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.

Note

A common error for the exercises in this chapter is that students don't implement the functions to meet the requirements even though the output from the main program is correct. For an example of this type of error see https://liangpy.pearsoncmg.com/supplement/CommonFunctionErrorPython.pdf.

Sections 6.2–6. 9

6.1 (*Math: pentagonal numbers*) A pentagonal number is defined as $n(3n-1)/2$ for $n = 1, 2, \ldots$, and so on. So, the first few numbers are 1, 5, 12, 22, … . Write a function with the following header that returns a pentagonal number:

```
def getPentagonalNumber(n):
```

For example, `getPentagonalNumber(1)` returns 1 and `getPentagonalNumber(2)` returns 5. Write a test program that uses this function to display the first 100 pentagonal numbers with 10 numbers on each line.

***6.2** (*Sum the digits in an integer*) Write a function that computes the sum of the digits in an integer. Use the following function header:

```
def sumDigits(n):
```

For example, `sumDigits(234)` returns `9` $(2 + 3 + 4)$. (Hint: Use the `%` operator to extract digits, and the `//` operator to remove the extracted digit. For instance, to extract `4` from `234`, use `234 % 10` $(= 4)$. To remove `4` from `234`, use `234 // 10` $(= 23)$. Use a loop to repeatedly extract and remove the digits until all the digits are extracted.) Write a test program that prompts the user to enter an integer and displays the sum of all its digits.

****6.3** (*Palindrome integer*) Write two functions with the following headers:

```
 # Return the reversal of an integer, e.g. reverse(456) returns
654
def reverse(number):

# Return true if number is a palindrome
def isPalindrome(number):
```

Use the `reverse` function to implement `isPalindrome`. A number is a palindrome if its reversal is the same as itself. Write a test program that prompts the user to enter an integer and reports whether the integer is a palindrome.

***6.4** (*Display an integer reversed*) Write a function with the following header to display an integer in reverse order:

```
def reverse(number):
```

For example, `reverse(3456)` displays `6543`. Write a test program that prompts the user to enter an integer and displays its reversal.

***6.5** (*Sort three numbers*) Write a function with the following header to display three numbers in increasing order:

```
def displaySortedNumbers(num1, num2, num3):
```

Write a test program that prompts the user to enter three numbers and invokes the function to display them in increasing order.

***6.6** (*Display patterns*) Write a function to display a pattern as follows:

```
                 1
               2 1
             3 2 1
...
n n-1 ... 3 2 1
```

The function header is

```
def displayPattern(n):
```

Write a test program that prompts the user to enter a number `n` and invokes `displayPattern(n)` to display the pattern.

***6.7** (*Financial application: compute the future investment value*) Write a function that computes a future investment value at a given interest rate for a specified number of years. The future investment is determined using the formula in Programming Exercise 2.19.
Use the following function header:

```
def futureInvestmentValue(
    investmentAmount, monthlyInterestRate, years):
```

For example, `futureInvestmentValue(10000, 0.05/12, 5)` returns `12833.59`.
Write a test program that prompts the user to enter the investment amount and the annual interest rate in percent and prints a table that displays the future value for the years from 1 to 30.

6.8 (Conversions between Celsius and Fahrenheit) Write a module that contains the following two functions:

```
# Convert from Celsius to Fahrenheit
def celsiusToFahrenheit(celsius):

# Convert from Fahrenheit to Celsius
def fahrenheitToCelsius(fahrenheit):
```

The formulas for the conversion are:

```
celsius = (5 / 9) * (fahrenheit – 32)
fahrenheit = (9 / 5) * celsius + 32
```

Write a test program that invokes these functions to display the following tables:

6.9 (Conversions between feet and meters) Write a module that contains the following two functions:

```
# Convert from feet to meters
def footToMeter(foot):

# Convert from meters to feet
def meterToFoot(meter):
```

The formulas for the conversion are:

```
foot = meter / 0.305
meter = 0.305 * foot
```

Write a test program that invokes these functions to display the following tables:

6.10 (*Use the* `isPrime` *Function*) Listing 6.7, PrimeNumberFunction.py, provides the `isPrime(number)` function for testing whether a number is prime. Use this function to find the number of prime numbers less than 10,000.

6.11 (*Financial application: compute commissions*) Write a function that computes the commission, using the scheme in Programming Exercise 5.39. The header of the function is as follows:

```
def computeCommission(salesAmount):
```

Write a test program that displays the following table:

```
Sales Amount      Commission
10000                  900.0
15000                 1500.0
...
95000                11100.0
100000               11700.0
```

6.12 (*Display characters*) Write a function that prints characters using the following header:

```
def printChars(ch1, ch2, numberPerLine):
```

This function prints the characters between `ch1` and `ch2` with the specified numbers per line. Write a test program that prints ten characters per line from `1` to `Z`. The characters are separated by exactly one space.

