Chapter 3

Numbers

This chapter focuses on numbers and simple mathematics in Python.

3.1 Integers and Decimal Numbers

Because of the way computer chips are designed, integers and decimal numbers are represented differently on computers. Decimal numbers are represented by what are called floating point numbers. The important thing to remember about them is you typically only get about 15 or so digits of precision. It would be nice if there were no limit to the precision, but calculations run a lot more quickly if you cut off the numbers at some point.

On the other hand, integers in Python have no restrictions. They can be arbitrarily large.

For decimal numbers, the last digit is sometimes slightly off due to the fact that computers work in binary (base 2) whereas our human number system is base 10. As an example, mathematically, we know that the decimal expansion of 7/3 is $2.333\cdots$, with the threes repeating forever. But when we type `7/3` into the Python shell, we get `2.3333333333333335`. This is called *roundoff error*. For most practical purposes this is not too big of a deal, but it actually can cause problems for some mathematical and scientific calculations. If you really need more precision, there are ways. See Section 22.5.

3.2 Math Operators

Here is a list of the common operators in Python:

Operator	Description
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`**`	exponentiation
`//`	integer division
`%`	modulo (remainder)

Exponentiation Python uses `**` for exponentiation. The caret, `^`, is used for something else.

Integer division The integer division operator, `//`, requires some explanation. Basically, for positive numbers it behaves like ordinary division except that it throws away the decimal part of the result. For instance, while `8/5` is `1.6`, we have `8//5` equal to `1`. We will see uses for this operator later. Note that in many other programming languages and in older versions of Python, the usual division operator `/` actually does integer division on integers.

Modulo The modulo operator, `%`, returns the remainder from a division. For instance, the result of `18%7` is 4 because 4 is the remainder when 18 is divided by 7. This operation is surprisingly useful. For instance, a number is divisible by n precisely when it leaves a remainder of 0 when divided by `n`. Thus to check if a number, `n`, is even, see if `n%2` is equal to 0. To check if `n` is divisible by 3, see if `n%3` is 0.

One use of this is if you want to schedule something in a loop to happen only every other time through the loop, you could check to see if the loop variable modulo 2 is equal to 0, and if it is, then do that something.

The modulo operator shows up surprisingly often in formulas. If you need to "wrap around" and come back to the start, the modulo is useful. For example, think of a clock. If you go six hours past 8 o'clock, the result is 2 o'clock. Mathematically, this can be accomplished by doing a modulo by 12. That is, `(8+6)%12` is equal to 2.

As another example, take a game with players 1 through 5. Say you have a variable `player` that keeps track of the current player. After player 5 goes, it's player 1's turn again. The modulo operator can be used to take care of this:

```
player = player%5+1
```

When `player` is 5, `player%5` will be 0 and expression will set `player` to 1.

3.3 Order of operations

Exponentiation gets done first, followed by multiplication and division (including `//` and `%`), and addition and subtraction come last. The classic math class mnemonic, PEMDAS (Please Excuse My Dear Aunt Sally), might be helpful.

This comes into play in calculating an average. Say you have three variables `x`, `y`, and `z`, and you want to calculate the average of their values. To expression `x+y+z/3` would not work. Because division comes before addition, you would actually be calculating $x + y + \frac{z}{3}$ instead of $\frac{x+y+z}{3}$. This is easily fixed by using parentheses: `(x+y+z)/3`.

In general, if you're not sure about something, adding parentheses might help and usually doesn't do any harm.

3.4 Random numbers

To make an interesting computer game, it's good to introduce some randomness into it. Python comes with a module, called `random`, that allows us to use random numbers in our programs.

Before we get to random numbers, we should first explain what a *module* is. The core part of the Python language consists of things like for loops, if statements, math operators, and some functions, like **`print`** and **`input`**. Everything else is contained in modules, and if we want to use something from a module we have to first *import* it—that is, tell Python that we want to use it.

At this point, there is only one function, called `randint`, that we will need from the `random` module. To load this function, we use the following statement:

```
from random import randint
```

Using `randint` is simple: `randint(a,b)` will return a random integer between `a` and `b` including both `a` and `b`. (Note that `randint` includes the right endpoint `b` unlike the **`range`** function). Here is a short example:

```
from random import randint
x = randint(1,10)
print('A random number between 1 and 10: ', x)
```

```
A random number between 1 and 10: 7
```

The random number will be different every time we run the program.

3.5 Math functions

The `math` module Python has a module called `math` that contains familiar math functions, including `sin`, `cos`, `tan`, `exp`, `log`, `log10`, `factorial`, `sqrt`, `floor`, and `ceil`. There are also the inverse trig functions, hyperbolic functions, and the constants `pi` and `e`. Here is a short example:

```
from math import sin, pi
print('Pi is roughly', pi)
print('sin(0) =', sin(0))
```

```
Pi is roughly 3.14159265359
sin(0) = 0.0
```

Built-in math functions There are two built in math functions, **abs** (absolute value) and **round** that are available without importing the `math` module. Here are some examples:

```
print(abs(-4.3))
print(round(3.336, 2))
print(round(345.2, -1))
```

```
4.3
3.37
350.0
```

The **round** function takes two arguments: the first is the number to be rounded and the second is the number of decimal places to round to. The second argument can be negative.

