Lesson 3. Introducing Python: a programming language

After reading lesson 3, you'll be able to

- Understand Python,the programming language you'll be using
- Use a program to write your programs
- Understand the components of a programming development environment

3.1. INSTALLING PYTHON

The Python programming language is, at the time of this writing, the most popular language for teaching introductory computer science. The language is used by top universities to expose students to programming, and many students are citing Python as a language they're familiar with upon entering college. Broadly, Python is used to build applications and websites, and is being used behind the scenes by companies such as NASA, Google, Facebook, and Pinterest to maintain features and analyze collected data.

Python is a great general-purpose language that can be used to write quick and simple programs. After you set up a working environment, writing a program in Python doesn't require much setup.

3.1.1. What is Python?

Python is a programming language created by Guido van Rossum at Centrum Wiskunde & Informatica in the Netherlands. But the name *Python* is also used to refer to the interpreter.

Definition

A Python interpreter is a program used to run programs written in the Python programming language.

In the Python programming language, every *thing*, called an *object*, has characteristics (data) associated with it and ways to interact with it. For example, any word is a thing, or object, in Python. The data associated with the word *summer* is the letter characters in that sequence. One way that you can interact with the word is to change every letter to be uppercase. An example of a more complicated object is a bicycle. The data associated with a bicycle could be the number of wheels, its height, its length, and its color. The actions that a bike can do might be that it can fall over, a person can ride it, and you can repaint it.

In this book, you'll be writing programs in the latest Python version at the time of this writing, version 3.5.

3.1.2. Downloading Python version 3.5

You can download Python version 3.5 in various ways; you can get it from the official Python website, www.python.org (http://www.python.org), or through any third-party programs that offer the Python language as well as extra packages preinstalled. In this book, I recommend that you download a specific third-party program called the *Anaconda Python Distribution*.

3.1.3. Anaconda Python Distribution

You can download the Anaconda Python Distribution from www.anaconda.com (http://www.anaconda.com). This free Python distribution offers various versions of Python and includes more than 400 of the most popular packages for science, math, engineering, and data analysis. There's also a lighter version, without any of the extra packages, called *Miniconda*.

Go to the downloads page, www.anaconda.com/downloads (http://www.anaconda.com/downloads), and choose the Python 3.5 download link for your appropriate operating system. Follow the installation instructions with the default values to install the distribution on your computer. Note that the latest ver-

3.1.4. Integrated development environments

After the installation is complete, open Spyder, a program part of Anaconda. Spyder is an integrated development environment (IDE) that you'll use to write and run your programs in this book.

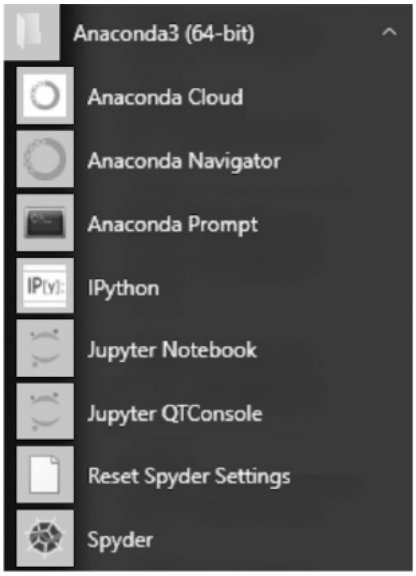
Definition

An *integrated development environment* (IDE) is a complete programming environment that helps make your program writing experience a lot nicer.

Open Spyder

In Windows only, you can open Spyder from the Anaconda folder in the Start menu, shown in figure 3.1.

