

TRAINING & REFERENCE

murach's Python programming

2ND EDITION

(Chapter 1)

Thanks for downloading this chapter from [Murach's Python Programming \(2nd Edition\)](#). We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its “how-to” headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our [website](#). From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books on related topics.

Thanks for your interest in our books!



MIKE MURACH & ASSOCIATES, INC.

1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963

murachbooks@murach.com • www.murach.com

Copyright © 2021 Mike Murach & Associates. All rights reserved.

What developers have said about previous editions

“This is by far the best Python tutorial I have come across. Tried Udemy, Udacity, some other video and printed tutorials, but didn’t get the feel of it. Murach’s side-by-side lecture & example really works for me.”

Posted at an online bookseller

“This is not only a book to learn Python but also a guide to refer to whenever I face a Python problem. What I really like is that it is very well-organized and easy to follow, and it does not make you bored by providing unnecessary details.”

Posted at an online bookseller

“The material in the book itself is worth 6 stars [out of 5]. It’s second to none in its quality.”

Posted at an online bookseller

“This is now my third text book for Python, and it is the ONLY one that has made me feel comfortable solving problems and reading code. The paired-pages approach is fantastic, and it makes learning the syntax, rules, and conventions understandable for me.”

Posted at an online bookseller

“[Look at the contents] and see where they put testing/debug: Chapter 5. I love the placement! Folks going through the book get Python installed, receive positive returns with some basic code, and then take a breather with testing/debugging before more complex things come their way.”

Jeremy Johnson, DreamInCode.net

“*Murach’s Rocks*: Murach’s usual excellent coverage with examples that are actually practical.”

Posted at an online bookseller

Section 1

Essential concepts and skills

The eight chapters in this section get you off to a fast start by presenting a complete subset of the essential concepts and skills that you need for Python programming. First, chapter 1 introduces you to Python and shows you how to use an integrated development environment (IDE) called IDLE to develop and run programs. Then, chapters 2, 3, and 4 present skills that let you develop substantial programs of your own.

At that point, you're going to need to improve your testing and debugging skills, so that's what chapter 5 shows you how to do. Then, chapters 6, 7, and 8 continue your development. When you complete this section, you'll be able to design, code, test, and debug Python programs that can work with data that's stored in files.

1

An introduction to Python programming

This chapter starts by showing why Python is considered by many to be the best language for teaching beginners how to program. Next, this chapter presents the concepts and terms that you need to know before you start programming. Then, it shows how to use an integrated development environment (IDE) called IDLE to develop and test Python programs.

Introduction to Python.....	4
Why Python works so well as your first programming language.....	4
Three types of Python applications.....	6
The source code for a console application.....	8
How Python compiles and runs source code	10
How disk storage and main memory work together.....	12
How to use IDLE to develop programs	14
How to use the interactive shell.....	14
How to work with source files	16
How to compile and run a program.....	18
How to fix syntax and runtime errors.....	20
Perspective	22

Introduction to Python

The *Python* programming language was developed in the 1990s by Guido van Rossum of the Netherlands. It was intended to be a simple, intuitive language that is as powerful as traditional languages, and it succeeded at that. Guido named the language Python because he was a big fan of “Monty Python’s Flying Circus,” and he continues to be involved in the development of Python. In fact, some developers in the Python community refer to him as the “Benevolent Dictator for Life (BDFL)”, although he officially stepped down from that role in 2018.

Why Python works so well as your first programming language

Figure 1-1 summarizes the case for learning Python as your first programming language. Here, you can see a short excerpt from a Java program and an excerpt from a Python program that does the same thing. This shows how much closer to plain English Python is and how much simpler the syntax for Python is. In fact, you’ll often be able to write a Python program with far less code than the same program would take in C++, Java, C#, or other traditional programming languages. And yet, you can learn most of the concepts and skills that you need for those languages when you learn Python, without getting slowed down by trivial coding details.

Beyond that, Python supports a wide range of applications. It is used by many successful companies. And it is *open source*, which means that its source code is available to the entire Python community.

In short, Python is a powerful programming language that lets you develop programs in less time and with less code than other programming languages. That’s why it’s a great language for learning how to program. But it’s also a great language if you already know how to use another language or two.

In this figure, the Python timeline shows that Python 2 became available in the year 2000. This was replaced by Python 3 in 2008. However, Python 3 isn’t *backward compatible*. That means that Python 3 can’t run most programs written for Python 2. Conversely, Python 2 isn’t *forward compatible*. That means that it can’t run most programs written for Python 3.

In this book, you’ll learn how to write programs for Python 3, and you’ll learn how to use Python 3 to run those programs. As a result, compatibility won’t be a problem. That’s a great way to learn Python.

Four general-purpose programming languages

C++
Java
C#
Python

The Python timeline

Year	Month	Release	Description
2000	October	2.0	First release of Python 2
2008	December	3.0	A redesign of Python that isn't backward compatible
2010	July	2.7	Last release of Python 2 with support until 2020
2020	October	3.9	The latest release at this writing

Syntax differences between Python and Java

Some Java code

```
private static double calculateFutureValue(
    double monthlyInvestment, double monthlyRate, int months)
{
    double futureValue = 0.0;
    for (int i = 1; i <= months; i++) {
        futureValue =
            (futureValue + monthlyInvestment) * (1 + monthlyRate);
    }
    return futureValue;
}
```

Python code that works the same

```
def calculateFutureValue(monthlyInvestment, monthlyRate, months):
    futureValue = 0.0
    for i in range(months):
        futureValue =
            (futureValue + monthlyInvestment) * (1 + monthlyRate)
    return futureValue
```

Why Python is a great first language

- Python has a simple syntax that's easier to read and use than most other languages.
- Python has most of the features of traditional programming languages. As a result, you can use Python to learn the concepts and skills that apply to those languages too.
- Python supports the development of a wide range of programs, including games, web applications, and system administration.
- Python is used by many successful companies, including Google, IBM, Disney, and EA Games. As a result, knowing Python is a valuable skill.
- Python is *open source*. There are many advantages to being open source.

