STUDENT SOLUTIONS MANUAL

to accompany

An Introduction to Programming Using Python

by David I. Schneider

CONTENTS

Comments

Chapter 1

Section 1.1: The most important questions in Section 1.1 are as follows:

> ***Question:*** *Many programming languages, including Python, use a zero-based numbering system. What is a zero- based numbering system?* (See page 3.)
>
> ***Question:*** *What conventions are used to show keystrokes?* (See page 4.)
>
> ***Question:*** *How can the programs for the examples in this textbook be obtained?* (See page 4.)
>
> ***Question:*** *Where will new programs be saved?* (See page 4.)

Section 1.3: Flowcharts appear primarily in Chapter 3. At that time, you may want to review the material on flowcharts on page 7.

Section 1.4

Figure 1.30 on page 20 illustrates a most important aspect of Python programs; the fact that indentation and blocks are heavily used to delineate different parts of programs.

Of the items in the menu in Fig. 1.31 on page 20, I rely heavily on the *Recent Files* command. I use it to get to the folder I am currently using to store and retrieve my programs. I often open any program in or near the proper folder. Then the *Open* or *New File* commands will default to that folder.

On page 34 you are asked to run a program that was downloaded from the Pearson website for the book. All of the programs appearing in Examples can be downloaded from that website. There is never any need for you to manually type in the code for a program in this textbook. The website also contains all the data files needed for the exercises. These programs and data files are contained in the subfolders "Ch2", "Ch3", "Ch4", and so on. Each program has a name of the form *chapte number-section number–example number*. For instance, the program in Chapter 3, Section 1, Example 2 is contained in the folder Ch3 and has the name **`3-1-2.py`**.

Chapter 2

Section 2.1

The first example containing a program is Example 1 on page 24. To open this program in Python, click on *Open* in the IDLE *File* menu, navigate to the subfolder *Ch2* of the *Programs* folder, and double-click on **`2-1-1.py`**. The next time you click on *Open* in the IDLE File menu, Python will use the folder *Programs*/*Ch2* as the default folder. That is, you will not have to navigate to that folder; its programs will automatically appear.

A number whose absolute value is less than 1 can be written with or without a 0 preceding its decimal point. Both representations are treated the same. That is, 0.8 and .8 are the same number. Python displays such numbers with a 0 preceding the decimal point.

Mathematical comment: x ** .5 has the same value as $\sqrt{x}$.

Comment 4 shows an unexpected feature of the round function. Most people expect the number 2.5 to be rounded to 3, not 2.

A common error is to forget to include the closing parenthesis at the end of a print function.

Section 2.2

We favor using double-quotes to delimit strings since strings sometimes contain contractions; that is, words such as *it's* and *can't*.

Correction: On page 41, the value of `int(-4.8)` should be -4.

Section 2.3

Mastering the format method (discussed on pages 52 and 53) is a bit challenging. However, once you have become comfortable using it, you will appreciate its power. It is the modern way of formatting output will be the preferred method of formatting output in this book.

The ljust, rjust, and center methods discussed on page 51 are much less important than the format method. They are presented to introduce the concept of a *field*.

Section 2.4

The material in the first three sections of this chapter is fundamental and usually is introduced early in computer programming books. I made the atypical decision to introduce lists and tuples earlier (in Section 2.4) than other Python textbooks and also to show how to fill them with the contents of a file. So doing allows us to solve interesting and non-trivial programs early in the book. For now, just think of the three-line method for filling a list with the contents of a text file as boilerplate. Later you will understand exactly how the three-line method carries out its mission. With many Python/Operating System combinations, the three lines can be replaced with a single line of the following form:

```
listName = [line.rstrip() for line in open("Data.txt", 'r')]
```

We have learned four ways to fill a list. Let's illustrate the ways by filling a list with the names of the first three presidents.

```
(a) pres = ["Washington", "Jefferson", "Adams"]
```

```
(b) infile = open("Pres.txt", 'r')
    pres = [line.rstrip() for line in infile]
    infile.close
```

where the text file `Pres.txt` is located in the same folder as the program and has three lines with each line containing the name of a president.

```
(c) strPres = "Washington,Jefferson,Adams"
    pres = strPres.split(",")
```

```
(d) pres = []
    pres.append("Washington")
    pres.append("Jefferson")
    pres.append("Adams")
```

Lists are incredibly useful structures and are used extensively throughout this book. The join method (as used in Example 3) provides an excellent way to display the items in a list.

The main idea to take away from the discussion of mutable objects is that numbers, strings, and tuples are handled differently in memory than are lists. This fact will rarely affect the programs that you write. However, it will give you some insights into what is going on in the background. In Chapters 4 and 5, the concept of mutability becomes important. Mutable objects can have their values changed when passed to functions. Also, mutable objects cannot be used as keys in dictionaries.

Chapter 3

Section 3.1

This section is unusual in that you are not asked to write any programs in the exercise set. However, the concepts presented in this section are fundamental to the rest of the textbook.

The sort method sorts the items of a list in ascending order. The sort method followed by the reverse method will sort the items in descending order. In Section 4.1, we will learn how to sort the items by many other different criteria. For instance, we will be able to sort a list of words by their length or by the number of vowels they contain.

You can sort a list of numbers even if some are *ints* and some are *floats*. However you cannot sort a list when some items are numbers and some are strings.

Section 3.2

Python is said to have "block-structured syntax" due to its use of consistent indentation to delimit blocks of code, such as the indented block of code following an if statement. This book indents blocks of statements by four spaces. Most Python programmers use four spaces. However, the use of any number of spaces of indentation to delimit a block is recognized by Python as long as each line of the block uses the same number of spaces of indentation.

There are two ways to obtain the largest of a collection of numbers. One way is to place the numbers into a list and then use the max functions to obtain the largest value. Another way is to use the procedure employed in Example 3. That procedure, a fundamental algorithm in computer science, is useful in many situations, such as situations where the numbers cannot practically be placed in a list. For instance, the algorithm can be used to find the largest value in a very large text file. The algorithm is as follows:

(a) Declare a variable named *max.*
(b) Assign a value to the variable that is guaranteed to be less than or equal to the maximum value in the collection of numbers. One possibility is to set the value of *max* equal to the first number in the collection. Another possibility requires some knowledge of the numbers in the list. For instance, if the numbers are all grades on an exam, the initial value of *max* can be set to 0.
(c) Examine the numbers one at a time. Whenever a number is greater than the current value of *max*, change the value of *max* to be that number. After all numbers have been considered, the value of *max* will be the largest number in the collection.

Correction: In Exercise 22, the last two lines should be indented.

Section 3.3

The while loop is the first of two repetition structures that appear in this textbook. The number of repetitions (called passes) through a while loop is usually not known in advance. With the second repetition structure, the for loop studied in the next section, the number of passes is frequently known in advance. The while loop is often referred to as an *event-controlled loop*.

Loops with the header `while True` are referred to as *posttest* loops since the terminating condition appears at the end of the indented block. Loops with the header `while condition` are referred to as *pretest* loops.

Section 3.4

The *for* loop is often referred to as a *count-controlled loop* since the number of passes through the loop can be determined by counting the number of items in the sequence it iterates through. Both the for loop and the while loop are invaluable programming tools.

Correction: In the first printing of the book, the outputs appearing in Example 2 on page 120 and Example 3 on page 121 should have more space between the two columns.

Chapter 4

Section 4.1

The extensive use of user-defined functions is part of good programming practice, but is not essential. Any computer program can be written without such functions. However, there are many programs in this textbook that would be difficult to write and read without user-defined functions.

Function names are case sensitive and often long. You can avoid mistyping the names by using copy and paste each time you reference the function.

The return statement returns the value of the function and also exits the function.

Section 4.2

This section discusses Python's different ways of passing arguments to parameters. Most of the time we will use passing by position. Default values will be heavily used in Section 6.3 (Turtle Graphics) and Chapter 7 (Object-Oriented Programming).

Custom sorting will be used extensively in Chapter 5 (Processing Data). Historically, computer programming courses devoted considerable time to teaching complicated and difficult-to understand sorting algorithms. Python's powerful sorting capabilities spare us from having to incorporate tedious algorithms into our programs.

Section 4.3

This section discusses the importance of breaking problems down into smaller problems. There are no exercises in Section 4.3.

Chapter 5

As you have already seen, this book makes extensive use of text files to obtain input. This chapter is primarily about files and contains some large text files. Two large text files we have already seen (and will be used in Section 5.1) are as follows:

(a) `States.txt`: contains the names of the fifty states in the order they joined the union.

(b) `USPres.txt`: contains the names of the first 44 U.S. presidents in the order in which they served.

Text files are either created with a text editor or created with a Python program. In both instances, the last line of the file may or may not end with a newline character. This fact often requires us to add a few lines of code to programs in order to make them execute properly in both cases.

Section 5.1

File names are not case-sensitive. Therefore the file `USPres.txt` also can be referred to as `USpres.txt` or `USPRES.TXT`.

In this section we learn how to write programs that create new text files or alter and combine existing text files.

The data-type *set* is a handy tool for working with text files. It is introduced in this section and used to carry out tasks such as removing duplicates from a text file and merging two files in several different ways.

When you display the contents of a set with the print function, there is no way to predict the order in which the elements will appear. For instance, when you run the program in Example 6, you will get the same sets of elements shown in the textbook. However, the order of the elements in some of the sets might differ from that in the textbook.

Section 5.2

Until now, each line of a text file contained just one piece of data. However, real-life data processing requires text files with each line containing several pieces of related data. Some of the text files introduced in this chapter are as follows:

(a) **`Justices.txt`**: contains information about the 112 people who have been appointed to the U.S. Supreme Court prior to January 2015.

(b) **`UN.txt`**: contains information about the 193 member countries of the United Nations.

(c) **`DOW.txt`**: contains information about the 30 stocks in the Dow Jones Industrial Average.

(d) **`StatesANC.txt`**: contains the name, abbreviation, nickname, and capital of each of the 50 states in the USA.

(e) **`Senate113.txt`** and **`Senate114.txt`**: contains the name, state, and party affiliation of each of the 100 senators of the 113th and 114th U.S. Senates.

Correction to answer for Exercise 15: The second line of the *displayOutput* function should be

```
print("The gifts for day", day, "are")
```

Section 5.3

The dictionary object is not always taught in beginning Python courses. However, since it it so useful in problem solving, we decided to add this object to our arsenal.

When the keys of a dictionary are placed into a list with a statement of the form **`listName = list(dictName.keys())`**, there is no way to predict the order in which the keys will appear in the list.

Chapter 6

Section 6.1

Although if statements can be used to make a program robust, if statements have limitations. There are certain situations beyond the programmer's control where the more powerful try/except structure is needed to guarantee that a program will not crash.

Section 6.2

In Example 3, the function *isOdd* could have been written

```
def isOdd(n):
    return ((1 <= n <= 36) and (n % 2))
```

Although this definition is terse and elegant, some people find the definition appearing in Example 3 to be easier to understand.

The names in Exercise 7 are the first names of some famous world-class track stars.

Correction: In Exercise 20, the function referred to should be `random.sample(range(1,60), 5)`.

Section 6.3

Turtle graphics has many capabilities that we do not cover in this section. However, the few methods we present can be used to create many different types of drawings.

Often trial and error guessing is required to determine the locations for the text in a drawing. For instance, that was the case when the principal languages program in Example 5 was written.

Chapter 7

Section 7.1

The topic of object-oriented programming is so fundamental to modern programming that is supported by nearly every currently used programing language.

Section 7.2

Inheritance is considered one of the most important capabilities of object-oriented programing.

Correction: On page 296, the if blocks inside the *CalcSemGrade* function should be shifted four spaces to the right.

Chapter 8

We do not create actual graphic user interface (GUI) programs until the third section of this chapter. The first two sections are preparatory. The first section explains what widgets are and how to use them, and the second section shows how to place widgets in a window.

Section 8.1

There are other widgets that are typically covered in Python books. The most common are check boxes and radio buttons. However, we decided to cover list boxes and scroll bars instead, since list boxes and scroll bars give programs a capability that cannot be achieved with standard Python programs.

Each GUI program ends with the statement `window.mainloop()`, since this statement is needed in certain circumstances. However, you may be able to get by without the statement. As a test, try running Example 1 with the statement commented out.

Section 8.2

In one sense this section is the easiest section of the chapter since the basic concepts are straightforward. In another sense, it is the most difficult section since placing the widgets exactly where you want takes considerable time and patience.

Section 8.3

Since we show how to write GUI programs with and without classes, this chapter can be used in courses that do not cover Chapter 7.

Correction: The last line on page 337 should read `bird = StateBirds()`

Answers to Odd-Numbered Exercises

CHAPTER 2

EXERCISES 2.1

1. 12 **3.** .125 **5.** 8 **7.** 2 **9.** 1 **11.** 1 **13.** Not valid **15.** Valid

17. Not valid **19.** 10 **21.** 16 **23.** 9 **25.** `print((7 * 8) + 5)`

27. `print(.055 * 20)` **29.** `print(17 * (3 + 162))`

31.

	x	y
`x = 2`	2	does not exist
`y = 3 * x`	2	6
`x = y + 5`	11	6
`print(x + 4)`	11	6
`y = y + 1`	11	7

33. `24` **35.** `10` **37.** `2 15` **39.** The third line should read `c = a + b`.

41. The first line should read `interest = 0.05`. **43.** 10 **45.** 7 **47.** 3.128

49. -2 **51.** 0 **53.** 6 **55.** `cost += 5` **57.** `cost /= 6` **59.** `sum %= 2`

61.
```
revenue = 98456
costs = 45000
profit = revenue - costs
print(profit)
```

63.
```
price = 19.95
discountPercent = 30
markdown = (discountPercent / 100) * price
price -= markdown
print(round(price, 2))
```

65.
```
balance = 100
balance += 0.05 * balance
balance += 0.05 * balance
balance += 0.05 * balance
print(round(balance, 2))
```

67.
```
balance = 100
balance *= 1.05 ** 10
print(round(balance, 2))
```

```
69. tonsPerAcre = 18
    acres = 30
    totalTonsProduced = tonsPerAcre * acres
    print(totalTonsProduced)

71. distance = 233
    elapsedTime = 7 - 2
    averageSpeed = distance / elapsedTime
    print(averageSpeed)

73. gallonsPerPersonDaily = 1600
    numberOfPeople = 315000000
    numberOfDays = 365
    gallonsPerYear = gallonsPerPersonDaily * numberOfPeople * numberOfDays
    print(gallonsPerYear)

75. numberOfPizzarias = 70000
    percentage = .12
    numberOfRestaurants = numberOfPizzarias / percentage
    print(round(numberOfRestaurants))

77. nationalDebt = 1.68e+13
    population = 3.1588e+8
    perCapitaDebt = nationalDebt / population
    print(round(perCapitaDebt))
```

EXERCISES 2.2

1. Python 3. Ernie 5. "o" 7. "h" 9. "Pyt" 11. "Py" 13. "h"

15. "th" 17. "Python" 19. 2 21. -1 23. 10 25. 2 27. -1

29. 3 31. "8 ball" 33. "8 BALL" 35. "The Artist" 39. 5

41. 7 43. 2 45. "King Kong" 47.
```
12
MUNICIPALITY
city
6
```

49. flute 51. Your age is 21. 53. A ROSE IS A ROSE IS A ROSE

55. WALLAWALLA 57. goodbye 59. Mmmmmmm. 61. a b

63. 76 trombones 65. 17 67. 8 69. The Great 9

71. s[:-1] 73. -8 75. True 77. True

79. `234-5678` should be surrounded with quotation marks.

81. *for* is a reserved word and cannot be used as a variable name.

83. The string should be replaced with `"Say it ain't so."`

85. `Upper` should be changed to `upper`.

87. A string cannot be concatenated with a number. The second line should be written
```
print("Age: " + str(age))
```

89. *find* is a not an allowable method for a number; only for a string.

91. The string "Python" does not have a character of index 8.

93.
```
## Display an inventor's name and year of birth.
firstName = "Thomas"
middleName = "Alva"
lastName = "Edison"
yearOfBirth = 1847
print(firstName,  middleName, lastName + ',', yearOfBirth)
```

95.
```
## Display a copyright statement.
publisher = "Pearson"
print("(c)", publisher)
```

97.
```
## Calculate the distance from a storm.
prompt = "Enter number of seconds between lightning and thunder: "
numberOfSeconds = float(input(prompt))
distance = numberOfSeconds / 5
distance = round(distance, 2)
print("Distance from storm:", distance, "miles.")
```

```
Enter number of seconds between lightning and thunder: 1.25
Distance from storm: 0.25 miles.
```

99.
```
## Calculate weight loss during a triathlon.
cycling = float(input("Enter number of hours cycling: "))
running = float(input("Enter number of hours running: "))
swimming = float(input("Enter number of hours swimming: "))
pounds = (200 * cycling + 475 * running + 275 * swimming) / 3500
pounds =round(pounds, 1)
print("Weight loss:", pounds, "pounds")
```

```
Enter number of hours cycling: 2
Enter number of hours running: 3
Enter number of hours swimming: 1
Weight loss: 0.6 pounds
```

101.
```
## Calculate percentage of games won by a baseball team.
name = input("Enter name of team: ")
gamesWon = int(input("Enter number of games won: "))
gamesList = int(input("Enter number of games lost: "))
percentageWon = round(100 * (gamesWon) / (gamesWon + gamesList), 1)
print(name, "won", str(percentageWon) + '%', "of their games.")
```

```
Enter name of team: Yankees
Enter number of games won: 68
Enter number of games lost: 52
Yankees won 56.7% of their games.
```

103.
```
## Determine the speed of a skidding car.
distance = float(input("Enter distance skidded (in feet): "))
speed = (24 * distance) ** .5
speed = round(speed, 2)
print("Estimated speed:", speed, "miles per hour")
```

```
Enter distance skidded: 54
Estimated speed: 36.0 miles per hour
```

105.
```
## Convert speed from kph to mph.
speedInKPH = float(input("Enter speed in KPH: "))
speedInMPH = speedInKPH * .6214
print("Speed in MPH:", round(speedInMPH, 2))
```

```
Enter speed in KPH: 112.6541
Speed in MPH: 70.00
```

Note: The world's fastest animal, the cheetah, can run at the speed of 112.6541 kilometers per hour.

107.
```
## Calculate equivalent CD interest rate for municipal bond rate.
taxBracket = float(input("Enter tax bracket (as decimal): "))
bondRate = float(input("Enter municipal bond interest rate (as %): "))
equivCDrate = bondRate / (1 - taxBracket)
print("Equivalent CD interest rate:", str(round(equivCDrate, 3)) + '%')
```

```
Enter tax bracket (as decimal): .37
Enter municipal bond interest rate (as %): 3.26
Equivalent CD interest rate: 5.175%
```

109.
```
## Analyze a number.
number = input("Enter number: ")
decimalPoint = number.find('.')
print(decimalPoint, "digits to left of decimal point")
print(len(number) - decimalPoint - 1, "digits to right of decimal point")
```

```
Enter number: 76.543
2 digits to left of decimal point
3 digits to right of decimal point
```

111.
```
## Convert a number of months to years and months.
numberOfMonths = int(input("Enter number of months: "))
years = numberOfMonths // 12
months = numberOfMonths % 12
print(numberOfMonths, "months is", years, "years and", months, "months.")
```

```
Enter number of months: 234
234 months is 19 years and 6 months.
```

EXERCISES 2.3

1. Bon Voyage!

3. Portion: 90%

5. 1 x 2 x 3

7. father-in-law

9. T-shirt

11. Python

13. Hello

    World!

