

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

If the program prompts the user to enter a list of values, enter the values on one line separated by spaces.

Sections 7.2–7.3

***7.1** (*Assign grades*) Write a program that reads a list of scores and then assigns grades based on the following scheme:

The grade is A if score is $\geq \text{best} - 10$.

The grade is B if score is $\geq \text{best} - 20$.

The grade is C if score is $\geq \text{best} - 30$.




The grade is D if score is $\geq \text{best} - 40$.

The grade is F otherwise.

7.2 (*Reverse the numbers entered*) Write a program that reads a list of integers and displays them in the reverse order in which they were read.

****7.3** (*Count occurrence of numbers*) Write a program that reads some integers between 1 and 100 in one line and counts the occurrences of each.

Note that if a number occurs more than one time, the plural word “times” is used in the output.


- 7.4** (*Analyze scores*) Write a program that reads an unspecified number of scores and determines how many scores are above or equal to the average and how many scores are below the average. Assume the input numbers are separated by one space in one line.
- **7.5** (*Print distinct numbers*) Write a program that reads in integers separated by a space in one line and displays distinct numbers in their input order and separated by exactly one space (i.e., if a number appears multiple times, it is displayed only once). (Hint: Read all the numbers and store them in `list1`. Create a new list `list2`. Add a number in `list1` to `list2`. If the number is already in the list, ignore it.)
- *7.6** (*Revise Listing 5.14* , *PrimeNumber.py*) Listing 5.14  determines whether a number `n` is prime by checking whether `2`, `3`, `4`, `5`, `6`, ..., `n/2` is a divisor for `n`. If a divisor is found, `n` is not prime. A more efficient approach is to check whether any of the prime numbers less than or equal to \sqrt{n} can divide `n` evenly. If not, `n` is prime. Rewrite Listing 5.14  to display the first 50 prime numbers using this approach. You need to use a list to store the prime numbers and later use them to check whether they are possible divisors for `n`.
- *7.7** (*Count single digits*) Write a program that generates 1,000 random integers between 0 and 9 and displays the count for each number. (Hint: Use a list of ten integers, say `counts`, to store the counts for the number of 0s, 1s, ..., 9s.)

Sections 7.4–7.7

- 7.8** (*Find the index of the smallest element*) Write a function that returns the index of the smallest element in a list of integers. If the number of such elements is greater than 1, return the smallest index. Use the following header:

```
def indexOfSmallestElement(lst):
```

Write a test program that prompts the user to enter a list of numbers in one line, invokes this function to return the index of the smallest element, and displays the index.

- *7.9** (*Statistics: compute deviation*) **Programming Exercise 5.46**  computes the standard deviation of numbers. This exercise uses a different but equivalent formula to compute the standard deviation of n numbers.

$$mean = \frac{\sum_{i=1}^n x_i}{n} = \frac{x_1 + x_2 + \dots + x_n}{n} \quad deviation = \sqrt{\frac{\sum_{i=1}^n (x_i - mean)^2}{n - 1}}$$


To compute the standard deviation with this formula, you have to store the individual numbers using a list so that they can be used after the mean is obtained.

Your program should contain the following functions:

```
# Compute the standard deviation of values
def deviation(x):

# Compute the mean of a list of values
def mean(x):
```

Write a test program that prompts the user to enter a list of numbers in one line and displays the mean and standard deviation, as shown in the following sample run:

- *7.10** (*Reverse a list*) The `reverse` function in **Section 7.8**  reverses a list by copying it to a new list. Rewrite the function that reverses the list passed in the argument and returns

this list. Write a test program that prompts the user to enter a list of numbers in one line, invokes the function to reverse the numbers, and displays the numbers.

Section 7.8

- *7.11** (*Random number chooser*) You can shuffle a list using `random.shuffle(lst)`. Write your own function without using `random.shuffle(lst)` to shuffle a list and return the list. Use the following function header:

```
def shuffle(lst):
```

Write a test program that prompts the user to enter a list of numbers in one line, invokes the function to shuffle the numbers, and displays the numbers.

- 7.12** (*Compute gcd*) Write a function that returns the greatest common divisor (gcd) of integers in a list. Use the following function header:

```
def gcd(numbers):
```

Write a test program that prompts the user to enter a list of numbers in one line, invokes the function to find the gcd of these numbers, and displays the gcd.

