Introduction to Programming

1

Computers help us perform many different tasks. They allow us to read the news, watch videos, play games, write books, purchase goods and services, perform complex mathematical analyses, communicate with friends and family, and so much more. All of these tasks require the user to provide input, such as clicking on a video to watch, or typing the sentences that should be included in a book. In response, the computer generates output, such as printing a book, playing sounds, or displaying text and images on the screen.

Consider the examples in the previous paragraph. How did the computer know what input to request? How did it know what actions to take in response to the input? How did it know what output to generate, and in what form it should be presented? The answer to all of these questions is "a person gave the computer instructions and the computer carried them out".

An *algorithm* is a finite sequence of effective steps that solve a problem. A step is effective if it is unambiguous and possible to perform. The number of steps must be finite (rather than infinite) so that all of the steps can be completed. Recipes, assembly instructions for furniture or toys, and the steps needed to open a combination lock are examples of algorithms that we encounter in everyday life.

The form in which an algorithm is presented is flexible and can be tailored to the problem that the algorithm solves. Words, numbers, lines, arrows, pictures and other symbols can all be used to convey the steps that must be performed. While the forms that algorithms take vary, all algorithms describe steps that can be followed to complete a task successfully.

A *computer program* is a sequence of instructions that control the behaviour of a computer. The instructions tell the computer when to perform tasks like reading input and displaying results, and how to transform and manipulate values to achieve a desired outcome. An algorithm must be translated into a computer program before a computer can be used to solve a problem. The translation process is called *programming* and the person who performs the translation is referred to as a *programmer*.

B. Stephenson, *The Python Workbook*, Texts in Computer Science,
https://doi.org/10.1007/978-3-030-18873-3_1

Computer programs are written in computer programming languages. Programming languages have precise syntax rules that must be followed carefully. Failing to do so will cause the computer to report an error instead of executing the programmer's instructions. A wide variety of different languages have been created, each of which has its own strengths and weaknesses. Popular programming languages currently include Java, C++, JavaScript, PHP, C# and Python, among others. While there are significant differences between these languages all of them allow a programmer to control the computer's behaviour.

This book uses the Python programming language because it is relatively easy for new programmers to learn, and it can be used to solve a wide variety of problems. Python statements that read keyboard input from the user, perform calculations, and generate text output are described in the sections that follow. Later chapters describe additional programming language constructs that can be used to solve larger and more complex problems.

1.1 Storing and Manipulating Values

A *variable* is a named location in a computer's memory that holds a value. In Python, variable names must begin with a letter or an underscore, followed by any combination of letters, underscores and numbers.[1] Variables are created using assignment statements. The name of the variable that we want to create appears to the left of the assignment operator, which is denoted by =, and the value that will be stored in the variable appears to the right of the assignment operator. For example, the following statement creates a variable named `x` and stores `5` in it:

```
x = 5
```

The right side of an assignment statement can be an arbitrarily complex calculation that includes parentheses, mathematical operators, numbers, and variables that were created by earlier assignment statements (among other things). Familiar mathematical operators that Python provides include addition (+), subtraction (−), multiplication (*), division (/), and exponentiation (**). Operators are also provided for floor division (//) and modulo (%). The floor division operator computes the floor of the quotient that results when one number is divided by another while the modulo operator computes the remainder when one number is divided by another.

The following assignment statement computes the value of one plus `x` squared and stores it in a new variable named `y`.

```
y = 1 + x ** 2
```

[1] Variable names are case sensitive. As a result, `count`, `Count` and `COUNT` are distinct variable names, despite their similarity.

Python respects the usual order of operations rules for mathematical operators. Since x is 5 (from the previous assignment statement) and exponentiation has higher precedence than addition, the expression to the right of the assignment operator evaluates to 26. Then this value is stored in y.

The same variable can appear on both sides of an assignment operator. For example:

```
y = y - 6
```

While your initial reaction might be that such a statement is unreasonable, it is, in fact, a valid Python statement that is evaluated just like the assignment statements we examined previously. Specifically, the expression to the right of the assignment operator is evaluated and then the result is stored into the variable to the left of the assignment operator. In this particular case y is 26 when the statement starts executing, so 6 is subtracted from y resulting in 20. Then 20 is stored into y, replacing the 26 that was stored there previously. Subsequent uses of y will evaluate to the newly stored value of 20 (until it is changed with another assignment statement).

1.2 Calling Functions

There are some tasks that many programs have to perform such as reading input values from the keyboard, sorting a list, and computing the square root of a number. Python provides functions that perform these common tasks, as well as many others. The programs that we create will call these functions so that we don't have to solve these problems ourselves.

A function is called by using its name, followed by parentheses. Many functions require values when they are called, such as a list of names to sort or the number for which the square root will be computed. These values, called *arguments*, are placed inside the parentheses when the function is called. When a function call has multiple arguments they are separated by commas.

Many functions compute a result. This result can be stored in a variable using an assignment statement. The name of the variable appears to the left of the assignment operator and the function call appears to the right of the assignment operator. For example, the following assignment statement calls the round function, which rounds a number to the closest integer.

```
r = round(q)
```

The variable q (which must have been assigned a value previously) is passed as an argument to the round function. When the round function executes it identifies the integer that is closest to q and returns it. Then the returned integer is stored in r.

1.2.1 Reading Input

Python programs can read input from the keyboard by calling the `input` function. This function causes the program to stop and wait for the user to type something. When the user presses the `enter` key the characters typed by the user are returned by the `input` function. Then the program continues executing. Input values are normally stored in a variable using an assignment statement so that they can be used later in the program. For example, the following statement reads a value typed by the user and stores it in a variable named `a`.

```
a = input()
```

The `input` function always returns a *string*, which is computer science terminology for a sequence of characters. If the value being read is a person's name, the title of a book, or the name of a street, then storing the value as a string is appropriate. But if the value is numeric, such as an age, a temperature, or the cost of a meal at a restaurant, then the string entered by the user is normally converted to a number. The programmer must decide whether the result of the conversion should be an integer or a floating-point number (a number that can include digits to the right of the decimal point). Conversion to an integer is performed by calling the `int` function while conversion to a floating-point number is performed by calling the `float` function.

It is common to call the `int` and `float` functions in the same assignment statement that reads an input value from the user. For example, the following statements read a customer's name, the quantity of an item that they would like to purchase, and the item's price. Each of these values is stored in its own variable with an assignment statement. The name is stored as a string, the quantity is stored as an integer, and the price is stored as a floating-point number.

```
name = input("Enter your name: ")
quantity = int(input("How many items? "))
price = float(input("Cost per item? "))
```

Notice that an argument was provided to the `input` function each time it was called. This argument, which is optional, is a prompt that tells the user what to enter. The prompt must be string. It is enclosed in double quotes so that Python knows to treat the characters as a string instead of interpreting them as the names of functions or variables.

Mathematical calculations can be performed on both integers and floating-point numbers. For example, another variable can be created that holds the total cost of the items with the following assignment statement:

```
total = quantity * price
```

This statement will only execute successfully if `quantity` and `price` have been converted to numbers using the `int` and `float` functions described previously. Attempting to multiply these values without converting them to numbers will cause your Python program to crash.

1.2.2 Displaying Output

Text output is generated using the `print` function. It can be called with one argument, which is the value that will be displayed. For example, the following statements print the number `1`, the string `Hello!`, and whatever is currently stored in the variable `x`. The value in `x` could be an integer, a floating-point number, a string, or a value of some other type that we have not yet discussed. Each item is displayed on its own line.

```
print(1)
print("Hello!")
print(x)
```

Multiple values can be printed with one function call by providing several arguments to the `print` function. The additional arguments are separated by commas. For example:

```
print("When x is", x, "the value of y is", y)
```

All of these values are printed on the same line. The arguments that are enclosed in double quotes are strings that are displayed exactly as typed. The other arguments are variables. When a variable is printed, Python displays the value that is currently stored in it. A space is automatically included between each item when multiple items are printed.

The arguments to a function call can be values and variables, as shown previously. They can also be arbitrarily complex expressions involving parentheses, mathematical operators and other function calls. Consider the following statement:

```
print("The product of", x, "and", y, "is", x * y)
```

When it executes, the product, `x * y`, is computed and then displayed along with all of the other arguments to the `print` function.

1.2.3 Importing Additional Functions

Some functions, like `input` and `print` are used in many programs while others are not used as broadly. The most commonly used functions are available in all programs, while other less commonly used functions are stored in *modules* that the programmer can import when they are needed. For example, additional mathematical functions are located in the `math` module. It can be imported by including the following statement at the beginning of your program:

```
import math
```

Functions in the `math` module include `sqrt`, `ceil` and `sin`, among many others. A function imported from a module is called by using the module name,

followed by a period, followed by the name of the function and its arguments. For example, the following statement computes the square root of `y` (which must have been initialized previously) and stores the result in `z` by calling the `math` module's `sqrt` function.

```
z = math.sqrt(y)
```

Other commonly used Python modules include `random`, `time` and `sys`, among others. More information about all of these modules can be found online.

1.3 Comments

Comments give programmers the opportunity to explain what, how or why they are doing something in their program. This information can be very helpful when returning to a project after being away from it for a period of time, or when working on a program that was initially created by someone else. The computer ignores all of the comments in the program. They are only included to benefit people.

In Python, the beginning of a comment is denoted by the # character. The comment continues from the # character to the end of the line. A comment can occupy an entire line, or just part of it, with the comment appearing to the right of a Python statement.

Python files commonly begin with a comment that briefly describes the program's purpose. This allows anyone looking at the file to quickly determine what the program does without carefully examining its code. Commenting your code also makes it much easier to identify which lines perform each of the tasks needed to compute the program's results. You are strongly encouraged to write thorough comments when completing all of the exercises in this book.

1.4 Formatting Values

Sometimes the result of a mathematical calculation will be a floating-point number that has many digits to the right of the decimal point. While one might want to display all of the digits in some programs, there are other circumstances where the value must be rounded to a particular number of decimal places. Another unrelated program might output a large number of integers that need to be lined up in columns. Python's formatting constructs allow us to accomplish these, and many other, tasks.

A programmer tells Python how to format a value using a *format specifier*. The specifier is a sequence of characters that describe a variety of formatting details. It uses one character to indicate what type of formatting should be performed. For example, an `f` indicates that a value should be formatted as a floating-point number while a `d` or an `i` indicates that a value should be formatted as a decimal (base-10) integer and an `s` indicates that a value should be formatted as a string. Characters

can precede the `f`, `d`, `i` or `s` to control additional formatting details. We will only consider the problems of formatting a floating-point number so that it includes a specific number of digits to the right of the decimal point and formatting values so that they occupy some minimum number of characters (which allows values to be printed in columns that line up nicely). Many additional formatting tasks can be performed using format specifiers, but these tasks are outside the scope of this book.

A floating-point number can be formatted to include a specific number of decimal places by including a decimal point and the desired number of digits immediately ahead of the `f` in the format specifier. For example, `.2f` is used to indicate that a value should be formatted as a floating-point number with two digits to the right of the decimal point while `.7f` indicates that 7 digits should appear to the right of the decimal point. Rounding is performed when the number of digits to the right of the decimal point is reduced. Zeros are added if the number of digits is increased. The number of digits to the right of the decimal point cannot be specified when formatting integers and strings.

Integers, floating-point numbers and strings can all be formatted so that they occupy at least some minimum width. Specifying a minimum width is useful when generating output that includes columns of values that need to be lined up. The minimum number of characters to use is placed before the `d`, `i`, `f` or `s`, and before the decimal point and number of digits to the right of the decimal point (if present). For example, `8d` indicates that a value should be formatted as a decimal integer occupying a minimum of 8 characters while `6.2f` indicates that a value should be formatted as a floating-point number using a minimum of 6 characters, including the decimal point and the two digits to its right. Leading spaces are added to the formatted value, when needed, to reach the minimum number of characters.

Finally, once the correct formatting characters have been identified, a percent sign (`%`) is prepended to them. A format specifier normally appears in a string. It can be the only characters in the string, or it can be part of a longer message. Examples of complete format specifier strings include `"%8d"`, `"The amount owing is %.2f"` and `"Hello %s! Welcome aboard!"`.

Once the format specifier has been created the formatting operator, denoted by `%`, is used to format a value.[2] The string containing the format specifier appears to the left of the formatting operator. The value being formatted appears to its right. When the formatting operator is evaluated, the value on the right is inserted into the string on the left (at the location of the format specifier using the indicated formatting) to compute the operator's result. Any characters in the string that are not part of a format specifier are retained without modification. Multiple values can be formatted simultaneously by including multiple format specifiers in the string to the left of the formatting operator, and by comma separating all of the values to be formatted inside parentheses to the right of the formatting operator.

[2]Python provides several different mechanisms for formatting strings including the formatting operator, the `format` function and `format` method, template strings and, most recently, f-strings. We will use the formatting operator for all of the examples and exercises in this book but the other techniques can also be used to achieve the same results.

String formatting is often performed as part of a `print` statement. The first `print` statement in the following code segment displays the value of the variable `x`, with exactly two digits to the right of the decimal point. The second `print` statement formats two values before displaying them as part of a larger output message.

```
print("%.2f" % x)
print("%s ate %d cookies!" % (name, numCookies))
```

Several additional formatting examples are shown in the following table. The variables x, y and z have previously been assigned `12`, `-2.75` and `"Andrew"` respectively.

Code Segment:	`"%d" % x`
Result:	`"12"`
Explanation:	The value stored in `x` is formatted as a decimal (base 10) integer.
Code Segment:	`"%f" % y`
Result:	`"-2.75"`
Explanation:	The value stored in `y` is formatted as a floating-point number.
Code Segment:	`"%d and %f" % (x, y)`
Result:	`"12 and -2.75"`
Explanation:	The value stored in `x` is formatted as a decimal (base 10) integer and the value stored in `y` is formatted as a floating-point number. The other characters in the string are retained without modification.
Code Segment:	`"%.4f" % x`
Result:	`"12.0000"`
Explanation:	The value stored in `x` is formatted as a floating-point number with 4 digits to the right of the decimal point.
Code Segment:	`"%.1f" % y`
Result:	`"-2.8"`
Explanation:	The value stored in `y` is formatted as a floating-point number with 1 digit to the right of the decimal point. The value was rounded when it was formatted because the number of digits to the right of the decimal point was reduced.
Code Segment:	`"%10s" % z`
Result:	`"    Andrew"`
Explanation:	The value stored in `z` is formatted as a string so that it occupies at least 10 spaces. Because `z` is only 6 characters long, 4 leading spaces are included in the result.
Code Segment:	`"%4s" % z`
Result:	`"Andrew"`
Explanation:	The value stored in `z` is formatted as a string so that it occupies at least 4 spaces. Because `z` is longer than the indicated minimum length, the resulting string is equal to `z`.
Code Segment:	`"%8i%8i" % (x, y)`
Result:	`"      12      -2"`
Explanation:	Both `x` and `y` are formatted as decimal (base 10) integers occupying a minimum of 8 spaces. Leading spaces are added as necessary. The digits to the right of decimal point are truncated (not rounded) when `y` (a floating-point number) is formatted as an integer.

1.5 Working with Strings

Like numbers, strings can be manipulated with operators and passed to functions. Operations that are commonly performed on strings include concatenating two strings, computing the length of a string, and extracting individual characters from a string. These common operations are described in the remainder of this section. Information about other string operations can be found online.

Strings can be concatenated using the + operator. The string to the right of the operator is appended to the string to the left of the operator to form the new string. For example, the following program reads two strings from the user which are a person's first and last names. It then uses string concatenation to construct a new string which is the person's last name, followed by a comma and a space, followed by the person's first name. Then the result of the concatenation is displayed.

```
# Read the names from the user
first = input("Enter the first name: ")
last = input("Enter the last name: ")

# Concatenate the strings
both = last + ", " + first

# Display the result
print(both)
```

The number of characters in a string is referred to as a string's length. This value, which is always a non-negative integer, is computed by calling the `len` function. A string is passed to the function as its only argument and the length of that string is returned as its only result. The following example demonstrates the `len` function by computing the length of a person's name.

```
# Read the name from the user
first = input("Enter your first name: ")

# Compute its length
num_chars = len(first)

# Display the result
print("Your first name contains", num_chars, "characters")
```

Sometimes it is necessary to access individual characters within a string. For example, one might want to extract the first character from each of three strings containing a first name, middle name and last name, in order to display a person's initials.

Each character in a string has a unique integer *index*. The first character in the string has index 0 while the last character in the string has an index which is equal to the length of the string, minus one. A single character in a string is accessed by placing its index inside square brackets after the name of the variable containing the string. The following program demonstrates this by displaying a person's initials.

```
# Read the user's name
first = input("Enter your first name: ")
middle = input("Enter your middle name: ")
last = input("Enter your last name: ")

# Extract the first character from each string and concatenate them
initials = first[0] + middle[0] + last[0]

# Display the initials
print("Your initials are", initials)
```

Several consecutive characters in a string can be accessed by including two indices, separated by a colon, inside the square brackets. This is referred to as slicing a string. String slicing can be used to access multiple characters within a string in an efficient manner.

1.6 Exercises

The exercises in this chapter will allow you to put the concepts discussed previously into practice. While the tasks that they ask you to complete are generally small, solving these exercises is an important step toward the creation of larger programs that solve more interesting problems.

Exercise 1: Mailing Address

(*Solved, 9 Lines*)

Create a program that displays your name and complete mailing address. The address should be printed in the format that is normally used in the area where you live. Your program does not need to read any input from the user.

Exercise 2: Hello

(*9 Lines*)

Write a program that asks the user to enter his or her name. The program should respond with a message that says hello to the user, using his or her name.

Exercise 3: Area of a Room

(*Solved, 13 Lines*)

Write a program that asks the user to enter the width and length of a room. Once these values have been read, your program should compute and display the area of the room. The length and the width will be entered as floating-point numbers. Include units in your prompt and output message; either feet or meters, depending on which unit you are more comfortable working with.

Exercise 4: Area of a Field

(*Solved, 15 Lines*)

Create a program that reads the length and width of a farmer's field from the user in feet. Display the area of the field in acres.

> Hint: There are 43,560 square feet in an acre.

Exercise 5: Bottle Deposits

(*Solved, 15 Lines*)

In many jurisdictions a small deposit is added to drink containers to encourage people to recycle them. In one particular jurisdiction, drink containers holding one liter or less have a $0.10 deposit, and drink containers holding more than one liter have a $0.25 deposit.

Write a program that reads the number of containers of each size from the user. Your program should continue by computing and displaying the refund that will be received for returning those containers. Format the output so that it includes a dollar sign and two digits to the right of the decimal point.

Exercise 6: Tax and Tip

(*Solved, 17 Lines*)

The program that you create for this exercise will begin by reading the cost of a meal ordered at a restaurant from the user. Then your program will compute the tax and tip for the meal. Use your local tax rate when computing the amount of tax owing. Compute the tip as 18 percent of the meal amount (without the tax). The output from your program should include the tax amount, the tip amount, and the grand total for the meal including both the tax and the tip. Format the output so that all of the values are displayed using two decimal places.

Exercise 7: Sum of the First n Positive Integers

(*Solved, 11 Lines*)

Write a program that reads a positive integer, n, from the user and then displays the sum of all of the integers from 1 to n. The sum of the first n positive integers can be computed using the formula:

$$\text{sum} = \frac{(n)(n+1)}{2}$$

Exercise 8: Widgets and Gizmos

(*15 Lines*)

An online retailer sells two products: widgets and gizmos. Each widget weighs 75 grams. Each gizmo weighs 112 grams. Write a program that reads the number of widgets and the number of gizmos from the user. Then your program should compute and display the total weight of the parts.

Exercise 9: Compound Interest

(*19 Lines*)

Pretend that you have just opened a new savings account that earns 4 percent interest per year. The interest that you earn is paid at the end of the year, and is added to the balance of the savings account. Write a program that begins by reading the amount of money deposited into the account from the user. Then your program should compute and display the amount in the savings account after 1, 2, and 3 years. Display each amount so that it is rounded to 2 decimal places.

Exercise 10: Arithmetic

(*Solved, 22 Lines*)

Create a program that reads two integers, a and b, from the user. Your program should compute and display:

- The sum of a and b
- The difference when b is subtracted from a
- The product of a and b
- The quotient when a is divided by b
- The remainder when a is divided by b
- The result of $\log_{10} a$
- The result of a^b

Hint: You will probably find the `log10` function in the `math` module helpful for computing the second last item in the list.

Exercise 11: Fuel Efficiency

(*13 Lines*)

In the United States, fuel efficiency for vehicles is normally expressed in miles-per-gallon (MPG). In Canada, fuel efficiency is normally expressed in liters-per-hundred kilometers (L/100 km). Use your research skills to determine how to convert from MPG to L/100 km. Then create a program that reads a value from the user in American units and displays the equivalent fuel efficiency in Canadian units.

Exercise 12: Distance Between Two Points on Earth

(*27 Lines*)

The surface of the Earth is curved, and the distance between degrees of longitude varies with latitude. As a result, finding the distance between two points on the surface of the Earth is more complicated than simply using the Pythagorean theorem.

Let (t_1, g_1) and (t_2, g_2) be the latitude and longitude of two points on the Earth's surface. The distance between these points, following the surface of the Earth, in kilometers is:

$$\text{distance} = 6371.01 \times \arccos(\sin(t_1) \times \sin(t_2) + \cos(t_1) \times \cos(t_2) \times \cos(g_1 - g_2))$$

> The value 6371.01 in the previous equation wasn't selected at random. It is the average radius of the Earth in kilometers.

Create a program that allows the user to enter the latitude and longitude of two points on the Earth in degrees. Your program should display the distance between the points, following the surface of the earth, in kilometers.

> Hint: Python's trigonometric functions operate in radians. As a result, you will need to convert the user's input from degrees to radians before computing the distance with the formula discussed previously. The `math` module contains a function named `radians` which converts from degrees to radians.

Exercise 13: Making Change

(*Solved, 35 Lines*)

Consider the software that runs on a self-checkout machine. One task that it must be able to perform is to determine how much change to provide when the shopper pays for a purchase with cash.

Write a program that begins by reading a number of cents from the user as an integer. Then your program should compute and display the denominations of the coins that should be used to give that amount of change to the shopper. The change should be given using as few coins as possible. Assume that the machine is loaded with pennies, nickels, dimes, quarters, loonies and toonies.

> A one dollar coin was introduced in Canada in 1987. It is referred to as a loonie because one side of the coin has a loon (a type of bird) on it. The two dollar coin, referred to as a toonie, was introduced 9 years later. It's name is derived from the combination of the number two and the name of the loonie.

Exercise 14: Height Units

(*Solved, 16 Lines*)

Many people think about their height in feet and inches, even in some countries that primarily use the metric system. Write a program that reads a number of feet from the user, followed by a number of inches. Once these values are read, your program should compute and display the equivalent number of centimeters.

Hint: One foot is 12 inches. One inch is 2.54 centimeters.

Exercise 15: Distance Units

(*20 Lines*)

In this exercise, you will create a program that begins by reading a measurement in feet from the user. Then your program should display the equivalent distance in inches, yards and miles. Use the Internet to look up the necessary conversion factors if you don't have them memorized.

Exercise 16: Area and Volume

(*15 Lines*)

Write a program that begins by reading a radius, r, from the user. The program will continue by computing and displaying the area of a circle with radius r and the volume of a sphere with radius r. Use the `pi` constant in the `math` module in your calculations.

Hint: The area of a circle is computed using the formula $area = \pi r^2$. The volume of a sphere is computed using the formula $volume = \frac{4}{3}\pi r^3$.

Exercise 17: Heat Capacity

(*Solved, 23 Lines*)

The amount of energy required to increase the temperature of one gram of a material by one degree Celsius is the material's specific heat capacity, C. The total amount of energy, q, required to raise m grams of a material by ΔT degrees Celsius can be computed using the formula:

$$q = mC\Delta T$$

Write a program that reads the mass of some water and the temperature change from the user. Your program should display the total amount of energy that must be added or removed to achieve the desired temperature change.

Hint: The specific heat capacity of water is $4.186 \frac{J}{g°C}$. Because water has a density of 1.0 grams per milliliter, you can use grams and milliliters interchangeably in this exercise.

Extend your program so that it also computes the cost of heating the water. Electricity is normally billed using units of kilowatt hours rather than Joules. In this exercise, you should assume that electricity costs 8.9 cents per kilowatt hour. Use your program to compute the cost of boiling the water needed for a cup of coffee.

Hint: You will need to look up the factor for converting between Joules and kilowatt hours to complete the last part of this exercise.

Exercise 18: Volume of a Cylinder

(*15 Lines*)

The volume of a cylinder can be computed by multiplying the area of its circular base by its height. Write a program that reads the radius of the cylinder, along with its height, from the user and computes its volume. Display the result rounded to one decimal place.