15. One Two
    Three Four

17. NUMBER SQUARE
    2 4
    3 9

19. Hello World!
    Hello World!

21. 01234567890
        A B C

23. 01234567890123456
      one two three

25. 0123456789
        12.30%
       123.0%
     1,230.00%

27. $1,234.57

29. 1

31. Language Native speakers % of World Pop.
    Mandarin 935,000,000 14.10%
    Spanish 387,000,000 5.85%
    English 365,000,000 5.52%

33. Be yourself – everyone else is taken.

35. Always look on the bright side of life.

37. The product of 3 and 4 is 12.

39. The square root of 2 is about 1.4142.

41. In a randomly selected group of 23 people, the probability is 0.51 that 2 people have the same birthday.

43. You miss 100% of the shots you never take. - Wayne Gretsky

45. 22.28% of the UN nations are in Europe.

47. abracadabra

49. Be kind whenever possible. It is always possible. - Dalai Lama

51. Yes

53.
```
## Calculate a server's tip.
bill = float(input("Enter amount of bill: "))
percentage = float(input("Enter percentage tip: "))
tip = (bill * percentage) / 100
print("Tip: ${0:.2f}".format(tip))
```

```
Enter amount of bill: 45.50
Enter percentage tip: 20
Tip: $9.10
```

55.
```
## Calculate a new salary.
beginningSalary = float(input("Enter beginning salary: "))
raisedSalary = 1.1 * beginningSalary
cutSalary = .9 * raisedSalary
percentChange = (cutSalary - beginningSalary) / beginningSalary
print("New salary: ${0:,.2f}".format(cutSalary))
print("Change: {0:.2%}".format(percentChange))
```

```
Enter beginning salary: 42500
New salary: $42,075.00
Change: -1.00%
```

57.
```
## Calculate a future value.
p = float(input("Enter principal: "))
r = float(input("Enter interest rate (as %): "))
n = int(input("Enter number of years: "))
futureValue = p * (1 + (r / 100)) ** n
print("Future value: ${0:,.2f}".format(futureValue))
```

```
Enter principal: 2500
Enter interest rate (as %): 3.5
Enter number of years: 2
Future value: $2,678.06
```

EXERCISES 2.4

1. Pennsylvania Hawaii 3. Alaska Hawaii 5. Delaware Delaware

7. 48 9. Ohio 11. DELAWARE 13. ['Puerto Rico'] 15. United States

17. ['New Jersey', 'Georgia', 'Connecticut']

19. ['Oklahoma', 'New Mexico', 'Arizona']

21. ['Delaware', 'Pennsylvania', 'New Jersey', 'Georgia']

23. ['Arizona', 'Alaska', 'Hawaii'] 25. [] 27. Georgia

29. ['Alaska', 'Hawaii'] 31. New Mexico 33. 10 35. 0 37. 48

39. ['Hawaii', 'Puerto Rico', 'Guam']

41. ['Hawaii', 'Puerto Rico', 'Guam']

43. ['Delaware', 'Commonwealth of Pennsylvania', 'New Jersey']

45.
```
['New', 'Mexico']
['New', 'Jersey']
```

47. Pennsylvania,New Jersey,Georgia 49. 8 51. 100

53. 0 55. Largest Number: 8 57. Total: 16

59. This sentence contains five words.
    This sentence contains six different words.

61. Babbage, Charles

63. Middle Name: van

65. When in the course of human events

67. editor-in-chief

69. e**pluribus**unum

71. ['New York', 'NY', 'Empire State', 'Albany']

73. ['France', 'England', 'Spain']

75. programmer

77. Follow your own star.

79. 987-654-3219

81. [3, 9, 6]

83. each

85. ['soprano', 'tenor', 'alto', 'bass']

87. ['gold', 'silver', 'bronze']

89. murmur

91. ('Happy', 'Sneezy', 'Bashful')

93. 1

95. Index out of range. The list does not have an item of index 3.

97. The join method only can be applied to a list consisting entirely of strings.

99. The second line is not valid. Items in a tuple cannot be reassigned values directly.

101.
```
## Count the number of words in a sentence.
sentence = input("Enter a sentence: ")
L = sentence.split(" ")
print("Number of words:", len(L))
```

```
Enter a sentence: This sentence contains five words.
Number of words: 5
```

103.
```
## Display a name.
name = input("Enter a 2-part name: ")
L = name.split()
print("{0:s}, {1:s}".format(L[1], L[0]))
```

```
Enter a 2-part name: Charles Babbage
Revised form: Babbage, Charles
```

CHAPTER 3

EXERCISES 3.1

1. hi

3. The letter before G is F.

5. Minimum: 3
   Maximum: 17

7. D is the 4th letter of the alphabet.

9. True

11. True 13. True 15. True 17. False 19. False 21. True

23. True 25. False 27. False 29. False 31. False 33. False

35. True 37. False 39. False 41. True 43. False 45. Equivalent

47. Not equivalent 49. Equivalent 51. Equivalent 53. Equivalent

55. `a <= b` 57. `(a >= b) or (c == d)`

59. `a > b` 61. `ans in ['Y', 'y', "Yes", "yes"]`

63. `2010 <= year <= 2013` 65. `3 <= n < 9` 67. `-20 < n <= 10`

69. True 71. True 73. True 75. True

77. True 79. False 81. False 83. False

85.
```
print("He said " + chr(34) + "How ya doin?" + chr(34) + " to me.")
```

EXERCISES 3.2

1. Less than ten. 3. False 5. Remember, tomorrow is another day.

7. 10 9. To be, or not to be. 11. Hi

13. A nonempty string is true.

15. Syntax error and logic error. Second line should be `if n == 7:`. Third line should be `print("The square is", n ** 2)`.

17. Syntax error. Second line is full of errors. It should be as follows:
```
if (major == "Business") or (major == "Computer Science"):
```

19.
```
a = 5
```

21.
```
if (j == 7):
    b = 1
else:
    b = 2
```

23.
```
answer = input("Is Alaska bigger than Texas and California combined? ")
if answer[0].upper() == 'Y':
    print("Correct")
else:
    print("Wrong")
```

25.
```
## Calculate a tip.
bill = float(input("Enter amount of bill: "))
tip = bill * 0.15
if (tip < 2):
    tip = 2
print("Tip is ${0:,.2f}".format(tip))
```

```
Enter amount of bill: 13.00
Tip is $2.00
```

```
Enter amount of bill: 52.00
Tip is $8.55
```

27.
```
## Calculate the cost of widgets.
num = int(input("Enter number of widgets: "))
if num < 100:
    cost = num * 0.25
else:
    cost = num * 0.20
print("Cost is ${0:,.2f}".format(cost))
```

```
Enter number of widgets: 325
Cost is $65.00
```

29.
```
## A quiz
response = input("Who was the first Ronald McDonald? ")
if response == "Willard Scott":
    print("You are correct.")
else:
    print("Nice try.")
```

```
Who was the first Ronald McDonald? Willard Scott
You are correct.
```

31.
```
## Calculate an average after dropping the lowest score.
scores = []
scores.append(eval(input("Enter first score: ")))
scores.append(eval(input("Enter second score: ")))
scores.append(eval(input("Enter third score: ")))
scores.remove(min(scores))
average = sum(scores) / 2
print("Average of the two highest scores is {0:.2f}".format(average))
```

```
Enter first score: 90
Enter second score: 80
Enter third score: 90
Average of two highest two scores is 90.
```

33.
```
## Make change for a purchase of apples.
weight = float(input("Enter weight in pounds: "))
payment = float(input("Enter payment in dollars: "))
cost = (2.5 * weight)
if payment >= cost:
    change = payment - cost
    print("Your change is ${0:,.2f}.".format(change))
else:
    amountOwed = cost - payment
    print("You owe ${0:,.2f} more.".format(amountOwed))
```

```
Enter weight in pounds: 5
Enter payment in dollars: 6
You owe $2.50 more.
```

```
Enter weight in pounds: 3
Enter payment in dollars: 10
Your change is $2.50.
```

35.
```
## Validate input.
letter = input("Enter a single uppercase letter: ")
if (len(letter) != 1) or (letter != letter.upper()):
    print("You did not comply with the request.")
```

```
Enter a single uppercase letter: y
You did not comply with the request.
```

37.
```
## Convert military time to regular time.
militaryTime = input("Enter a military time (0000 to 2359): ")
hours = int(militaryTime[0:2])
minutes = int(militaryTime[2:4])
if hours >= 12:
    cycle = "pm"
    hours %= 12
else:
    cycle = "am"
if hours == 0:
    hours = 12
print("The regular time is {0}:{1} {2}.".format(hours, minutes, cycle))
```

```
Enter a military time (0000 to 2359): 0040
The regular time is 12:40 am.
```

39.
```
## Use APYs to compare interest rates offered by two banks.
r1 = float(input("Enter annual rate of interest for Bank 1: "))
m1 = float(input("Enter number of compounding periods for Bank 1: "))
r2 = float(input("Enter annual rate of interest for Bank 2: "))
m2 = float(input("Enter number of compounding periods for Bank 2: "))
ipp1 = r1 / (100 * m1)   # interest rate per period
ipp2 = r2 / (100 * m2)
apy1 = ((1 + ipp1) ** m1) - 1
apy2 = ((1 + ipp2) ** m2) - 1
print("APY for Bank 1 is {0:,.3%}".format(apy1))
print("APY for Bank 2 is {0:,.3%}".format(apy2))
if (apy1 == apy2):
    print("Bank 1 and Bank 2 are equally good.")
else:
    if (apy1 > apy2):
        betterBank = 1
    else:
        betterBank = 2
    print("Bank", betterBank, "is the better bank.")
```

```
Enter annual rate of interest for Bank 1: 3.1
Enter number of compounding periods for Bank 1: 2
Enter annual rate of interest for Bank 2: 3
Enter number of compounding periods for Bank 2: 4
APY for Bank 1 is 3.124%.
APY for Bank 2 is 3.034%.
Bank 1 is the better bank.
```

41.
```
## Bestow graduation honors.
# Request grade point average.
gpa = eval(input("Enter your grade point average (2 through 4): "))
# Validate that GPA is between 2 and 4
if not (2 <= gpa <=4):
    print("Invalid grade point average. GPA must be between 2 and 4.")
else:
    # Determine if honors are warranted and display conclusion.
    if gpa >= 3.9:
        honors = " summa cum laude."
    elif gpa >= 3.6:
        honors = " magna cum laude."
    elif gpa >= 3.3:
        honors = " cum laude."
    else:
        honors = "."
    print("You graduated" + honors)
```

```
Enter your gpa: 2.5
You graduated.
```

43.
```
## Calculate a person's state income tax.
income = float(input("Enter your taxable income: "))
if income <= 20000:
    tax =.02 * income
else:
    if income <= 50000:
        tax = 400 + .025 * (income - 20000)
    else:
        tax = 1150 + .035 * (income - 50000)
print("Your tax is ${0:,.0f}.".format(tax))
```

```
Enter your taxable income: 60000
Your tax is $1,500.
```

EXERCISES 3.3

1. 24 3. 10 5. 20 7. a b c d 9. Infinite loop

11. `i` should be initialized to -1 in order to iterate over all the elements

13.
```
for i in range(3):
    name = input("Enter a name: ")
    print(name)
```

15.
```
## Display a Celsius-to-Fahrenheit conversion table.
print("Celsius\t\tFahrenheit")
for celsius in range(10, 31, 5):
    fahrenheit = (celsius * (9 / 5)) + 32
    print("{0}\t\t{1:.0f}".format(celsius, fahrenheit))
```

```
Celsius     Fahrenheit
  10           50
  15           59
  20           68
  25           77
  30           86
```

17.
```
## Find the GCD of two numbers.
m = int(input("Enter value of M: "))
n = int(input("Enter value of N: "))
while n != 0:
    t = n
    n = m % n    # remainder after m is divided by n
    m = t
print("Greatest common divisor:", m)
```

```
Enter value of M: 49
Enter value of N: 28
Greatest common divisor:7
```

19.
```
## Find special age.
age = 1
while (1980 + age) != (age * age):
    age += 1
print("Person will be {0} \nin the year {1}.".format(age, age * age))
```

```
Person will be 45
in the year 2024.
```

21.
```
## Radioactive decay
mass = 100    # weight in grams
year = 0
while(mass > 1):
    mass /= 2
    year += 28
print("The decay time is")
print(year, "years.")
```

```
The decay time is
196 years.
```

23.
```
## Determine when a car loan will be half paid off.
principal = 15000
balance = principal   # initial balance
monthlyPayment = 290
monthlyFactor = 1.005   # multiplier due to interest
month = 0
while(balance >= principal / 2):
    balance =  (monthlyFactor * balance) - monthlyPayment
    month += 1
print("Loan will be half paid \noff after", month, "months.")
```

```
Loan will be half paid
off after 33 months.
```

25.
```
## Annuity with withdrawals
balance = 10000
interestMultiplier = 1.003   # multiplier due to interest
monthlyWithdrawal = 600
month = 0
while balance > 600:
    balance = (interestMultiplier * balance) - monthlyWithdrawal
    month += 1
print("Balance will be ${0:,.2f} \nafter {1} months.".
      format(balance, month))
```

```
Balance will be $73.19
after 17 months.
```

27.
```
## Determine class size for which the probability is greater
## than 50% that someone has the same birthday as you.
num  = 1
while (364 / 365) ** num > 0.5:
    num += 1
print("With", num, "students, the probability")
print("is greater than 50% that someone")
print("has the same birthday as you.")
```

```
With 253 students, the probability
is greater than 50% that someone
has the same birthday as you.
```

29.
```
## Determine when India's population will surpass China's population.
chinaPop = 1.37
indiaPop = 1.26
year = 2014
while indiaPop < chinaPop:
    year += 1
    chinaPop *= 1.0051
    indiaPop *= 1.0135
print("India's population will exceed China's")
print("population in the year", str(year) + '.')
```

```
India’s population will exceed China’s
population in the year 2025.
```

```
31. ## Maintain a savings account.
    print("Options:")
    print("1. Make a Deposit")
    print("2. Make a Withdrawal")
    print("3. Obtain Balance")
    print("4. Quit")
    balance = 1000
    while True:
        num = int(input("Make a selection from the options menu: "))
        if num == 1:
            deposit = float(input("Enter amount of deposit: "))
            balance += deposit
            print("Deposit Processed.")
        elif num == 2:
            withdrawal = float(input("Enter amount of withdrawal: "))
            while (withdrawal > balance):
                print("Denied. Maximum withdrawal is ${0:,.2f}"
                      .format(balance))
                withdrawal = float(input("Enter amount of withdrawal: "))
            balance -= withdrawal
            print("Withdrawal Processed.")
        elif num == 3:
            print("Balance: ${0:,.2f}".format(balance))
        elif num == 4:
            break
        else:
            print("You did not enter a proper number.")
```

```
Options:
1. Make a Deposit
2. Make a Withdrawal
3. Obtain Balance
4. Quit
Make a selection from the options menu: 1
Enter amount of deposit: 500
Deposit Processed.
Make a selection from the options menu: 2
Enter amount of withdrawal: 2000
Denied. Maximum withdrawal is $1,500.00
Enter amount of withdrawal: 600
Withdrawal Processed.
Make a selection from the options menu: 3
Balance: $900.00
Make a selection from the options menu: 4
```

EXERCISES 3.4

1. 7, 8, 9, 10

3. 2, 5, 8, 11

5. 0, 1, 2, 3, 4, 5

7. 11, 10, 9, 8

9. range(4, 20, 5)

11. range(-21, -17)

13. range(20, 13, -3)

15. range(5, -1, -1)

17.
```
Pass #1
Pass #2
Pass #3
Pass #4
```

19.
```
5
6
7
```

21.
```
¢¢¢¢¢¢¢¢¢¢
```

23.
```
2
4
6
8
Who do we appreciate?
```

25.
```
3
```

27.
```
15
```

29.
```
n
```

31.
```
3 20
```

33.
```
The shortest word has length 5
```

35.
```
Three
```

37.
```
18
```

39.
```
North Carolina
North Dakota
```

41. The range generates no elements because the step argument's direction is opposite the direction from start to stop.

43. The print function call is missing parentheses.

45. The range constructor should read `range(0, 20)` or `range(20)` because `range(20,0)` will not generate any values. Also, the print statement must be indented twice so it belongs to the *if* block.

47.
```
for num in range(1, 10, 2):
    print(num)
```

49.
```
lakes = ["Erie", "Huron", "Michigan", "Ontario", "Superior"]
print(", ".join(lakes))
```

51.
```
## Determine amount of radioactive material remaining after five years.
amount = 10
for i in range(5):
    amount *= .88
print("The amount of cobalt-60 remaining")
print("after five years is {0:.2f} grams.".format(amount))
```

```
The amount of cobalt-60 remaining
after five years is 5.28 grams.
```

53.
```
## Count the number of vowels in a phrase.
total = 0
phrase = input("Enter a phrase: ")
phrase = phrase.lower()
for ch in phrase:
    if ch in "aeiou":
        total += 1
print("The phrase contains", total, "vowels.")
```

```
Enter a phrase: E PLURIBUS UNUM
The phrase contains 6 vowels.
```

55.
```
## Total the fractions 1/n for n = 1 through 100.
sum = 0
for i in range(1, 101):
    sum += 1 / i
print("The sum of 1 + 1/2 + 1/3 + ... + 1/100")
print("is {0:.5f} to five decimal places.".format(sum))
```

```
The sum 1 + 1/2 + 1/3 + ... + 1/100
is 5.18738 to five decimal places.
```

57.
```
## Determine if the letters of a word are in alphabetical order.
word = input("Enter a word: ")
word = word.lower()
firstLetter = ""
secondLetter = ""
flag = True
for i in range(0, len(word) - 1):
    firstLetter = word[i]
    secondLetter = word[i + 1]
    if firstLetter > secondLetter:
        flag = False
        break
if flag:
    print("Letters are in alphabetical order.")
else:
    print("Letters are not in alphabetical order.")
```

```
Enter a word: python
Letters are not in alphabetical order.
```

59.
```
## Calculate a person's lifetime earnings.
name = input("Enter name: ")
age = int(input("Enter age: "))
salary = float(input("Enter starting salary: "))
earnings = 0
for i in range(age, 65):
    earnings += salary
    salary += .05 * salary
print("{0} will earn about ${1:,.0f}.".format(name, earnings))
```

```
Enter name: Ethan
Enter age: 22
Enter starting salary: 27000
Helen will earn about $3,860,820.
```

61.
```
## Display the balances on a car loan.
print("         AMOUNT OWED AT")
print("YEAR    ", "END OF YEAR")
balance = 15000
year = 2012
for i in range(1, 49):
    balance = (1.005 * balance) - 290
    if i % 12 == 0:
        year += 1
        print(year, "    ${0:,.2f}".format(balance))
print(year + 1, "    $0.00")
```

```
          AMOUNT OWED AT
YEAR      END OF YEAR
2013      $12,347.85
2014      $9,532.13
2015      $6,542.74
2016      $3,368.97
2017      $0.00
```

63.
```
## Calculate the average of the best two of three grades.
grades = []
for i in range(3):
    grade = int(input("Enter a grade: "))
    grades.append(grade)
grades.sort()
average = (grades[1] + grades[2]) / 2
print("Average: {0:n}".format(average))
```

```
Enter a grade: 70
Enter a grade: 90
Enter a grade: 80
Average: 85
```

65.
```
## Display the effects of supply and demand.
print("YEAR   QUANTITY    PRICE")
quantity = 80
price = 20 - (.1 * quantity)
print("{0:d}     {1:.2f}     ${2:.2f}".format(2014, quantity, price))
for i in range(4):
    quantity = (5 * price) - 10
    price = 20 - (.1 * quantity)
    print("{0:d}     {1:.2f}     ${2:.2f}".format(i + 2015, quantity, price))
```

```
YEAR   QUANTITY    PRICE
2014     80.00     $12.00
2015     50.00     $15.00
2016     65.00     $13.50
2017     57.50     $14.25
2018     61.25     $13.88
```

