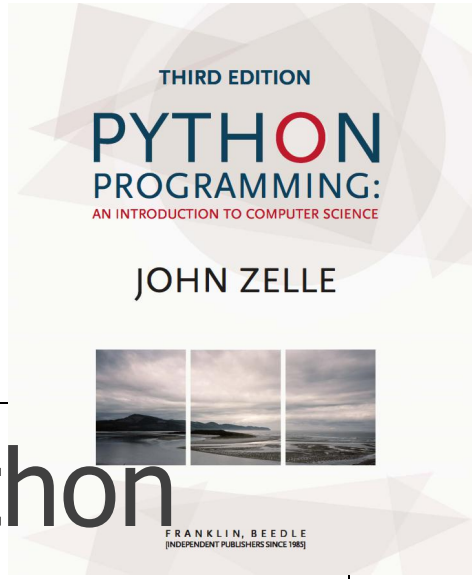


An Introduction to Python Programming

Chapter 3:Computing with Numbers

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.)

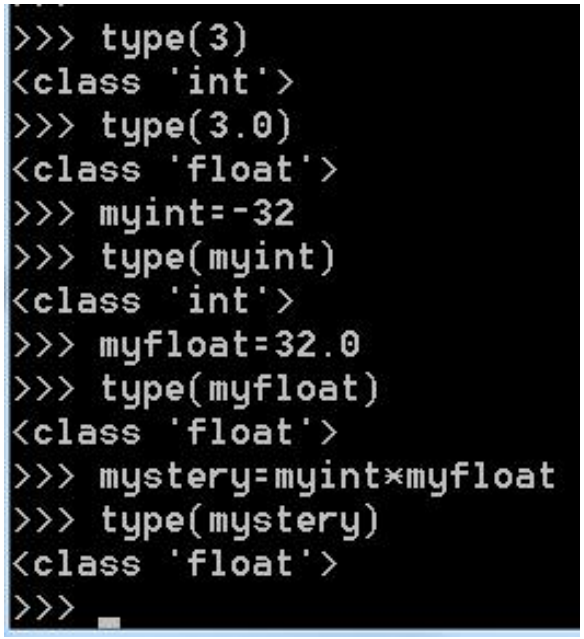
- To be able to use the Python math library.
- To understand the accumulator program pattern.
- To be able to read and write programs that process numerical data.

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---***integer*** (*int* for short) data type.
  - (.25, .10, .05, .01) are decimal fractions---***floating point*** (or *float*) values.

Numeric Data Types

- How can we tell which is which?
- Python has a special function to tell us the data type of any value.



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

- Pthon built-in numeric operations

operator	operation
+	addition
−	subtraction
*	multiplication
/	float division
**	exponentiation
abs()	absolute value
//	integer division
%	remainder

```
>>> 10 / 3
3.3333333333333335
>>> 10.0 / 3.0
3.3333333333333335
>>> 10 / 5
2.0
>>> 10 // 3
3
>>> 10.0 // 3.0
3.0
>>> 10 % 3
1
>>> 10.0 % 3.0
1.0
```

Type Conversions and Rounding

- How you think Python should handle this situation?

```
x = 5.0 * 2
```

- Sometimes we may want to perform a type conversion ourselves. This iscalled an explicit type conversion.

Numeric Data Types

- Pthon provides the built-in functions int\float\round

```
>>> int(4.5)
4
>>> int(3.9)
3
>>> float(4)
4.0
>>> float(4.5)
4.5
>>> float(int(3.3))
3.0
>>> int(float(3.3))
3
>>> int(float(3))
3
```

```
>>> round(3.14)
3
>>> round(3.5)
4
>>> pi = 3.141592653589793
>>> round(pi, 2)
3.14
>>> round(pi,3)
3.142
>>> int("32")
32
>>> float("32")
32.0
```

Numeric Data Types

- An improved version of the change-counting program

```
# change2.py
#   A program to calculate the value of some change in dollars

def main():
    print("Change Counter")
    print()
    print("Please enter the count of each coin type.")
    quarters = int(input("Quarters: "))
    dimes = int(input("Dimes: "))
    nickels = int(input("Nickels: "))
    pennies = int(input("Pennies: "))
    total = .25*quarters + .10*dimes + .05*nickels + .01*pennies
    print()
    print("The total value of your change is", total)

main()
```

Numeric Data Types

- Use numeric type conversions in place of eval

```
>>> # simultaneous input using eval
>>> x,y = eval(input("Enter (x,y): "))
Enter (x,y): 3,4
>>> x
3
>>> y
4
>>> # does not work with float
>>> x,y = float(input("Enter (x,y): "))
Enter (x,y): 3,4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: '3,4'
```

Using the Math Library

- A *library* is a module with some useful definitions/functions.
- Let's write a program to compute the roots of a quadratic equation $ax^2+bx+c=0$.

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

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 = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient 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

- What do you suppose this means?

```
Enter coefficient a: 1
Enter coefficient b: 2
Enter coefficient c: 3

Traceback (most recent call last):
  File "quadratic.py", line 21, in ?
    main()
  File "quadratic.py", line 14, in main
    discRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
```

- Just assume that the user will give us solvable equations.

