Chapter 2

THE INTERACTIVE SHELL

Topics Covered In This Chapter:

- Integers and Floating Point Numbers
- Expressions
- Values
- Operators
- Evaluating Expressions
- Storing Values in Variables
- Overwriting variables

Before you can make games, you need to learn a few basic programming concepts. You won't make games in this chapter, but learning these concepts is the first step to programming video games. Start by learning how to use Python's interactive shell.

Some Simple Math Stuff

Open IDLE using the steps in Chapter 1, then get Python to solve some simple math stuff. The interactive shell can work just like a calculator. Type `2 + 2` into the interactive shell and press the ENTER key on your keyboard. (On some keyboards, this is the RETURN key.) Figure 2-1 shows how the IDLE responds with the number `4`.

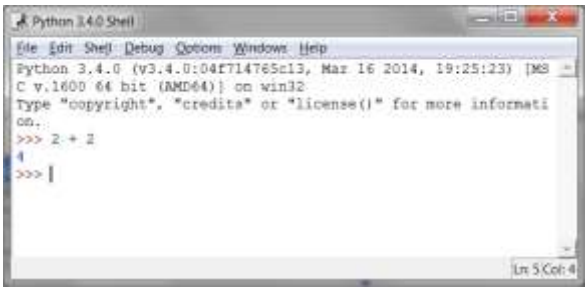

Figure 2-1: Enter 2+2 into the interactive shell.

This math problem is a simple programming instruction. The + sign tells the computer to add the numbers 2 and 2. Table 2-1 lists the other math symbols available in Python. The - sign will subtract numbers. The * asterisk will multiply numbers. The / slash will divide numbers.

Table 2-1: The various math operators in Python.

Operator	Operation
+	addition
-	subtraction
*	multiplication
/	division

When used in this way, +, -, *, and / are called **operators**. Operators tell Python which mathematical operation on the numbers surrounding them.

Integers and Floating Point Numbers

In programming, **integers** or **ints** are whole numbers such as 4, 99, and 0. **Floating point numbers** or **floats** are fractions or numbers with decimal points like 3.5, 42.1 and 5.0.. In Python, the number 5 is an integer, but 5.0 is a float.

Expressions

These math problems are examples of expressions. Computers can solve millions of these problems in seconds. Expressions are made up of **values** (the numbers) connected by **operators** (the math signs). Let's learn exactly what values and operators are. Try entering some of these math problems into the interactive shell, pressing ENTER key after each one.

```
2+2+2+2+2
8*6
10-5+6
2   +         2
```

After you type in the above instructions, the interactive shell will look like Figure 2-2.

Figure 2-2: What the IDLE window looks like after entering instructions.

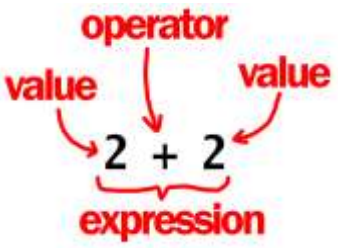

Figure 2-3: An expression is a made up of values and operators.

An **expression** is made up of values (such as integers like `8` and `6`) connected by an operator (such as the `*` multiplication sign). A single value by itself is also an expression. In the `2 + 2` example, notice that there can be any amount of spaces between the values and operators. However, don't put any spaces at the beginning of the instruction when entering them into the interactive shell.

In the next chapter, you will learn about working with text in expressions. Python isn't limited to just numbers; it's more than a calculator!

Evaluating Expressions

When a computer solves the expression `10 + 5` and gets the value `15`, it has **evaluated** the expression. Evaluating an expression reduces the it to a single value, just like solving a math problem reduces the problem to a single number: the answer.

The expressions `10 + 5` and `10 + 3 + 2` have the same value. They both evaluate to `15`.

Expressions can be of any size, but they will always evaluate down to a single value. Even single values are expressions: The expression `15` evaluates to the value `15`. For example, the expression `8 * 3 / 2 + 2 + 7 - 9` will evaluate down to the value `12.0` through the following steps:

```
8 * 3 / 2 + 2 + 7 - 9
                  ▼
   24 / 2 + 2 + 7 - 9
                  ▼
     12.0 + 2 + 7 - 9
                  ▼
         14.0 + 7 - 9
                  ▼
             21.0 - 9
                  ▼
                12.0
```

You don't see all of these steps in the interactive shell. The interactive shell does them and just shows you the results:

```
>>> 8 * 3 / 2 + 2 + 7 - 9
12.0
```

Notice that the `/` division operator evaluates to a float value, as in `24 / 2` evaluating to `12.0`. Also, math operations with float values also evaluate to float values, as in `12.0 + 2` evaluating to `14.0`.

Syntax Errors

If you enter `5 +` into the interactive shell, you'll get an error message.

```
>>> 5 +
SyntaxError: invalid syntax
```

This error happened because `5 +` isn't an expression. Expressions have values connected by operators. But the + operator expects a value after the + sign. An error message appears when this value is missing.

`SyntaxError` means Python doesn't understand the instruction because you typed it incorrectly. A lot of computer programming isn't just telling the computer what to do, but also knowing exactly how to tell it.

Don't worry about making mistakes though. Errors don't damage your computer. Just retype the instruction correctly into the interactive shell at the next >>> prompt.

Storing Values in Variables

You'll often want to save the value an expression evaluates to so you can use it later in the program. You can store values in **variables**. Think of variables like a box that can hold values.