Sections 7.9– 7.11

- 7.13** (*Eliminate duplicates*) Write a function that returns a new list by eliminating the duplicate values in the list. Use the following function header:

```
def eliminateDuplicates(lst):
```

Write a test program that reads in a list of integers in one line, invokes the function, and displays the result.

- *7.14** (*Revise selection sort*) In [Section 7.10](#), you used selection sort to sort a list. The selection-sort function repeatedly finds the smallest number in the current list and swaps it with the first one. Rewrite this program by finding the largest number and swapping it with the last one. Write a test program that reads in numbers entered in one line, invokes the function, and displays the sorted numbers.
- **7.15** (*Sorted?*) Write the following function that returns true if the list is already sorted in increasing order:

```
def isSorted(lst):
```

Write a test program that prompts the user to enter a list of numbers in one line and displays whether the list is sorted or not.

- **7.16** (*Bubble sort*) Write a sort function that uses the bubble-sort algorithm. The bubble-sort algorithm makes several passes through the list. On each pass, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged. The technique is called a *bubble sort* or *sinking sort* because the smaller values gradually “bubble” their way to the top and the larger values “sink” to the bottom. Write a test program that reads in numbers entered in one line, invokes the function, and displays the sorted numbers.
- **7.17** (*Anagrams*) Write a function that checks whether two words are anagrams. Two words are anagrams if they contain the same letters. For example, *silent* and *listen* are anagrams. The header of the function is:

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

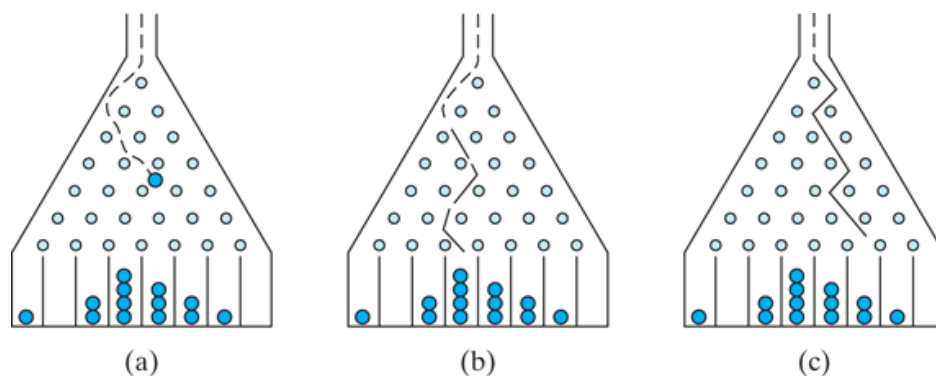
(Hint: Obtain two lists for the two strings. Sort the lists and check if two lists are identical.)

Write a test program that prompts the user to enter two strings and checks whether they are anagrams.

***7.18 (Game: *Eight Queens*) The classic Eight Queens puzzle is to place eight queens on a chessboard such that no two queens can attack each other (i.e., no two queens are in the same row, same column, or same diagonal). There are many possible solutions. Write a program that displays one such solution.

***7.19 (Game: *bean machine*) The bean machine, also known as a quincunx or the Galton box, is a device for statistics experiments named after English scientist Sir Francis Galton. It consists of an upright board with evenly spaced nails (or pegs) in a triangular pattern, as shown in Figure 7.9.

Figure 7.9



Each ball takes a random path and falls into a slot.

Balls are dropped from the opening of the board. Every time a ball hits a nail, it has a 50% chance of falling to the left or to the right. The piles of balls are accumulated in the slots at the bottom of the board.

Write a program that simulates the bean machine. Your program should prompt the user to enter the number of the balls and the number of the slots in the machine. Simulate the falling of each ball by printing its path. For example, the path for the ball in Figure 7.9b is LLRLLLR and the path for the ball in Figure 7.9c is RLRRLRR. Display the final buildup of the balls in the slots in a histogram.

(Hint: Create a list named `slots`. Each element in `slots` stores the number of balls in a slot. Each ball falls into a slot via a path. The number of Rs in a path is the position of the slot where the ball falls. For example, for the path LRLRLRR, the ball falls into `slots[4]`, and for the path RRLLLLL, the ball falls into `slots[2]`.)

*****7.20** (Game: multiple Eight Queens solutions) Programming Exercise 7.18 asks you to find one solution for the Eight Queens puzzle. Write a program to count all possible solutions for the Eight Queens problem and display all the solutions.