***6.13** (*Sum series*) Write a function to compute the following series:

$$m(i) = \frac{1}{2} + \frac{2}{3} + \ldots + \frac{i}{i+1}$$

Write a test program that displays the following table:

***6.14** (*Estimate*
π)
π can be computed using the following series:

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

Write a function that returns `m(i)` for a given `i` and write a test program that displays the following table:

***6.15** (*Financial application: print a tax table*) Listing 3.6, ComputeTax.py, gives a program to compute tax. Write a function for computing tax using the following header:

```
def computeTax(status, taxableIncome):
```

Use this function to write a program that prints a tax table for taxable income from $50,000 to $60,000 with intervals of $50 for all four statuses, as follows:

***6.16** (*Number of days in a year)* Write a function that returns the number of days in a year using the following header:

```
def numberOfDaysInAYear(year):
```

Write a test program that displays the year and the number of days in the year from `2010` to `2020` .

Sections 6.10– 6.11

***6.17** (*The `MyTriangle` module*) Create a module named `MyTriangle` that contains the following two functions:

```
# Return true if the sum of any two sides is
# greater than the third side.
def isValid(side1, side2, side3):

# Return the area of the triangle.
def area(side1, side2, side3):
```

Write a test program that reads three sides for a triangle and computes the area if the input is valid. Otherwise, it displays that the input is invalid. The formula for computing the area of a triangle is given in Programming Exercise 2.14.

***6.18** (*Display matrix of 0s and 1s*) Write a function that displays an *n* -by- *n* matrix using the following header:

```
def printMatrix(n):
```

Each element is 0 or 1, which is generated randomly. Write a test program that prompts the user to enter `n` and displays an *n*-by-*n* matrix.

***6.19** (*Geometry: point position*) Programming Exercise 3.31 shows how to test whether a point is on the left side of a directed line, on the right, or on the same line. Write the functions with the following headers:

```
# Return true if point (x2, y2) is on the left side of the
# directed line from (x0, y0) to (x1, y1)
def leftOfTheLine(x0, y0, x1, y1, x2, y2):

# Return true if point (x2, y2) is on the same
# line from (x0, y0) to (x1, y1)
def onTheSameLine(x0, y0, x1, y1, x2, y2):

# Return true if point (x2, y2) is on the
# line segment from (x0, y0) to (x1, y1)
def onTheLineSegment(x0, y0, x1, y1, x2, y2):
```

Write a program that prompts the user to enter the three points for p0, p1, and p2 and displays whether p2 is on the left of the line from p0 to p1, on the right, on the same line, or on the line segment.

***6.20** (*Geometry: display angles*) Rewrite Listing 4.2, ComputeAngles.py, using the following function for computing the distance between two points.

```
def distance(x1, y1, x2, y2):
```

****6.21** (*Math: approximate the square root*) There are several techniques for implementing the `sqrt` function in the `math` module. One such technique is known as the *Babylonian function*. It approximates the square root of a number, `n`, by repeatedly performing a calculation using the following formula:

```
nextGuess = (lastGuess + (n / lastGuess)) / 2
```

When `nextGuess` and `lastGuess` are almost identical, `nextGuess` is the approximated square root. The initial guess can be any positive value (e.g., `1`). This value will be the starting value for `lastGuess` . If the difference between `nextGuess` and `lastGuess` is less than a very small number, such as `0.0001` , you can claim that `nextGuess` is the approximated square root of `n` . If not, `nextGuess` becomes `lastGuess` and the approximation process continues. Implement the following function that returns the square root of `n` .

```
def sqrt(n):
```

Sections 6.12– 6.13

**6.22 (*Display current date and time*) Listing 2.7, ShowCurrentTime.py, displays the current time. Enhance this program to display the current date and time. (Hint: The calendar example in LiveExample 6.13, PrintCalendar.py, should give you some ideas on how to find the year, month, and day.)

**6.23 (*Convert milliseconds to hours, minutes, and seconds*) Write a function that converts milliseconds to hours, minutes, and seconds using the following header:

```
def convertMillis(millis):
```

The function returns a string as hours:minutes:seconds. For example, `convertMillis(5500)` returns the string 0:0:5, `convertMillis(100000)` returns the string 0:1:40, and `convertMillis(555550000)` returns the string 154:19:10.
Write a test program that prompts the user to enter a value for milliseconds and displays a string in the format of hours:minutes:seconds.