Figure 1-1 Why Python works so well as your first programming language

Three types of Python applications

An *application*, or *app*, is computer software that performs a task or related set of tasks. However, applications can also be referred to as *programs*, even though one application may actually consist of many related programs. In practice, most people use these terms interchangeably. In this book, we use the term *program* to refer to the short applications that it presents.

In any event, figure 1-2 shows three types of applications that you can create with Python. The first type is a *console application*. In this type of application, you enter commands at the *command prompt* in the *console* that's available from your operating system.

In this case, a console application is being run on a Windows system. As a result, it uses the Command Prompt window that's available from Windows. On a macOS system, though, a console application runs in the Terminal window. Because console applications are the easiest type of application to develop, you'll work with console applications in the first three sections of this book.

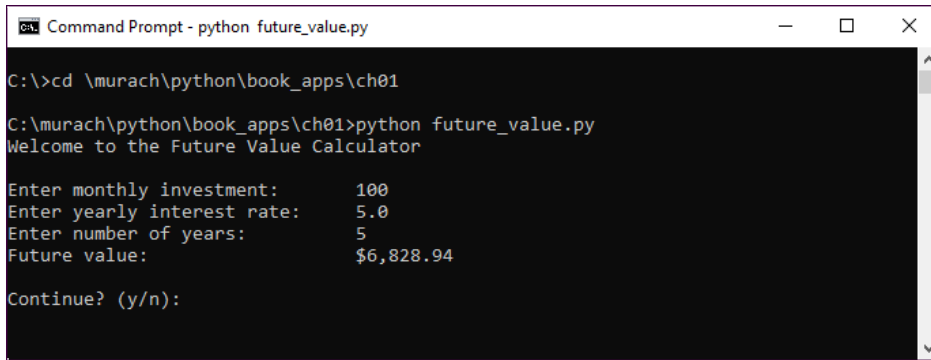
The second type of Python application is a *GUI application*. That's an application that has a *graphical user interface (GUI)*. In this figure, the GUI application performs the same tasks as the console application. In other words, it gets the same input from the user, performs the same calculation, and displays the same result. However, the GUI application is more user-friendly and intuitive. You'll learn how to develop this type of application in the last section of this book.

The third type of application shown in this figure is a *web application*. Unlike a *desktop application*, which runs directly on your computer, a web application can be called by a web browser that's running on a computer or mobile device and can use a server to process or store data. In this figure, the web application performs the same task as the console and GUI applications. Although this book doesn't show how to develop web applications, you should know that Python frameworks are available to help you develop that type of application with Python.

You should also know that you can use Python as a *scripting language* to work with other software applications or to develop system administration programs that perform tasks like automatically backing up a server or rotating log files. Because programs like this usually run in the background, they don't always have a user interface. When you finish this book, though, you will have the skills you need to write this type of program.

In addition, Python is a popular language for data analysis and visualization. *Data analysis* is the process of cleaning, preparing, analyzing, and modeling data to better understand and use it to make decisions. *Data visualization* involves creating a graphic representation of data by using visual elements like charts, graphs, and maps. This often makes the data easier to interpret. Python is popular in these areas partly due to the existence of many open-source Python libraries. For example, Pandas is commonly used for Python data analysis. Similarly, Matplotlib and Seaborn are commonly used for data visualization. To get started with these subjects, you can check out our book titled *Murach's Python Data Analysis*. It should be available in the second half of 2021.

A console application



```
Command Prompt - python future_value.py

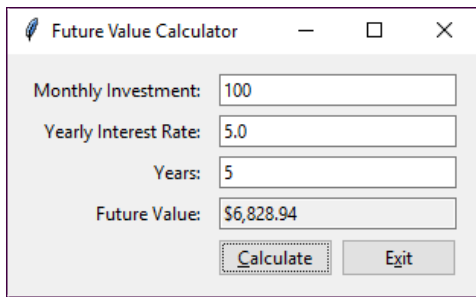
C:\>cd \murach\python\book_apps\ch01

C:\murach\python\book_apps\ch01>python future_value.py
Welcome to the Future Value Calculator

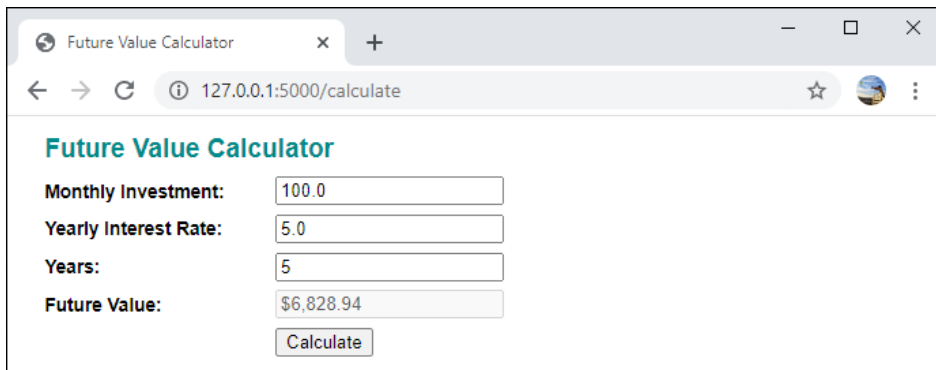
Enter monthly investment:      100
Enter yearly interest rate:    5.0
Enter number of years:        5
Future value:                  $6,828.94

Continue? (y/n):
```

A GUI application



A web application



Description

- A *console application* is a desktop application that uses the *console* to interact with the user.
- A *GUI application* is an application that uses a *graphical user interface (GUI)* to interact with the user.
- A *web application* typically gets requests from a web browser, processes them on a web server, and returns the responses to the web browser.

Figure 1-2 Three types of Python applications

The source code for a console application

To show you what a Python program looks like, figure 1-3 shows the source code for a console application. This application gets three values from the user: a monthly investment amount, a yearly interest rate, and a number of years. Then, the program takes these entries and calculates the future value of the monthly investments with the interest calculated each month.

For now, just note that the code begins with a line that starts with a hash (#) and a bang (!). This line is commonly called the *shebang line*. It is ignored by Windows, but it's used by Unix-like operating systems, including macOS, to specify the interpreter for the program. In this case, the shebang specifies that this program should be run using Python 3. As a result, you can use this shebang line for all programs in this book.