Math Library

- some of the other functions available in the math library:

Python	mathematics	English
`pi`	π	An approximation of pi.
`e`	e	An approximation of e.
`sqrt(x)`	$\sqrt{x}$	The square root of x.
`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 $<= x$.

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 --- the factorial of 6 (or 6!)

- Factorial is defined as: *n! = n(n-1)(n-2)...(1)*

  *So, 6! = 6*5*4*3*2*1 = 720*

- 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

- Obviously the trick part here is in the second step.

```
Initialize the accumulator variable
Loop until final result is reached
    update the value of accumulator variable
```

- Realizing this is just need to fill in the details.

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

Accumulating Results: Factorial

- Whenever you use the accumulator pattern, make sure you include the proper initialization.
- Since multiplication is associative and commutative, we can rewrite our program as:

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

- Any questions?

Accumulating Results: Factorial

- But what if we want to find the factorial of some other number??
- Use the Python range function!
- What about *range(n)* ,*range(start, n)* ,*range(start, n, step)?*

Accumulating Results: Factorial

- Let's try some examples!

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

Accumulating Results: Factorial

- We can do the range for our loop in different ways.

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

def main():
    n = int(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()
```

Limitations of Computer Arithmetic

- What is 100!?

The factorial of 100 is

933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000

- Here's what happens in several runs of a similar program written using Java:

```
# run 2
Please enter a whole number: 12
The factorial is: 479001600

# run 3
Please enter a whole number: 13
The factorial is: 1932053504
```

Limitations of Computer Arithmetic

- It is important to keep in mind that computer representations of numbers do not always behave exactly like the numbers that they stand for.
- Inside the computer, ints are stored in a fxed-sized binary representation.
- This range depends on the number of *bits* a particular CPU uses to represent an integer value. Typical PCs use 32 bits.

Limitations of Computer Arithmetic

- What's going on at the hardware level?
- Computer memory is composed of electrical "switches" .
- Typical PCs today use 32 or 64 bits. For a 32-bit CPU, the range of integers that can be represented is -2^{31} to $2^{31}- 1$.

bit 2	bit 1
0	0
0	1
1	0
1	1

bit 3	bit 2	bit 1
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Limitations of Computer Arithmetic

- Python uses the float data type to get us around the size limitation of the ints?

```
>>> def main():
...     n=int(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()
please enter a whole number: 30
The factorial of 30 is 265252859812191058636308480000000
>>> 30.0
30.0
>>> main()
please enter a whole number: 30.0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in main
ValueError: invalid literal for int() with base 10: '30.0'
>>> _
```

Limitations of Computer Arithmetic

- In fact, a computer stores floating-point numbers as a pair of fixed-length integers,the **mantissa** (represents the string of digits) and the **exponent** (keeps track of where the whole part ends and the factional part begins)
- Only fractions that involve powers of 2 can be represented exactly.
- A Python Integer Is More Than Just an Integer!

Limitations of Computer Arithmetic

- This sort of flexibility is one piece that makes Python and other dynamically-typed languages convenient and easy to use.

```python
# Python code
x = 4
x = "four"
```

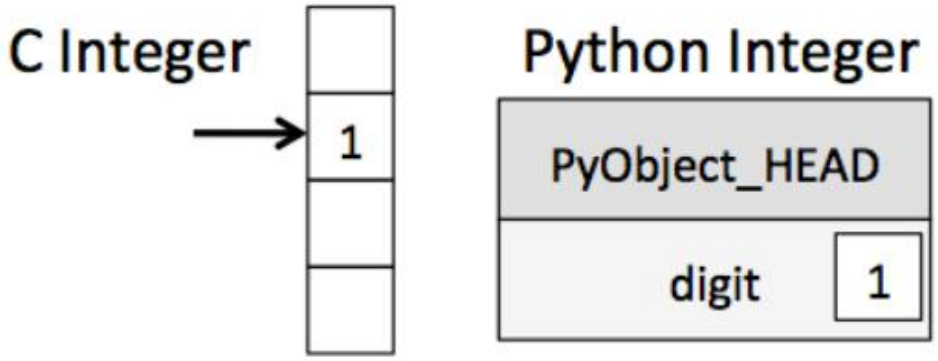
```c
/* C code */
int x = 4;
x = "four";  // FAILS
```

- In Python 3.4 source code, the integer (long) type definition effectively looks like this :

```c
struct _longobject {
    long ob_refcnt;
    PyTypeObject *ob_type;
    size_t ob_size;
    long ob_digit[1];
};
```

Limitations of Computer Arithmetic

- **ob_refcnt,** a reference count that helps Python silently handle memory allocation and deallocation
- **ob_type**, which encodes the type of the variable
- **ob_size**, which specifies the size of the following data members
- **ob_digit**, which contains the actual integer value that we expect the Python variable to represent.



Programming Exercise

- Write a program to calculate the volume and surface area of a sphere from its radius, given as input. Here are some formulas that might be useful:

$$V = 4/3\pi r^3$$

$$A = 4\pi r^2$$

- *Then write your .py files,carry out them and give me the results.*