67.
```
## Compare two salary options.
# Calculate amount earned in ten years with Option 1.
salary = 20000
option1 = 0
for i in range(10):
    option1 += salary
    salary += 1000
print("Option 1 earns ${0:,d}.".format(option1))
# Calculate amount earned in ten years with Option 2.
salary = 10000
option2 = 0
for i in range(20):
    option2 += salary
    salary += 250
print("Option 2 earns ${0:,d}.".format(option2))
```

```
Option1 earns $245,000.
Option2 earns $247,500.
```

69.
```
## Determine the number of Super Bowl wins for the Pittsburg Steelers.
teams = open("SBWinners.txt", 'r')
numberOfWins = 0
for team in teams:
    if team.rstrip() == "Steelers":
        numberOfWins += 1
print("The Steelers won")
print(numberOfWins, "Super Bowl games.")
```

```
The Steelers won
6 Super Bowl games.
```

71.
```
## Analyze grades on a final exam.
infile = open("Final.txt", 'r')
grades = [line.rstrip() for line in infile]
infile.close()
for i in range(len(grades)):
    grades[i] = int(grades[i])
average = sum(grades) / len(grades)
num = 0    # number of grades above average
for grade in grades:
    if grade > average:
        num += 1
print("Number of grades:", len(grades))
print("Average grade:", average)
print("Percentage of grades above average: {0:.2f}%"
                  .format(100 * num / len(grades)))
```

```
Number of grades: 24
Average grade: 83.25
Percentage of grades above average: 54.17%
```

73.
```
## Count the number of different vowels in a word.
word = input("Enter a word: ")
word = word.upper()
vowels = "AEIOU"
vowelsFound = []
numVowels = 0
for letter in word:
    if (letter in vowels) and (letter not in vowelsFound):
        numVowels += 1
        vowelsFound.append(letter)
print("Number of vowels:", numVowels)
```

```
Enter a word: Mississippi
Number of different vowels: 1
```

75.
```
## Calculate probabilities that at least two
## people in a group have the same birthday.
print("{0:17} {1}".format("NUMBER OF PEOPLE", "PROBABILITY"))
# r = size of group
for r in range(21, 26):
    product = 1
    for t in range(1, r):
        product *= ((365 - t) / 365)
    print("{0:<17} {1:.3f}".format(r, 1 - product))
```

```
NUMBER OF PEOPLE  PROBABILITY
21                0.444
22                0.476
23                0.507
24                0.538
25                0.569
```

77.
```
## Display sentence with Boston accent.
sentence = input("Enter a sentence: ")
newSentence = ""
for ch in sentence:
    if ch.upper() != 'R':
        newSentence += ch
print(newSentence)
```

```
Enter a sentence: Park the car in Harvard Yard.
Revised sentence: Pak the ca in Havad Yad.
```

79.
```
## Identify president by number.
infile = open("USPres.txt", 'r')
for i in range(15):
    infile.readline()
print("The 16th president was")
print(infile.readline().rstrip() + '.')
infile.close()
```

```
The 16th president was
Abraham Lincoln.
```

81.
```
## Calculate number of odometer readings containing the digit 1.
total = 0
for n in range(1000000):
    if '1' in str(n):
        total += 1
print("{0:,d} numbers on the odometer".format(total))
print("contain the digit 1.")
```

```
468,559 numbers on the odometer
contain the digit 1.
```

83.
```
## Display justices by party of appointing president.
justices = ["Scalia R", "Kennedy R", "Thomas R", "Ginsburg D",
            "Breyer D", "Roberts R", "Alito R", "Sotomayor D", "Kagan D"]
demAppointees = []
repAppointees = []
for justice in justices:
    if justice[-1] == 'D':
        demAppointees.append(justice[:-2])
    else:
        repAppointees.append(justice[:-2])
namesD = ", ".join(demAppointees)
namesR = ", ".join(repAppointees)
print("Democratic appointees:", namesD)
print("Republican appointees:", namesR)
```

```
Democratic appointees: Ginsburg, Breyer, Sotomayor, Kagan
Republican appointees: Scalia, Kennedy, Thomas, Roberts, Alito
```

CHAPTER 4

EXERCISES 4.1

1.
```
H
W
```

3.
```
Enter the population growth as a percent: 2
The population will double in about 36.00 years.
```

5.
```
Your income tax is $499.00
```

7.
```
Why do clocks run clockwise?

Because they were invented in the northern
hemisphere where sundials go clockwise.
```

9.
```
168 hours in a week
76 trombones in the big parade
```

11.
```
President Bush is a graduate of Yale.
President Obama is a graduate of Columbia.
```

13.
```
7
5
```

15. Fredrick

17. Total cost: $106.00

19. 5

21.
```
When in the course of human events
```

23.
```
Enter grade on midterm exam: 85
Enter grade on final exam: 94
Enter type of student (Pass/Fail) or (Letter Grade): Letter Grade
Semester grade: A

Enter grade on midterm exam: 50
Enter grade on final exam: 62
Enter type of student (Pass/Fail) or (Letter Grade): Pass/Fail
Semester grade: Fail

Enter grade on midterm exam: 56
Enter grade on final exam: 67
Enter type of student (Pass/Fail) or (Letter Grade): Letter Grade
Semester grade: D
```

25.
```
def maximum(list1):
    largestNumber = list1[0]
    for number in list1:
        if number > largestNumber:
            largestNumber = number
    return largestNumber
```

27.
```
def main():
    word = input("Enter a word: ")
    if isQwerty(word):
        print(word, "is a Qwerty word.")
    else:
        print(word, "is not a Qwerty word.")

def isQwerty(word):
    word = word.upper()
    for ch in word:
        if ch not in "QWERTYUIOP":
            return False
    return True

main()
```

```
Enter a word: YET
YET is a Qwerty word.
```

```
Enter a word: Python
Python is not a Qwerty word.
```

29.
```
def main():
    ## Compare salary options
    opt1 = option1()
    opt2 = option2()
    print("Option 1 = ${0:,.2f}.".format(opt1))
    print("Option 2 = ${0:,.2f}.".format(opt2))
    if opt1 > opt2:
        print("Option 1 pays better.")
    elif opt1 == opt2:
        print("Options pay the same.")
    else:
        print("Option 2 is better.")
```

```
def option1():
    ## Compute the total salary for 10 days,
    ## with a flat salary of $100/day.
    sum = 0
    for i in range(10):
        sum += 100
    return sum

def option2():
    ## Compute the total salary for 10 days,
    ## starting at $1 and doubling each day.
    sum = 0
    daySalary = 1
    for i in range(10):
        sum += daySalary
        daySalary *= 2
    return sum

main()
```

```
Option 1 pays $1,000.00
Option 2 pays $1,023.00
Option 2 is better.
```

31.
```
# Named constants.
WAGE_BASE = 117000 # There is no social security benefits
                   # tax on income above this level.
SOCIAL_SECURITY_TAX_RATE = 0.062      # 6.2%
MEDICARE_TAX_RATE = 0.0145            # 1.45%
ADDITIONAL_MEDICARE_TAX_RATE = .009   # 0.9%

def main():
    ## Calculate FICA tax for a single employee.
    ytdEarnings, curEarnings, totalEarnings = obtainEarnings()
    socialSecurityBenTax = calculateBenTax(ytdEarnings, curEarnings,
                                           totalEarnings)
    calculateFICAtax(ytdEarnings, curEarnings, totalEarnings,
                     socialSecurityBenTax)

def obtainEarnings():
    str1 = "Enter total earnings for this year prior to current pay period: "
    ytdEarnings = eval(input(str1))     # year-to-date earnings
    curEarnings = eval(input("Enter earnings for the current pay period: "))
    totalEarnings = ytdEarnings + curEarnings
    return(ytdEarnings, curEarnings, totalEarnings)

def calculateBenTax(ytdEarnings, curEarnings, totalEarnings):
    ## Calculate the Social Security Benefits tax.
    socialSecurityBenTax = 0
    if totalEarnings <= WAGE_BASE:
        socialSecurityBenTax = SOCIAL_SECURITY_TAX_RATE * curEarnings
    elif ytdEarnings < WAGE_BASE:
        socialSecurityBenTax = SOCIAL_SECURITY_TAX_RATE * (WAGE_BASE -
                               ytdEarnings)
    return socialSecurityBenTax
```

```
def calculateFICAtax(ytdEarnings, curEarnings, totalEarnings,
                     socialSecurityBenTax):
    ## Calculate and display the FICA tax.
    medicareTax = MEDICARE_TAX_RATE * curEarnings
    if ytdEarnings >= 200000:
        medicareTax += ADDITIONAL_MEDICARE_TAX_RATE * curEarnings
    elif totalEarnings > 200000:
        medicareTax += ADDITIONAL_MEDICARE_TAX_RATE * (totalEarnings - 200000)
    ficaTax = socialSecurityBenTax + medicareTax
    print("FICA tax for the current pay period: ${0:,.2f}".format(ficaTax))

main()
```

```
Enter total earnings for this year prior to current pay period: 200000
Enter earnings for the current pay period: 2500
FICA tax for the current pay period: $58.75
```

33.
```
colors = []

def main():
    ## Display colors beginning with a specified letter.
    letter = requestLetter()
    fillListWithColors(letter)
    displayColors()

def requestLetter():
    letter = input("Enter a letter: ")
    return letter.upper()

def fillListWithColors(letter):
     global colors
     infile = open("Colors.txt", 'r')
     for color in infile:
         if color.startswith(letter):
             colors.append(color.rstrip())
     infile.close()

def displayColors():
    for color in colors:
        print(color)

main()
```

```
Enter a letter: a
Almond
Antique Brass
Apricot
Aquamarine
Asparagus
Atomic Tangerine
```

EXERCISES 4.2

1.
```
24 blackbirds baked in a pie.
```

3.
```
Cost: $250.00
Shipping cost: $15.00
Total cost: $265.00
```

5.
```
Enter first grade: 88
Enter second grade: 99
Enter third grade: 92
[88, 92, 99]
```

7.
```
['Banana', 'apple', 'pear']
['apple', 'Banana', 'pear']
```

9.
```
nudge nudge
nudge nudge nudge nudge
```

11.
```
spam     and     eggs
spam and eggs
```

13.
```
George Washington
John Adams
```

15.
```
Amadeus
Joseph
Sebastian
Vaughan
```

17.
```
['M', 'S', 'a', 'l', 'o', 't']
['a', 'l', 'M', 'o', 'S', 't']
```

19.
```
VB  Ruby  Python  PHP  Java  C++  C
```

21.
```
Python  Java  Ruby  C++  PHP  VB  C
```

23.
```
-3  -2  4  5  6
```

25.
```
[10, 7, 6, 4, 5, 3]
```

27.
```
['BRRR', 'TWO']
```

29.
```
['c', 'a']
```

31.
```
names = ["George Boole", "Charles Babbage", "Grace Hopper"]
lastNames = [name.split()[-1] for name in names]
```

33. A list consisting of the 50 states in uppercase characters.

35. A list consisting of the 50 states ordered by the lengths of the names in ascending order.

37. Valid

39. Valid

41. Not valid

43. Valid

45. Not valid

47.
```
almost
```

49.
```
def main():
    ## Calculate the original cost of mailing an airmail letter.
    weight = float(input("Enter the number of ounces: "))
    print("Cost: ${0:0,.2f}".format(cost(weight)))

def cost(weight):
    return 0.05 + 0.1 * ceil(weight - 1)

def ceil(x):
    if int(x) != x:
        return int(x + 1)
    else:
        return x
main()
```

```
Enter the number of ounces: 3.2
Cost: $0.35
```

51.
```
def main():
    ## Determine whether two words are anagrams.
    string1 = input("Enter the first word or phrase: ")
    string2 = input("Enter the second word or phrase: ")
    if areAnagrams(string1, string2):
        print("Are anagrams.")
    else:
        print("Are not anagrams.")

def areAnagrams(string1, string2):
    firstString = string1.lower()
    secondString = string2.lower()
    # In the next two lines, the if clauses remove all
    # punctuation and spaces.
    letters1 = [ch for ch in firstString if 'a' <= ch <= 'z']
    letters2 = [ch for ch in secondString if 'a' <= ch <= 'z']
    letters1.sort()
    letters2.sort()
    return (letters1 == letters2)

main()
```

```
Enter the first word or phrase: silent
Enter the second word or phrase: listen
Are anagrams.
```

53.
```
def main():
    ## Sort three names.
    pres = [("Lyndon", "Johnson"),("John", "Kennedy"),("Andrew", "Johnson")]
    pres.sort(key=lambda person: person[0])  # sort by first name
    pres.sort(key=lambda person: person[1])  # sort by last name
    for person in pres:
        print(person[1] + ',', person[0])

main()
```

```
Johnson, Andrew
Johnson, Lyndon
Kennedy, John
```

55.
```
def main():
    ## Sort New England states by land area.
    NE = [("Maine", 30840, 1.329), ("Vermont", 9217, .626),
          ("New Hampshire", 8953, 1.321), ("Massachusetts", 7800, 6.646),
          ("Connecticut", 4842, 3.59), ("Rhode Island", 1044, 1.05)]
    NE.sort(key=lambda state: state[1], reverse=True)
    print("Sorted by land area in descending order:")
    for state in NE:
        print(state[0], " ", end="")
    print()

main()
```

```
Sorted by land area in descending order:
Maine  Vermont  New Hampshire  Massachusetts  Connecticut  Rhode Island
```

57.
```
def main():
    ## Sort New England states by population density.
    NE = [("Maine", 30840, 1.329), ("Vermont", 9217, .626),
          ("New Hampshire", 8953, 1.321), ("Massachusetts", 7800, 6.646),
          ("Connecticut", 4842, 3.59), ("Rhode Island", 1044, 1.05)]
    NE.sort(key=sortByPopulationDensity)
    print("Sorted by population density in ascending order:")
    for state in NE:
        print(state[0], " ", end="")
    print()

def sortByPopulationDensity(state):
    return state[2] / state[1]

main()
```

```
Sorted by population density in ascending order:
Maine  Vermont  New Hampshire  Connecticut  Massachusetts  Rhode Island
```

59.
```
def main():
    ## Sort numbers by largest prime factor.
    numbers = [865, 1169, 1208, 1243, 290]
    numbers.sort(key=largestPrimeFactor)
    print("Sorted by largest prime factor:")
    print(numbers)

def largestPrimeFactor(num):
    n = num
    f = 2
    max = 1
    while n > 1:
        if n % f == 0:
            n = int(n / f)
            if f > max:
                max = f
        else:
            f += 1
    return max

main()
```

```
Sorted by largest prime factor:
[290, 1243, 1208, 1169, 865]
```

61.
```
def main():
    ## Sort numbers by the sum of their odd digits in descending order.
    numbers = [865, 1169, 1208, 1243, 290]
    numbers.sort(key=sumOfOddDigits, reverse=True)
    print("Sorted by sum of odd digits:")
    print(numbers)
```

```
def sumOfOddDigits(num):
    listNums = list(str(num))
    total = 0
    for i in range(len(listNums)):
        if int(listNums[i]) % 2 == 1:
            total += int(listNums[i])
    return total

main()
```

```
Sorted by sum of odd digits:
[1169, 290, 865, 1243, 1208]
```

63.
```
def main():
    ## Display presidents ordered by length of first name.
    infile = open("USPres.txt", 'r')
    listPres = [pres.rstrip() for pres in infile]
    infile.close()
    listPres.sort(key=sortByLengthOfFirstName)
    for i in range(6):
        print(listPres[i])

def sortByLengthOfFirstName(pres):
    return len(pres.split()[0])

main()
```

```
John Adams
John Q. Adams
John Tyler
John Kennedy
Bill Clinton
James Madison
```

65.
```
def main():
    ## Sort states by number of vowels in descending order.
    infile = open("States.txt", 'r')
    listStates = [state.rstrip() for state in infile]
    infile.close()
    listStates.sort(key=numberOfVowels, reverse=True)
    for i in range(6):
        print(listStates[i])

def numberOfVowels(word):
    vowels = ('a', 'e', 'i', 'o', 'u')
    total = 0
    for vowel in vowels:
        total += word.lower().count(vowel)
    return total

main()
```

```
South Carolina
Louisiana
North Carolina
California
West Virginia
South Dakota
```

67.
```
def main():
    ## Calculate new balance and minimum payment for a credit card.
    (oldBalance, charges, credits) = inputData()
    (newBalance, minimumPayment) = calculateNewValues(oldBalance,
                                   charges, credits)
    displayNewData(newBalance, minimumPayment)

def inputData():
    oldBalance = float(input("Enter old balance: "))
    charges = float(input("Enter charges for month: "))
    credits = float(input("Enter credits: "))
    return (oldBalance, charges, credits)

def calculateNewValues(oldBalance, charges, credits):
    newBalance =  (1.015) * oldBalance + charges - credits
    if newBalance <= 20:
        minimumPayment = newBalance
    else:
        minimumPayment = 20 + 0.1 * (newBalance - 20)
    return (newBalance, minimumPayment)

def displayNewData(newBalance, minimumPayment):
    print("New balance: ${0:0,.2f}".format(newBalance))
    print("Minimum payment: ${0:0,.2f}".format(minimumPayment))

main()
```

```
Enter old balance: 175
Enter charges for month: 40
Enter credits: 50
New balance: $167.62.
Minimum payment: $34.76
```

69.
```
def main():
    ## Determine a person's earnings for a week.
    (wage, hours) = getWageAndHours()
    payForWeek = pay(wage, hours)
    displayEarnings(payForWeek)

def getWageAndHours():
    hoursworked = eval(input("Enter hours worked: "))
    hourlyWage = eval(input("Enter hourly pay: "))
    return(hourlyWage, hoursworked)

def pay(wage, hours):
    ## Calculate weekly pay with time-and-a-half for overtime.
    if hours <= 40:
        amount = wage * hours
    else:
        amount = (wage * 40) + ((1.5) * wage * (hours - 40))
    return amount

def displayEarnings(payForWeek):
    print("Week's pay: ${0:,.2f}".format(payForWeek))

main()
```

```
Enter hours worked: 45
Enter hourly pay: 15.00
Week's pay: $712.50
```

CHAPTER 5

EXERCISES 5.1

1. `Aloha`

3.
```
Hello
Aloha
```

5. `6`

7. `[4, 1, 0, 1, 4]`

9. `Believe in yourself.`

11. `['a', 'c', 't']`

13. `ABC.txt` should be open for reading, not for writing.

15. `close()` should be called on the file object, *infile*, not on `ABC.txt`. That is, the last line should read `infile.close()`.

17. The argument for write() must be a string, not an integer.

19. The code should close the file after writing it. Otherwise, the value of *list1* will still be in the buffer and not on the disk drive when the file is opened for reading.

21. The file is cannot be read since it has been closed.

23. The file `ABC.txt` is created. Nothing is displayed on the monitor.

25.
```
def removeDuplicates(list1):
    set1 = set(list1)
    return list(set1)
```

27.
```
def findItemsInEither(list1, list2):
    set1 = set(list1).union(list2)
    return list(set1)
```

29.
```
## Count the words in the Gettysburg Address.
infile = open("Gettysburg.txt")
originalLine = infile.readline()
print(originalLine[:89])
originalLine = originalLine.lower()
# Remove punctuation marks from the original line.
line = ""
for ch in originalLine:
    if ('a' <= ch <= 'z') or (ch == " "):
        line += ch
# Place the words into a list.
listOfWords = line.split()
# Form a set of the words without duplications.
setOfWords = set(listOfWords)
print("The Gettysburg Address contains", len(listOfWords), "words.")
print("The Gettysburg Address contains", len(setOfWords),
      "different words.")
```

```
Four score and seven years ago, our fathers brought
forth on this continent a new nation:
The Gettysburg Address contains 268 words.
The Gettysburg Address contains 139 different words.
```

31. The new file will contain the names of the people who subscribe to both the New York Times and the Wall Street Journal.