In the next chapter, you'll start learning how to write the code in this program. Then, by the time you complete chapter 3, you'll know how most of the code in this program works, and you'll be able to write comparable programs of your own.

The source code for a console application

```
#!/usr/bin/env python3

import locale

# set the locale for use in currency formatting
locale.setlocale(locale.LC_ALL, 'en_US')

# display a welcome message
print("Welcome to the Future Value Calculator")
print()

choice = "y"
while choice.lower() == "y":

    # get input from the user
    monthly_investment = float(input("Enter monthly investment:\t"))
    yearly_interest_rate = float(input("Enter yearly interest rate:\t"))
    years = int(input("Enter number of years:\t\t"))

    # convert yearly values to monthly values
    monthly_interest_rate = yearly_interest_rate / 12 / 100
    months = years * 12

    # calculate the future value
    future_value = 0
    for i in range(months):
        future_value = future_value + monthly_investment
        monthly_interest_amount = future_value * monthly_interest_rate
        future_value = future_value + monthly_interest_amount

    # format and display the result
    print("Future value:\t\t\t" + locale.currency(
        future_value, grouping=True))
    print()

    # see if the user wants to continue
    choice = input("Continue? (y/n): ")
    print()

print("Bye!")
```

Discussion

- This application gets three values from the user: monthly investment amount, yearly interest rate, and number of years. Then, it calculates and displays the future value of the investments at the specified interest rate for the specified number of years.
- The first line in this program is called the *shebang line*, or *shebang*. It specifies the interpreter that should be used to run this program. In this book, all programs should be run by Python 3.

How Python compiles and runs source code

The code in a program like the one in the last figure can be referred to as Python *source code*. To create the file that contains this source code, the programmer uses a text editor or integrated development environment (IDE), which you'll learn more about in a moment.

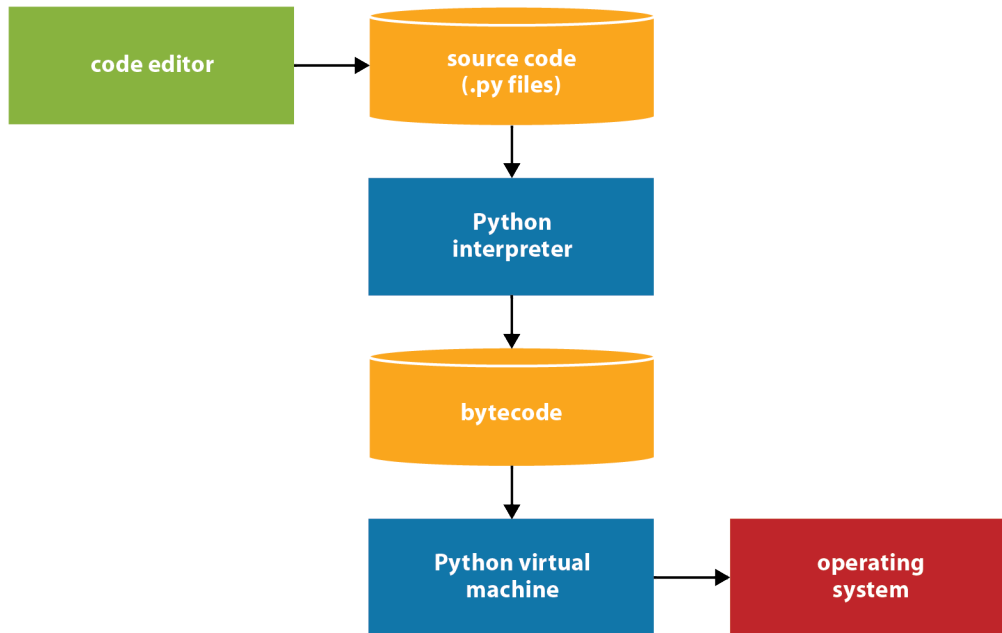
However, before the Python source code can be run (or executed) on a computer, the code must be translated into code that the computer can understand, as shown in figure 1-4. Here, you can see that the files that contain the source code have the .py extension. Then, the *Python interpreter* is used to translate (or *compile*) the source code into *bytecode*.

This bytecode can be run by any computer that has the *Python virtual machine* installed on it. This virtual machine translates the bytecode so it can be run by the operating system of the computer.

Since the Python virtual machine is part of the Python interpreter, it is available on all platforms that support Python. In fact, that's why Python is said to be *platform-independent*. In other words, it is the virtual machine that makes it possible for Python to run on a wide variety of operating systems.

Although this compilation process is complicated, it's done automatically with the push of a single key when you're using an IDE to develop your Python programs as shown later in this chapter. For now, all you need to take away from this figure is that Python source code is compiled into bytecode that's run by the computer's operating system.

How Python compiles and runs source code



Procedure

- Step 1** The programmer uses a *text editor* or *IDE* to enter and edit the *source code*. Then, the programmer saves the source code to a file with a `.py` extension.
- Step 2** The source code is *compiled* by the *Python interpreter* into *bytecode*.
- Step 3** The bytecode is translated by the *Python virtual machine* into instructions that can interact with the operating system of the computer.

Description

- Although this procedure is complicated, it runs behind the scenes when you're developing Python programs. As a result, it's mostly invisible to the developer.
- Python bytecode can be run on any operating system that has a Python virtual machine. That's why Python is said to be *platform-independent*.
- Most programs import one or more modules, which are files that contain reusable code. The Python interpreter saves the compiled bytecode for those modules in files that have `.pyc` or `.pyo` extensions. This is an import optimization that can improve the startup time for a program.

Figure 1-4 How Python compiles and runs source code

How disk storage and main memory work together

Now that you know how a program is compiled and run, it's worth taking a minute to go over how the main memory of a computer and the disk storage of a computer work together as an application runs. This conceptual background can help you understand what needs to be done when you develop a program that works with data in disk storage. In addition, figure 1-5 presents the terms that you need to be familiar with.