Figure 3.1. Anaconda folder in the Start menu

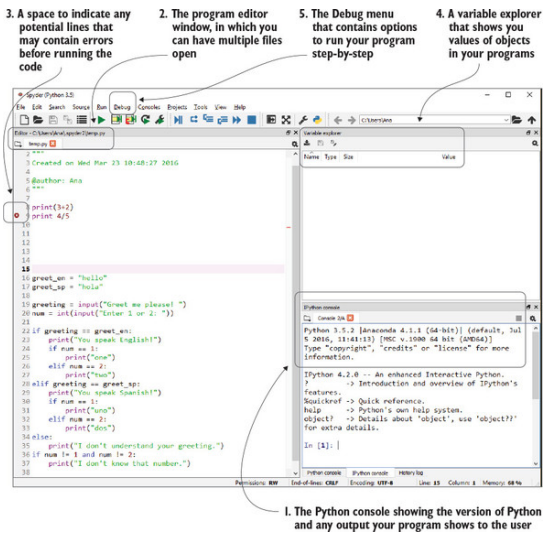


Some of the important features that the Spyder IDE offers, shown in figure 3.2, are as follows:

- An editor to write your Python programs
- A way to see lines of code, before running your program, that may contain potential errors or inefficiencies
- A console to interact with the user of your programs, through input and output
- A way to see values of variables in your program
- A way to step through your code line by line

Figure 3.2 shows the entire Spyder IDE and some code written in the code editor. You don't have to understand the code.

Figure 3.2. The Spyder IDE with the code editor, console, and variable explorer windows



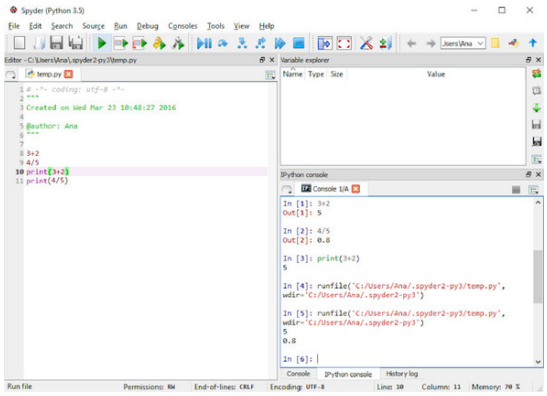
3.2. SETTING UP YOUR WORKSPACE

When you open Spyder, as shown in figure 3.2, you see that the program

- The left pane is the editor, originally containing no code, only a few lines of text. You'll notice that this text is green, meaning that this is a multi-line comment—not code that will be run.
- The top-right pane might contain the object inspector, variable explorer, or file explorer. You won't be using this window pane, but the variable explorer, for example, shows you the values for each variable in your program after the program is finished.
- The bottom-right pane is, by default, the IPython console. In this lesson, you'll see some basics regarding the IPython console and the file editor.

The next two sections will guide you through simple computations in Spyder. You'll see how to enter the computations directly in the console and how to write more-complicated programs in the code editor. At the end of the next two sections, your Spyder session should look like figure 3.3.

Figure 3.3. Spyder session after entering expressions in the IPython console and the code editor



3.2.1. The IPython console

The IPython console is the primary way that you can quickly test commands to see what they do. More important, users will be using the console to interact with your programs. The *I* in *IPython* stands for *interactive*. The IPython console is an advanced console that gives its users neat features including autocompletion, a history of previous commands typed in, and color highlighting of special words.

Writing commands directly into the console

You can write single commands directly in the IPython console to try things and see what they do. If you're just beginning to program, you should be trying things out a lot. That's the best way to start gaining intuition about what statements do and what expressions evaluate to.

Type `3 + 2` in the console and hit Enter to perform this addition. You'll see the result `5` preceded by the text `Out[]`. Now type `4 / 5` to perform this division and you'll see the result `0.8` preceded by the text `Out[]`.

You can think of this console as something that lets you peek into the values of the expressions that you type in. Why do I say *peek*? Because the results of these expressions aren't visible to a user. To make them visible to a user, you must explicitly print their values to the console. Type `print(3 + 2)` in the console. The number `5` is printed again, except that there's no `Out[]` right before it.

Both `3 + 2` and `4 / 5` are called Python *expressions*. In general, anything in Python that can be evaluated to a value is called an expression. You'll see more examples of expressions in lesson 4. In the next section, you'll see how to enter commands in the file editor to write more-complicated programs.