33.
```
def main():
    ## Update colors.
    setOfNewColors = getSetOfNewColors()
    createFileOfNewColors(setOfNewColors)

def getSetOfNewColors():
    infile = open("Pre1990.txt", 'r')
    colors = {line.rstrip() for line in infile}
    infile.close()
    infile = open("Retired.txt", 'r')
    retiredColors = {line.rstrip() for line in infile}
    infile.close()
    infile = open("Added.txt", 'r')
    addedColors = {line.rstrip() for line in infile}
    infile.close()
    colorSet = colors.difference(retiredColors)
    colorSet = colorSet.union(addedColors)
    return colorSet

def createFileOfNewColors(setOfNewColors):
    orderedListOfColors = sorted(setOfNewColors)
    orderedListOfColorsString =('\n').join(orderedListOfColors)
    outfile = open("NewColors.txt", 'w')
    outfile.write(orderedListOfColorsString)
    outfile.close()

main()
```

35.
```
def main():
    ## Display the largest number in the file Numbers.txt
    max = getMax("Numbers.txt")
    print("The largest number in the \nfile Numbers.txt is",
          str(max) + ".")

def getMax(fileName):
    infile = open("Numbers.txt", 'r')
    max = int(infile.readline())
    for line in infile:
        num = int(line)
        if num > max:
            max = num
    infile.close()
    return max

main()
```

```
The largest number in the
file Numbers.txt is 9.
```

37.
```
def main():
    ## Display the sum of the numbers in the file Numbers.txt.
    sum = getSum("Numbers.txt")
    print("The sum of the numbers in \nthe file Numbers.txt is",
          str(sum) + ".")

def getSum(fileName):
    infile = open("Numbers.txt", 'r')
    sum = 0
    for line in infile:
        sum += int(line)
    infile.close()
    return sum

main()
```

```
The sum of the numbers in
the file Numbers.txt is 30.
```

39.
```
def main():
    ## Display the last number in the file Numbers.txt.
    lastNumber = getLastNumber("Numbers.txt")
    print("The last number in the \nfile Numbers.txt is",
           str(lastNumber) + '.')

def getLastNumber(fileName):
    infile = open("Numbers.txt", 'r')
    for line in infile:
        pass
    lastNumber = eval(line)
    infile.close()
    return lastNumber

main()
```

```
The last number in the
file Numbers.txt is 4.
```

41.
```
import os

## Delete colors having more than six characters.
infile = open("ShortColors.txt", 'r')
outfile = open("Temp.txt", 'w')
for color in infile:
    if len(color.rstrip()) <= 6:
        outfile.write(color)
infile.close()
outfile.close()
os.remove("ShortColors.txt")
os.rename("Temp.txt", "ShortColors.txt")
```

43.
```
def main():
    ## Create alphabetical file of last 37 states to join the union.
    lastStates = getListOfLastStates()
    createFileOfLastStates(lastStates)

def getListOfLastStates():
    infile = open("AllStates.txt", 'r')
    states = {state.rstrip() for state in infile}
    infile.close()
    infile = open("FirstStates.txt", 'r')
    infile.close()
    firstStates = {state.rstrip() for state in infile}
    lastStates = list(states.difference(firstStates))
    lastStates.sort()
    return lastStates

def createFileOfLastStates(lastStates):
    outfile = open("LastStates.txt", 'w')
    for state in lastStates:
        outfile.write(state + "\n")
    outfile.close()

main()
```

45.
```
def main():
    ## Display a range of presidents.
    lowerNumber, upperNumber = getRange()
    displayPresidents(lowerNumber, upperNumber)

def getRange():
    lowerNumber = int(input("Enter the lower number for the range: "))
    upperNumber = int(input("Enter the upper number for the range: "))
    return (lowerNumber, upperNumber)

def displayPresidents(lowerNumber, upperNumber):
    infile = open("USPres.txt", 'r')
    count = 0
    for pres in infile:
        count += 1
        if lowerNumber <= count <= upperNumber:
            print(" ", count, pres, end="")
    infile.close()

main()
```

```
Enter the lower number for the range: 40
Enter the upper number for the range: 44
  40 Ronald Reagan
  41 George H. W. Bush
  42 Bill Clinton
  43 George W. Bush
  44 Barack Obama
```

EXERCISES 5.2

1. **The area of Afghanistan is 251,772 sq. miles.**
   The area of Albania is 11,100 sq. miles.

3. **Afghanistan,Asia,251772**
   Albania,Europe,11100

5. Each line of the new file contains the name of a European country and its population in millions. The countries in descending order by population. The first two lines of the file contain the data **Russian Federation,142.5** and **Germany,81.0.**

7.
```
def main():
    ## Display information about a DOW stock.
    symbols = placeSymbolsIntoList("DOW.txt")
    displaySymbols(symbols)
    print()
    symbol = input("Enter a symbol: ")
    infile = open("DOW.txt", 'r')
    abbrev = ""
    while abbrev != symbol:
        line = infile.readline()
        lineList = line.split(',')
        abbrev = lineList[1]
    infile.close()
    print("Company:", lineList[0])
    print("Industry:", lineList[3])
    print("Exchange:", lineList[2])
    increase = ((float(lineList[5]) - float(lineList[4])) /
                float(lineList[4]))
    print("Growth in 2013: {0:0,.2f}%".format(100 * increase))
    priceEarningsRatio = float(lineList[5]) / float(lineList[6])
    print("Price/Earnings ratio in 2013: {0:0,.2f}".
                      format(priceEarningsRatio))

def placeSymbolsIntoList(fileName):
    symbolList = [""] * 30
    infile = open(fileName, 'r')
    for i in range(30):
        line = infile.readline()
        lineList = line.split(',')
        symbolList[i] = lineList[1]
    infile.close()
    return symbolList

def displaySymbols(symbols):
    ## Display symbols in alphabetical order
    symbols.sort()
    print("Symbols for the Thirty DOW Stocks")
    for symbol in symbols:
        print("{0:5} \t".format(symbol), end='')

main()
```

```
Symbols for the Thirty DOW Stocks
AXP      BA     CAT    CSCO   CVX    DD     DIS    GE     GS     HD
IBM      INTC   JNJ    JPM    KO     MCD    MMM    MRK    MSFT   NKE
PFE      PG     T      TRV    UNH    UTX    V      VZ     WMT    XOM

Enter a symbol: MSFT
Company: Microsoft
Industry: Software
Exchange: NASDAQ
Growth in 2013: 40.06%
Price/Earnings ratio in 2013: 14.22
```

9.
```
def main():
    ## Determine the dogs of the DOW.
    stockList = placeDataIntoList("DOW.txt")
    stockList.sort(key=byDividendToPriceRatio, reverse=True)
    displayDogs(stockList)

def placeDataIntoList(fileName):
    infile = open(fileName, 'r')
    listOfLines = [line.rstrip() for line in infile]
    infile.close()
    for i in range(len(listOfLines)):
        listOfLines[i] = listOfLines[i].split(',')
        listOfLines[i][4] = eval(listOfLines[i][4])
        listOfLines[i][5] = eval(listOfLines[i][5])
        listOfLines[i][6] = eval(listOfLines[i][6])
        listOfLines[i][7] = eval(listOfLines[i][7])
    return listOfLines

def byDividendToPriceRatio(stock):
    return stock[7] / stock[5]

def displayDogs(listOfStocks):
    print("{0:25} {1:11} {2:s}".
              format("Company", "Symbol", "Yield as of 12/31/2013"))
    for i in range(10):
        print("{0:25} {1:11} {2:0.2f}%".format(listOfStocks[i][0],
         listOfStocks[i][1], 100 * listOfStocks[i][7] / listOfStocks[i][5]))

main()
```

```
Company                   Symbol      Yield as of 12/31/2013
AT&T                      T           5.15%
Verizon                   VZ          4.19%
Intel                     INTC        3.47%
Merck                     MRK         3.46%
McDonald's                MCD         3.22%
Cisco Systems             CSCO        3.21%
Chevron Corporation       CVX         3.20%
Pfizer                    PFE         3.20%
Procter & Gamble          PG          3.06%
Microsoft                 MSFT        2.86%
```

11.
```
def main():
    ## Display justices appointed by a given president.
    president = input("Enter the name of a president: ")
    justices = getJusticesByPresident(president)
    fixCurrentJustices(justices)
    justices.sort(key=lambda justice: justice[5] - justice[4], reverse=True)
    if len(justices) > 0:
        print("Justices Appointed:")
        for justice in justices:
            print("  " + justice[0] + " " + justice[1])
    else:
        print(president, "did not appoint any justices.")
```

```
def getJusticesByPresident(president):
    infile = open("Justices.txt", 'r')
    listOfRecords = [line for line in infile
                     if line.split(',')[2] == president]
    infile.close()
    for i in range(len(listOfRecords)):
        listOfRecords[i] = listOfRecords[i].split(',')
        listOfRecords[i][4] = int(listOfRecords[i][4])
        listOfRecords[i][5] = int(listOfRecords[i][5])
    return listOfRecords

def fixCurrentJustices(justices):
    for justice in justices:
        if justice[5] == 0:
            justice[5] = 2015

main()
```

```
Enter the name of a president: Barack Obama
Justices Appointed:
  Sonia Sotomayor
  Elena Kagan
```

13.
```
def main():
    ## Makeup of Supreme Court in 1980.
    infile = open("Justices.txt", 'r')
    justices = [line for line in infile
                if (int(line.split(',')[4]) < 1980)
                and (int(line.split(',')[5]) >= 1980)]
    justices.sort(key=lambda x: int(x.split(',')[4]))
    print("{0:20} {1}".format("Justice", "Appointing President"))
    for justice in justices:
        print("{0:20} {1}".format(justice.split(',')[0] + " " +
        justice.split(',')[1], justice.split(',')[2]))

main()
```

```
Justice              Appointing President
William Brennan      Dwight Eisenhower
Potter Stewart       Dwight Eisenhower
Byron White          John Kennedy
Thurgood Marshall    Lyndon Johnson
Warren Burger        Richard Nixon
Harry Blackman       Richard Nixon
Lewis Powell         Richard Nixon
William Rehnquist    Richard Nixon
John Stevens         Gerald Ford
```

15.
```
def main():
    ## Twelve Days of Christmas
    listOfDaysCosts = createListOfDaysCosts()
    day = int(input("Enter a number from 1 through 12: "))
    displayOutput(day, listOfDaysCosts)

def createListOfDaysCosts():
    infile = open("Gifts.txt", 'r')
    costs = [float(line.split(',')[2]) for line in infile]
    infile.close()
    listOfDaysCosts = [0] * 12
    for i in range(12):
        listOfDaysCosts[i] = (i + 1) * costs[i]
    return listOfDaysCosts

def displayOutput(day, listOfDaysCosts):
    print("The gifts for day", day, "are")
    infile = open("Gifts.txt", 'r')
    for i in range(day):
        data = infile.readline().split(',')
        print(int(data[0]), data[1])
    print()
    print("Cost for day {0}: ${1:,.2f}".
          format(day, sum(listOfDaysCosts[:day])))
    totalCosts = 0
    for i in range(day):
        totalCosts += sum(listOfDaysCosts[:i + 1])
    print("Total cost for the first {0} days: ${1:,.2f}"
          .format(day, totalCosts))

main()
```

```
Enter a number from 1 through 12: 4
The gifts for day 4 are
1 partridge in a pear tree
2 turtle doves
3 French hens
4 calling birds

Cost for day 4: $1,114.14
Total cost for the first 4 days: $2,168.68
```

17.
```
def main():
    ## Display colleges from requested state.
    colleges = getOrderedListOfColleges()
    displayListOfColleges(colleges)

def getOrderedListOfColleges():
    infile = open("Colleges.txt", 'r')
    colleges = [line.rstrip() for line in infile]
    infile.close()
    colleges.sort()
    return colleges
```

```
def displayListOfColleges(colleges):
    found = False
    abbrev = input("Enter a state abbreviation: ")
    for college in colleges:
        college = college.split(",")
        if college[1] == abbrev:
            print(college[0], college[2])
            found = True
    if not found:
        print("There are no early colleges from ", abbrev, ".", sep="")

main()
```

```
Enter a state abbreviation: VA
Hampton-Sydney College 1776
Washington and Lee University 1749
William and Mary College 1693
```

19.
```
def main():
    ## Find states whose name and capital begin with the same letter.
    infile = open("StatesANC.txt", 'r')
    for line in infile:
        data = line.split(",")
        letter = data[0][0:1]
        if data[3].startswith(letter):
            print((data[3].rstrip()) + ",", data[0])
    infile.close()

main()
```

```
Dover, Delaware
Honolulu, Hawaii
Indianapolis, Indiana
Oklahoma City, Oklahoma
```

21.
```
def main():
    ## Display Oscar-winning films of requested genre.
    displayGenres()
    displayFilms()

def displayGenres():
    print("The different film genres are as follows:")
    print("{0:12}{1:12}{2:10}{3:11}{4:11}".
          format("adventure", "bioptic", "comedy", "crime", "drama"))
    print("{0:12}{1:12}{2:10}{3:11}{4:11}".
          format("epic", "fantasy", "musical", "romance", "silent"))
    print("{0:12}{1:12}{2:10}{3:11}".
          format("sports", "thriller", "war", "western"))
    print()
```

```
def displayFilms():
    films = open("Oscars.txt",'r')
    genre = input("Enter a genre: ")
    print()
    print("The Academy Award winners are")
    for line in films:
        if line.endswith(genre + "\n"):
            temp = line.split(",")
            print("  " + temp[0])
    films.close()

main()
```

```
The different film genres are as follows:
adventure  bioptic   comedy   crime    drama
epic       fantasy   musical  romance  silent
sports     thriller  war      western

Enter a genre: sports

The Academy Award winners are
  Rocky
  Million Dollar Baby
```

23.
```
def main():
    ## Create file of articles purchased by cowboys.
    articles = ["Colt Peacemaker,12.20\n", "Holster,2.00\n",
        "Levi Strauss jeans,1.35\n", "Saddle,40.00\n", "Stetson,10.00\n"]
    outfile = open("Cowboy.txt", 'w')
    outfile.writelines(articles)
    outfile.close()

main()
```

25.
```
def main():
    ## Create receipt
    createOrderFile()
    total = 0
    infile1 = open("Cowboy.txt", 'r')
    infile2 = open("Order.txt", 'r')
    for line in infile1:
        quantity = int(infile2.readline())
        cost = quantity * float(line.split(',')[1])
        print("{0} {1}: ${2:,.2f}".format(quantity, line.split(',')[0],
                                          cost))
        total += cost
    print("{0}: ${1:,.2f}".format("TOTAL", total))

def createOrderFile():
    orders = ["3\n", "2\n", "10\n", "1\n", "4\n"]
    outfile = open("Order.txt", 'w')
    outfile.writelines(orders)
    outfile.close()

main()
```

```
27. def main():
        ## Determine the day of the week for a date.
        infile = open("Calendar2015.txt", 'r')
        date = input("Enter a date in 2015: ")
        for line in infile:
            temp = line.split(',')
            if temp[0] == date:
                print(date, "falls on a",  temp[1].rstrip())
                break

    main()
```

```
Enter a date in 2015: 7/4/2015
7/4/2015 falls on a Saturday
```

EXERCISES 5.3

1. 6.5 3. ['NH', 'CT', 'ME', 'VT', 'MA', 'RI']

5. [('NH', 1.5), ('CT', 3.6), ('ME', 1.3), ('VT', 0.6), ('MA', 6.5), ('RI', 1.1)]

7. absent 9. VT 11. 2 13. 2 15. VT CT MA RI ME NH

17. 14.6 19. 5 21. 2 23. False 25. Aaron

27. ['Aaron', 'Bonds'] 29. [755, 762] 31. 762 33. {'Aaron': 755}

35. 0 37. Bonds 39. 762 41. 762
    Aaron 755

43. {'Bonds': 761, 'Aaron': 755, 'Ruth': 714}

```
45. pres = input("Who was the youngest U.S. president? ")
    pres = pres.upper()
    trResponse = "Correct. He became president at age 42\n" + \
                 "when President McKinley was assassinated."
    jfkResponse = "Incorrect. He became president at age 43. However,\n" + \
                  "he was the youngest person elected president."
    responses = {}
    responses["THEODORE ROOSEVELT"] = trResponse
    responses["TEDDY ROOSEVELT"] = trResponse
    responses["JFK"] = jfkResponse
    responses["JOHN KENNEDY"] = jfkResponse
    responses["JOHN F. KENNEDY"] = jfkResponse
    print(responses.get(pres, "Nope."))
```

```
47. def main():
        ## Display batting averages of top hitters.
        topHitters = {"Gehrig":{"atBats":8061, "hits":2721},
                      "Ruth":{"atBats":8399, "hits":2873},
                      "Williams":{"atBats":7706, "hits":2654}}
        displayBattingAverage(topHitters)
```

```
def displayBattingAverage(topHitters):
    for hitter in topHitters:
        print("{0:10} {1:.3f}".format(hitter,
              topHitters[hitter]["hits"] / topHitters[hitter]["atBats"]))

main()
```

```
Ruth       0.342
Williams   0.344
Gehrig     0.338
```

49.
```
def main():
    ## Display average number of hits by the top three hitters.
    topHitters = {"Gehrig":{"atBats":8061, "hits":2721},
                  "Ruth":{"atBats":8399, "hits":2873},
                  "Williams":{"atBats":7706, "hits":2654}}
    displayAveNumberOfHits(topHitters)

def displayAveNumberOfHits(topHitters):
    hitList = []
    for hitter in topHitters:
        hitList.append(topHitters[hitter]["hits"])
    value = "{0:.1f}".format(sum(hitList) / len(hitList))
    print("The average number of hits by")
    print("the baseball players was", value + '.')

main()
```

```
The average number of hits by
the baseball players was 2749.3.
```

51.
```
import pickle

def main():
    ## Display justices appointed by a specified president.
    justicesDict = createDictFromFile("JusticesDict.dat")
    displayPresidentialAppointees(justicesDict)

def createDictFromFile(fileName):  # from binary file
    infile = open(fileName, 'rb')
    dictionaryName = pickle.load(infile)
    infile.close()
    return dictionaryName

def displayPresidentialAppointees(dictionaryName) :
    pres = input("Enter a president: ")
    for x in dictionaryName:
        if dictionaryName[x]["pres"] == pres:
            print("  {0:16} {1:d}"
                  .format(x, dictionaryName[x]["yrAppt"]))

main()
```

```
Enter a president: Ronald Reagan
  Anthony Kennedy  1987
  Sandra O'Connor  1981
  Antonin Scalia   1986
```

53.
```
import pickle

def main():
    ## Display information about a specific justice.
    justicesDict = createDictFromFile("JusticesDict.dat")
    displayInfoAboutJustice(justicesDict)

def createDictFromFile(fileName):  # from binary file
    infile = open(fileName, 'rb')
    dictionaryName = pickle.load(infile)
    infile.close()
    return dictionaryName

def displayInfoAboutJustice(dictionaryName):
    justice = input("Enter name of a justice: ")
    print("Appointed by", dictionaryName[justice]["pres"])
    print("State:", dictionaryName[justice]["state"])
    print("Year of appointment:", dictionaryName[justice]["yrAppt"])
    if dictionaryName[justice]["yrLeft"] == 0:
        print("Currently serving on the Supreme Court.")
    else:
        print("Left court in", dictionaryName[justice]["yrLeft"])

main()
```

```
Enter name of a justice: Anthony Kennedy
Appointed by Ronald Reagan
State: CA
Year of appointment: 1987
Currently serving on the Supreme Court.
```