**6.24 (*Palindromic prime*) A *palindromic prime* is a prime number that is also palindromic. For example, `131` is a prime and also a palindromic prime, as are `313` and `757` . Write a program that displays the first 100 palindromic prime numbers. Display 10 numbers per line and align the numbers properly, as follows:

```
 2     3     5     7    11   101   131   151   181   191
313   353   373   383   727   757   787   797   919   929
..
```

**6.25 (*Emirp*) An *emirp* (*prime* spelled backward) is a nonpalindromic prime number whose reversal is also a prime. For example, both `17` and `71` are prime numbers, so `17` and `71` are emirps. Write a program that displays the first 100 emirps. Display 10 numbers per line and align the numbers properly, as follows:

```
 13   17   31   37   71   73   79   97  107  113
149  157  167  179  199  311  337  347  359  389
...
```

**6.26 (*Mersenne prime*) A prime number is called a *Mersenne prime* if it can be written in the form
$2^p - 1$ for some positive integer p. Write a program that finds all Mersenne primes with
$p \leq 10$ and displays the output as follows:

```
    p        2^p - 1
    2           3
    3           7
    5          31
...
```

**6.27 (*Twin primes*) Twin primes are a pair of prime numbers that differ by 2. For example, 3 and 5, 5 and 7, and 11 and 13 are twin primes. Write a program to find all twin primes less than 1,000. Display the output as follows:

```
  (3, 5)
 (5, 7)...
```

**6.28 (*Game: craps*) Craps is a popular dice game played in casinos. Write a program to play a variation of the game, as follows:

Roll two dice. Each die has six faces representing values `1`, `2`, ..., and `6`, respectively. Check the sum of the two dice. If the sum is `2`, `3`, or `12` (called *craps*), you lose; if the sum is `7` or `11` (called *natural*), you win; if the sum is another value (i.e., `4`, `5`, `6`, `8`, `9`, or `10`), a *point* is established. Continue to roll the dice until either a `7` or the same point value is rolled. If `7` is rolled, you lose. Otherwise, you win.

Your program acts as a single player.

**6.29 (*Financial: credit card number validation*) Credit card numbers follow certain patterns: It must have between 13 and 16 digits, and the number must start with:

- 4 for Visa cards
- 5 for MasterCard credit cards
- 37 for American Express cards
- 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the *Luhn check* or the *Mod 10 check*, which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.

2. Now add all single-digit numbers from Step 1.

$$4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37$$

3. Add all digits in the odd places from right to left in the card number.

$$6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38$$

4. Sum the results from Steps 2 and 3.

$$37 + 38 = 75$$

5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number `4388576018402626` is invalid, but the number `4388576018410707` is valid.

Write a program that prompts the user to enter a credit card number as an integer. Display whether the number is valid or invalid. Design your program to use the following functions:

```
 # Return true if the card number is valid
def isValid(number):

# Get the result from Step 2
def sumOfDoubleEvenPlace(number):

# Return this number if it is a single digit, otherwise, return
# the sum of the two digits
def getDigit(number):

# Return sum of odd place digits in number
def sumOfOddPlace(number):

# Return true if the digit d is a prefix for number
def prefixMatched(number, d):

# Return the number of digits in d
def getSize(d):

# Return the first k number of digits from number. If the
# number of digits in number is less than k, return number.
def getPrefix(number, k):
```

****6.30** (*Game: chance of winning at craps*) Revise Programming Exercise 6.28 to run it `10,000` times and display the number of winning games.

*****6.31** (*Current date and time*) Invoking `time.time()` returns the elapsed time in seconds since midnight of January 1, 1970. Write a program that displays the date and time.

****6.32** (*Print calendar*) Programming Exercise 3.21 uses Zeller's congruence to calculate the day of the week. Simplify LiveExample 6.13, PrintCalendar.py, using Zeller's algorithm to get the start day of the month.

6.33 (Geometry: area of a pentagon) Rewrite Programming Exercise 4.4 using the following function to return the area of a pentagon:

```
def area(s):
```

***6.34** (*Geometry: area of a regular polygon*) Rewrite Programming Exercise 4.5 using the following function to return the area of a regular polygon:

```
def area(n, side):
```

***6.35** (*Compute the probability*) Use the functions in `RandomCharacter` in Listing 6.10 to generate 10,000 uppercase letters and count the occurrence of letter `A`.

***6.36** (*Generate random characters*) Use the functions in `RandomCharacter` in Listing 6.10 to print 100 uppercase letters and then 100 single digits, printing ten per line.

Section 6.14

***6.37** (*Turtle: generate random characters*) Use the functions in `RandomCharacter` in Listing 6.10 to display 100 lowercase letters, fifteen per line, as shown in Figure 6.12a.