****7.21** (Game: locker puzzle) A school has 100 lockers and 100 students. All lockers are closed on the first day of school. As the students enter, the first student, denoted S1, opens every locker. Then the second student, S2, begins with the second locker, denoted L2, and closes every other locker. Student S3 begins with the third locker and changes every third locker (closes it if it was open, and opens it if it was closed). Student S4 begins with locker L4 and changes every fourth locker. Student S5 starts with L5 and changes every fifth locker, and so on, until student S100 changes L100. After all the students have passed through the building and changed the lockers, which lockers are open? Write a program to find your answer and display all open locker numbers separated by exactly one space.

(Hint: Use a list of 100 Boolean elements, each of which indicates whether a locker is open (`True`) or closed (`False`). Initially, all lockers are closed.)

****7.22** (Simulation: coupon collector's problem) Coupon Collector is a classic statistics problem with many practical applications. The problem is to pick objects from a set of objects repeatedly and find out how many picks are needed for all the objects to be picked at least once. A variation of the problem is to pick cards from a shuffled deck of 52 cards repeatedly and find out how many picks are needed before you see one of each suit. Assume a picked card is placed back in the deck before picking another. Write a program to simulate the number of picks needed to get four cards, one from each suit and display the four cards picked (it is possible a card may be picked twice).

- 7.23** (*Algebra: solve quadratic equations*) Write a function for solving a quadratic equation using the following header:

```
def solveQuadratic(eqn, roots):
```

The coefficients of a quadratic equation $ax^2 + bx + c = 0$ are passed to the list `eqn` and the noncomplex roots are stored in `roots`. The function returns the number of roots. See [Programming Exercise 3.1](#) on how to solve a quadratic equation.

Write a program that prompts the user to enter values for a , b , and c and displays the number of roots and all noncomplex roots.

- *7.24** (*Math: combinations*) Write a program that prompts the user to enter 10 integers and displays all the combinations of picking two numbers from the 10.
- *7.25** (*Game: pick four cards*) Write a program that picks four cards from a deck of 52 cards and computes their sum. An ace, king, queen, and jack represent `1`, `13`, `12`, and `11`, respectively. Your program should display the number of picks that yield the sum of `24`.
- **7.26** (*Merge two sorted lists*) Write the following function that merges two sorted lists into a new sorted list:

```
def merge(list1, list2):
```

Implement the function in a way that takes `len(list1) + len(list2)` comparisons. Write a test program that prompts the user to enter two sorted lists and displays the merged list.

Here is an animation to show how to merge two sorted lists.

- *7.27** (*Pattern recognition: four consecutive equal numbers*) Write the following function that tests whether the list has four consecutive numbers with the same value:

```
def isConsecutiveFour(values):
```

Write a test program that prompts the user to enter a series of integers in one line and reports whether the series contains four consecutive numbers with the same value.

- **7.28** (*Partition of a list*) Write the following function that partitions the list using the first element, called a *pivot*:

```
def partition(lst):
```

After the partition, the elements in the list are rearranged so that all the elements before the pivot are less than or equal to the pivot and the element after the pivot are greater than the pivot. The function also returns the index where the pivot is located in the new list. For example, suppose the list is [5 , 2 , 9 , 3 , 6 , 8]. After the partition, the list becomes [3 , 2 , 5 , 9 , 6 , 8]. Implement the function in a way that takes `len(lst)` comparisons. Write a test program that prompts the user to enter a list in one line and displays the list after the partition.

Here is an animation to show how to partition a list.

- ***7.29** (*Game: hangman*) Write a hangman game that randomly generates a word and prompts the user to guess one letter at a time, as shown in the sample run. Each letter in the word is displayed as an asterisk. When the user makes a correct guess, the actual letter is then displayed. When the user finishes a word, display the number of misses and ask the user whether to continue playing. Create a list to store the words, as follows:

```
# Use any words you wish
words = ["write", "that", "program", ...]
```

***7.30** (*Culture: Chinese Zodiac*) Simplify Listing 3.4, ChineseZodiac.py, using a list of strings to store the animals' names.