55.
```
def main():
    ## Calculate letter frequencies for a sentence.
    sentence = input("Enter a sentence: ")
    sentence = sentence.upper()
    letterDict = dict([(chr(n),0) for n in range(65, 91)])
    for char in sentence:
        if 'A' <= char <= 'Z':
            letterDict[char] += 1
    displaySortedResults(letterDict)

def displaySortedResults(dictionaryName):
    letterList = list(dictionaryName.items())
    letterList.sort(key=f,reverse=True)
    for x in letterList:
        if x[1] != 0:
            print("  " + x[0] + ':', x[1])

def f(k):
    return k[1]

main()
```

```
Enter a sentence: To fail to plan is to plan to fail.
  O: 4
  A: 4
  L: 4
  T: 4
  I: 3
  N: 2
  P: 2
  F: 2
  S: 1
```

57.
```
import pickle

def main():
    ## Determine states that were home to three or more presidents.
    presidents = getDictionary("USpresStatesDict.dat")
    states = createStatesDict(presidents)
    sortedStates = [state for state in states if states[state] > 2]
    sortedStates.sort(key=lambda state: states[state], reverse=True)
    print("States that produced three or")
    print("more presidents as of 2016:")
    for state in sortedStates:
        print(" ", state + ":", states[state])

def getDictionary(fileName):
    infile = open(fileName, 'rb')
    dictName = pickle.load(infile)
    infile.close()
    return dictName

def createStatesDict(presidents):
    states = {}
    for state in presidents.values():
        if not states.get(state, False):
            states[state] = 1
        else:
            states[state] += 1
    return states

main()
```

```
States that produced three or
more presidents as of 2016:
  Ohio: 6
  New York: 6
  Virginia: 5
  Massachusetts: 4
  Tennessee: 3
  California: 3
  Texas: 3
  Illinois: 3
```

59.
```
def main():
    ## Determine the day of the week for a date.
    calendar2015Dict = createDictionary("Calendar2015.txt")
    date = input("Enter a date in 2015: ")
    print(date, "falls on a", calendar2015Dict[date])

def createDictionary(fileName):
    infile = open(fileName, 'r')
    textList = [line.rstrip() for line in infile]
    infile.close()
    return dict([x.split(',') for x in textList])

main()
```

```
Enter a date in 2015: 2/14/2015
11/3/2015 falls on a Saturday
```

61.
```
import pickle

def main():
    ## Determine states having a specified number of large cities.
    largeCities = createDictionaryFromBinaryFile("LargeCitiesDict.dat")
    number = int(input("Enter an integer from 1 to 13: "))
    states = sorted(getStates(number, largeCities))
    displayResult(number, states)

def createDictionaryFromBinaryFile(fileName):
    infile = open(fileName, 'rb')
    dictionaryName = pickle.load(infile)
    infile.close()
    return dictionaryName

def getStates(number,dictionaryName):
    states = []
    for state in dictionaryName:
        if len(dictionaryName[state]) == number:
            states.append(state)
    return states

def displayResult(number, states):
    if len(states) == 0:
        print("No states have exactly", number, "large cities.")
    else:
        print("The following states have exactly", number, "large cities:")
        print("  ".join(states))

main()
```

```
Enter an integer from 1 to 13: 4
The following states have exactly 4 large cities:
Ohio
```

CHAPTER 6

EXERCISES 6.1

1. f 3. l 5. B 7. i 9. s 11. o 13. g 15. n 17. d 19. h 21. r

23.
```
You must enter a number.
```

25.
```
string index out of range
Oops
```

27.
```
File Salaries.txt contains an invalid salary.
Thank you for using our program.
```

29.
```
while True:
    try:
        n = int(input("Enter a nonzero integer: "))
        reciprocal = 1 / n
        print("The reciprocal of {0} is {1:,.3f}".format(n, reciprocal))
        break
    except ValueError:
        print("You did not enter a nonzero integer. Try again.")
    except ZeroDivisionError:
        print("You entered zero. Try again.")
```

```
Enter a nonzero integer: 0
You entered zero. Try again.
Enter a nonzero integer: eight
You did not enter a nonzero integer. Try again.
Enter a nonzero integer: 8
The reciprocal of 8 is 0.125
```

31.
```
while True:
    try:
        num = int(input("Enter an integer from 1 to 100: "))
        if 1 <= num <= 100:
            print("Your number is", str(num) + '.')
            break
        else:
            print("Your number was not between 1 and 100.")
    except ValueError:
        print("You did not enter an integer.")
```

```
Enter an integer from 1 to 100: 5.5
You did not enter an integer.
Enter an integer from 1 to 100: five
You did not enter an integer.
Enter an integer from 1 to 100: 555
Your number was not between 1 and 100.
Enter an integer from 1 to 100: 5
Your number is 5.
```

EXERCISES 6.2

1. A free throw by a basketball player who makes 75% of his or her free throws.

3. The result of an at-bat by a baseball player with a 0.275 batting average.

5. The random selection of two people to be co-chairs of a club.

7. Randomly assigning starting positions in a one-mile race.

9.
```
## Select three letters at random from the alphabet.
# Create a list of the 26 uppercase letters of the alphabet.
list1 = [chr(n) for n in range(ord('A'), ord('Z') + 1)]
# Select three letters at random.
list2 = random.sample(list1, 3)
# Display the three letters
print(", ".join(list2))
```

11.
```
## Randomly select two even numbers from 2 through 100.
# Create a list of the even numbers from 2 through 100.
list1 = [n for n in range(2, 101, 2)]
# Select two of the even numbers at random.
list2 = random.sample(list1, 2)
# Display the two numbers.
print(list2[0], list2[1])
```

13.
```
## Count the number of "Heads" in 100 coin tosses.
numberOfHeads = 0
for i in range(100):
    if (random.choice(["Head","Tail"]) == "Head"):
        numberOfHeads += 1
print("In 100 tosses, Heads occurred {0} times.".format(numberOfHeads))
```

15.
```
import random
## Select three states at random from a file containing the 50 states.
allNumbers = [n for n in range(1, 51)]
# Randomly select three numbers from 1 through 50.
threeNumbers = random.sample(allNumbers, 3)
infile = open("StatesAlpha.txt", 'r')
lineNumber = 1    for line in infile:
    if lineNumber in threeNumbers:
        print(line.rstrip())
    lineNumber += 1
infile.close()
```

```
Illinois
New Hampshire
South Dakota
```

Possible output.

17.
```
import random
import pickle

NUMBER_OF_TRIALS = 10000

def main():
    ## Carry out matching process NUMBER_OF_TRIALS times.
    totalNumberOfMatches = 0
    for i in range(NUMBER_OF_TRIALS):
        totalNumberOfMatches += matchTwoDecks()
    averageNumberOfMatches = totalNumberOfMatches / NUMBER_OF_TRIALS
    print("The average number of cards that")
    print("matched was {0:.3f}.".format(averageNumberOfMatches))

def matchTwoDecks():
    ## Determine the number of matches when comparing
    ## two shuffled decks of cards.
    # Create two decks as lists using the binary file
    # DeckOfCardsList.dat from Example 2.
    infile = open("DeckOfCardsList.dat", 'rb')
    deck1 = pickle.load(infile)
    infile.close()
    infile = open("DeckOfCardsList.dat", 'rb')
    deck2 = pickle.load(infile)
    infile.close()
```

```
    # Shuffle both decks of cards.
    random.shuffle(deck1)
    random.shuffle(deck2)
    # Compare cards and determine the number of matches.
    numberOfMatches = 0
    for i in range(52):
        if (deck1[i] == deck2[i]):
            numberOfMatches += 1
    return numberOfMatches

main()
```

```
The average number of cards
that matched was 1.002.
```

19.
```
import random
## Simulate a Powerball Drawing.
whiteBalls = [num for num in range(1, 60)]
# Randomly sample and display five white balls.
whiteBallSelection = random.sample(whiteBalls, 5)
for i in range(5):
    whiteBallSelection[i] = str(whiteBallSelection[i])
print("White Balls:", " ".join(whiteBallSelection))
# Randomly select and display the Powerball.
powerBall = random.randint(1, 35)
print("Powerball:", powerBall)
```

```
White Balls: 15 48 38 22 20
Powerball: 2
```

21.
```
import random
## Simulate 32 coin tosses and check for runs of length five.
coin = ['T', 'H']
result = ""
for i in range(32):
    result += random.choice(coin)
print(result)
if ("TTTTT" in result) or ("HHHHH" in result):
    print("There was a run of five consecutive")
    print("same outcomes.")
else:
    print("There was no run of five consecutive same outcomes.")
```

```
HTTTTHTHTTTTHHHHTTHTTHHTHTHTTTHH
There was not a run of five
consecutive
```

23.
```
import random
import pickle

def main():
    ## Calculate the High Point Count for a bridge hand.
    bridgeHand = getHand()
    print(", ".join(bridgeHand))  # Display the bridge hand.
    HCP = calculateHighCardPointCount(bridgeHand)
    print("HPC =", HCP)
```

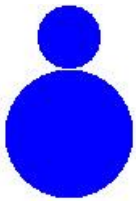
```
def getHand():
    infile = open("DeckOfCardsList.dat", 'rb')
    deckOfCards = pickle.load(infile)
    infile.close()
    bridgeHand = random.sample(deckOfCards, 13)
    return bridgeHand

def calculateHighCardPointCount(bridgeHand):
    countDict = {'A':4, 'K':3, 'Q':2, 'J':1}
    HPC = 0
    for card in bridgeHand:
        rank = card[0]  # Each card is a string of
                        # two characters.
    if rank in "AKQJ":
        HPC += countDict[rank]
    return HPC

main()
```

```
4♦, J♣, K♠, 4♥, 7♦, 3♣, 7♠, 6♣, 3♥, 8♥, Q♦, J♥, K♦
HPC = 10
```

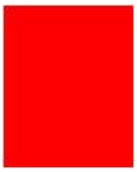
EXERCISES 6.3

1.
```
import turtle
t = turtle.Turtle()
t.pencolor("blue")
t.hideturtle()
t.up()
t.goto(20, 30)
t.dot(5)
t.down()
t.goto(80, 90)
t.dot(5)
```

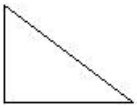
3.
```
import turtle
t = turtle.Turtle()
t.hideturtle()
t.dot(80, "blue")
t.up()
t.goto(0, 60)
t.dot(40, "blue")
```



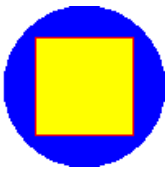
5.
```
import turtle
t = turtle.Turtle()
t.hideturtle()
t.color("red", "red")
t.up()
t.goto(-30, -40)
t.down()
t.begin_fill()
t.goto(-30, 60)
t.goto(50, 60)
t.goto(50, -40)
t.goto(-30, -40)
t.end_fill()
```



7.
```
import turtle
t = turtle.Turtle()
t.hideturtle()
t.goto(0, 60)
t.goto(80, 0)
t.goto(0, 0)
```



9.
```
import turtle

def main():
    ## Draw a yellow square inside a blue dot.
    t = turtle.Turtle()
    t.hideturtle()
    drawDot(t, 50, 50, 100, "blue")
    drawFilledRectangle(t, 20, 20, 60, 60, "red", "yellow")

def drawFilledRectangle(t, x, y, w, h, colorP="black", colorF="black"):
    ## Draw a filled rectangle with bottom-left corner (x, y),
    ## width w, height h, pen color colorP, and fill color colorF.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()                  # Disable drawing of lines.
    t.goto(x, y)            # Move to bottom-left corner of rectangle.
    t.down()                # Enable drawing of lines.
    t.begin_fill()
    t.goto(x + w, y)        # Draw line to bottom-right corner.
    t.goto(x + w, y + h)    # Draw line to top-right corner.
    t.goto(x, y + h)        # Draw line to top-left corner.
    t.goto(x, y)            # Draw line to bottom-left corner.
    t.end_fill()

def drawDot(t, x, y, diameter, colorP):
    ## Draw dot with center (x, y) and color colorP.
    t.up()
    t.goto(x, y)
    t.dot(diameter, colorP)

main()
```

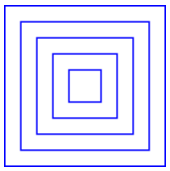


11.
```
import turtle

def main():
    t = turtle.Turtle()
    t.speed(10)
    t.hideturtle()
    colors = ["black", "white", "dark blue", "red", "yellow"]
    diameter = 300
    for color in colors:
        t.pencolor(color)
        t.dot(diameter)
    diameter -= 60

main()
```

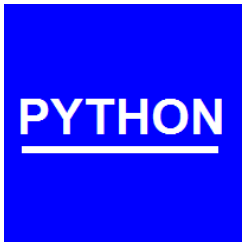
13.
```
import turtle

def main():
    ## Draw a partial moon.
    t = turtle.Turtle()
    t.hideturtle()
    drawDot(t, 0, 0, 200, "orange")     # Draw moon.
    drawDot(t, -100,0, 200, "white")    # Take bite out of moon.

def drawDot(t, x, y, diameter, colorP):
    ## Draw a dot with center (x, y) having color colorP.
    t.up()
    t.goto(x, y)
    t.dot(diameter, colorP)

main()
```

15.
```
import turtle

def main():
    ## Draw nested set of five squares.
    t = turtle.Turtle()
    t.hideturtle()
    for i in range(1, 6):
        drawRectangle(t, -10 * i, -10 * i, 20 * i, 20 * i, "blue")
```

```
def drawRectangle(t, x, y, w, h, colorP="black"):
    ## Draw a rectangle with bottom-left corner (x, y),
    ## width w, height h, and pencolor colorP.
    t.pencolor(colorP)
    t.up()
    t.goto(x, y)          # start at bottom-left corner of rectangle
    t.down()
    t.goto(x + w, y)      # draw line to bottom-right corner
    t.goto(x + w, y + h)  # draw line to top-right corner
    t.goto(x, y + h)      # draw line to top-left corner
    t.goto(x, y)          # draw line to bottom-left corner

main()
```

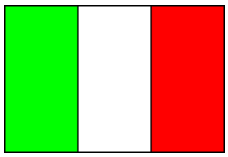


17.
```
import turtle

def main():
    ## Draw a blue square containing the underlined word PYTHON.
    t = turtle.Turtle()
    t.hideturtle()
    drawFilledRectangle(t, 0, 0, 200, 200, "blue", "blue")   # Square
    drawFilledRectangle(t, 15, 75, 165, 5, "white", "white") # Underline
    t.up()
    t.goto(100, 80)
    t.pencolor("white")
    t.write("PYTHON", align="center", font=("Arial", 25, "bold"))

def drawFilledRectangle(t, x, y, w, h, colorP="black", colorF="black"):
    ## Draw a filled rectangle with bottom-left corner (x, y),
    ## width w, height h, pen color colorP, and fill color colorF.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x, y)          # Start at bottom-left corner of rectangle.
    t.down()
    t.begin_fill()
    t.goto(x + w, y)      # Draw line to bottom-right corner.
    t.goto(x + w, y + h)  # Draw line to top-right corner.
    t.goto(x, y + h)      # Draw line to top-left corner.
    t.goto(x, y)          # Draw line to bottom-left corner.
    t.end_fill()

    main()
```

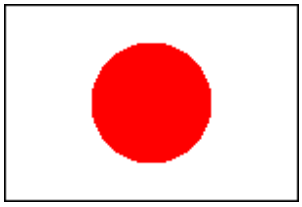


19.
```
import turtle

def main():
    t = turtle.Turtle()
    t.hideturtle()
    drawFilledRectangle(t, 0, 0, 200, 40)
    t.goto(100,0)
    t.pencolor("white")
    t.write("PYTHON", align="center", font=("Ariel", 20, "italic bold"))

def drawFilledRectangle(t, x, y, w, h, colorP="black", colorF="black"):
    ## Draw a filled rectangle with bottom-left corner (x, y),
    ## width w, height h, pen color colorP, and fill color colorF.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x, y)          # start at bottom-left corner of rectangle
    t.down()
    t.begin_fill()
    t.goto(x + w, y)      # draw line to bottom-right corner
    t.goto(x + w, y + h)  # draw line to top-right corner
    t.goto(x, y + h)      # draw line to top-left corner
    t.goto(x, y)          # draw line to bottom-left corner
    t.end_fill()

main()
```

21.
```
import turtle
def main():
    ## Draw the Italian flag.
    t = turtle.Turtle()
    t.hideturtle()
    drawFilledRectangle(t, 0, 0, 50, 100, "black", "green")
    drawFilledRectangle(t, 50, 0, 50, 100, "black", "white")
    drawFilledRectangle(t, 100, 0, 50, 100, "black", "red")
def drawFilledRectangle(t, x, y, w, h, colorP="black", colorF="black"):
    ## Draw a filled rectangle with bottom-left corner (x, y),
    ## width w, height h, pen color colorP, and fill color colorF.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x, y)          # Start at bottom-left corner of rectangle.
    t.down()
    t.begin_fill()
    t.goto(x + w, y)      # Draw line to bottom-right corner.
    t.goto(x + w, y + h)  # Draw line to top-right corner.
    t.goto(x, y + h)      # Draw line to top-left corner.
    t.goto(x, y)          # Draw line to bottom-left corner.
    t.end_fill()

main()
```



23.
```
import turtle

def main():
    ## Draw flag of Japan.
    t = turtle.Turtle()
    t.hideturtle()
    drawFilledRectangle(t, 0, 0, 150, 100, "black", "white")
    t.up()
    t.goto(75,50)
    t.color("red")
    t.dot(62)

def drawFilledRectangle(t, x, y, w, h, colorP="black", colorF="black"):
    ## Draw a filled rectangle with bottom-left corner (x, y),
    ## width w, height h, pen color colorP, and fill color colorF.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x, y)          # start at bottom-left corner of rectangle
    t.down()
    t.begin_fill()
    t.goto(x + w, y)      # draw line to bottom-right corner
    t.goto(x + w, y + h)  # draw line to top-right corner
    t.goto(x, y + h)      # draw line to top-left corner
    t.goto(x, y)          # draw line to bottom-left corner
    t.end_fill()

main()
```

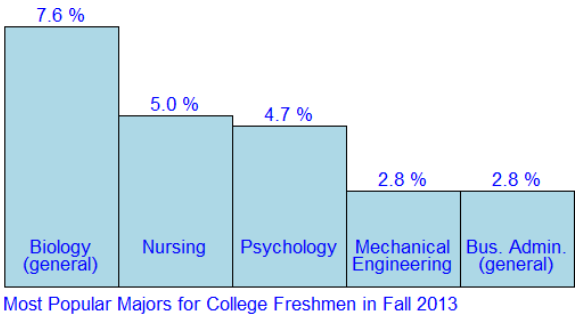


25.
```
import turtle

def main():
    ## Draw the flag of Burkina Faso.
    t = turtle.Turtle()
    t.hideturtle()
    t.down()
    drawFilledRectangle(t, 0, 50, 150, 50, "red", "red")
    drawFilledRectangle(t, 0, 0, 150, 50, "forest green", "forest green")
    drawFivePointStar(t, 65, 33, 40, "yellow", "yellow")
```

```
def drawFivePointStar(t, x, y, lenthOfSide, colorP="black",
                      colorF="white"):
    # Drawing begins at (x, y) and moves in a north-east direction.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x, y)
    t.setheading(0)
    t.left(36)
    t.down()
    t.begin_fill()
    for i in range(6):
        t.forward(lenthOfSide)
        t.left(144)   # 144 = 180 - 36
    t.end_fill()

def drawFilledRectangle(t, x, y, w, h, colorP="black",
                        colorF="black"):
    ## Draw a filled rectangle with bottom-left corner (x, y),
    ## width w, height h, pen color colorP, and fill color colorF.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x, y)          # Start at bottom-left corner of rectangle.
    t.down()
    t.begin_fill()
    t.goto(x + w, y)      # Draw line to bottom-right corner.
    t.goto(x + w, y + h)  # Draw line to top-right corner.
    t.goto(x, y + h)      # Draw line to top-left corner.
    t.goto(x, y)          # Draw line to bottom-left corner.
    t.end_fill()

main()
```