Quick check 3.1

Will the following expressions show output to the user, or are they just letting you peek into their value? Type the expressions in the console to check yourself!

1

```
6 < 7
```

2

```
print(0)
```

3

```
7 * 0 + 4
```

```
print("hello")
```

Primary uses of the console

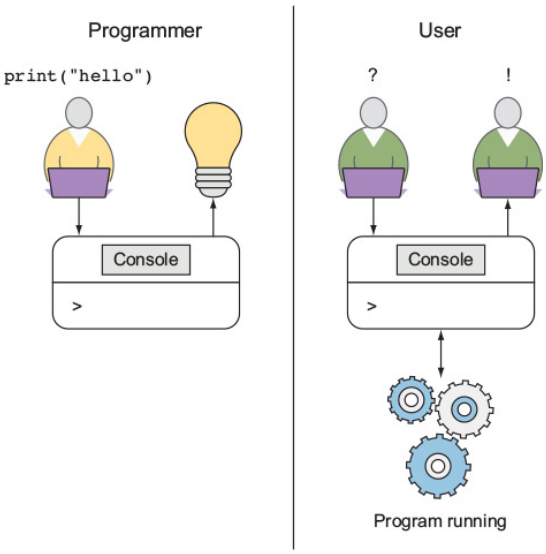
Few programmers can write a perfect program on the first go. Even experienced programmers make mistakes. Your first try to write a program will be a little unsteady, and you'll have *bugs* (errors) that will show up when you try to run your program.

Definition

A *bug* is an error in a program.

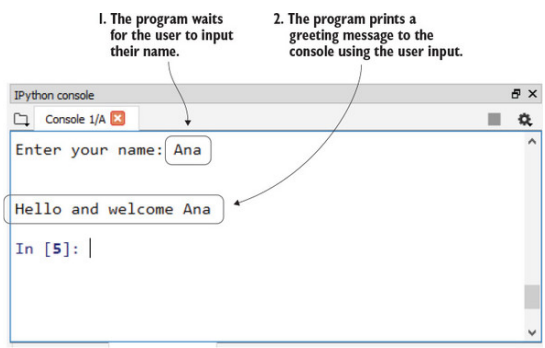
If a program has bugs, big or small, you have to try to fix them. You can learn a lot from the debugging process. As you start to write more-complicated programs, you can think of using the console from the point of view of two roles: you as a programmer, and as a person interacting with your program (the user). Figure 3.4 shows the dual role the console lets you play. A programmer primarily uses the console to test commands and debug programs. A user uses the console to interact with a program that's running by typing in input and seeing what the program outputs.

Figure 3.4. Programmers use the console for their own testing and debugging. They type commands directly in the console and look at the output. Users interact with a program via the console. They type input to a program and view the output from a program in the console.



The majority of the programs you'll see in this book don't have a visual interface. Instead, you'll write programs that interact with users via text in the console; users will be given the opportunity to enter text/numbers/symbols when prompted in the console, and your program will display results in the console. Figure 3.5 shows an example of how the user may interact with the programs you write.

Figure 3.5. An example of a user interacting with a program



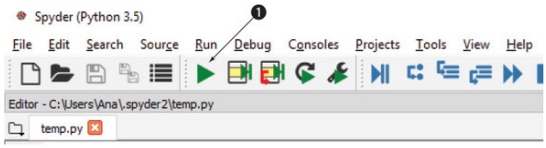
As a programmer, you'll be using the console to take on the role of a user of your program. This is most useful when debugging programs (when you're trying to figure out why your program isn't working as expected). When you use the file editor to write more-complicated programs, it's often useful to have the console print values of any computations or objects in your programs, not just the final value. Doing so can help you determine intermediary values in your programs and help you debug. If running your program is like trying out a recipe, printing intermediary values is like tasting items in

The console is useful for trying out single expressions and seeing their values. You can retype expressions in the console if you want to run them again, or you can use the up arrow to see expressions you previously typed and hit Enter to run them again. A file editor saves your expressions to a file so you don't need to retype them. This saves a lot of time when you want to write programs that are longer than one line.