7.31 (*Occurrences of each digit in a string*) Write a function that counts the occurrences of each digit in a string using the following header:

```
def count(s):
```

The function counts how many times a digit appears in the string. The return value is a list of ten elements, each of which holds the count for a digit. For example, after executing `counts = count("12203AB3")`, `counts[0]` is 1, `counts[1]` is 1, `counts[2]` is 2, and `counts[3]` is 2.

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

****7.32** (*Sort characters in a string*) Write a function that returns a sorted string using the following header:

```
def sort(s):
```

For example, `sort("acb")` returns `abc`. Write a test program that prompts the user to enter a string and displays the sorted string.

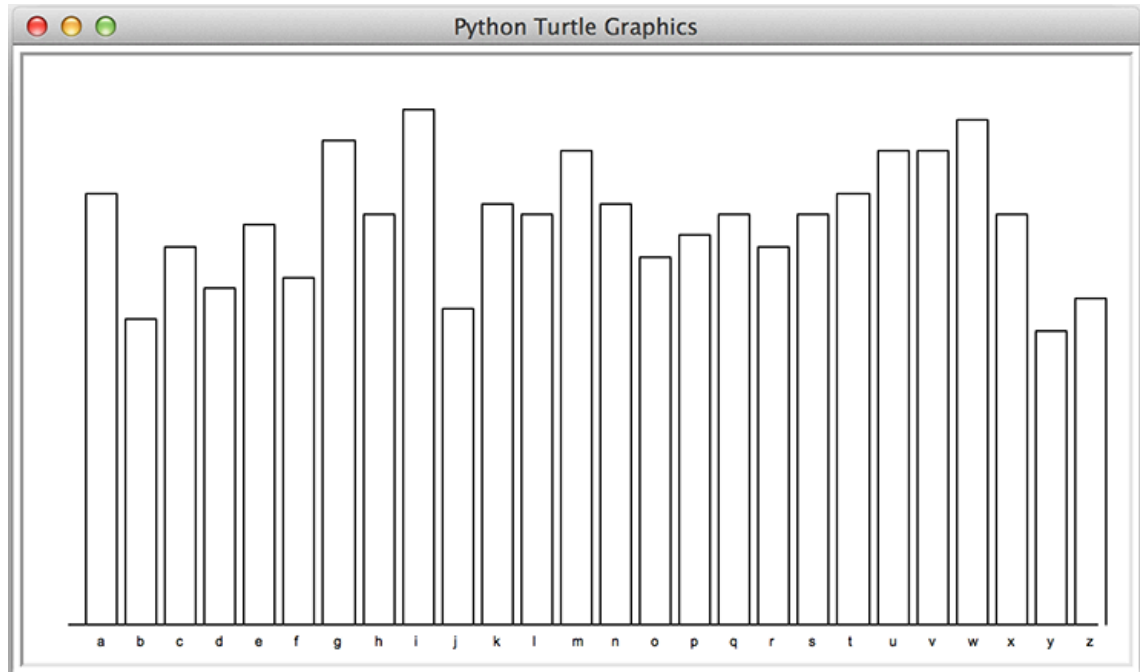
- *7.33** (*Area of a convex polygon*) A polygon is convex if it contains any line segments that connects two points of the polygon. Write a program that prompts the user to enter the points counter-clockwise in one line, and displays the area of the polygon. For the formula for computing the area of a polygon, see

http://www.mathwords.com/a/area_convex_polygon.htm.

- 7.34** (*Turtle: draw a line*) Write the following function that draws a line from point p1 ([x1, y1]) to point p2 ([x2, y2]).

```
# Draw a line
def drawLine(p1, p2):
```

- 7.35** (*Turtle: draw histograms*) Write a program that generates 1,000 lowercase letters randomly, counts the occurrence of each letter, and displays a histogram for the occurrences, as shown in [Figure 7.10](#).

Figure 7.10

A histogram is drawn for the count of each letter.

(Screenshot courtesy of Apple.)

7.36 (*Subtraction quiz*) Rewrite Listing 5.1  RepeatSubtractionQuiz.py to alert the user if an answer is entered again. *Hint*: Use a list to store answers.

****7.37** (*Algebra: perfect square*) Write a program that prompts the user to enter an integer m and find the smallest integer n such that $m * n$ is a perfect square. (*Hint*: Store all smallest factors of m into a list. n is the product of the factors that appear an odd number of times in the list. For example, consider $m = 90$, store the factors 2, 3, 3, 5 in a list. 2 and 5 appear an odd number of times in the vector. So, n is 10.)