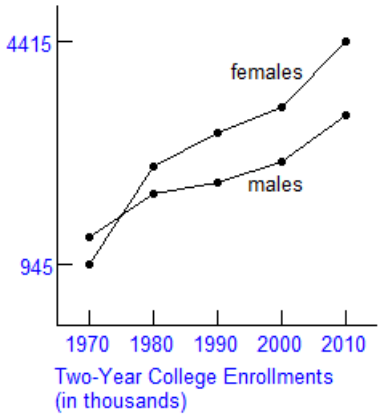
27.
```
import turtle

values = [7.6, 5.0, 4.7, 2.8, 2.8]

def main():
    ## Draw bar chart for popular majors.
    t = turtle.Turtle()
    t.speed(10)
    t.hideturtle()
    for i in range(5):
        height = 30 * values[i]
        drawFilledRectangle(t, (-250 + 100 * i), 0, 100, height,
                            "black", "light blue")
    insertText(t)
```

```
def drawFilledRectangle(t, x, y, w, h, colorP="black", colorF="black"):
    ## Draw a filled rectangle with bottom-left corner (x, y),
    ## width w, height h, pen color colorP, and fill color colorF.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x, y)          # start at bottom-left corner of rectangle
    t.down()
    t.begin_fill()
    t.goto(x + w, y)      # draw line to bottom-right corner
    t.goto(x + w, y + h)  # draw line to top-right corner
    t.goto(x, y + h)      # draw line to top-left corner
    t.goto(x, y)          # draw line to bottom-left corner
    t.end_fill()

def insertText(t):
    t.up()
    labels1 = ["Biology", "Nursing", "Psychology", "Mechanical", "Bus. Admin."]
    labels2 = ["(general)", "", "", "Engineering", "(general)"]
    for i in range(5):
        t.pencolor("blue")
        t.goto(-200 + 100 * i, 30 * values[i])
        t.write(str(values[i]) + '%', align="center",font=("Ariel", 10, "normal"))
        t.goto(-200 + 100 * i, 25)
        t.write(labels1[i], align="center",font=("Ariel", 10, "normal"))
        t.goto(-200 + 100 * i, 10)
        t.write(labels2[i], align="center",font=("Ariel", 10, "normal"))
    t.goto(-250, -25)
    t.write("Most Popular Majors for College Freshmen in Fall 2013",
            font=("Ariel", 10, "normal"))

main()
```



```
29. import turtle

    MALE_ENROLLMENTS = [1375, 2047, 2233, 2559, 3265]
    FEMALE_ENROLLMENTS = [945, 2479, 3007, 3390, 4415]

    def main():
        ## Draw line chart of two-year college enrollments.
        t = turtle.Turtle()
        t.hideturtle()
        drawLine(t, 0, 0, 200, 0)    # Draw x-axis.
        drawLine(t, 0, 0, 0, 200)    # Draw y-axis.
        ## Draw graphs.
        for i in range(4):
            drawLineWithDots(t, 20 + (40 * i), MALE_ENROLLMENTS[i]/ 25,
                             60 + 40 * i, MALE_ENROLLMENTS[i+1]/25, "black")
        for i in range(4):
            drawLineWithDots(t, 20 + (40 * i), FEMALE_ENROLLMENTS[i]/ 25,
                            60 + 40 * i, FEMALE_ENROLLMENTS[i+1]/25, "black")
        drawTickMarks(t)
        insertText(t)

    def drawLine(t, x1, y1, x2, y2, colorP="black"):
        ## Draw line segment from (x1, y1) to (x2, y2) having color colorP.
        t.up()
        t.goto(x1, y1)
        t.down()
        t.color(colorP)
        t.goto(x2, y2)

    def drawLineWithDots(t, x1, y1, x2, y2, colorP="black"):
        ## Draw line segment from (x1, y1) to (x2, y2) having color
        ## colorP and insert dots at both ends of the line segment.
        t.pencolor(colorP)
        t.up()
        t.goto(x1, y1)
        t.dot(5)
        t.down()
        t.goto(x2, y2)
        t.dot(5)

    def drawTickMarks(t):
        for i in range(5):
            drawLine(t, 20 + (40 * i), 0, 20 + 40 * i , 10)
        drawLine(t, 0, max(FEMALE_ENROLLMENTS)/25, 10,
                 max(FEMALE_ENROLLMENTS)/25)
        drawLine(t, 0, min(FEMALE_ENROLLMENTS)/25, 10,
                 min(FEMALE_ENROLLMENTS)/25)
```

```
def insertText(t):
    t.up()
    t.pencolor("black")
    t.goto(110, 150)
    t.write("Females")
    t.goto(120, 80)
    t.write("Males")
    # Display greatest enrollment value.
    t.color("blue")
    t.goto(-30, (max(FEMALE_ENROLLMENTS)/25)-10)
    t.write(max(FEMALE_ENROLLMENTS))
    # Display least enrollment value.
    t.goto(-22, (min(FEMALE_ENROLLMENTS)/25) - 10)
    t.write(min(FEMALE_ENROLLMENTS))
    # Display labels for tick marks on x-axis.
    t.goto(0, -20)
    x = 20
    for i in range(1970, 2011, 10):
        t.goto(x, -20)
        t.write(str(i), align="center")
        x += 40
    # Display title of line chart.
    t.goto(0, -40)
    t.write("Two-Year College Enrollments")
    t.goto(0, -55)
    t.write("(in thousands)")

        main()
```



EXERCISES 6.4

1. 15 3. ****** 5. harpo

7.
```
def isAlpha(L):
    ## Determine whether items in a list are in alphabetical order.
    if len(L) == 1:
        return True
    elif L[0] > L[1]:
        return False
    else:
        return isAlpha(L[1:])
```

9.
```
def main():
    ## Determine the coefficients in a binomial expansion.
    n = int(input("Enter a positive integer: "))
    for r in range(0, n + 1):
        print(C(n, r), end=" ")

def C(n, r):
    if (n == 0) or (r == 0) or (n == r):
        return 1
    else:
        return C(n - 1, r - 1) + C(n - 1, r)
main()
```

```
Enter a positive integer: 6
1  6  15  20  15  6  1
```

11.
```
def main():
    ## Find the greatest common divisor of two non-negative integers.
    m = int(input("Enter the first integer: "))
    n = int(input("Enter the second integer: "))
    print("GCD =", GCD(m, n))

def GCD(m, n):
    if n == 0:
        return m
    else:
        return GCD(n, m % n)

main()
```

```
Enter the first integer: 15
Enter the second integer: 21
GCD = 3
```

13.
```
def main():
    ## Reverse the order of items entered by the user.
    state = ""
    getState(state)

def getState(state):
    state = input("Enter a state: ")
    if state != "End":
        getState(state)
        print(state)

main()
```

```
Enter a state: Maine
Enter a state: Utah
Enter a state: Wyoming
Enter a state: End
Wyoming
Utah
Maine
```

CHAPTER 7

EXERCISES 7.1

1. The *self* parameter is missing from the second line.

3. The pair of parentheses in the first line should be replaced by a colon. Also, a colon should be placed at the end of the second line.

5. 1 7. 4 9. 12.56 11. 18.84

13.
```
import point

def main():
    ## Determine the distance of a point from the origin.
    x = float(input("Enter x-coordinate of point: "))
    y = float(input("Enter y-coordinate of point: "))
    p = point.Point(x, y)
    print("Distance from origin: {0:,.2f}".
          format(p.distanceFromOrigin()))

main()
```
```
Enter the x-coordinate of point: -4
Enter the y-coordinate of point: 3
Distance from origin: 5.00
```

15.
```
import pairOfDice

def main():
    ## Roll a pair of dice.
    dice = pairOfDice.PairOfDice()
    dice.roll()
    print("Red die:", dice.getRedDie())
    print("Blue die:", dice.getBlueDie())
    print("Sum of the dice:", dice.sum())

main()
```
```
Red die: 1
Blue die: 4
Total: 5
```

17.
```
import pairOfDice

def main():
    ## Determine the likelihood of obtaining 7
    ## when rolling a pair of dice.
    numberOfSevens = 0
    for i in range(100000):
        dice = pairOfDice.PairOfDice()
        dice.roll()
        if dice.sum() == 7:
            numberOfSevens += 1
    print("7 occurred {0:.2%} of the time.".
          format(numberOfSevens / 100000))

main()
```

19. queen of hearts 21. 10 of clubs 23. 7 of hearts

25.
```
import pCard
import random

def main():
    ## Randomly select a face card.
    c = pCard.PlayingCard()
    c.selectAtRandom()
    picture = random.choice(["jack", "queen", "king"])
    c.setRank(picture)
    print(c)

main()
```

27.
```
class Fraction:
    def __init__(self, numerator=0, denominator=1):
        self._numerator = numerator
        self._denominator = denominator

    def setNumerator(self, numerator):
        self._numerator = numerator

    def getNumerator(self):
        return self._numerator

    def setDenominator(self, denominator):
        self._denominator = denominator

    def getDenominator(self):
        return self._denominator

    def GCD(self, m, n):   # Greatest Common Divisor
        while n != 0:
            t = n
            n = m % n
            m = t
        return m

    def reduce(self):
        gcd = self.GCD(self._numerator, self._denominator)
        self._numerator = int(self._numerator / gcd)
        self._denominator = int(self._denominator / gcd)
```

29.
```
import fraction

def main():
    ## Convert a decimal number to a fraction.
    decimal = input("Enter a positive decimal number less than 1: ")
    decimal = decimal[1:]     # Strip off decimal point.
    f = fraction.Fraction()
    f.setNumerator(int(decimal))
    f.setDenominator(10 ** len(decimal))
    f.reduce()
    msg = "Converted to fraction:"
    print(msg, str(f.getNumerator()) + '/' + str(f.getDenominator()))
```

```
main()
```

```
Enter a positive decimal number less than 1: .15625
Converted to fraction: 5/32
```

31.
```
def main():
    ## Calculate a workers weekly pay.
    salary = Wages()
    name = input("Enter person's name: ")
    salary.setName(name)
    hours = float(input("Enter number of hours worked: "))
    salary.setHours(hours)
    wage = float(input("Enter hourly wage: "))
    salary.setWage(wage)
    print("Pay for", salary.getName() + ':', salary.payForWeek())

class Wages:
    def __init__(self, name="", hours=0.0, wage=0.0):
        self._name = name
        self._hours = hours    # Number of hours worked during week
        self._wage = wage      # Hourly wage

    def setName(self, name):
        self._name = name

    def getName(self):
        return self._name

    def setHours(self, hours):
        self._hours = hours

    def getHours(self):
        return self._hours

    def setWage(self, wage):
        self._wage = wage

    def getHours(self):
        return self._hours

    def payForWeek(self):
        amount = self._hours * self._wage
        if self._hours > 40:
            amount = 40 * self._wage + ((self._hours - 40) *
                     (1.5 * self._wage))
        return "${0:,.2f}".format(amount)

main()
```

```
Enter person's name: Sophia
Enter number of hours worked: 42
Enter hourly wage: 35
Pay for Sophia: $1,505.00
```

33.
```
import random
import pCard

def main():
   ## Randomly select a poker hand.
   deckOfCards = []
   ranks = ['2', '3', '4', '5', '6', '7', '8', '9',
            "10", "jack", "queen", "king", "ace"]
   suits = ["spades", "hearts", "clubs", "diamonds"]
   for i in ranks:
       for j in suits:
           c = pCard.PlayingCard(i, j)
           deckOfCards.append(c)
   pokerHand = random.sample(deckOfCards, 5)
   pokerHand.sort(key = lambda x: x.getRank())
   for k in pokerHand:
       print(k)

main()
```

```
3 of clubs
4 of clubs
5 of spades
7 of diamonds
queen of clubs
```

35.
```
def main():
    ## Check out at a shopping Web site.
    myPurchases = Cart()
    carryOn = 'Y'
    while carryOn.upper() == 'Y':
        description = input("Enter description of article: ")
        price = float(input("Enter price of article: "))
        quantity = int(input("Enter quantity of article: "))
        article = Purchase(description, price, quantity)
        myPurchases.addItemToCart(article)
        carryOn = input("Do you want to enter more articles (Y/N)? ")
    printReceipt(myPurchases)

def printReceipt(myPurchases):
    print("\n{0:12}  {1:<s}  {2:<12}".format("ARTICLE",
          "PRICE", "QUANTITY"))
    for purchase in myPurchases.getItems():
        print("{0:12s} ${1:,.2f}  {2:5}".format(purchase.getDescription(),
                                 purchase.getPrice(), purchase.getQuantity()))
    print("\nTOTAL COST: ${0:,.2f}".format(myPurchases.calculateTotal()))
```

```
class Purchase:
    def __init__(self, description="", price=0, quantity=0):
        self._description = description
        self._price = price
        self._quantity = quantity

    def setDescription(self, description):
        self._description = description

    def getDescription(self):
        return self._description

    def setPrice(self, price):
        self._price = price

    def getPrice(self):
        return self._price

    def setQuantity(self, quantity):
        self._quantity = quantity

    def getQuantity(self):
        return self._quantity

class Cart:
    def __init__(self, items=[]):
        self._items = items

    def addItemToCart(self, item):
        self._items.append(item)

    def getItems(self):
       return self._items

    def calculateTotal(self):
        amount = 0
        for item in self._items:
            amount += item.getPrice() * item.getQuantity()
        return amount

main()
```

```
Enter description of article: shirt
Enter price of article: 35
Enter quantity of article: 3
Do you want to enter more articles (Y/N)? Y
Enter description of article: tie
Enter price of article: 15
Enter quantity of article: 2
Do you want to enter more articles (Y/N)? N

ARTICLE       PRICE   QUANTITY
shirt        $35.00       3
tie          $15.00       2

TOTAL COST: $135.00
```

EXERCISES 7.2

1. 4 **3.** 6.928 **5.** The rectangle has area 6.00. **7.** Howdy
G'day mate

9. Change function *displayResults* to the following:

```
def displayResults(listOfStudents):
    listOfStudents.sort(key=lambda x: x.getName())
    for pupil in listOfStudents:
        if pupil.calcSemGrade() == 'A':
            print(pupil.getName())
```

11.

```
import random

def main():
    ## Play three games of rock, paper, scissors.
    # Get names of contestants and instantiate an object for each.
    nameOfHuman = input("Enter name of human: ")
    h = Human(nameOfHuman)
    nameOfComputer = input("Enter name of computer: ")
    c = Computer(nameOfComputer)
    print()
    # Play three games and keep score.
    for i in range(3):
        humanChoice = h.makeChoice()
        computerChoice = c.makeChoice()
        print("{0} chooses {1}".format(c.getName(), computerChoice))
        if humanChoice == "rock":
            if computerChoice == "scissors":
                h.incrementScore()
            elif computerChoice == "paper":
                c.incrementScore()
        elif humanChoice == "paper":
            if computerChoice == "rock":
                h.incrementScore()
            elif computerChoice == "scissors":
                c.incrementScore()
        else:  # humanChoice = scissors
            if computerChoice == "rock":
                c.incrementScore()
            elif computerChoice == "paper":
                h.incrementScore()
        print(h, end="  ")
        print(c)
        print()
    if h.getScore() > c.getScore():
        print(h.getName().upper(), "WINS")
    elif c.getScore() > h.getScore():
        print(c.getName().upper(), "WINS")
    else:
        print("TIE")
```

```
class Contestant():
    def __init__(self, name="", score=0):
        self._name = name
        self._score = score

    def getName(self):
        return self._name

    def getScore(self):
        return self._score

    def incrementScore(self):
        self._score += 1

    def __str__(self):
        return "{0}: {1}".format(self._name, self._score)

class Human(Contestant):
    def makeChoice(self):
        choices = ["rock", "paper", "scissors"]
        while True:
            choice = input(self._name + ", enter your choice: ")
            if choice.lower() in choices:
                break
        return choice.lower()

class Computer(Contestant):
    def makeChoice(self):
        choices = ["rock", "paper", "scissors"]
        selection = random.choice(choices)
        return selection

 main()
```

```
Enter name of human: Garry
Enter name of computer: Big Blue

Garry, enter your choice: rock
Big Blue chooses scissors
Garry: 1  Big Blue: 0

Garry, enter your choice: scissors
Big Blue chooses paper
Garry: 2  Big Blue: 0

Garry, enter your choice: rock
Big Blue chooses rock
Garry: 2  Big Blue: 0

GARRY WINS
```

13.
```
class Mortgage:
    def __init__(self, principal, interestRate, term):
        self._principal = principal
        self._interestRate = interestRate
        self._term = term

    def calculateMonthlyPayment(self):
        i = self._interestRate / 1200
        return ((i / (1 - ((1 + i) ** (-12 * self._term))))
                * self._principal)
```

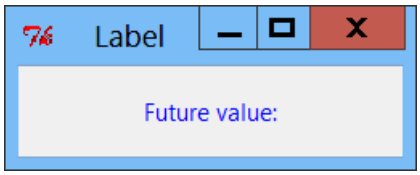
15.
```
def main():
    ## Calculate the values for an interest-only mortgage.
    principal = float(input("Enter principal amount of mortgage: "))
    interestRate = float(input("Enter percent interest rate: "))
    term = float(input("Enter duration of mortgage in years: "))
    numberOfInterestOnlyYears = \
                    float(input("Enter number of interest-only years: "))
    mort = InterestOnlyMortgage(principal, interestRate,
                                term, numberOfInterestOnlyYears)
    print("Monthly payment for first {0:.0f} years: ${1:,.2f}"
        .format(numberOfInterestOnlyYears, mort.initialMonthlyPayment()))
    mort.setTerm(term - numberOfInterestOnlyYears)
    print("Monthly payment for last {0:.0f} years: ${1:,.2f}"
          .format(mort.getTerm(), mort.calculateMonthlyPayment()))

class Mortgage:
    def __init__(self, principal, interestRate, term):
        self._principal = principal
        self._interestRate = interestRate
        self._term = term

    def calculateMonthlyPayment(self):
        i = self._interestRate / 1200
        return ((i / (1 - ((1 + i) ** (-12 * self._term))))
                 * self._principal)

class InterestOnlyMortgage(Mortgage):
    def __init__(self, principal, interestRate,
                 term, numberOfInterestOnlyYears):
        super().__init__(principal, interestRate, term)
        self._numberOfInterestOnlyYears = numberOfInterestOnlyYears

    def initialMonthlyPayment(self):
        return self._principal * (self._interestRate / 1200)

    def setTerm(self, numberOfInterestOnlyYears):
        self._term -= self._numberOfInterestOnlyYears

    def getTerm(self):
        return self._term

main()
```

```
Enter principal amount of mortgage: 275000
Enter percent interest rate: 4.5
Enter duration of mortgage in years: 30
Enter number of interest-only years: 5
Monthly payment for first 5 years: $1,031.25
Monthly payment for last 25 years: $1,528.54
```

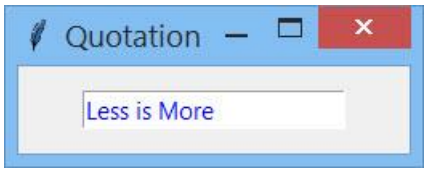
CHAPTER 8

EXERCISES 8.1

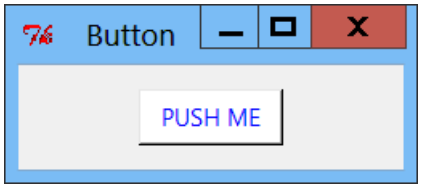
1.
```
from tkinter import *
window = Tk()
window.title("Label")
lblFV = Label(window, text="Future value:", fg="blue")
lblFV.grid(padx=75, pady=15)
window.mainloop()
```