Figure 6.12

(a) The program displays random lowercase letters. (b) The program draws a star. (c) The program draws random points in a rectangle and in a circle.

(Screenshot courtesy of Apple.)

****6.38** (*Turtle: draw a line*) Write a function with the following header that draws a line from point (`x1` , `y1`) to (`x2` , `y2`) with color (default to black) and line size (default to 1).

```
def drawLine(x1, y1, x2, y2, color = "black", size = 1):
```

****6.39** (*Turtle: draw a star*) Write a program that draws a star, as shown in Figure 6.12b. Use the `drawLine` function defined in Programming Exercise 6.38.

****6.40** (*Turtle: filled rectangle and circle*) Write functions with the following headers that fill a rectangle with the specified color, center, width, and height, and a circle with the specified color, center, and radius.

```
 # Fill a rectangle
def drawRectangle(color = "black",
    x = 0, y = 0, width = 30, height = 30):

# Fill a circle
def drawCircle(color = "black", x = 0, y = 0, radius = 50):
```

****6.41** (*Turtle: draw points, rectangles, and circles*) Use the functions defined in Listing 6.14 to write a program that displays a rectangle centered at $(-75, 0)$ with width and height 100 and a circle centered at (50, 0) with radius 50. Fill 10 random points inside the rectangle and 10 inside the circle, as shown in Figure 6.12c.

**6.42 (*Turtle: plot the sine function*) Simplify the code for Programming Exercise 5.52 by using the functions in Listing 6.14.

**6.43 (*Turtle: plot the sine and cosine functions*) Simplify the code for Programming Exercise 5.53 by using the functions in Listing 6.14.

**6.44 (*Turtle: plot the square function*) Simplify the code for Programming Exercise 5.54 by using the functions in Listing 6.14.

**6.45 (*Turtle: draw a regular polygon*) Write a function with the following header to draw a regular polygon:

```
def drawPolygon(x = 0, y = 0, radius = 50, numberOfSides = 3):
```

The polygon is centered at (x, y) with a specified radius for the bounding circle for the polygon and the number of sides. Write a test program that displays a triangle, square, pentagon, hexagon, heptagon, and octagon, as shown in Figure 6.13a.

Figure 6.13

(a) The program displays several n-sided polygons. (b) The program displays a hexagon with all points connected.

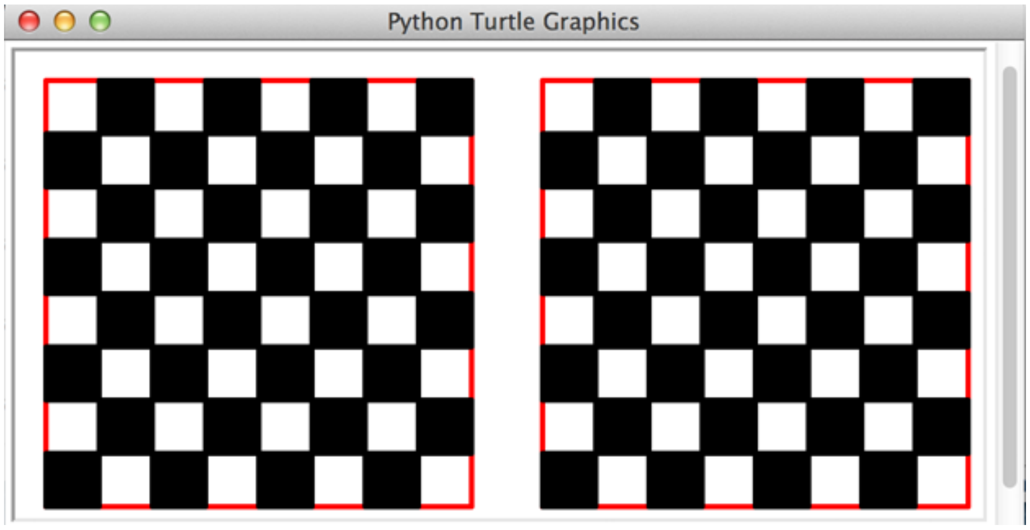
(Screenshots courtesy of Apple.)

*6.46 (*Turtle: connect all points in a hexagon*) Write a program that displays a hexagon with all the points connected, as shown in Figure 6.13b.

*6.47 (*Turtle: two chessboards*) Write a program that displays two chessboards, as shown in Figure 6.14. Your program should define at least the following function:

```
# Draw one chessboard whose upper-left corner is at
# (startx, starty) and bottom-right corner is at (endx, endy)
def drawChessboard(startx, endx, starty, endy):
```

Figure 6.14



The program draws two chessboards.