An **assignment statement** instruction will store a value (or the value an expression evaluates to) inside a variable. Make up a name for the variable, followed by the = sign (called **the assignment operator**), and then the value to store in the variable. For example, enter `spam = 15` into the interactive shell:

```
>>> spam = 15
>>>
```

Figure 2-4: Variables are like boxes that can hold values in them.

The `spam` variable's box will have the value 15 stored in it, as shown in Figure 2-4. The name "spam" is the label on the box (so Python can tell variables apart) and the value is written on a small note inside the box.

When you press ENTER you won't see anything in response. In Python, assume the instruction executed successfully if no error message appears. The `>>>` prompt will appear so you can type in the next instruction.

Unlike expressions, **statements** are instructions that do not evaluate to any value, which is why there is no value displayed on the next line in the interactive shell. It might be confusing to know which instructions are expressions and which are statements. Just remember that expressions evaluate to a single value. Any other kind of instruction is a statement.

Variables store values, not expressions. For example, consider the expression in the statements `spam = 10 + 5` and `spam = 10 + 7 - 2`. They both evaluate to `15`. The end result is the same: Both statements store the value `15` in the variable `spam`.

The first time a variable is used in an assignment statement, Python will create that variable. To check what value is in a variable, type the variable name into the interactive shell:

```
>>> spam = 15
>>> spam
15
```

The expression `spam` evaluates to the value inside the `spam` variable: `15`. You can use variables in expressions. Try entering the following in the interactive shell:

```
>>> spam = 15
>>> spam + 5
```

```
20
```

You've set the value of the variable `spam` to `15`, so writing `spam + 5` is like writing the expression `15 + 5`.

You cannot use a variable before an assignment statement creates it. Python will give you an error because no such variable exists yet. Mistyping the variable name also causes this error:

```
>>> spam = 15
>>> spma
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    spma
NameError: name 'spma' is not defined
```

The error appeared because there is `spam` variable but not `spma`.

You can change the value stored in a variable by entering another assignment statement. For example, try the following:

```
>>> spam = 15
>>> spam + 5
20
>>> spam = 3
>>> spam + 5
8
```

When you first enter `spam + 5`, the expression evaluates to `20` because you stored `15` inside `spam`. However, when you enter `spam = 3`, the value `15` is replaced, or **overwritten**, with the value `3`. Now when you enter `spam + 5`, the expression evaluates to `8` because the value of `spam` is now `3`.

You can even use the value in the `spam` variable to assign a new value to `spam`:

```
>>> spam = 15
>>> spam = spam + 5
20
```

The assignment statement `spam = spam + 5` is like saying, "the new value of the `spam` variable will be the current value of `spam` plus five." Keep increasing the value in `spam` by `5` several times:

```
>>> spam = 15
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam
30
```

Using More Than One Variable

Create as many variables as you need in your programs. For example, let's assign different values to two variables named `eggs` and `fizz`, like so:

```
>>> fizz = 10
>>> eggs = 15
```

Now the `fizz` variable has `10` inside it, and `eggs` has `15` inside it.

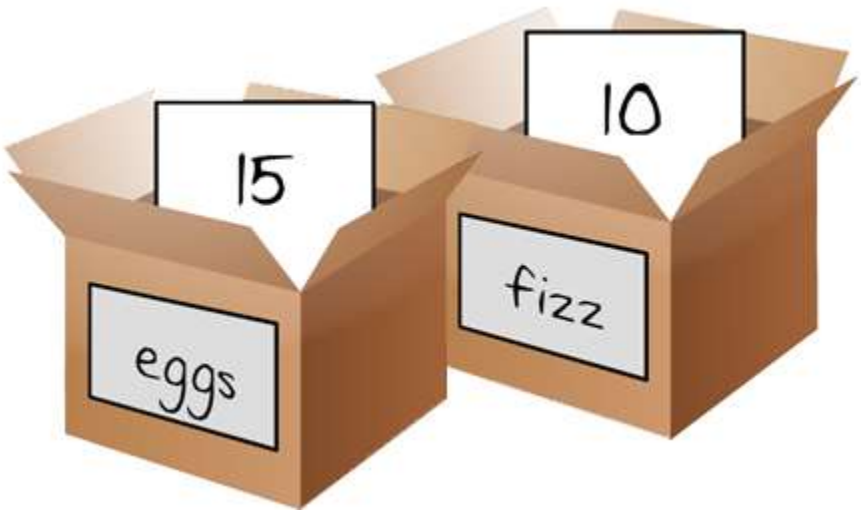

Figure 2-5: The "fizz" and "eggs" variables have values stored in them.

Try entering `spam = fizz + eggs` into the interactive shell, then check the new value of `spam`:

```
>>> fizz = 10
>>> eggs = 15
>>> spam = fizz + eggs
>>> spam
25
```

The value in `spam` is now `25`. When you added `fizz` and `eggs` you are adding their values, which are 10 and 15, respectively.

Summary

In this chapter, you learned the basics about writing Python instructions. Python needs you to tell it exactly what to do in a strict way. Computers don't have common sense and only understand specific instructions.

Expressions are values (such as `2` or `5`) combined with operators (such as `+` or `-`). Python can evaluate expressions (that is, reduce the expression to a single value). You can store values inside of variables so that your program can remember them and use them later.

There are many other types of operators and values in Python. In the next chapter, you'll go over some more basic concepts and write your first program!