Chapter 1 - Some points:

Reference: *Python for Everyone, 2nd Edition, Cay S. Horstmann, Rance D. Necaise, Wiley, 2016.*

Programming Tip 1.1

Interactive Mode

When you write a complete program, you place the program instructions in a file and let the Python interpreter execute your program file. The interpreter, however, also provides an interactive mode in which Python instructions can be entered one at a time. To launch the Python interactive mode from a terminal window, enter the command

```
python
```

(On systems where multiple versions of Python are installed, use the command `python3` to run version 3 of Python.) Interactive mode can also be started from within most Python integrated development environments.

The interface for working in interactive mode is known as the **Python shell**. First, you will see an informational message similar to the following:

```
Python 3.1.4 (default, Nov 3 2014, 14:38:10)
[GCC 4.9.1 20140930 (Red Hat 4.9.1-11)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The `>>>` at the bottom of the output is the **prompt**. It indicates that you can enter Python instructions. (Your prompt may look different, such as `In [1]:`.) After you type an instruction and press the Enter key, the code is immediately executed by the Python interpreter. For example, if you enter

```
print("Hello, World!")
```

the interpreter will respond by executing the `print` function and displaying the output, followed by another prompt:

```
>>> print("Hello, World!")
Hello World
>>>
```

Interactive mode is very useful when you are first learning to program. It allows you to experiment and test individual Python instructions to see what happens. You can also use interactive mode as a simple calculator. Just enter mathematical expressions using Python syntax:

```
>>> 7035 * 0.15
1055.25
>>>
```

Make it a habit to use interactive mode as you experiment with new language constructs.

Programming Tip 1.2

Backup Copies

© Tatiana Popova/iStockphoto.

You will spend many hours creating and improving Python programs. It is easy to delete a file by accident, and occasionally files are lost because of a computer malfunction. Retyping the contents of lost files is frustrating and time-consuming. It is therefore crucially important that you learn how to safeguard files and get in the habit of doing so *before* disaster strikes. Backing up files on a memory stick is an easy and convenient storage method for many people. Another increasingly popular form of backup is Internet file storage. Here are a few pointers to keep in mind:

Develop a strategy for keeping backup copies of your work before disaster strikes.

- *Back up often.* Backing up a file takes only a few seconds, and you will hate yourself if you have to spend many hours recreating work that you could have saved easily. We recommend that you back up your work once every thirty minutes.
- *Rotate backups.* Use more than one directory for backups, and rotate them. That is, first back up onto the first directory. Then back up onto the second directory. Then use the

third, and then go back to the first. That way you always have three recent backups. If your recent changes made matters worse, you can then go back to the older version.

- *Pay attention to the backup direction.* Backing up involves copying files from one place to another. It is important that you do this right—that is, copy from your work location to the backup location. If you do it the wrong way, you will overwrite a newer file with an older version.
- *Check your backups once in a while.* Double-check that your backups are where you think they are. There is nothing more frustrating than to find out that the backups are not there when you need them.
- *Relax, then restore.* When you lose a file and need to restore it from a backup, you are likely to be in an unhappy, nervous state. Take a deep breath and think through the recovery process before you start. It is not uncommon for an agitated computer user to wipe out the last backup when trying to restore a damaged file.

Special Topic 1.1

The Python Interpreter

When you use the Python interpreter to execute a program, you can imagine it reading your program and executing it, one step at a time. However, that is not actually what is happening. Because one typically runs a program many times, the Python interpreter employs a division of labor. The time-consuming task of reading a program and comprehending its instructions is carried out once, by a component called a **compiler**. The compiler reads the file containing your **source code** (that is, the Python instructions that you wrote), and translates the instructions into **byte code**. Byte codes are very simple instructions understood by the **virtual machine**, a separate program that is similar to the CPU of a computer. After the compiler has translated your program into virtual machine instructions, they are executed by the virtual machine, as often as you like.

Your source code doesn't contain all the information that the virtual machine needs. For example, it does not contain the implementation of the `print` function. The virtual machine locates functions such as `print` in library modules. Generally, you need not be concerned with library modules. However, when you want to do specialized tasks, such as graphics programming, you may need to install the required libraries. The details depend on your Python environment.

You may find files containing virtual machine instructions in your file system. These files have the extension `.pyc` and are produced by the compiler. You don't have to pay much attention to these files, but don't turn them in for grading. They are only useful for the Python virtual machine, not a human grader.

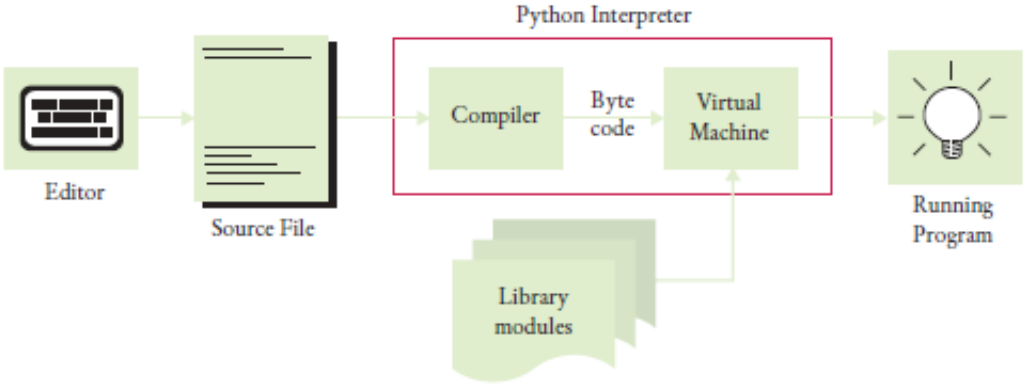

Figure 7 From Source Code to Running Program

Common Error 1.1

Misspelling Words

If you accidentally misspell a word, then strange things may happen, and it may not always be completely obvious from the error message what went wrong. Here is a good example of how simple spelling errors can cause trouble:

```
Print("Hello, World!")
print("How are you?")
```

The first statement calls the `Print` function. This is not the same as the `print` function because `Print` starts with an uppercase letter and the Python language is case sensitive. Upper- and lowercase letters are considered to be completely different from each other; to the interpreter `Print` is no better match for `print` than `pint`. Of course, the message `Name 'Print' is not defined` should give you a clue where to look for the error.

If you get an error message that seems to indicate that the Python interpreter is on the wrong track, it is a good idea to check for spelling and capitalization.

1.6 Errors

Experiment a little with the `hello.py` program. What happens if you make a typing error such as

```
print("Hello, World!)
```

(Note the missing quotation marks at the end of the greeting.) When you attempt to run the program, the interpreter will stop and display the following message

```
File "hello.py", line 2
   print("Hello, World)
                      ^
SyntaxError: EOL while scanning string literal
```

Programmers spend a fair amount of time fixing compile-time and run-time errors.

A compile-time error is a violation of the programming language rules that is detected when the code is translated into executable form.

This is a **compile-time** error. (The process of transforming Python instructions into an executable form is called *compilation*—see Special Topic 1.1). Something is wrong according to the rules of the language, and the error is detected before your program is actually

run. For this reason, compile-time errors are sometimes called **syntax errors**. When such an error is found, no executable program is created. You must fix the error and attempt to run the program again. The interpreter is quite picky, and it is common to go through several rounds of fixing compile-time errors before the program runs for the first time. In this case, the fix is simple: add a quotation mark at the end of the string.

Unfortunately, the interpreter is not very smart and often provides no help in identifying the syntax error. For example, suppose you forget both quotation marks around a string

```
print(Hello, World!)
```

The error report looks like this:

```
File "hello.py", line 2
   print(Hello, World!)
                      ^
SyntaxError: invalid syntax
```

It is up to you to realize that you need to enclose strings in quotation marks.

Some errors can only be found when the program executes. For example, suppose your program includes the statement

```
print(1 / 0)
```

This statement does not violate the rules of the Python language, and the program will start running. However, when the division by zero occurs, the program will stop and display the following error message:

```
Traceback (most recent call last):
   File "hello.py", line 3, in <module>
ZeroDivisionError: int division or modulo by zero
```

An exception occurs when an instruction is syntactically correct, but impossible to perform.

This is called an **exception**. Unlike a compile-time error, which is reported as the program code is analyzed, an exception occurs when the program runs. An exception is a **run-time error**.

There is another kind of run-time error. Consider a program that contains the following statement:

```
print("Hello, Word!")
```

A run-time error is any error that occurs when the program compiles and runs, but produces unexpected results.

The program is syntactically correct and runs without exceptions, but it doesn't produce the results we expected. Instead of printing "Hello, World!", it prints "Word" in place of "World".

Some people use the term **logic error** instead of run-time error. After all, when the program misbehaves, something is wrong with the program logic. A well-written program would make sure that there are no divisions by zero and no faulty outputs.

During program development, errors are unavoidable. Once a program is longer than a few lines, it would require superhuman concentration to enter it correctly without slipping up once. You will find yourself misspelling words, omitting quotation marks, or trying to perform an invalid operation more often than you would like. Fortunately, these problems are reported at compile-time, and you can fix them.

Run-time errors are more troublesome. They are the harder to find and fix because the interpreter cannot flag them for us. It is the responsibility of the program author to test the program and prevent any run-time errors.

CHAPTER SUMMARY

Define "computer program" and programming.

- Computers execute very basic instructions in rapid succession.
- A computer program is a sequence of instructions and decisions.
- Programming is the act of designing and implementing computer programs.

Describe the components of a computer.

- The central processing unit (CPU) performs program control and data processing.
- Storage devices include memory and secondary storage.

Describe the benefits of the Python language.

- Python is portable and easy to learn and use.

Become familiar with your Python programming environment.

- Set aside some time to become familiar with the programming environment that you will use for your class work.
- A text editor is a program for entering and modifying text, such as a Python program.
- Python is case sensitive. You must be careful about distinguishing between upper- and lowercase letters.
- The Python interpreter reads Python programs and executes the program instructions.
- Develop a strategy for keeping backup copies of your work before disaster strikes.

Describe the building blocks of a simple program.

- A comment provides information to the programmer.
- A function is a collection of instructions that perform a particular task.
- A function is called by specifying the function name and its arguments.
- A string is a sequence of characters enclosed in a pair of single or double quotation marks.

Classify program errors as compile-time and run-time errors.

- A compile-time error is a violation of the programming language rules that is detected when the code is translated into executable form.
- An exception occurs when an instruction is syntactically correct, but impossible to perform.
- A run-time error is any error that occurs when the program compiles and runs, but produces unexpected results.

Write pseudocode for simple algorithms.

- Pseudocode is an informal description of a sequence of steps for solving a problem.
- An algorithm for solving a problem is a sequence of steps that is unambiguous, executable, and terminating.

Reference: *Python for Everyone, 2nd Edition, Cay S. Horstmann, Rance D. Necaise, Wiley, 2016.*