To start, you should know that a computer consists of a *Central Processing Unit (CPU)* and *main memory*, which is often referred to as *RAM (Random Access Memory)*. When an application runs, its bytecode is stored in main memory and its instructions are executed by the CPU. In addition, the data that the application is currently using is stored in main memory. This type of storage is temporary, however, so the data is lost when the application ends.

To store the data of an application, *disk storage* is commonly used. This type of storage remains after the application ends, so it is called *persistent data storage*. In chapter 7, you'll learn how to work with persistent data that's stored in a file. Then, in chapter 17, you'll learn how to work with persistent data that's stored in a database.

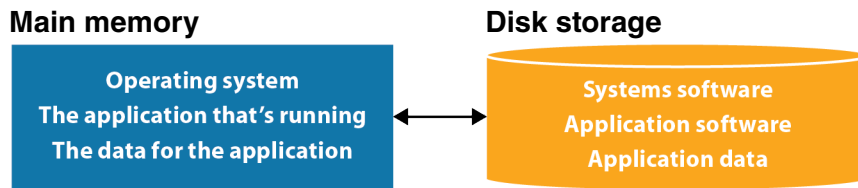
In general, two types of software are stored on disk. *Systems software* consists of the programs that control the operation of the system. That includes the *operating system* for the computer, like Windows for a PC or macOS for a Mac. By contrast, *application software* consists of the applications that are available on the system. That includes applications such as web browsers, word processors, and spreadsheet programs.

When a computer starts, it loads the operating system from disk storage into main memory. Then, when you start an application, the operating system loads the application from disk storage into main memory, and it runs the application. If the application requires persistent data while it is running, it reads the data from disk storage into main memory and then processes it. If the application updates the data or creates new data, it writes the data from main memory back to disk storage.

The storage capacity of main memory and disk storage is measured in *bytes*. For instance, main memory used to be measured in *megabytes (MB)*, which is millions of bytes, but now it is usually measured in *gigabytes (GB)*, which is billions of bytes. For the record, a *byte* consists of eight *binary digits*, or *bits*, and each byte is roughly equivalent to one character of data, like an A or a B.

If you're comfortable with these concepts and terms, you're ready to learn how to use an IDE to create, test, and debug Python programs as shown in the next few figures. Otherwise, it's worth taking a few minutes to review the terms in this figure because they're terms that every programmer should know.

Main memory and disk storage as an application runs



How disk storage and main memory work together

- When you start the computer, it loads the operating system into main memory. Then, you use the features of the operating system to start an application.
- When you start an application, the operating system loads it into main memory. Then, it runs the application.
- As the application runs, it may read data from disk storage into main memory or write data from main memory to disk storage.

Description

- The *systems software* for a computer provides all of the software that is needed for running the applications, including the *operating system*.
- The *applications software* consists of the applications that are available on the system.
- A computer consists of a *CPU* (*Central Processing Unit*) and *main memory*, also known as *RAM* (*Random Access Memory*).
- The data in main memory is lost when an application ends. *Disk storage* is used for *persistent data storage*, which remains after an application ends.
- Main memory and disk storage are commonly measured in *megabytes* (*MB*) and *gigabytes* (*GB*), and a *byte* is roughly equivalent to one character of data.

Figure 1-5 How disk storage and main memory work together

How to use IDLE to develop programs

Although you can use a text editor to develop Python programs, most programmers use an *integrated development environment (IDE)* because an IDE usually helps you get more done in less time. One IDE that's popular for Python programming is PyCharm, which is a free, open-source IDE that runs on most operating systems. Another is Eclipse, which is a general-purpose IDE that has a plugin called pyDev for doing Python development.

However, for this book, we recommend using an IDE named *IDLE*. This IDE is included with the Python distributions for Windows and macOS and can be easily installed on Linux. Besides that, it's easy to use. Note that the name of this IDE is the same as the last name of Eric Idle, a sly reference to one of the founding members of the Monty Python comedy group.

How to use the interactive shell

Figure 1-6 shows how to use IDLE's *interactive shell*. This shell makes it easy to enter and test Python statements without writing full programs. This is another feature that makes Python work so well as a first programming language.

When you start the shell, it displays some version and copyright information followed by its prompt (`>>>`). At this prompt, you can type any Python code that you want to run. Then, when you press the Enter key, Python runs the code, displays the results if there are any, and displays another prompt.

In this figure, the first prompt executes a `print()` function that prints "Hello out there!" to the console. This displays the characters coded within the quotation marks immediately after the prompt. Then, the next three prompts evaluate arithmetic expressions and display the result of each expression immediately after the prompt.

The fifth prompt executes another `print()` function. However, this `print()` function doesn't include quotation marks around the characters, and those quotation marks are required. As a result, the shell displays an error message for a `SyntaxError` that says "invalid syntax".

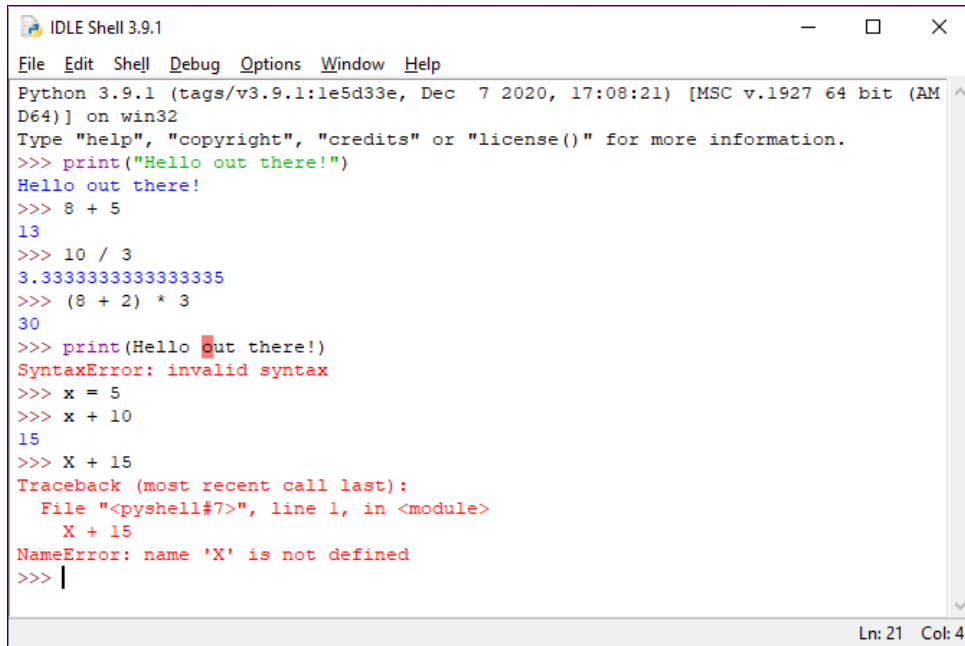
The next two prompts show what happens when you set a variable named `x` equal to 5 and then add 10 to that variable. The result is 15.

