ldots, n-1
- But wait! There’s more! $\text{range}(\text{start}, n, \text{step})$ start, start+step, \ldots, 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]

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

>>> 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)
    ```
Outline

- Numeric data type
- Using Math library
- The limits of int
- Handling large numbers
- Type conversion
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?
Outline

- Numeric data type
- Using Math library
- The limits of int
- Handling large numbers
- Type conversion
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

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

• We no longer get an exact answer!
Handling Large Numbers: Long Int

- 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

- Numeric data type
- Using Math library
- The limits of int
- Handling large numbers
- 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*.