3.6 Getting help from Python

There is documentation built into Python. To get help on the `math` module, for example, go to the Python shell and type the following two lines:

```
>>> import math
>>> dir(math)
```

```
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp',
'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

This gives a list of all the functions and variables in the `math` module. You can ignore all of the ones that start with underscores. To get help on a specific function, say the `floor` function, you can type **help**`(math.floor)`. Typing **help**`(math)` will give you help for everything in the `math` module.

3.7 Using the Shell as a Calculator

The Python shell can be used as a very handy and powerful calculator. Here is an example session:

```
>>> 23**2
529
>>> s = 0
>>> for n in range(1,10001):
        s = s + 1/n**2

>>> s
1.6448340718480652
>>> from math import *
>>> factorial(10)
3628800
```

The second example here sums the numbers $1 + 1/4 + 1/9 + \cdots + 1/10000^2$. The result is stored in the variable `s`. To inspect the value of that variable, just type its name and press enter. Inspecting variables is useful for debugging your programs. If a program is not working properly, you can type your variable names into the shell after the program has finished to see what their values are.

The statement `from math import *` imports every function from the `math` module, which can make the shell a lot like a scientific calculator.

Note Under the Shell menu, select `Restart shell` if you want to clear the values of all the variables.

3.8 Exercises

1. Write a program that generates and prints 50 random integers, each between 3 and 6.
2. Write a program that generates a random number, x, between 1 and 50, a random number y between 2 and 5, and computes x^y.
3. Write a program that generates a random number between 1 and 10 and prints your name that many times.
4. Write a program that generates a random decimal number between 1 and 10 with two decimal places of accuracy. Examples are 1.23, 3.45, 9.80, and 5.00.
5. Write a program that generates 50 random numbers such that the first number is between 1 and 2, the second is between 1 and 3, the third is between 1 and 4, ..., and the last is between 1 and 51.
6. Write a program that asks the user to enter two numbers, x and y, and computes $\frac{|x-y|}{x+y}$.
7. Write a program that asks the user to enter an angle between $-180°$ and $180°$. Using an expression with the modulo operator, convert the angle to its equivalent between $0°$ and $360°$.

8. Write a program that asks the user for a number of seconds and prints out how many minutes and seconds that is. For instance, 200 seconds is 3 minutes and 20 seconds. [Hint: Use the `//` operator to get minutes and the `%` operator to get seconds.]

9. Write a program that asks the user for an hour between 1 and 12 and for how many hours in the future they want to go. Print out what the hour will be that many hours into the future. An example is shown below.

```
Enter hour: 8
How many hours ahead? 5
New hour: 1 o'clock
```

10. (a) One way to find out the last digit of a number is to mod the number by 10. Write a program that asks the user to enter a power. Then find the last digit of 2 raised to that power.

    (b) One way to find out the last two digits of a number is to mod the number by 100. Write a program that asks the user to enter a power. Then find the last two digits of 2 raised to that power.

    (c) Write a program that asks the user to enter a power and how many digits they want. Find the last that many digits of 2 raised to the power the user entered.

11. Write a program that asks the user to enter a weight in kilograms. The program should convert it to pounds, printing the answer rounded to the nearest tenth of a pound.

12. Write a program that asks the user for a number and prints out the factorial of that number.

13. Write a program that asks the user for a number and then prints out the sine, cosine, and tangent of that number.

14. Write a program that asks the user to enter an angle in degrees and prints out the sine of that angle.

15. Write a program that prints out the sine and cosine of the angles ranging from 0 to 345° in 15° increments. Each result should be rounded to 4 decimal places. Sample output is shown below:

```
0 --- 0.0 1.0
15 --- 0.2588 0.9659
30 --- 0.5 0.866
...
345 --- -0.2588 0.9659
```

16. Below is described how to find the date of Easter in any year. Despite its intimidating appearance, this is not a hard problem. Note that $\lfloor x \rfloor$ is the *floor* function, which for positive numbers just drops the decimal part of the number. For instance $\lfloor 3.14 \rfloor = 3$. The floor function is part of the `math` module.

    C = century (1900's $\rightarrow C = 19$)

Y = year (all four digits)

$m = (15 + C - \lfloor \frac{C}{4} \rfloor - \lfloor \frac{8C+13}{25} \rfloor) \bmod 30$

$n = (4 + C - \lfloor \frac{C}{4} \rfloor) \bmod 7$

$a = Y \bmod 4$

$b = Y \bmod 7$

$c = Y \bmod 19$

$d = (19c + m) \bmod 30$

$e = (2a + 4b + 6d + n) \bmod 7$

Easter is either March $(22 + d + e)$ or April $(d + e - 9)$. There is an exception if $d = 29$ and $e = 6$. In this case, Easter falls one week earlier on April 19. There is another exception if $d = 28$, $e = 6$, and $m = 2, 5, 10, 13, 16, 21, 24$, or 39. In this case, Easter falls one week earlier on April 18. Write a program that asks the user to enter a year and prints out the date of Easter in that year. (See Tattersall, *Elementary Number Theory in Nine Chapters*, 2nd ed., page 167)

17. A year is a leap year if it is divisible by 4, except that years divisible by 100 are not leap years unless they are also divisible by 400. Ask the user to enter a year, and, using the // operator, determine how many leap years there have been between 1600 and that year.

18. Write a program that given an amount of change less than $1.00 will print out exactly how many quarters, dimes, nickels, and pennies will be needed to efficiently make that change. [Hint: the // operator may be useful.]

19. Write a program that draws "modular rectangles" like the ones below. The user specifies the width and height of the rectangle, and the entries start at 0 and increase typewriter fashion from left to right and top to bottom, but are all done mod 10. Below are examples of a 3×5 rectangle and a 4×8.

```
0 1 2 3 4
5 6 7 8 9
0 1 2 3 4

0 1 2 3 4 5 6 7
8 9 0 1 2 3 4 5
6 7 8 9 0 1 2 3
4 5 6 7 8 9 0 1
```