The last statement attempts to add 15 to `X`. This displays a `NameError` message that says "name 'X' is not defined". That's because variable names are case sensitive, so `x` and `X` refer to two different variables. The variable named `x` has been defined earlier in this session, but the variable named `X` has not been defined yet in this session.

This interactive session shows how the Python shell can help you build your coding skills. That's why this book encourages you to use it whenever you need to experiment with Python code. In fact, one of the exercises at the end of this chapter walks you through your first use of this shell.

Incidentally, when you see a notation like `File→Close`, it means to pull down the File menu and select the Close command. Similarly, `Run→Python Shell` means to pull down the Run menu and select the Python Shell command. You'll see this notation throughout the book.

IDLE's interactive shell



```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello out there!")
Hello out there!
>>> 8 + 5
13
>>> 10 / 3
3.3333333333333335
>>> (8 + 2) * 3
30
>>> print(Hello out there!)
SyntaxError: invalid syntax
>>> x = 5
>>> x + 10
15
>>> X + 15
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    X + 15
NameError: name 'X' is not defined
>>> |
  
```

How to open, close, and restart the interactive shell

- To start IDLE, use the features of your operating system. This opens an interactive shell.
- To close the interactive shell, click on its close button or select File→Close.
- To restart an interactive shell, select Run→Python Shell from another IDLE window.

How to use the interactive shell

- Enter Python code after the >>> prompt. Then, press Enter.
- If you enter valid code that produces a result, the shell displays the results.
- If you enter invalid code, the shell displays an error message.

Description

- The *interactive shell* makes it easy to experiment with Python code and view the results right away. This is another feature that makes Python a good first language for beginning programmers.

How to work with source files

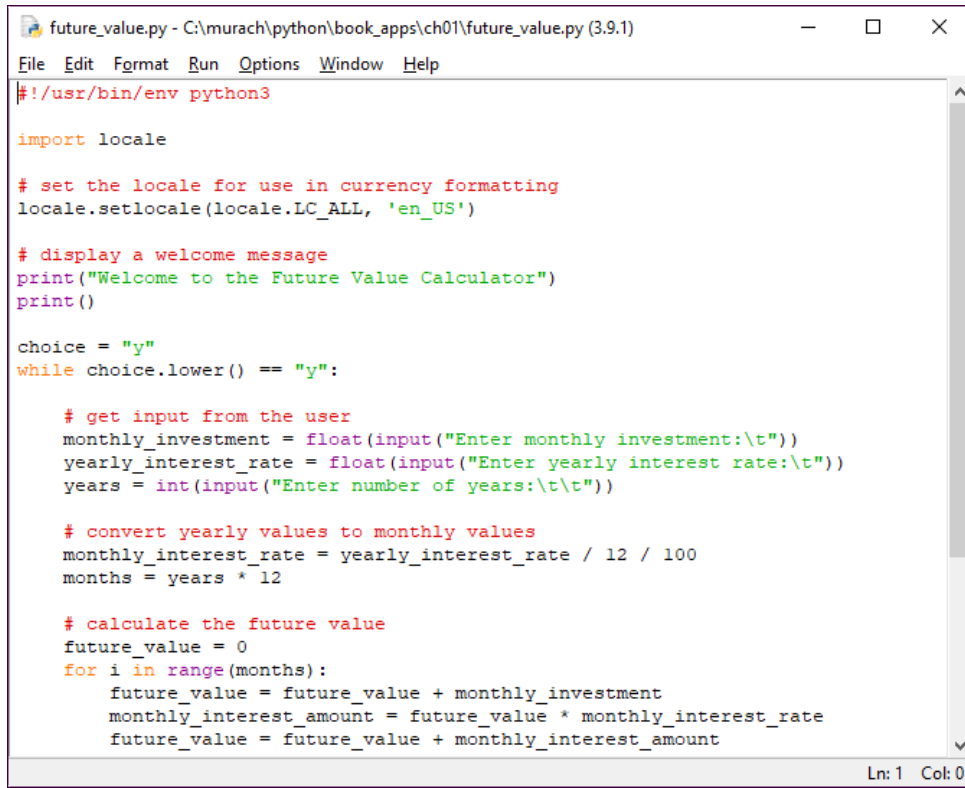
Figure 1-7 shows IDLE's editor window. In general, this editor works like a simple word processor. So, to edit the source code for a program, you can use many of the same techniques that you use with your word processor.

On a Windows system, for example, you can use Ctrl+X or Ctrl+C to cut or copy the selected characters, Ctrl+V to paste the characters in a new location, and Ctrl+Z to undo the last change that you made. You can also use Ctrl+Del or Ctrl+Backspace to delete an entire word at a time. And you can right-click on a selection to get a context menu with commands that are appropriate for the selected text.

You can also use the drop-down menus to find the commands that you may want to use. For instance, you can use the commands in the File menu to create, open, save, and close source files. You can use the Window menu to switch between open files. And you can use the Edit and Format menus to edit and format a selection.

If you want to make a copy of a source file that you're working on, you can use the Save As command. This command is useful when you want to experiment with a variation of the same program. It's also useful if you want to start a new program from an existing program. Then, you can delete the parts of the program that you don't want, keep the parts that you do want, and get your program off to a fast start.