3.2.2. The file editor

When you write more-complicated Python programs containing more than just a couple of lines, you should use the file editor pane. Here, you can type the commands (in programming, called *statements*), one on each line, as in figure 3.3. After you finish writing a set of commands, you can run the program by clicking the green arrow in the toolbar at the top of Spyder, shown in figure 3.6. Editing and running files is the same for all operating systems that Anaconda supports: PC, Mac, and Linux. This book shows screenshots from a Windows operating system.

Figure 3.6. Click the green arrow button to run the program.



Not all lines of code produce output visible to the user

In the empty file, type `3 + 2` on line 8, as shown previously in figure 3.3. On the next line, type `4 / 5`. Don't type anything else yet. Now click the green arrow to run the program. The first time you click the arrow, you may get a pop-up that asks you for the working directory; it's OK to accept the default values. What happens? Your console at the bottom right shows some red text, similar to the following:

```
runfile('C:/Users/Ana/.spyder2-py3/temp.py',
➥wdir='C:/Users/Ana/.spyder2-py3')
```

That line indicates that your program ran, but nothing was shown to the user.

Now make the following additions. On line 10, type `print(3 + 2)`. And on the following line, type `print(4 / 5)`. Run the program again. Now what happens? You should see the same thing as in figure 3.3. The console shows the results of the calculations to the user, each on a different line.

How does that work? The Python interpreter executes each line in the file. It first runs the statement `3 + 2` and internally calculates the result of this expression. Then it internally calculates `4 / 5`. Because these two statements don't tell Python to show the output of the calculations, their values don't show up on the console.

A keyword in Python, `print`, is reserved for when you want to output the value of whatever is in the parentheses following `print` to the console. In this case, you show the result of evaluating the expressions `3 + 2` and `4 / 5`, in that order.

Quick check 3.2

Which of these expressions will the user see on the console? Type the expressions in the file editor and click Run to check!

1

```
print(4 - 4 * 4)
```

2

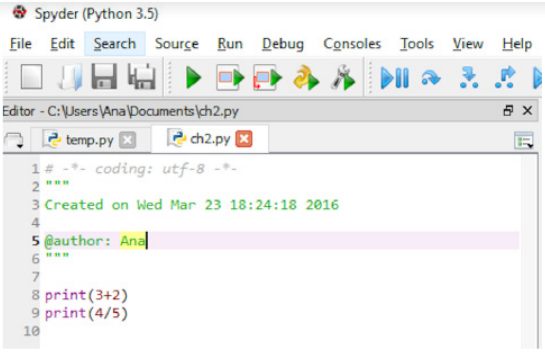
```
print(19)
```

3

```
19 - 10
```

Saving files

You should save every program you write in a separate file to keep you organized. The file in which you wrote the previous code is a temporary file, saved in some location in the Anaconda installation folder. Open a new file from the Spyder menu bar, as shown in figure 3.7. Type the previous two `print` statements again in the new file.



Tip

I strongly encourage you to type the commands again instead of copying and pasting. Repetition is a great way to help you get the hang of programming. Forcing yourself, especially at the beginning of your programming career, to type commands will help speed up your learning process and make writing code second nature.

Now save the file in a directory of your choosing. You must save it with a .py extension. If you don't save it with this extension, you won't be able to run the program (the green Run button will be gray). After you save the file, click the green Run button. The same output as before should show up in the console.

If you close the file you just saved, your program isn't lost. You can reopen the file from the File menu. All the code is still there, and you can run the program as if you just wrote it.

SUMMARY

In this lesson, my objective was to teach you

- How to install a Python distribution called Anaconda, using Python version 3.5 and an IDE called Spyder
- How to open a new file, write a simple program in the file, save the file, and run a program
- How to write code in the file editor and open many files in the editor pane
- That the console allows you to peek into values of variables or to show output to the user
- How to use `print` statements to print expression values to the console

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
Unit 1. Variables, types, expressions, and statements

NEXT
Lesson 4. Variables and expressions: giving names and value…