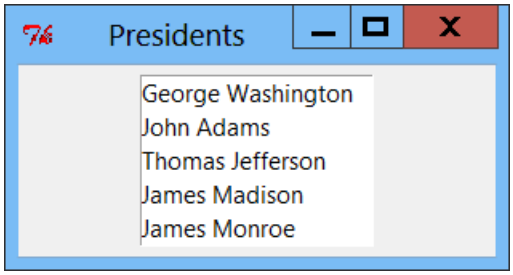
3.
```
from tkinter import *
window = Tk()
window.title("Quotation")
conOFentQuote = StringVar()   # contents of the Entry widget
entQuote = Entry(window, fg="blue", textvariable=conOFentQuote)
entQuote.grid(padx=40, pady=15)
conOFentQuote.set("Less is More")
window.mainloop()
```



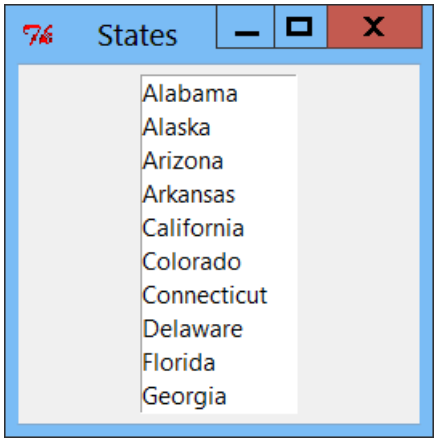
5.
```
from tkinter import *
window = Tk()
window.title("Button")
btnPush = Button(window, text="PUSH ME", fg="blue", bg="white", width=10)
btnPush.grid(padx=75, pady=15)
window.mainloop()
```



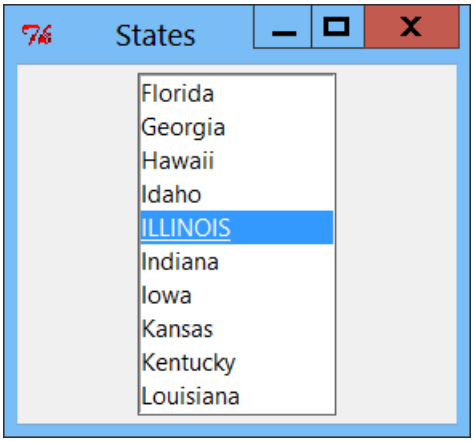
9.
```
from tkinter import *
window = Tk()
window.title("Presidents")
infile = open("USPres.txt", 'r')
listOfPresidents = [line.rstrip() for line in infile]
infile.close()
conOFlstPres = StringVar()
lstPres = Listbox(window, height=5, width=18,
                  listvariable=conOFlstPres)
lstPres.grid(padx=75, pady=5)
conOFlstPres.set(tuple(listOfPresidents))
window.mainloop()
```



11.
```
from tkinter import *
window = Tk()
window.title("States")
infile = open("StatesANC.txt", 'r')
listOfStates = [line.split(',')[0] for line in infile]
infile.close()
conOFlstStates = StringVar()
lstStates = Listbox(window, height=10, width=12,
                    listvariable=conOFlstStates)
lstStates.grid(padx=75, pady=5)
conOFlstStates.set(tuple(listOfStates))
window.mainloop()
```

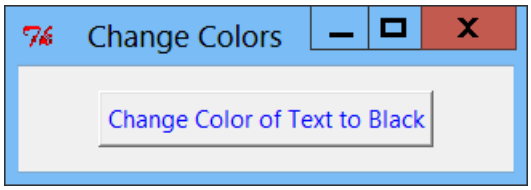


13.
```
def convertToUC(event):
    state = lstStates.get(lstStates.curselection())
    n = listOfStates.index(state)
    listOfStates.remove(state)
    listOfStates.insert(n, state.upper())
    conOFlstStates.set(tuple(listOfStates))

from tkinter import *
window = Tk()
window.title("States")
infile = open("StatesANC.txt", 'r')
listOfStates = [line.split(',')[0] for line in infile]
infile.close()
conOFlstStates = StringVar()
lstStates = Listbox(window, height=10,
                    width=15, listvariable=conOFlstStates)
lstStates.grid(padx=75, pady=5)
conOFlstStates.set(tuple(listOfStates))
lstStates.bind("<<ListboxSelect>>", convertToUC)
window.mainloop()
```

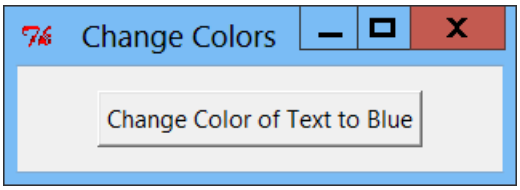


15.
```
from tkinter import *
window = Tk()
window.title("Nations")
infile = open("UN.txt", 'r')
listOfNations = [line.split(',')[0] for line in infile]
infile.close()
conOFlstNations = StringVar()
lstNations = Listbox(window, height=10,
                     width=38, listvariable=conOFlstNations)
lstNations.grid(padx=15, pady=5)
conOFlstNations.set(tuple(listOfNations))
window.mainloop()
```

17.
```
from tkinter import *

def changeColorandText():
    if btnChange["fg"] == "blue":
        btnChange["fg"] = "black"
        btnChange["text"] = "Change Color of Text to Blue"
    else:
        btnChange["fg"] = "blue"
        btnChange["text"] = "Change Color of Text to Black"
window = Tk()
window.title("Change Colors")
btnChange = Button(window, text="Change Color of Text to Black",
                   command=changeColorandText, fg="blue")
btnChange.grid(padx=50, pady=15)
window.mainloop()
```



(a) **Original display.**



(b) **Display after first left-click.**

```
19. from tkinter import *
    def changeText():
        if btnTest["text"] == "HELLO":
            btnTest["text"] = "GOODBYE"
        else:
            btnTest["text"] = "HELLO"
    window = Tk()
    window.title("Salutation")
    btnTest = Button(window, text="HELLO", fg="blue", command=changeText)
    btnTest.grid(padx=100, pady=15)
    window.mainloop()
```

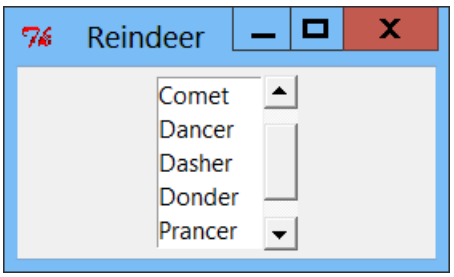
(a) Original display.

(b) Display after first left-click.

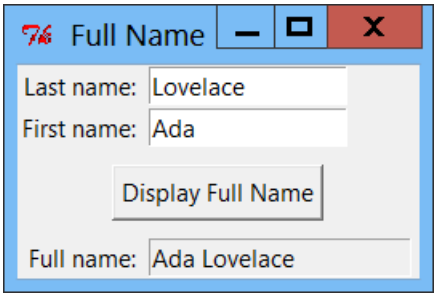
EXERCISES 8.2

1. D **3.** B **5.** A

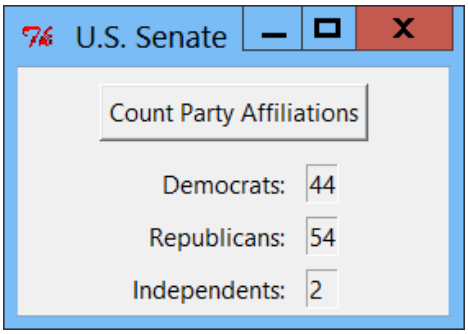
```
7. from tkinter import *
   window = Tk()
   window.title("Reindeer")
   Label(window, text="", width = 10).grid(row=0, column=0)
   Label(window, text="", width = 10).grid(row=0, column=3)
   yscroll = Scrollbar(window, orient=VERTICAL)
   yscroll.grid(row=0, column=2, rowspan=9, pady=5, sticky=NS)
   deerList = ["Blitzen", "Comet", "Dancer", "Dasher","Donder",
               "Prancer", "Vixen"]
   conOFlstDeer = StringVar()
   lstDeer = Listbox(window, width=8, height=5, listvariable=conOFlstDeer,
                     yscrollcommand=yscroll.set)
   lstDeer.grid(row=0, column=1, rowspan=4, pady=5, sticky=E)
   conOFlstDeer.set(tuple(deerList))
   yscroll["command"] = lstDeer.yview
   window.mainloop()
```



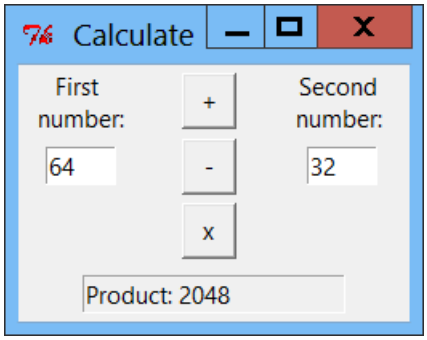
9.
```
from tkinter import *
window = Tk()
window.title("Full Name")
Label(window, text="Last name:").grid(row=0, column=0, sticky=E)
entLastName = Entry(window, width=15)
entLastName.grid(row=0, column=1, padx=5, sticky=W)
Label(window, text="First name:").grid(row=1, column=0, sticky=E)
entFirstName = Entry(window, width=15)
entFirstName.grid(row=1, column=1, padx=5, sticky=W)
btnDisplay = Button(text="Display Full Name")
btnDisplay.grid(row=2, column=0, columnspan=2, pady = 10)
Label(window, text="Full name:").grid(row=3, column=0, sticky=E)
entFullName = Entry(window, state="readonly")
entFullName.grid(row=3, column=1, padx=5)
window.mainloop()
```



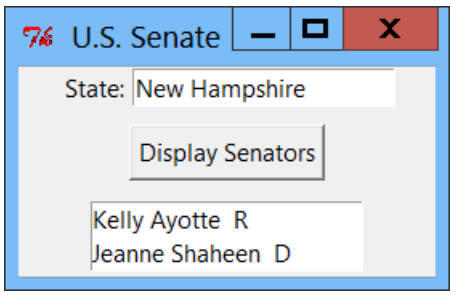
11.
```
from tkinter import *
window = Tk()
window.title("U.S. Senate")
lblDemocrats = Label(window, text="Democrats:")
lblRepublicans = Label(window, text="Republicans:")
lblIndependents = Label(window, text="Independents:")
entDemocrats = Entry(window, width=2, state="readonly")
entRepublicans = Entry(window, width=2, state="readonly")
entIndependents = Entry(window, width=2, state="readonly")
lblDemocrats.grid(row=1, column=1, padx=5,pady=3,sticky=E)
lblRepublicans.grid(row=2, column=1, padx=5,pady=3,sticky=E)
lblIndependents.grid(row=3, column=1, padx=5,pady=3,sticky=E)
entDemocrats.grid(row=1, column=2, pady=3, padx=5, sticky=W)
entRepublicans.grid(row=2, column=2, padx=5,pady=3,sticky=W)
entIndependents.grid(row=3, column=2, padx=5,pady=3,sticky=W)
btnDisplay = Button(text="Count Party Affiliations")
btnDisplay.grid(row=0, columnspan=4, padx=50, pady=10)
window.mainloop()
```



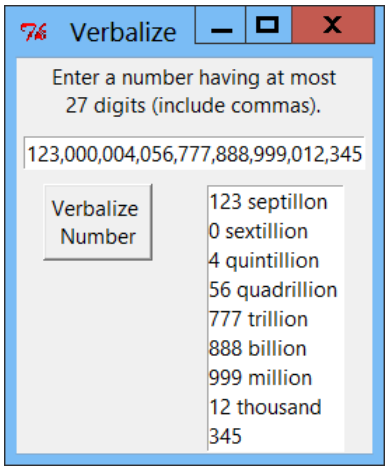
13.
```
from tkinter import *
window = Tk()
window.title("Calculate")
Label(window, text="First \nnumber:").grid(row=0, column=0)
Label(window, text="Second \nnumber: ").grid(row=0, column=2)
entFirst = Entry(window, width=5)
entFirst.grid(row=1, column=0)
entSecond = Entry(window, width=5)
entSecond.grid(row=1, column=2)
btnAdd = Button(window, text='+', width=3)
btnAdd.grid(row=0, column=1, padx=15)
btnSubtract = Button(window, text='-', width=3)
btnSubtract.grid(row=1, column=1, padx=15)
btnMultiply = Button(window, text='x', width=3)
btnMultiply.grid(row=2, column=1, padx=15, pady=5)
entResult = Entry(window, state="readonly", width=20)
entResult.grid(row=3, column=0, columnspan=3, padx=40, pady=5)
window.mainloop()
```



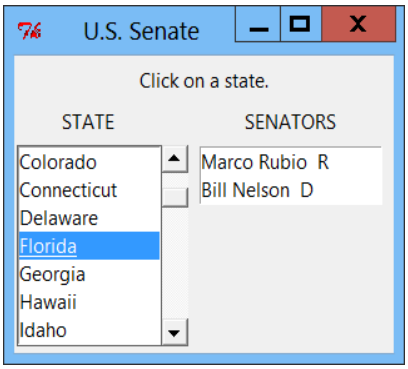
15.
```
from tkinter import *
window = Tk()
window.title("U.S. Senate")
Label(window, text="State:", width=5).grid(row=0, column=0, sticky=E)
state = StringVar()
entState = Entry(window, textvariable=state)
entState.grid(row=0, column=1, sticky=W)
btnDisplay = Button(text="Display Senators")
btnDisplay.grid(row=1, columnspan=2, pady = 10)
lstSenators = Listbox(window, height=2, width=21)
lstSenators.grid(row=2,column=0, columnspan=2, padx=44, pady=2)
window.mainloop()
```



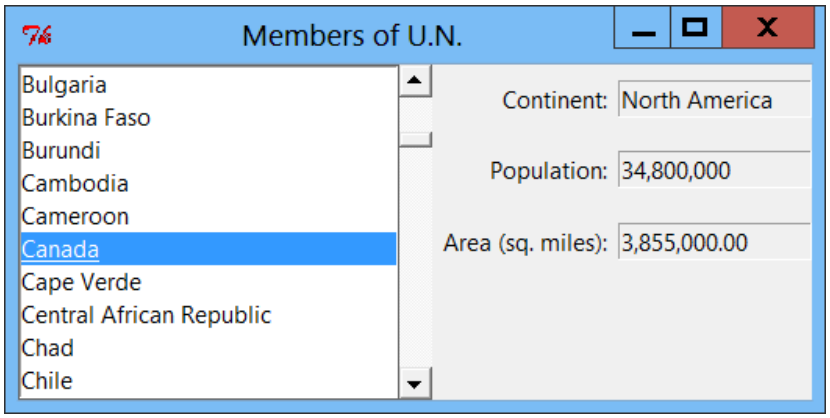
17.
```
from tkinter import *
window = Tk()
window.title("Verbalize")
instruction = "Enter a number having at most\n" + \
              "27 digits (include commas)."
Label(window, text=instruction).grid(row=0, column=0,
      columnspan=2, padx=15)
entNum = Entry(window, width=27)
entNum.grid(row=1, column=0, columnspan=2, pady=5)
btnVerbalize = Button(window, text="Verbalize\nNumber")
btnVerbalize.grid(row=2, column=0, sticky=N)
lstEnglish = Listbox(window, height=9, width=14)
lstEnglish.grid(row=2, column=1)
window.mainloop()
```



19.
```
from tkinter import *
window = Tk()
window.title("U.S. Senate")
instruction = "Click on a state."
Label(window, text=instruction).grid(row=0, column=0, columnspan=3, pady=5)
Label(window, text="STATE", width=14).grid(row=1, column=0)
Label(window, text="SENATORS").grid(row=1, column=2)
yscroll = Scrollbar(window, orient=VERTICAL)
yscroll.grid(row=2, column=1, pady=5, sticky=NS)
lstStates = Listbox(window, width=14, height=7, yscrollcommand=yscroll.set)
lstStates.grid(row=2, column=0, pady=5, sticky=E)
lstSenators = Listbox(window, width=18, height=2)
lstSenators.grid(row=2, column=2, padx=8, pady=5, sticky=N)
yscroll["command"] = lstStates.yview
window.mainloop()
```



```
21. from tkinter import *
    import pickle
    window = Tk()
    window.title("Members of U.N.")
    yscroll = Scrollbar(window, orient=VERTICAL)
    yscroll.grid(row=0, column=1, rowspan=7, sticky=NS)
    lstNations = Listbox(window, height=10, width=30, yscrollcommand=yscroll.set)
    lstNations.grid(row=0, column=0, rowspan=7, sticky=NSEW)
    yscroll["command"] = lstNations.yview
    Label(window, text="Continent:").grid(row=0, column=3, padx=4, sticky=E)
    Label(window, text="Population:").grid(row=1, column=3, padx=4, sticky=E)
    Label(window, text="Area (sq. miles):").grid(row=2, column=3,
                                                 padx=4,sticky=E)
    entContinent = Entry(window, width=15, state="readonly")
    entContinent.grid(row=0, column=4, sticky=W)
    entPopulation = Entry(window, width=15, state="readonly")
    entPopulation.grid(row=1, column=4,)
    entArea = Entry(window, width=15, state="readonly")
    entArea.grid(row=2, column=4)
    window.mainloop()
```

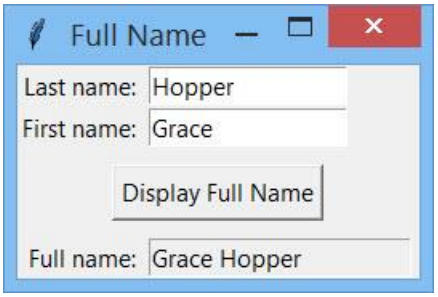


EXERCISES 8.3

(Each program is written in a direct coding style and in an object-oriented style.)

```
1. from tkinter import *

   def fullName():
       conOFentFullName.set(conOFentFirstName.get() + \
                            " " + conOFentLastName.get())
   window = Tk()
   window.title("Full Name")
   Label(window, text="Last name:").grid(row=0, column=0, sticky=E)
   conOFentLastName = StringVar()
   entLastName = Entry(window, width=15, textvariable=conOFentLastName)
   entLastName.grid(row=0, column=1, padx=5, sticky=W)
   Label(window, text="First name:").grid(row=1, column=0, sticky=E)
   conOFentFirstName = StringVar()
   entFirstName = Entry(window, width=15, textvariable=conOFentFirstName)
```

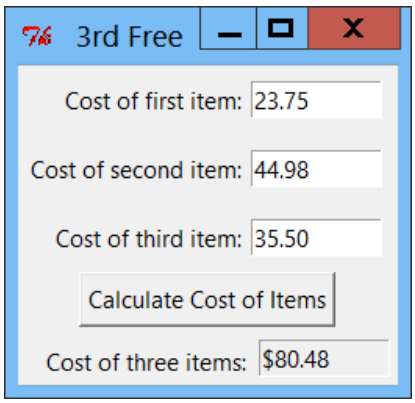
```
entFirstName.grid(row=1, column=1, padx=5, sticky=W)
btnDisplay = Button(text="Display Full Name", command=fullName)
btnDisplay.grid(row=2, column=0, columnspan=2, pady = 10)
Label(window, text="Full name:").grid(row=3, column=0, sticky=E)
conOFentFullName = StringVar()
entFullName = Entry(window, state="readonly", textvariable=conOFentFullName)
entFullName.grid(row=3, column=1, padx=5)
window.mainloop()
```