Exercise 19: Free Fall

(*Solved, 15 Lines*)

Create a program that determines how quickly an object is travelling when it hits the ground. The user will enter the height from which the object is dropped in meters (m). Because the object is dropped its initial speed is 0 m/s. Assume that the acceleration due to gravity is $9.8\,\mathrm{m/s^2}$. You can use the formula $v_f = \sqrt{v_i^2 + 2ad}$ to compute the final speed, v_f, when the initial speed, v_i, acceleration, a, and distance, d, are known.

Exercise 20: Ideal Gas Law

(*19 Lines*)

The ideal gas law is a mathematical approximation of the behavior of gasses as pressure, volume and temperature change. It is usually stated as:

$$PV = nRT$$

where P is the pressure in Pascals, V is the volume in liters, n is the amount of substance in moles, R is the ideal gas constant, equal to $8.314 \frac{\mathrm{J}}{\mathrm{mol\ K}}$, and T is the temperature in degrees Kelvin.

Write a program that computes the amount of gas in moles when the user supplies the pressure, volume and temperature. Test your program by determining the number of moles of gas in a SCUBA tank. A typical SCUBA tank holds 12 liters of gas at a pressure of 20,000,000 Pascals (approximately 3,000 PSI). Room temperature is approximately 20 degrees Celsius or 68 degrees Fahrenheit.

Hint: A temperature is converted from Celsius to Kelvin by adding 273.15 to it. To convert a temperature from Fahrenheit to Kelvin, deduct 32 from it, multiply it by $\frac{5}{9}$ and then add 273.15 to it.

Exercise 21: Area of a Triangle

(*13 Lines*)

The area of a triangle can be computed using the following formula, where b is the length of the base of the triangle, and h is its height:

$$\text{area} = \frac{b \times h}{2}$$

Write a program that allows the user to enter values for b and h. The program should then compute and display the area of a triangle with base length b and height h.

Exercise 22: Area of a Triangle (Again)

(*16 Lines*)

In the previous exercise you created a program that computed the area of a triangle when the length of its base and its height were known. It is also possible to compute the area of a triangle when the lengths of all three sides are known. Let s_1, s_2 and s_3 be the lengths of the sides. Let $s = (s_1 + s_2 + s_3)/2$. Then the area of the triangle can be calculated using the following formula:

$$\text{area} = \sqrt{s \times (s - s_1) \times (s - s_2) \times (s - s_3)}$$

Develop a program that reads the lengths of the sides of a triangle from the user and displays its area.

Exercise 23: Area of a Regular Polygon

(*Solved, 14 Lines*)

A polygon is regular if its sides are all the same length and the angles between all of the adjacent sides are equal. The area of a regular polygon can be computed using the following formula, where s is the length of a side and n is the number of sides:

$$\text{area} = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

Write a program that reads s and n from the user and then displays the area of a regular polygon constructed from these values.

Exercise 24: Units of Time

(*22 Lines*)

Create a program that reads a duration from the user as a number of days, hours, minutes, and seconds. Compute and display the total number of seconds represented by this duration.

Exercise 25: Units of Time (Again)

(*Solved, 24 Lines*)

In this exercise you will reverse the process described in Exercise 24. Develop a program that begins by reading a number of seconds from the user. Then your program should display the equivalent amount of time in the form D:HH:MM:SS, where D, HH, MM, and SS represent days, hours, minutes and seconds respectively. The hours, minutes and seconds should all be formatted so that they occupy exactly two digits. Use your research skills determine what additional character needs to be included in the format specifier so that leading zeros are used instead of leading spaces when a number is formatted to a particular width.

Exercise 26: Current Time

(*10 Lines*)

Python's `time` module includes several time-related functions. One of these is the `asctime` function which reads the current time from the computer's internal clock and returns it in a human-readable format. Use this function to write a program that displays the current time and date. Your program will not require any input from the user.

Exercise 27: When is Easter?

(*33 Lines*)

Easter is celebrated on the Sunday immediately after the first full moon following the spring equinox. Because its date includes a lunar component, Easter does not have a fixed date in the Gregorian calendar. Instead, it can occur on any date between

March 22 and April 25. The month and day for Easter can be computed for a given year using the Anonymous Gregorian Computus algorithm, which is shown below.