IDLE's editor with a source file displayed



```
future_value.py - C:\murach\python\book_apps\ch01\future_value.py (3.9.1)
File Edit Format Run Options Window Help
#!/usr/bin/env python3

import locale

# set the locale for use in currency formatting
locale.setlocale(locale.LC_ALL, 'en_US')

# display a welcome message
print("Welcome to the Future Value Calculator")
print()

choice = "y"
while choice.lower() == "y":

    # get input from the user
    monthly_investment = float(input("Enter monthly investment:\t"))
    yearly_interest_rate = float(input("Enter yearly interest rate:\t"))
    years = int(input("Enter number of years:\t\t"))

    # convert yearly values to monthly values
    monthly_interest_rate = yearly_interest_rate / 12 / 100
    months = years * 12

    # calculate the future value
    future_value = 0
    for i in range(months):
        future_value = future_value + monthly_investment
        monthly_interest_amount = future_value * monthly_interest_rate
        future_value = future_value + monthly_interest_amount

Ln: 1 Col: 0
```

How to create, open, save, and close source files

- Use the File menu and common techniques for your operating system.

How to switch between a source file window and its shell window

- Use the Window menu. Or, arrange the windows on your screen so both are visible, and then click on the window you want to use.

How to enter and edit Python code

- Use the Edit and Format menus, common editing keystrokes, and context menus.

Description

- Like other programs that you've used, IDLE provides menus for working with files, editing and formatting selected text, and switching from one window to another.

How to compile and run a program

To compile and run a Python program when you're using IDLE, you can just press the F5 key. Then, Python compiles the source code into bytecode and runs it. It's that simple. This is summarized in figure 1-8, and you can compare that to the procedure in figure 1-4, which is done automatically by IDLE.

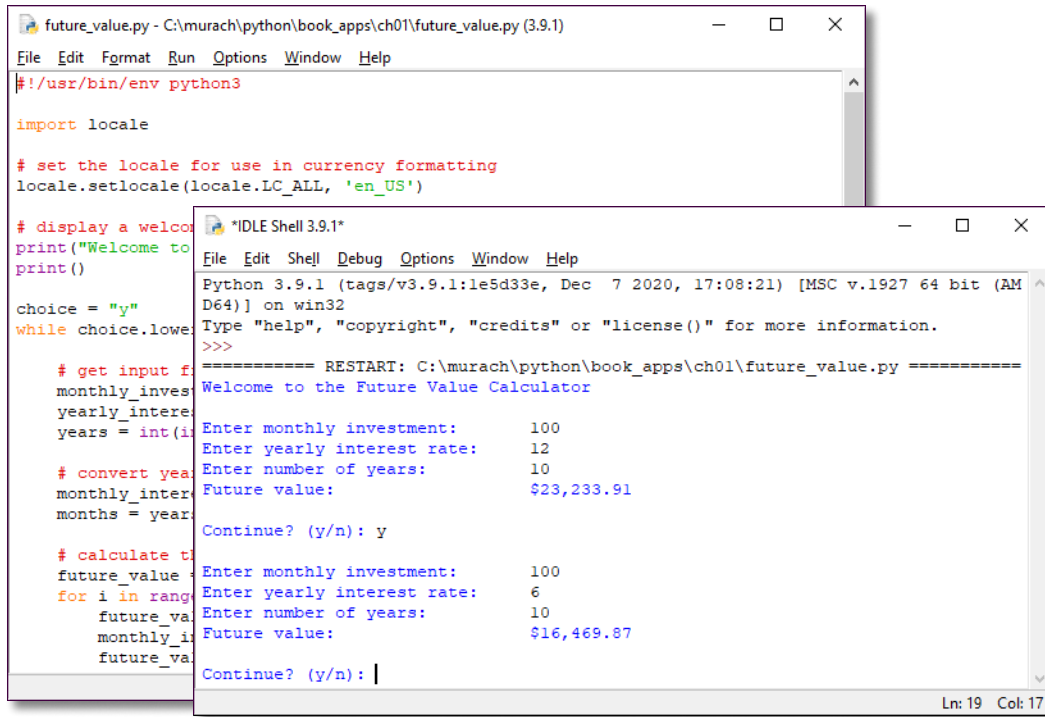
Note that on some laptop keyboards, you need to hold down the Fn key to access function keys like the F5 key. If that's not what you want, you can change the system preferences for your keyboard so it isn't necessary to use the Fn key.

If you're compiling and running a program that works with the console, IDLE uses the interactive shell as the console. This makes it easy to test the program and to switch between the shell and the editor as you do that.

This assumes that the source code doesn't have any errors. However, it's common for source code to have errors. In that case, you need to fix the errors before Python can compile and can run the program. The next figure shows how to do that.

Incidentally, if you're developing a professional program that uses the console, you have to do the final testing by using the real console of a system. However, you don't need to do that when you're learning Python with IDLE. Instead, you can use the IDLE shell as the console for all of your programs.

A console application that's being run in the shell



How to compile and run a Python program

- From the editor window, press the F5 key or select Run→Run Module.
- If IDLE displays a dialog box that indicates that you must save the program first, press the Enter key or click OK to save it. Then, if the program doesn't have any errors, IDLE runs the program in the interactive shell.

Description

- When you run a program, Python attempts to compile it, as shown in figure 1-4. If it doesn't detect any errors, it executes the code for the program.
- However, if Python detects any errors, it usually alerts you as shown in the next figure, and it doesn't execute the code.
- When you use IDLE to run a console application, it uses the shell as the console.
- On most platforms, IDLE's shell automatically becomes active when you run a console application. If it doesn't, you have to switch to the shell after you run the application.

A note about function keys (F1, F2, etc.)

- On some laptop keyboards, you need to hold down the Fn key to access the function keys. So, you must press Fn+F5 to access the F5 key. However, you can change this behavior by editing the system preferences for your keyboard.

Figure 1-8 How to compile and run a Python program

How to fix syntax and runtime errors

After you write the code for a program, you need to *test* the program. When you do that, your goal is to find all of the errors (or *bugs*) in the program. Whenever you find a bug, you need to *debug* it by fixing it. Then, you need to test the program again, fix the next bug that's found, and continue this process until you finish debugging the entire program.

