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: