Programming Exercises

The exercises in this section are optional and do not report to the performance dashboard. Instructors can decide whether to assign these exercises and students can check the correctness of their programs using the Check Exercise tool.

Pedagogical Note

For each problem, read it several times until you understand it. Think how to solve the problem before coding. Translate your logic into a program.

A problem often can be solved in many different ways. You should explore various solutions.

Sections 5.2–5.10

***5.1** (*Count positive and negative numbers and compute the average of numbers*) Write a program that reads an unspecified number of integers, determines how many positive and negative values have been read, and computes the total and average of the input values (not counting zeros). Your program ends with the input `0`. Display the average as a floating-point number.

5.2 (*Repeat additions*) LiveExample 5.4, SubtractionQuizLoop.py, generates five random subtraction questions. Revise the program to generate ten random addition questions for two integers between 1 and 15. Display the correct count and test time.

5.3 (*Conversion from kilograms to pounds*) Write a program that displays the following table (note that 1 kilogram is 2.2 pounds):

```
Kilograms    Pounds
1              2.2
3              6.6
...
197          433.4
199          437.8
```

5.4 (*Conversion from miles to kilometers*) Write a program that displays the following table (note that 1 mile is 1.609 kilometers):

```
Miles       Kilometers
1           1.609
2           3.218
...
9           14.481
10          16.090
```

***5.5** (*Conversion from kilograms to pounds*) Write a program that displays the following two tables side by side (note that 1 kilogram is 2.2 pounds):

```
Kilograms   Pounds     |      Pounds         Kilograms
1           2.2        |      20             9.09
3           6.6        |      25             11.36
...
197         433.4      |      510            231.82
199         437.8      |      515            234.09
```

***5.6** (*Conversion from miles to kilometers and kilometers to miles*) Write a program that displays the following two tables side by side (note that 1 mile is 1.609 kilometers):

```
Miles       Kilometers  |   Kilometers       Miles
1           1.609       |   20               12.430
2           3.218       |   25               15.538
...
9           15.481      |   60               37.290
10          16.090      |   65               40.398
```

5.7 (*Use trigonometric functions*) Print the following table to display the `sin` value and `cos` value of degrees from 0 to 360 with increments of 10 degrees. Round the value to keep four digits after the decimal point.

```
Degree      Sin         Cos
0           0.0000      1.0000
10          0.1736      0.9848
...
350         -0.1736     0.9848
360         0.0000      1.0000
```

5.8 (*Use the `math.sqrt` function*) Write a program that prints the following table using the `sqrt` function in the `math` module.

```
Number        Square Root
0             0.0000
2             1.4142
...
18            4.2426
20            4.4721
```

****5.9** (*Financial application: compute future tuition*) Suppose that the tuition for a university is $10,000 this year and increases 5% every year. In one year, the tuition will be $10,500.

Write a program that displays the tuition in 10 years and the total cost of four years' worth of tuition starting from the tenth year.

5.10 (*Find the highest score*) Write a program that prompts the user to enter the number of students and each student's name and score, and displays the name and highest score. Assume that the number of students is at least 1.

***5.11** (*Find the two highest scores*) Write a program that prompts the user to enter the number of students and each student's name and score, and displays the highest and second-highest name and scores. Assume that the number of students is at least 2.

5.12 (*Find numbers divisible by 5 and 6*) Write a program that displays, ten numbers per line, all the numbers from 100 to 1,000 that are divisible by 5 and 6. The numbers are separated by exactly one space.

5.13 (*Find numbers divisible by 5 or 6, but not both*) Write a program that displays, ten numbers per line, all the numbers from 100 to 200 that are divisible by 5 or 6, but not both. The numbers are separated by exactly one space.

5.14 (*Find the smallest* `n` *such that* $n^2 > 12,000$) Use a `while` loop to find the smallest integer `n` such that n^2 is greater than 12,000.

5.15 (*Find the largest* `n` *such that* $n^3 < 12,000$) Use a `while` loop to find the largest integer `n` such that n^3 is less than 12,000.

***5.16** (*Compute the greatest common divisor*) For Listing 5.8, another solution to find the greatest common divisor of two integers `n1` and `n2` is as follows: First find `d` to be the minimum of `n1` and `n2`, and then check whether `d`, `d - 1`, `d - 2`, ..., `2`, or `1` is a divisor for both `n1` and `n2` in this order. The first such common divisor is the greatest common divisor for `n1` and `n2`.

Section 5.11

***5.17** (*Display the ASCII character table*) Write a program that displays the characters in the ASCII character table from `!` to `~`. Display ten characters per line. The characters are separated by exactly one space.

****5.18** (*Find the factors of an integer*) Write a program that reads an integer and displays all its smallest factors, also known as *prime factors*. For example, if the input integer is 120, the output should be as follows:

```
2, 2, 2, 3, 5
```

****5.19** (*Display a pyramid*) Write a program that prompts the user to enter an integer from 1 to 15 and displays a pyramid, as shown in the following sample run:

***5.20** (*Display four patterns using loops*) Use nested loops that display the following patterns in four separate programs:

****5.21** (*Display numbers in a pyramid pattern*) Write a nested `for` loop that displays the following output:

***5.22** (*Display prime numbers between 2 and 1,000*) Modify Listing 5.14 to display all the prime numbers between 2 and 1,000, inclusive. Display eight prime numbers per line.

Comprehensive

**5.23 (*Financial application: compare loans with various interest rates*) Write a program that lets the user enter the loan amount and loan period in number of years and displays the monthly and total payments for each interest rate starting from 5% to 8%, with an increment of 1/8.

For the formula to compute monthly payment, see Listing 2.8, ComputeLoan.py.

**5.24 (*Financial application: loan amortization schedule*) The monthly payment for a given loan pays the principal and the interest. The monthly interest is computed by multiplying the monthly interest rate and the balance (the remaining principal). The principal paid for the month is therefore the monthly payment minus the monthly interest. Write a program that lets the user enter the loan amount, number of years, and interest rate, and then displays the amortization schedule for the loan.

> **Note**
>
> The balance after the last payment may not be zero. If so, the last payment should be the normal monthly payment plus the final balance.

HINT

Write a loop to display the table. Since the monthly payment is the same for each month, it should be computed before the loop. The balance is initially the loan amount. For each iteration in the loop, compute the interest and principal and update the balance. The loop may look like this:

```
for i in range(1, numberOfYears * 12 + 1):
    interest = monthlyInterestRate * balance
    principal = monthlyPayment - interest
    balance = balance - principal
    print(i, "\t\t", interest, "\t\t", principal,
"\t\t", balance)
```

***5.25** (*Demonstrate cancellation errors*) A *cancellation error* occurs when you are manipulating a very large number with a very small number. The large number may cancel out the smaller number. For example, the result of `100000000.0 + 0.000000001` is equal to `100000000.0`. To avoid cancellation errors and obtain more accurate results, carefully select the order of computation. For example, in computing the following series, you will obtain more accurate results by computing from right to left rather than from left to right:

$$1 + \frac{1}{2} + \frac{1}{3} \ \ldots \ + \frac{1}{n}$$

Write a program that compares the results of the summation of the preceding series, computing both from left to right and from right to left with `n = 50000`.

***5.26** (*Sum a series*) Write a program to sum the following series:

$$\frac{1}{3} + \frac{3}{5} + \frac{5}{7} + \frac{7}{9} + \frac{9}{11} + \frac{11}{13} + \ \ldots \ + \frac{95}{97} + \frac{97}{99}$$

**5.27 (*Compute π*) You can approximate π by using the following series:

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ \ldots \ + \frac{(-1)^{i+1}}{2i-1}\right)$$

Write a program that displays the π value for `i = 10000`, `20000`, ..., and `100000`.

**5.28 (*Compute e*) You can approximate `e` by using the following series:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ \ldots \ + \frac{1}{i!}$$

Write a program that displays the `e` value for `i = 10000`, `20000`, ..., and `100000`. (Hint: Since $i! = i \times (i-1) \times \ \ldots \ \times 2 \times 1$, then $\frac{1}{i!}$ is $\frac{1}{i(i-1)!}$ Initialize `e` and `item` to be `1` and keep adding a new `item` to `e`. The new item is the previous item divided by `i` for `i = 2.`, `3`, `4`, ...)

5.29 (*Display leap years*) Write a program that displays, ten per line, all the leap years from year 2001 to 2100. The years are separated by exactly one space. Also display the number of leap years in this period.

**5.30 (*Display the first days of each month*) Write a program that prompts the user to enter the year and first day of the year, and displays the first day of each month in the year on the console. For example, if the user entered year 2013, and 2 for Tuesday, January 1, 2013.

**5.31 (*Display calendars*) Write a program that prompts the user to enter the year and first day of the year, and displays on the console the calendar table for the year. For example, if the user entered year 2005, and 6 for Saturday, January 1, 2005, your program should display the calendar for each month in the year, as follows:

```
        January 2005
--------------------------
Sun Mon Tue Wed Thu Fri Sat
                        1
  2   3   4   5   6   7   8
  9  10  11  12  13  14  15
 16  17  18  19  20  21  22
 23  24  25  26  27  28  29
 30  31
```

. . .

```
        December 2005
--------------------------
Sun Mon Tue Wed Thu Fri Sat
                    1   2   3
  4   5   6   7   8   9  10
 11  12  13  14  15  16  17
 18  19  20  21  22  23  24
 25  26  27  28  29  30  31
```

***5.32** (*Financial application: compound value*) Suppose you save $100 *each* month into a savings account with the annual interest rate 5%. So, the monthly interest rate is $0.05/12 = 0.00417$. After the first month, the value in the account becomes

```
100 * (1 + 0.00417) = 100.417
```

After the second month, the value in the account becomes

```
(100 + 100.417) * (1 + 0.00417) = 201.252
```

After the third month, the value in the account becomes

```
(100 + 201.252) * (1 + 0.00417) = 302.507
```

and so on.

Write a program that prompts the user to enter an amount (e.g., 100), the annual interest rate (e.g., 5), and the number of months (e.g., 6), and displays the amount in the savings account after the given month.

***5.33** (*Financial application: compute CD value*) Suppose you put $10,000 into a CD with an annual percentage yield of 5.75%. After one month, the CD is worth

```
10000 + 10000 * 5.75 / 1200 = 10047.92
```

After two months, the CD is worth

```
10047.91 + 10047.91 * 5.75 / 1200 = 10096.06
```

After three months, the CD is worth

```
10096.06 + 10096.06 * 5.75 / 1200 = 10145.44
```

and so on.

Write a program that prompts the user to enter an amount (e.g., 10,000), the annual percentage yield (e.g., 5.75), and the number of months (e.g., 4), and displays a table as shown in the sample run.

***5.34** (*Game: lottery)* Revise Listing 3.9, Lottery.py, to generate a lottery of a two-digit number. The two digits in the number are distinct. (Hint: Generate the first digit. Use a loop to continuously generate the second digit until it is different from the first digit.)

****5.35** (*Perfect number*) A positive integer is called a *perfect number* if it is equal to the sum of all of its positive divisors, excluding itself. For example, 6 is the first perfect number, because

$6 = 3 + 2 + 1$ The next is

$28 = 14 + 7 + 4 + 2 + 1$ There are four perfect numbers less than 10,000. Write a program to find these four numbers.

*****5.36** (*Game: scissor, rock, paper)* Programming Exercise 3.17 gives a program that plays the scissor, rock, paper game. Revise the program to let the user play continuously until either the user or the computer wins more than two times than its opponent.

***5.37** (*Summation)* Write a program that computes the following summation.

$$\frac{1}{1+\sqrt{2}} + \frac{1}{\sqrt{2}+\sqrt{3}} + \frac{1}{\sqrt{3}+\sqrt{4}} + \ldots + \frac{1}{\sqrt{624}+\sqrt{625}}$$

***5.38** (*Longest common prefix*) Write a program that prompts the user to enter two strings and displays the longest common prefix of the two strings. If the two strings have no common prefix, display `No common prefix`.

***5.39** (*Financial application: find the sales amount*) You have just started a sales job in a department store. Your pay consists of a base salary plus a commission. The base salary is $5,000. The following scheme shows how to determine the commission rate:

```
Sales Amount            Commission Rate
$0.01–$5,000            8 percent
$5,000.01–$10,000       10 percent
$10,000.01 and above    12 percent
```

Note that this is a graduated rate. The rate for the first $5,000 is at 8%, the next $5,000 is at 10%, and the rest is at 12%. If the sales amount is 25,000, the commission is 5,000 * 8% + 5,000 * 10% + 15,000 * 12% = 2,700. Your goal is to earn $30,000 a year. Write a program that finds the minimum sales you have to generate in order to make $30,000.

5.40 (*Simulation: heads or tails*) Write a program that simulates flipping a coin one million times and displays the number of heads and tails.

***5.41** (*Occurrence of max numbers*) Write a program that reads integers, finds the largest of them, and counts its occurrences. Assume that the input ends with number `0`. Suppose that you entered `3 5 2 5 5 5 0`; the program finds that the largest is `5` and the occurrence count for `5` is `4`. (Hint: Maintain two variables, `max` and `count`. The variable `max` stores the current max number, and `count` stores its occurrences. Initially, assign the first number to `max` and `1` to `count`. Compare each subsequent number with `max`. If the number is greater than `max`, assign it to `max` and reset `count` to `1`. If the number is equal to `max`, increment `count` by `1`.)

***5.42** (*Process string*) Write a program that prompts the user to enter a string and displays the characters at odd positions.

***5.43** (*Math: combinations*) Write a program that displays all possible combinations for picking two numbers from integers `1` to `7`. Also display the total number of combinations.

****5.44** (*Decimal to binary*) Write a program that prompts the user to enter a decimal integer and displays its corresponding binary value.

**5.45 (*Decimal to hex*) Write a program that prompts the user to enter a decimal integer and displays its corresponding hexadecimal value.

**5.46 (*Statistics: compute mean and standard deviation*) In business applications, you are often asked to compute the mean and standard deviation of data. The mean is simply the average of the numbers. The standard deviation is a statistic that tells you how tightly all the various data are clustered around the mean in a set of data. For example, what is the average age of the students in a class? How close are the ages? If all the students are the same age, the deviation is 0. Write a program that prompts the user to enter ten numbers, and displays the mean and standard deviations of these numbers using the following formula:

$$mean = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{x_1 + x_2 + \ldots + x_n}{n} \qquad deviation = \sqrt{\frac{\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}}{n-1}}$$

**5.47 (*Turtle: draw random balls*) Write a program that displays 10 random balls in a rectangle with width 120 and height 100, centered at (0, 0), as shown in Figure 5.3a.

Figure 5.3

The program draws 10 random balls in (a), and 10 circles in (b).

(Screenshots courtesy of Apple.)

**5.48 (*Turtle: draw circles*) Write a program that draws 10 circles centered at (0, 0), as shown in Figure 5.3b.

**5.49 (*Turtle: display a multiplication table*) Write a program that displays a multiplication table, as shown in Figure 5.4a.

Figure 5.4

(a) The program displays a multiplication table. (b) The program displays numbers in a triangular pattern. (c) The program displays an 18-by-18 lattice.

(Screenshots courtesy of Apple.)

**5.50 (*Turtle: display numbers in a triangular pattern*) Write a program that displays numbers in a triangular pattern, as shown in Figure 5.4b.

**5.51 (*Turtle: display a lattice*) Write a program that displays an 18-by-18 lattice, as shown in Figure 5.4c.

**5.52 (*Turtle: plot the sine function*) Write a program that plots the sine function, as shown in Figure 5.5a.

Figure 5.5

(a) The program plots a sine function. (b) The program plots sine function in blue and cosine function in red.

(Screenshots courtesy of Apple.)

HINT

The Unicode for π is `\u03c0`. To display -2π use `turtle.write("-2\u03c0")`. For a trigonometric function like `sin(x)`, `x` is in radians. Use the following loop to plot the sine function:

```
for x in range(-175, 176):
    turtle.goto(x, 50 * math.sin((x / 100) * 2 *
math.pi))
```

-2π is displayed at $(-100, -15)$ the center of the axis is at $(0, 0)$, and 2π is displayed at $(100, -15)$

**5.53 (*Turtle: plot the sine and cosine functions*) Write a program that plots the sine function in red and cosine in blue, as shown in Figure 5.5b.

**5.54 (*Turtle: plot the square function*) Write a program that draws a diagram for the function $f(x) = x^2$ (see Figure 5.6a).

Figure 5.6

(a) The program plots a diagram for function $f(x) = x^2$. (b) The program draws a chessboard.

(Screenshots courtesy of Apple.)

**5.55 (*Turtle: chessboard*) Write a program to draw a chessboard, as shown in Figure 5.6b.

*5.56 (*Count uppercase letters*) Write a program that prompts the user to enter a string and displays the number of the uppercase letters in the string.

*5.57 (*Business: check ISBN-13*) **ISBN-13** is a new standard for identifying books. It uses 13 digits:
$d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}d_{11}d_{12}d_{13}$. The last digit
d_{13} is a checksum, which is calculated from the other digits using the following formula:

$$10 - (d_1 + 3d_2 + d_3 + 3d_4 + d_5 + 3d_6 + d_7 + 3d_8 + d_9 + 3d_{10} + d_{11} + 3d_{12})\%10$$

If the checksum is `10`, replace it with `0`. Your program should read the input as a string.

*5.58 (*Reverse a string*) Write a program that prompts the user to enter a string and displays the string in reverse order.

*5.59 (*Count vowels and consonants*) Assume letters `A`, `E`, `I`, `O`, and `U` as the vowels. Write a program that prompts the user to enter a string and displays the number of vowels and consonants in the string.