When you're ready to test a program and you try to compile and run it, two types of errors are likely to be detected. The first type is a *syntax error*. These are errors that occur because the Python coding rules have been violated. As a result, the program can't be compiled.

The first example in figure 1-9 shows the dialog box that's displayed when a syntax error is detected. In addition, the point at which the error was detected is highlighted in the source code. Keep in mind, though, that the error may not be exactly at that point. In fact, it is likely to be in the previous statement. In this example, the error occurs because the statement before the highlighted statement should end with another right parenthesis.

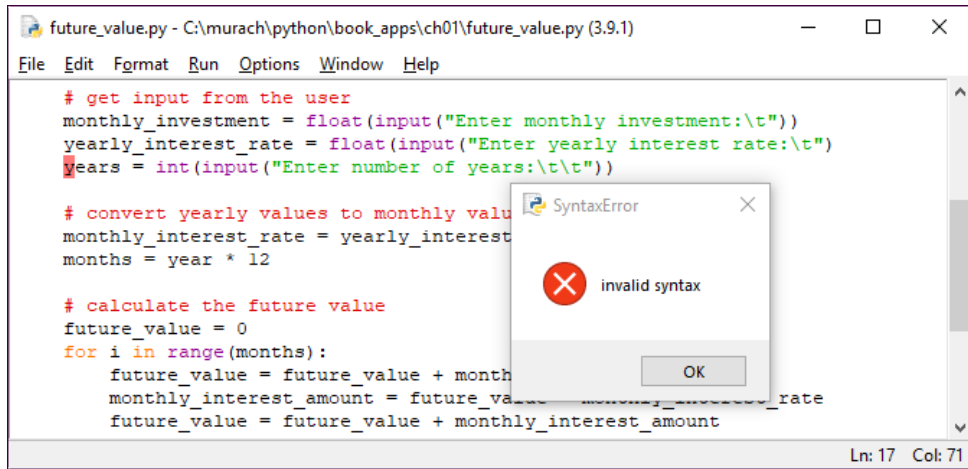
The other type of error is a *runtime error*. This type occurs after all the code has compiled cleanly and the program is being executed. Then, an error message is displayed that helps you identify the cause of the error. This type of error is also known as an *exception*.

When an exception occurs, the program ends, or "crashes," unless the exceptions are handled by the program. In chapter 8, you'll learn how to *handle* the *exceptions* that occur so your programs won't crash. However, you can often prevent exceptions from occurring in the first place by fixing the code that caused the exception.

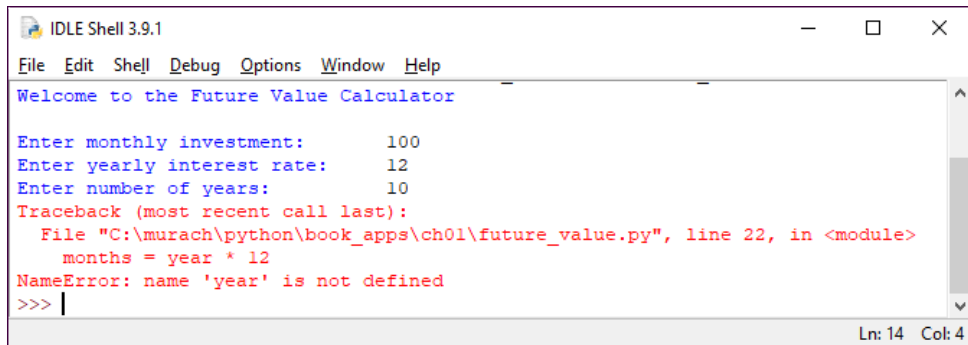
The second example in this figure shows a message that's displayed for an exception. Here, the last line of the message says that a `NameError` has occurred because "year" is not defined. If you look at the first example, you can see that this variable is coded as "years", not "year", which is why the exception occurred. To fix it, you just add the "s" to "year". Although this may not make sense to you right now, it will as soon as you start developing programs of your own.

When you fix the errors for a program, you'll note that IDLE catches just one error each time. As a result, you need to correct an error, try to run the program again, fix the next error, and continue until all of the syntax errors have been fixed. After that, you need to do the same for the runtime errors.

A dialog box for a syntax error



A message that's displayed for a runtime error



Description

- After you write the code for a Python program, you *test* the program to make sure it works. As part of that process, you will usually find errors (*bugs*) so you will have to *debug* the program.
- A *syntax error* violates one of the rules of Python coding so the source code can't be compiled. By contrast, a *runtime error* occurs when a Python statement can't be executed as the program is running.
- When you try to run a program, IDLE detects one syntax error at a time. For each error, a dialog box is displayed and the point at which the error was detected is highlighted. Then, you close the dialog box, fix the error, and try again.
- When all of the errors have been fixed, the program will run. But here again, only one runtime error is detected at a time. So you have to fix the error, run the program again, and keep going until all of runtime errors have been debugged.
- A runtime error is also known as an *exception*, and how to handle exceptions is one of the critical skills that you'll learn in chapter 8 of this book.

Figure 1-9 How to fix syntax and runtime errors

Perspective

Now that you've finished this chapter, you should understand the concepts and terms that have been presented. You should also be familiar with the way IDLE works. At this point, you're ready to start learning how to program. And that's what you're going to do in the next chapter.