Set a equal to the remainder when $year$ is divided by 19
Set b equal to the floor of $year$ divided by 100
Set c equal to the remainder when $year$ is divided by 100
Set d equal to the floor of b divided by 4
Set e equal to the remainder when b is divided by 4
Set f equal to the floor of $\frac{b+8}{25}$
Set g equal to the floor of $\frac{b-f+1}{3}$
Set h equal to the remainder when $19a+b-d-g+15$ is divided by 30
Set i equal to the floor of c divided by 4
Set k equal to the remainder when c is divided by 4
Set l equal to the remainder when $32+2e+2i-h-k$ is divided by 7
Set m equal to the floor of $\frac{a+11h+22l}{451}$
Set month equal to the floor of $\frac{h+l-7m+114}{31}$
Set day equal to one plus the remainder when $h+l-7m+114$ is divided by 31

Write a program that implements the Anonymous Gregorian Computus algorithm to compute the date of Easter. Your program should read the year from the user and then display a appropriate message that includes the date of Easter in that year.

Exercise 28: Body Mass Index

(*14 Lines*)

Write a program that computes the body mass index (BMI) of an individual. Your program should begin by reading a height and weight from the user. Then it should use one of the following two formulas to compute the BMI before displaying it. If you read the height in inches and the weight in pounds then body mass index is computed using the following formula:

$$\text{BMI} = \frac{\text{weight}}{\text{height} \times \text{height}} \times 703$$

If you read the height in meters and the weight in kilograms then body mass index is computed using this slightly simpler formula:

$$\text{BMI} = \frac{\text{weight}}{\text{height} \times \text{height}}$$

Exercise 29: Wind Chill

(*Solved, 22 Lines*)

When the wind blows in cold weather, the air feels even colder than it actually is because the movement of the air increases the rate of cooling for warm objects, like people. This effect is known as wind chill.

In 2001, Canada, the United Kingdom and the United States adopted the following formula for computing the wind chill index. Within the formula T_a is the air temperature in degrees Celsius and V is the wind speed in kilometers per hour. A similar formula with different constant values can be used for temperatures in degrees Fahrenheit and wind speeds in miles per hour.

$$WCI = 13.12 + 0.6215T_a - 11.37V^{0.16} + 0.3965T_aV^{0.16}$$

Write a program that begins by reading the air temperature and wind speed from the user. Once these values have been read your program should display the wind chill index rounded to the closest integer.

> The wind chill index is only considered valid for temperatures less than or equal to 10 degrees Celsius and wind speeds exceeding 4.8 kilometers per hour.

Exercise 30: Celsius to Fahrenheit and Kelvin

(*17 Lines*)

Write a program that begins by reading a temperature from the user in degrees Celsius. Then your program should display the equivalent temperature in degrees Fahrenheit and degrees Kelvin. The calculations needed to convert between different units of temperature can be found on the Internet.

Exercise 31: Units of Pressure

(*20 Lines*)

In this exercise you will create a program that reads a pressure from the user in kilopascals. Once the pressure has been read your program should report the equivalent pressure in pounds per square inch, millimeters of mercury and atmospheres. Use your research skills to determine the conversion factors between these units.

Exercise 32: Sum of the Digits in an Integer

(*18 Lines*)

Develop a program that reads a four-digit integer from the user and displays the sum of its digits. For example, if the user enters 3141 then your program should display $3+1+4+1=9$.

Exercise 33: Sort 3 Integers

(*Solved, 19 Lines*)

Create a program that reads three integers from the user and displays them in sorted order (from smallest to largest). Use the `min` and `max` functions to find the smallest and largest values. The middle value can be found by computing the sum of all three values, and then subtracting the minimum value and the maximum value.

Exercise 34: Day Old Bread

(*Solved, 19 Lines*)

A bakery sells loaves of bread for $3.49 each. Day old bread is discounted by 60 percent. Write a program that begins by reading the number of loaves of day old bread being purchased from the user. Then your program should display the regular price for the bread, the discount because it is a day old, and the total price. Each of these amounts should be displayed on its own line with an appropriate label. All of the values should be displayed using two decimal places, and the decimal points in all of the numbers should be aligned when reasonable values are entered by the user.