(Screenshot courtesy of Apple.)

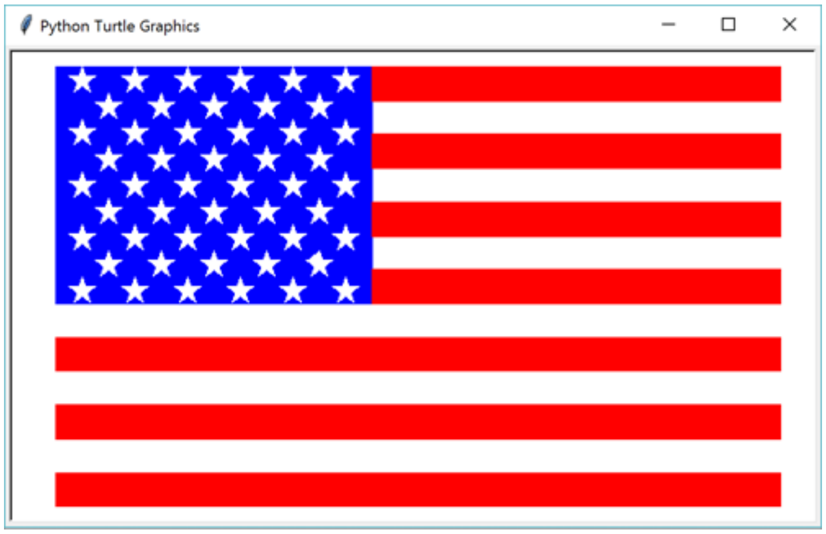
*__6.48__ (*Format an integer*) Write a function with the following header to format the integer with the specified width.

```
def format(number, width):
```

The function returns a string for the number with prefix `0` s. The size of the string is the width. For example, `format(34, 4)` returns `"0034"` and `format(34, 5)` returns `"00034"` . If the number is longer than the width, the function returns the string representation for the number. For example, `format(34, 1)` returns `"34"` . Write a test program that prompts the user to enter a number and its width and displays a string returned from invoking `format(number, width)` .

***__6.49__ (*US flag*) Write a program that draws a U.S. flag, as shown in Figure 6.15. Hint: Write a function that draws a filled star and use this function to draw 50 stars.

Figure 6.15



The program draws a U.S. flag.

(Screenshot courtesy of Microsoft Corporation.)

****6.50** (*Decimal to hex*) Write a function that parses a decimal number into a hex number as a string. The function headers are:

```python
def dec2Hex(value):
```

See Appendix C, "Number Systems," for converting a decimal into a hex. Write a test program that prompts the user to enter a decimal number and displays its equivalent hex value.

****6.51** (*Check substrings*) You can check whether a string is a substring of another string by using the `find` method in the `str` class. Write the following function to check whether string `s1` is a substring of string `s2`. The function returns the first index in `s2` if there is a match. Otherwise, return `-1`.

```
def indexOf(s1, s2):
```

Write a test program that reads two strings and checks whether the first string is a substring of the second string.

***6.52** (*Longest common prefix*) Write the `prefix` function using the following function header to return the longest common prefix between two strings:

```
def prefix(s1, s2):
```

Write a test program that prompts the user to enter two strings and displays their longest common prefix.

***6.53** (*Occurrences of a specified character*) Write a function that finds the number of occurrences of a specified character in the string using the following header:

```
def count(s, ch):
```

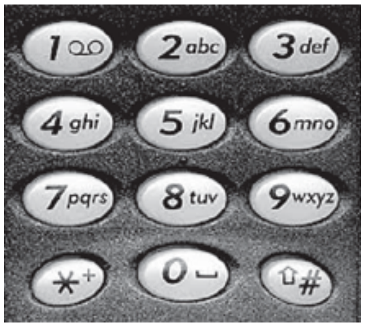
For example, `count("Welcome", 'e')` returns `2`. The `str` class has the `count` method. Implement your method without using the `count` method. Write a test program that reads a string and a character and displays the number of occurrences of the character in the string.

***6.54** (*Count the letters in a string*) Write a function that counts the number of letters in a string using the following header:

```python
def countLetters(s):
```

Write a test program that prompts the user to enter a string and displays the number of letters in the string.

***6.55** (*Phone keypads*) The international standard letter/number mapping for telephones is:



Write a function that returns a number, given an uppercase letter, as follows:

```python
def getNumber(uppercaseLetter):
```

Write a test program that prompts the user to enter a phone number as a string. The input number may contain letters. The program translates a letter (uppercase or lowercase) to a digit and leaves all other characters intact.