Terms

Python
open source
backward compatible
forward compatible
application
app
program
console application
console
command prompt
GUI application
web application
desktop application
scripting language
shebang line
source code
compile
interpreter
bytecode
virtual machine
platform-independent
module

CPU (Central Processing Unit)
main memory
RAM (Random Access Memory)
disk storage
persistent data storage
systems software
operating system
application software
byte
bit
megabyte (MB)
gigabyte (GB)
integrated development
 environment (IDE)
IDLE
interactive shell
testing
debugging
syntax error
runtime error
exception
exception handling

Summary

- *Python* is a powerful programming language with a simple syntax that's easy to read and understand. That's why it's such a good first language for beginners.
- Python can be used to develop *desktop applications* such as *console applications* and *GUI applications*. Python can also be used to develop *web applications*. And Python can be used as a *scripting language* for purposes such as system administration.
- Before the *source code* for a Python program can be run, it is *compiled* by the Python *interpreter* into *bytecode* that can be run by the Python *virtual machine* for a computer. The bytecode and virtual machine are what makes Python *platform-independent*.
- When a program is being run, the *operating system*, the program, and the program's data are all stored in the *main memory* of the computer.
- *Disk storage* provides *persistent data storage* for the data that's operated upon by a program. Disk storage also stores all of the *systems software* and *applications software* of the system.
- An *integrated development environment (IDE)* like *IDLE* makes it easier to develop Python programs.
- You can use IDLE's *interactive shell* to enter and test Python code such as expressions, functions, and statements. You use its editor to enter and edit Python source code.
- To run a program with IDLE, you can press the F5 key. If the program uses the console, the IDLE shell is used as the console.
- The goal of *testing* is to find all of the errors in a program. The goal of *debugging* is to fix all the errors.
- If a program has *syntax errors*, you need to correct them before the program can be compiled. If a program has *runtime errors*, you need to debug each one until the program runs correctly.
- A runtime error is also known as an *exception*. To make sure that a program doesn't crash, you must learn how to *handle* the *exceptions* that occur.

Before you do the exercises for this chapter

Before you do any of the exercises in this book, you need to install Python. In addition, you need to install the source code for this book from our website (www.murach.com). For details, see the appendix for your operating system.

Exercise 1-1 Experiment with IDLE's interactive shell

This exercise guides you through experimenting with Python's interactive shell.

1. Start IDLE. That should display a window for the interactive shell.
2. Enter the following `print()` functions and arithmetic expressions into the interactive console:

```
>>> print("The meaning of life")
The meaning of life
>>> print()

>>> 30 + 12
42
>>> 30-12
18
>>> 3 * 4
12
>>> 12/3
4.0
>>> 1 / 3
0.3333333333333333
>>> 3 * 4 + 30
42
>>> 3 * (4 + 30)
102
```

Note that (1) the `print()` function prints a blank line if you don't code anything between the parentheses, (2) the `*` is used for multiplication in an arithmetic expression, (3) the spaces before and after arithmetic operators aren't required, and (4) parentheses are used just as they are in algebraic expressions. If you make entry errors, the shell should display appropriate error messages.

3. Try entering these statements to see how they work:

```
var1 = 30
var2 = 25
var1 + 10
var1
var1 + var2
var_2
```

The last one is deliberately incorrect, so it should display an error message.

4. Continue experimenting if you like. However, this should make more sense after you read the next chapter and learn how to start writing Python code.

Exercise 1-2 Use IDLE to run programs

This exercise guides you through the process of using IDLE to compile and run three programs.

Run the console version of the Future Value program

1. Start IDLE and open the `future_value.py` file that should be in this folder:
`murach/python/book_apps/ch01`
2. Press F5 to compile and run the program. Then, enter values for monthly investment, yearly interest rate, and years when you're prompted for them. This should display the future value that's calculated from your entries.
3. When you're asked if you want to continue, enter "y" if you want to do another calculation or "n" to exit the program. Then, close the IDLE window for the source code.

Run the GUI version of the Future Value program

4. Open the `future_value_gui.py` file that should be in the same folder as the `future_value.py` file.
5. Select Run→Run Module to compile and run the program. Then, enter values for the first three text boxes, and click the Calculate button to view the future value that's calculated from your entries.
6. If you want to do another calculation, enter new values and click the Calculate button. When you're through, click the Exit button to exit the program. Then, close the IDLE window for the source code.

Run the Guess the Number program of chapter 4

7. Open the `guess_the_number.py` file that's in this folder:
`murach/python/book_apps/ch04`
8. Compile and run the program. Then, guess the number. This demonstrates the type of program that you'll be able to write by the time you complete chapter 4.
9. Close the window for the source code.

Exercise 1-3 Use IDLE to test and debug a program

This exercise guides you through the process of using IDLE to test and debug a program that has one syntax error and one runtime error. This walks you through the procedure that's shown in figure 1-9.

1. Start IDLE and open the `future_value_errors.py` file that should be in this folder:
`python/exercises/ch01`
2. Press F5 to compile and run the program. This should display a dialog box for a syntax error, and it should highlight the variable named “years” in the statement that gets the number of years.
3. Fix this syntax error by adding a right parenthesis at the end of the statement in the preceding line.
4. Press F5 again, and press Enter to save the updated file when the dialog box is displayed. This time, the program should compile cleanly and run in the interactive shell.
5. Enter the values for monthly investment, yearly interest rate, and years. This should cause a `NameError` exception that says “year” is not defined in this statement:
`months = year * 12`
6. Fix this runtime error by changing “year” to “years”.
7. Compile and run this program again. This time, the program should be able to calculate and display a future value for the three values that you enter.
8. To exit this program, enter “n” at the last prompt. Otherwise, the program will continue running.

How to become a Python programmer

The easiest way is to let [Murach's Python Programming \(2nd Edition\)](#) be your guide! So if you've enjoyed this chapter, I hope you'll get your own copy of the book today. You can use it to:

- Teach yourself the foundational skills that you'll use to code *any* Python program, in any field you choose to go on to (web development, game development, data analysis, scientific computing...there are many areas where Python programmers are in demand today)
- Plan the functions of your programs using hierarchy charts and test and debug them using the best Python techniques
- Use object-oriented programming techniques the way the pros do
- Pick up a new skill whenever you want or need to by focusing on material that's new to you
- Look up coding details or refresh your memory on forgotten details when you're in the middle of developing a Python program
- Loan to your colleagues who will be asking you more and more questions about Python



Mike Murach, Publisher

To get your copy, you can order online at www.murach.com or call us at 1-800-221-5528. And remember, when you order directly from us, this book comes with my personal guarantee:

100% Guarantee

When you buy directly from us, you must be satisfied. Try our books for 30 days or our eBooks for 14 days. They must outperform any competing book or course you've ever tried, or return your purchase for a prompt refund....no questions asked.

A handwritten signature in black ink, appearing to read 'Mike'.