In that case, go back to the program window and make the needed corrections, then re-save and re-execute the program. You may need to go through this process a number of times until all the syntax errors have been corrected.

1.7 A First Program—Calculating the Drake Equation

Vincedevries/LittleGreenMen/Wikimedia Commons

Dr. Frank Drake conducted the first search for radio signals from extraterrestrial civilizations in 1960. This established SETI (Search for Extraterrestrial Intelligence), a new area of scientific inquiry. In order to estimate the number of civilizations that may exist in our galaxy that we may be able to communicate with, he developed what is now called the *Drake equation*.

The Drake equation accounts for a number of different factors. The values used for some of these are the result of scientific study, while others are only the result of an "intelligent guess." The factors consist of R, the average rate of star creation per year in our galaxy; p, the percentage of those stars that have planets; n, the average number of planets that can potentially support life for each star with planets; f, the percentage of those planets that actually go on to develop life; i, the percentage of those planets that go on to develop intelligent life; c, the percentage of those that have the technology communicate with us; and L, the expected lifetime of civilizations (the period that they can communicate). The Drake equation is simply the multiplication of all these factors, giving N, the estimated number of detectable civilizations there are at any given time,

$$N = R \cdot p \cdot n \cdot f \cdot i \cdot c \cdot L$$

Figure 1-32 shows those parameters in the Drake equation that have some consensus as to their correct value.

Drake Equation Factor Values		Estimated Values
Rate of star creation	R	7†
Percentage of stars with planets	p	40%
Average number of planets that can potentially support life for each star with planets	n	(no consensus)
Percentage of those that go on to develop life	f	13%
Percentage of those that go on to intelligent develop life	i	(no consensus)
Percentage of those willing and able to communicate	c	(no consensus)
Expected lifetime of civilizations	L	(no consensus)
† Estimate of NASA and the European Space Agency		

FIGURE 1-32 Proposed Values for the Drake Equation

1.7.1 The Problem

The value of 7 for R, the rate of star creation, is the least disputed value in the Drake equation today. Given the uncertainty of the remaining factors, you are to develop a program that allows a user to enter their own estimated values for the remaining six factors (p, n, f, i, c, and L) and displays the calculated result.

1.7.2 Problem Analysis

This problem is very straightforward. We only need to understand the equation provided.

1.7.3 Program Design

The program design for this problem is also straightforward. The data to be represented consist of numerical values, with the Drake equation as the algorithm. The overall steps of the program are depicted in Figure 1-33.

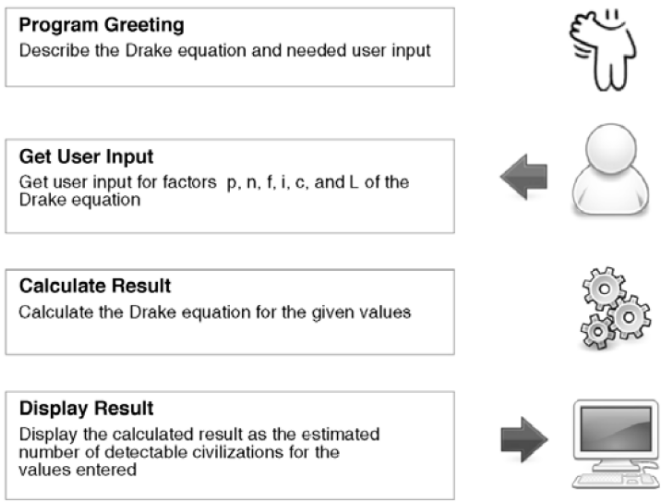

FIGURE 1-33 The Overall Steps of the Drake Equation Program

1.7.4 Program Implementation

The implementation of this program is fairly simple. The only programming elements needed are input, assignment, and print, along with the use of arithmetic operators. An implementation is given in Figure 1-34. Example execution of the program is given in Figure 1-35.

First, note the program lines beginning with the hash sign, #. In Python, this symbol is used to denote a *comment statement*. A **comment statement** contains information for persons reading the program. Comment statements are ignored during program execution—they have no effect on the program results. In this program, the initial series of comment statements (**lines 1–23**) explain the Drake equation and provide a brief summary of the purpose of the program.

```
1  # SETI Program
2  #
3  # The Drake equation, developed by Frank Drake in the 1960s, attempts to
4  # estimate how many extraterrestrial civilizations, N, may exist in our
5  # galaxy at any given time that we might come in contact with,
6  #
7  #    N = R * p * n * f * i * c * L
8  #
9  # where,
10 #
11 #    R ... estimated rate of star creation in our galaxy
12 #    p ... estimated percent of stars that have planets
13 #    n ... estimated average number of planets that can potentially support
14 #          life for each star with planets
15 #    f ... estimated percent of those planets that actually go on to develop life
16 #    i ... estimated percent of those planets go on to develop intelligent life
17 #    c ... estimated percent of those that are willing and able to communicate
18 #    L ... estimated expected lifetime of such civilizations
19 #
20 # Given that the value for R, 7 per year, is the least disputed of the values,
21 # the user will be prompted to enter estimated values for the remaining six
22 # factors. The estimated number of civilizations that may be detected in our
23 # galaxy will then be displayed.
24
25 # display program welcome
26 print('Welcome to the SETI program')
27 print('This program will allow you to enter specific values related to')
28 print('the likelihood of finding intelligent life in our galaxy. All')
29 print('percentages should be entered as integer values, e.g., 40 and not .40')
30 print()
31
32 # get user input
33 p = int(input('What percentage of stars do you think have planets?: '))
34 n = int(input('How many planets per star do you think can support life?: '))
35 f = int(input('What percentage do you think actually develop life?: '))
36 i = int(input('What percentage of those do you think have intelligent life?: '))
37 c = int(input('What percentage of those do you think can communicate with us?: '))
38 L = int(input('Number of years you think civilizations last?: '))
39
40 # calculate result
41 num_detectable_civilizations = 7 * (p/100) * n * (f/100) * (i/100) * (c/100) * L
42
43 # display result
44 print()
45 print('Based on the values entered ...')
46 print('there are an estimated', round(num_detectable_civilizations),
47       'potentially detectable civilizations in our galaxy')
```

FIGURE 1-34 Drake Equation Program

Comment statements are also used in the program to denote the beginning of each program section (**lines 25, 32, 40,** and **43**). These *section headers* provide a broad outline of the program following the program design depicted in Figure 1-33.

The program welcome section (**lines 25–30**) contains a series of print instructions displaying output to the screen. Each begins with the word `print` followed by a matching pair of parentheses. Within the parentheses are the items to be displayed. In this case, each contains a particular string of characters. The final "empty" print, `print()` on **line 30** (and **line 44**), does not display anything. It simply causes the screen cursor to move down to the next line, therefore creating a skipped line in the screen output. (Later we will see another way of creating the same result.)

Program Execution ...

```
Welcome to the SETI program
This program will allow you to enter specific values related to
the likelihood of finding intelligent life in our galaxy. All
percentages should be entered as integer values, e.g., 40 and not .40

What percentage of stars do you think have planets?: 40
How many planets per star do you think can support life?: 2
What percentage do you think actually develop life?: 5
What percentage of those do you think have intelligent life?: 3
What percentage of those do you think can communicate with us?: 5
Number of years you think civilizations last?: 10000

Based on the values entered...
there are an estimated 4.2 potentially detectable civilizations in
our galaxy
>>>
```

FIGURE 1-35 Execution of the Drake Equation Program

The following section (**lines 32–38**) contains the instructions for requesting the input from the user. Previously, we saw the `input` function used for inputting a name entered by the user. In that case, the instruction was of the form,

```
name = input('What is your name?:')
```

In this program, there is added syntax,

```
p = int(input('What percentage of stars do you think have planets?:'))
```

The `input` function always returns what the user enters as a string of characters. This is appropriate when a person's name is being entered. However, in this program, numbers are being entered, not names. Thus, in, the following,

```
What percentage of stars do you think have planets?: 40
```

`40` is to be read as single number and not as the characters `'4'` and `'0'`. The addition of the `int` (. . .) syntax around the input instruction accomplishes this. This will be discussed more fully when numeric and string (character) data are discussed in Chapter 2.

On **line 41** the Drake equation is calculated and stored in variable `num_detectable_civilizations`. Note that since some of the input values were meant to be percentages (`p`, `f`, `i`, and `c`), those values in the equation are each divided by 100. Finally, **lines 44–47** display the results.

1.7.5 Program Testing

To test the program, we can calculate the Drake equation for various other values using a calculator, providing a set of *test cases*. A **test case** is a set of input values and expected output of a given program. A **test plan** consists of a number of test cases to verify that a program meets all requirements. A good strategy is to include "average," as well as "extreme" or "special" cases in a test plan. Such a test plan is given in Figure 1-36.

Based on these results, we can be fairly confident that the program will give the correct results for all input values.

Input Values	p	n	f	i	c	L	Expected Results	Actual Results	Evaluation
Extreme Cases (no chance of contacting intelligent life)									
Zero Planets per Star	0	2	100%	1%	1%	10,000	0	0	passed
Zero percent of Planets Support Life	50%	0	100%	1%	1%	10,000	0	0	passed
Average Cases									
Average Case 1	30%	3	75%	1%	5%	5,000	12	12	passed
Average Case 2	50%	6	80%	5%	10%	10,000	840	840	passed
Extreme Cases (great chance of contacting intelligent life)									
Extreme Case 1	100%	10	100%	100%	100%	10,000	700,000	700,000	passed
Extreme Case 2	100%	12	100%	100%	100%	100,000	8,400,000	8,400,000	passed

FIGURE 1-36 Test Plan Results for Drake's Equation Program

CHAPTER SUMMARY

General Topics

Computational Problem Solving/Representation/Abstraction
Algorithms/Brute Force Approach
Computer Hardware/Transistors/Integrated Circuits/Moore's Law
Binary Representation/Bit/Byte/Binary-Decimal Conversion
Central Processing Unit (CPU)/Main and Secondary Memory
Input/Output Devices/Buses
Computer Software/System Software/Application Software
Operating Systems
Syntax and Semantics/Program Translation/Compiler vs. Interpreter
Program Debugging/Syntax Errors vs. Logic (Semantic) Errors
Procedural Programming vs. Object-Oriented Programming
The Process of Computational Problem Solving/Program Testing
Integrated Development Environments (IDE)/Program Editors/Debuggers
Comment Statements as Program Documentation
Test Cases/Test Plans

Python-Specific Programming Topics

The Python Programming Language/Guido van Rossum (creator of Python)
Comment Statements in Python

Page 36:

P3. Write a Python program that allows the user to enter any integer base and integer exponent, and displays the value of the base raised to that exponent. Your program should function as shown below.

```
What base? 10
What power of 10? 4
10 to the power of 4 is 10000
```

P4. Write a Python program that allows the user to enter a four-digit binary number and displays its value in base 10. *Each binary digit should be entered one per line, starting with the leftmost digit*, as shown below.

```
Enter leftmost digit: 1
Enter the next digit: 0
Enter the next digit: 0
Enter the next digit: 1
The value is 9
```

Page 37:

M2. Modify the Drake's Equation Program in section 1.7 so that it calculates results for a best case scenario, that is, so that factors `p` (percentage of stars that have planets), `f` (percentage of those planets that develop life), `i` (percentage of those planets that develop intelligent life), and `c` (percentage of those planets that can communicate with us) are all hard-coded as 100%. The value of R should remain as 7. Design the program so that the only values that the user is prompted (asked) for are how many planets per star can support life, `n`, and the estimated number of years civilizations last, `L`.

A **mixed-type expression** is an expression with operands of different type.

Coercion vs. Type Conversion

Coercion is the *implicit* (automatic) conversion of operands to a common type. Coercion is automatically performed on mixed-type expressions only if the operands can be safely converted, that is, if no loss of information will result. The conversion of integer `2` to floating-point `2.0` below is a safe conversion—the conversion of `4.5` to integer `4` is not, since the decimal digit would be lost,

```
2 + 4.5 → 2.0 + 4.5 → 6.5  safe (automatic conversion of int to float)
```

Type conversion is the *explicit* conversion of operands to a specific type. Type conversion can be applied even if loss of information results. Python provides built-in **type conversion functions** `int()` and `float()`, with the `int()` function truncating results as given in Figure 2-21.

```
float(2) + 4.5  →  2.0 + 4.5  →  6.5
2 + int(4.5)  →  2 + 4  →  6
```

Conversion Function		Converted Result		Conversion Function		Converted Result
`int()`	`int(10.8)`	10		`float()`	`float(10)`	10.0
	`int('10')`	10			`float('10')`	10.0
	`int('10.8')`	ERROR			`float('10.8')`	10.8

FIGURE 2-21 Conversion Functions int() and float() in Python

Note that numeric strings can also be converted to a numeric type. In fact, we have already been doing this when using `int` or `float` with the `input` function,

```
num_credits = int(input('How many credits do you have? '))
```

Coercion is the *implicit* (automatic) conversion of operands to a common type. **Type conversion** is the *explicit* conversion of operands to a specific type.

2.4.6 Let's Apply It—"Temperature Conversion Program"

The following Python program (Figure 2-22) requests from the user a temperature in degrees Fahrenheit, and displays the equivalent temperature in degrees Celsius. This program utilizes the following programming features:

➤ arithmetic expressions ➤ operator associativity ➤ format function

Program Execution ...

```
This program will convert degrees Fahrenheit to degrees Celsius
Enter degrees Fahrenheit: 100
100.0 degrees Fahrenheit equals 37.8 degrees Celsius
```

```
1  # Temperature Conversion Program (Fahrenheit to Celsius)
2
3  # This program will convert a temperature entered in Fahrenheit
4  # to the equivalent degrees in Celsius
5
6  # program greeting
7  print('This program will convert degrees Fahrenheit to degrees Celsius')
8
9  #get temperature in Fahrenheit
10 fahren = float(input('Enter degrees Fahrenheit: '))
11
12 # calc degrees Celsius
13 celsius = (fahren - 32) * 5 / 9
14
15 # output degrees Celsius
16 print(fahrenheit, 'degrees Fahrenheit equals',
17       format(celsius, '.1f'), 'degrees Celsius')
```

FIGURE 2-22 Temperature Conversion Program

Lines 1–4 contain the program description. **Line 7** provides the program greeting. **Line 10** reads the Fahrenheit temperature entered, assigned to variable `fahren`. Either an integer or a floating-point value may be entered, since the input is converted to float type. **Line 13** performs the calculation for converting Fahrenheit to Celsius. Recall that the division and multiplication operators have the same level of precedence. Since these operators associate left-to-right, the multiplication operator is applied first. Because of the use of the "true" division operator `/`, the result of the expression will have floating-point accuracy. Finally, **lines 16–17** output the converted temperature in degrees Celsius.

Self-Test Questions

1. What value does the following expression evaluate to?
   `2 + 9 * ((3 * 12) - 8) / 10`
   (a) `27` **(b)** `27.2` **(c)** `30.8`

2. Evaluate the following arithmetic expressions using the rules of operator precedence in Python.
   (a) `3 + 2 * 10` **(b)** `2 + 5 * 4 + 3` **(c)** `20 // 2 * 5` **(d)** `2 * 3 ** 2`

3. Evaluate the following arithmetic expressions based on Python's rules of operator associativity.
   (a) `24 // 4 // 2` **(b)** `2 ** 2 ** 3`

4. Which of the following is a mixed-type expression?
   (a) `2 + 3.0` **(b)** `2 + 3 * 4`

Page 76:

P1. Write a Python program that prompts the user for two integer values and displays the result of the first number divided by the second, with exactly two decimal places displayed.

M4. Modify the Temperature Conversion program in section 2.4.6 to convert from Celsius to Fahrenheit instead. The formula for the conversion is `f = (c * 9/5) + 32`.