1. (Object-oriented style)

```
from tkinter import *

class FullName:
    def __init__(self):
        window = Tk()
        window.title("Full Name")
        Label(window, text="Last name:").grid(row=0, column=0, sticky=E)
        self.conOFentLastName = StringVar()
        entLastName = Entry(window, width=15,
                            textvariable=self.conOFentLastName)
        entLastName.grid(row=0, column=1, padx=5, sticky=W)
        Label(window, text="First name:").grid(row=1, column=0, sticky=E)
        self.conOFentFirstName = StringVar()
        entFirstName = Entry(window, width=15,
                             textvariable=self.conOFentFirstName)
        entFirstName.grid(row=1, column=1, padx=5, sticky=W)
        btnDisplay = Button(text="Display Full Name",
                            command=self.fullName)
        btnDisplay.grid(row=2, column=0, columnspan=2, pady = 10)
        Label(window, text="Full name:").grid(row=3, column=0, sticky=E)
        self.conOFentFullName = StringVar()
        self.entFullName = Entry(window, state="readonly",
                                 textvariable=self.conOFentFullName)
        self.entFullName.grid(row=3, column=1, padx=5)
        window.mainloop()

    def fullName(self):
        self.conOFentFullName.set(self.conOFentFirstName.get() + \
                                  " " + self.conOFentLastName.get())
FullName()
```

3.
```
from tkinter import *

def calculateCost():
    costs = [float(conOFentFirst.get()),
             float(conOFentSecond.get()),float(conOFentThird.get())]
    totalCost = sum(costs) - min(costs)
    conOFentTotalCost.set("${0:,.2f}".format(totalCost))

window = Tk()
window.title("3rd Free")
Label(window, text="Cost of first item:").grid(row=0, column=0,
                                               padx=(5,3), pady=5, sticky=E)
Label(window, text="Cost of second item:").grid(row=1, column=0,
                                               padx=(5,3), pady=5, sticky=E)
Label(window, text="Cost of third item:").grid(row=2, column=0,
                                                padx=(5,3), pady=5, sticky=E)
conOFentFirst = StringVar()
entFirst = Entry(window, width=10, textvariable=conOFentFirst)
entFirst.grid(row=0, column=1, pady=10, sticky=W)
conOFentSecond = StringVar()
entSecond = Entry(window, width=10, textvariable=conOFentSecond)
entSecond.grid(row=1, column=1, pady=10, sticky=W)
conOFentThird = StringVar()
entThird = Entry(window, width=10, textvariable=conOFentThird)
entThird.grid(row=2, column=1, pady=10, sticky=W)
btnCalculate = Button(window, text="Calculate Cost of Items",
                      command=calculateCost)
btnCalculate.grid(row=3, column=0, columnspan=2, pady=(0,8))
Label(window, text="Cost of three items:").grid(row=4, column=0, sticky=E)
conOFentTotalCost = StringVar()
entTotalCost = Entry(window, width=10, textvariable=conOFentTotalCost,
                     state="readonly")
entTotalCost.grid(row=4, column=1, padx=5, pady=(0,5), sticky=W)
window.mainloop()
```



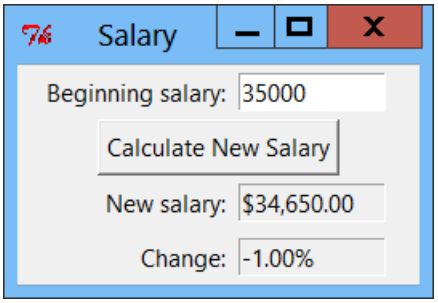
3. (Object-oriented style)

```
from tkinter import *

class Cost:
    def __init__(self):
        window = Tk()
        window.title("3rd Free")
        Label(window, text="Cost of first item:").grid(row=0, column=0,
                                            padx=(5,3), pady=5, sticky=E)
        Label(window, text="Cost of second item:").grid(row=1, column=0,
                                            padx=(5,3), pady=5, sticky=E)
        Label(window, text="Cost of third item:").grid(row=2, column=0,
                                            padx=(5,3), pady=5, sticky=E)
        self._conOFentFirst = StringVar()
        entFirst = Entry(window, width=10, textvariable=self._conOFentFirst)
        entFirst.grid(row=0, column=1, pady=10, sticky=W)
        self._conOFentSecond = StringVar()
        entSecond = Entry(window, width=10, textvariable=self._conOFentSecond)
        entSecond.grid(row=1, column=1, pady=10, sticky=W)
        self._conOFentThird = StringVar()
        entThird = Entry(window, width=10, textvariable=self._conOFentThird)
        entThird.grid(row=2, column=1, pady=10, sticky=W)
        btnCalculate = Button(window, text="Calculate Cost of Items",
                              command=self.calculateCost)
        btnCalculate.grid(row=3, column=0, columnspan=2, pady=(0,8))
        Label(window, text="Cost of three items:").grid(row=4, column=0,
              sticky=E)
        self._conOFentTotalCost = StringVar()
        entTotalCost = Entry(window, width=10,
                       textvariable=self._conOFentTotalCost, state="readonly")
        entTotalCost.grid(row=4, column=1, padx=5, pady=(0,5), sticky=W)
        window.mainloop()

    def calculateCost(self):
        costs = [float(self._conOFentFirst.get()),
            float(self._conOFentSecond.get()), float(self._conOFentThird.get())]
        totalCost = sum(costs) - min(costs)
        self._conOFentTotalCost.set("${0:,.2f}".format(totalCost))

Cost()
```

5.

```
from tkinter import *

def newSalary():
    begSalary = eval(conOFentBegSalary.get())
    salary = begSalary + (.1 * begSalary)
    salary = salary - (.1 * salary)
    conOFentNewSalary.set("${0:,.2f}".format(salary))
    begSalary = eval(conOFentBegSalary.get())
    change = (salary - begSalary) / begSalary
    conOFentChange.set("{0:,.2%}".format(change))
```

```
window = Tk()
window.title("Salary")
Label(window, text="Beginning salary:").grid(row=0, column=0, sticky=E)
conOFentBegSalary = StringVar()
entBegSalary = Entry(window, width=11, textvariable=conOFentBegSalary)
entBegSalary.grid(row=0, column=1, padx=5, pady=5, sticky=W)
btnCalculate = Button(text="Calculate New Salary", command=newSalary)
btnCalculate.grid(row=2, column=0, columnspan=2, padx=50)
Label(window, text="New salary:").grid(row=3, column=0, sticky=E)
conOFentNewSalary = StringVar()
entNewSalary = Entry(window, width=11, state="readonly",
                     textvariable=conOFentNewSalary)
entNewSalary.grid(row=3, column=1, padx=5, pady=5, sticky=W)
Label(window, text="Change:").grid(row=4, column=0, sticky=E)
conOFentChange = StringVar()
entChange = Entry(window, width=11, state="readonly",
                          textvariable=conOFentChange)
entChange.grid(row=4, column=1, padx=5, pady=5, sticky=W)
window.mainloop()
```

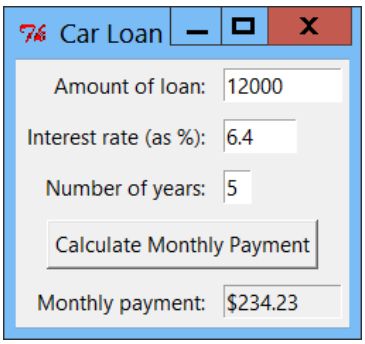


5. (Object-oriented style)

```
from tkinter import *
class Salary:
    def __init__(self):
        window = Tk()
        window.title("Salary")
        Label(window, text="Beginning salary:").grid(row=0, column=0,
              sticky=E)
        self.conOFentBegSalary = StringVar()
        entBegSalary = Entry(window, width=11,
                             textvariable=self.conOFentBegSalary)
        entBegSalary.grid(row=0, column=1, padx=5, pady=5, sticky=W)
        btnCalculate = Button(text="Calculate New Salary",
                              command=self.newSalary)
        btnCalculate.grid(row=2, column=0, columnspan=2, padx=50)
        Label(window, text="New salary:").grid(row=3, column=0, sticky=E)
        self.conOFentNewSalary = StringVar()
        self.entNewSalary = Entry(window, width=11, state="readonly",
                                  textvariable=self.conOFentNewSalary)
        self.entNewSalary.grid(row=3, column=1, padx=5, pady=5, sticky=W)
        Label(window, text="Change:").grid(row=4, column=0, sticky=E)
        self.conOFentChange = StringVar()
        self.entChange = Entry(window, width=11, state="readonly",
                               textvariable=self.conOFentChange)
        self.entChange.grid(row=4, column=1, padx=5, pady=5, sticky=W)
        window.mainloop()
```

```
    def newSalary(self):
        begSalary = eval(self.conOFentBegSalary.get())
        salary = begSalary + (.1 * begSalary)
        salary = salary - (.1 * salary)
        self.conOFentNewSalary.set("${0:,.2f}".format(salary))
        begSalary = eval(self.conOFentBegSalary.get())
        change = (salary - begSalary) / begSalary
        self.conOFentChange.set("{0:,.2%}".format(change))
Salary()
```

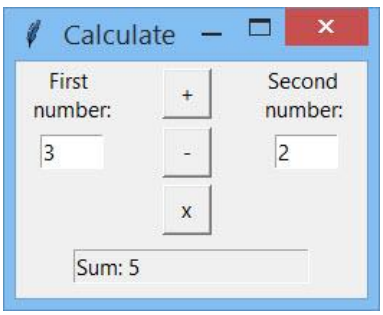
7.
```
from tkinter import *
def calculate():
   p = eval(principal.get())
   r = eval(interestRate.get())
   n = eval(numberOfYears.get())
   payment = (p*(r/1200)/(1 - (1 + (r/1200)) ** (-12*n)))
   payment = "${0:,.2f}".format(payment)
   monthlyPayment.set(payment)
window = Tk()
window.title("Car Loan")
lblPrincipal = Label(window, text="Amount of loan:", )
lblPrincipal.grid(row=0, column=0, padx=5, pady=5, sticky=E)
lblInterestRate = Label(window, text="Interest rate (as %):" )
lblInterestRate.grid(row=1, column=0, padx=5, pady=5, sticky=E)
lblNumberOfYears = Label(window, text="Number of years:" )
lblNumberOfYears.grid(row=2, column=0, padx=5, pady=5, sticky=E)
lblMonthlyPayment = Label(window, text="Monthly payment:")
lblMonthlyPayment.grid(row=5, column=0, padx=5, pady=5, sticky=E)
principal = StringVar()
interestRate = StringVar()
numberOfYears = StringVar()
monthlyPayment = StringVar()
entPrincipal = Entry(window, width=10, textvariable=principal)
entPrincipal.grid(row=0, column=1, padx=5, pady=5, sticky=W)
entInterestRate = Entry(window, width=6 ,textvariable=interestRate)
entInterestRate.grid(row=1, column=1, padx=5, pady=5, sticky=W)
entNumberOfYears = Entry(window, width=2 ,textvariable=numberOfYears)
entNumberOfYears.grid(row=2, column=1, padx=5, pady=5, sticky=W)
entMonthlyPayment = Entry(window, width=10, state="readonly",
                          textvariable=monthlyPayment)
entMonthlyPayment.grid(row=5, column=1, padx=5, pady=5, sticky=W)
btnCalculate = Button(window, text="Calculate Monthly Payment",
                      command=calculate)
btnCalculate.grid(row=3, column=0, columnspan=2, padx=5, pady=5)
window.mainloop()
```

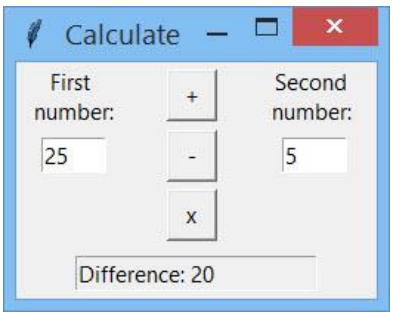


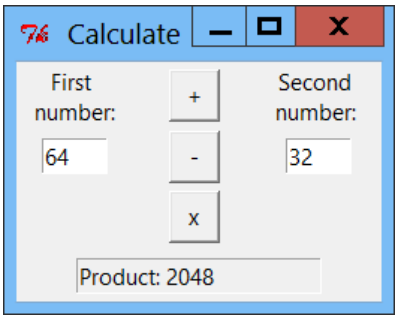
7. (Object-oriented style)

```
from tkinter import *

class CarLoan:
    def __init__(self):
        window = Tk()
        window.title("Car Loan")
        lblPrincipal = Label(window, text="Amount of loan:", )
        lblPrincipal.grid(row=0, column=0, padx=5, pady=5, sticky=E)
        lblInterestRate = Label(window, text="Interest rate (as %):" )
        lblInterestRate.grid(row=1, column=0, padx=5, pady=5, sticky=E)
        lblNumberOfYears = Label(window, text="Number of years:" )
        lblNumberOfYears.grid(row=2, column=0, padx=5, pady=5, sticky=E)
        lblMonthlyPayment = Label(window, text="Monthly payment:")
        lblMonthlyPayment.grid(row=5, column=0, padx=5, pady=5, sticky=E)
        self.principal = StringVar()
        self.interestRate = StringVar()
        self.numberOfYears = StringVar()
        self.monthlyPayment = StringVar()
        entPrincipal = Entry(window, width=10,
                             textvariable=self.principal)
        entPrincipal.grid(row=0, column=1, padx=5, pady=5, sticky=W)
        entInterestRate = Entry(window, width=6,
                                textvariable=self.interestRate)
        entInterestRate.grid(row=1, column=1, padx=5, pady=5, sticky=W)
        entNumberOfYears = Entry(window, width=2,
                                 textvariable=self.numberOfYears)
        entNumberOfYears.grid(row=2, column=1, padx=5, pady=5, sticky=W)
        entMonthlyPayment = Entry(window, width=10, state="readonly",
                                  textvariable=self.monthlyPayment)
        entMonthlyPayment.grid(row=5, column=1, padx=5, pady=5, sticky=W)
        btnCalculate = Button(window, text="Calculate Monthly Payment",
                              command=self.calculate)
        btnCalculate.grid(row=3, column=0, columnspan=2, padx=5, pady=5 )
        window.mainloop()

    def calculate(self):
        p = eval(self.principal.get())
        r = eval(self.interestRate.get())
        n = eval(self.numberOfYears.get())
        payment = (p*(r/1200)/(1 - (1 + (r/1200)) ** (-12*n)))
        payment = "${0:,.2f}".format(payment)
        self.monthlyPayment.set(payment)

CarLoan()
```

9.
```
from tkinter import *

def add():
    num1 = eval(conOFentFirst.get())
    num2 = eval(conOFentSecond.get())
    sum = num1 + num2
    conOFentResult.set("Sum: " + str(sum))

def subtract():
    num1 = eval(conOFentFirst.get())
    num2 = eval(conOFentSecond.get())
    difference = num1 - num2
    conOFentResult.set("Difference: " + str(difference))

def multiply():
    num1 = eval(conOFentFirst.get())
    num2 = eval(conOFentSecond.get())
    product = num1 * num2
    conOFentResult.set("Product: " + str(product))

window = Tk()
window.title("Calculate")
Label(window, text="First \nnumber:").grid(row=0, column=0)
Label(window, text="Second \nnumber: ").grid(row=0, column=2)
conOFentFirst = StringVar()
entFirst = Entry(window, width=5, textvariable=conOFentFirst)
entFirst.grid(row=1, column=0)
conOFentSecond = StringVar()
entSecond = Entry(window, width=5, textvariable=conOFentSecond)
entSecond.grid(row=1, column=2)
btnAdd = Button(window, text='+', width=3, command=add)
btnAdd.grid(row=0, column=1, padx=15)
btnSubtract = Button(window, text='-', width=3, command=subtract)
btnSubtract.grid(row=1, column=1, padx=15)
btnMultiply = Button(window, text='x', width=3, command=multiply)
btnMultiply.grid(row=2, column=1, padx=15, pady=5)
conOFentResult = StringVar()
entResult = Entry(window, state="readonly", width=20,
                  textvariable=conOFentResult)
entResult.grid(row=3, column=0, columnspan=3, padx=40, pady=5)
window.mainloop()
```



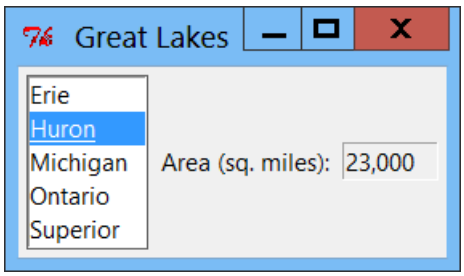




9. (Object-oriented style)

```
from tkinter import *

class Calculate:
    def __init__(self):
        window = Tk()
        window.title("Calculate")
        Label(window, text="First \nnumber:").grid(row=0, column=0)
        Label(window, text="Second \nnumber: ").grid(row=0, column=2)
        self._conOFentFirst = StringVar()
        self.entFirst = Entry(window, width=5,
                              textvariable=self._conOFentFirst)
        self.entFirst.grid(row=1, column=0)
        self._conOFentSecond = StringVar()
        self.entSecond = Entry(window, width=5,
                               textvariable=self._conOFentSecond)
        self.entSecond.grid(row=1, column=2)
        btnAdd = Button(window, text='+', width=3, command=self.add)
        btnAdd.grid(row=0, column=1, padx=15)
        btnSubtract = Button(window, text='-', width=3,
                             command=self.subtract)
        btnSubtract.grid(row=1, column=1, padx=15)
        btnMultiply = Button(window, text='x', width=3,
                             command=self.multiply)
        btnMultiply.grid(row=2, column=1, padx=15, pady=5)
        self.conOFentResult = StringVar()
        self.entResult = Entry(window, state="readonly", width=20,
                               textvariable=self.conOFentResult)
        self.entResult.grid(row=3, column=0, columnspan=3, padx=40,
                            pady=5)
        window.mainloop()

    def add(self):
        num1 = eval(self._conOFentFirst.get())
        num2 = eval(self._conOFentSecond.get())
        sum = num1 + num2
        self.conOFentResult.set("Sum: " + str(sum))

    def subtract(self):
        num1 = eval(self._conOFentFirst.get())
        num2 = eval(self._conOFentSecond.get())
        difference = num1 - num2
        self.conOFentResult.set("Difference: " + str(difference))

    def multiply(self):
        num1 = eval(self._conOFentFirst.get())
        num2 = eval(self._conOFentSecond.get())
        product = num1 * num2
        self.conOFentResult.set("Product: " + str(product))

Calculate()
```

11.
```
from tkinter import *
import pickle

def displayData(e):
    lake = lstLakes.get(lstLakes.curselection())
    conOFentArea.set("{0:,d}".format(lakesDict[lake]))

window = Tk()
window.title("Great Lakes")
global lakesDict
lakesDict = {"Huron":23000, "Ontario":8000, "Michigan":22000,
                     "Erie":10000, "Superior":32000}
lakeList = list((lakesDict).keys())
lakeList.sort()
conOFlstLakes = StringVar()
global lstLakes
lstLakes = Listbox(window, height=5, width=9, listvariable=conOFlstLakes)
lstLakes.grid(row=0, column=0, padx=5, pady=5, rowspan=5, sticky=NSEW)
conOFlstLakes.set(tuple(lakeList))
lstLakes.bind("<<ListboxSelect>>", displayData)
Label(window, text="Area (sq. miles):").grid(row=2, column=1, sticky=E)
conOFentContinent = StringVar()
conOFentArea = StringVar()
entArea = Entry(window, width=7, state="readonly", textvariable=conOFentArea)
entArea.grid(row=2, column=2, padx=5)
window.mainloop()
```

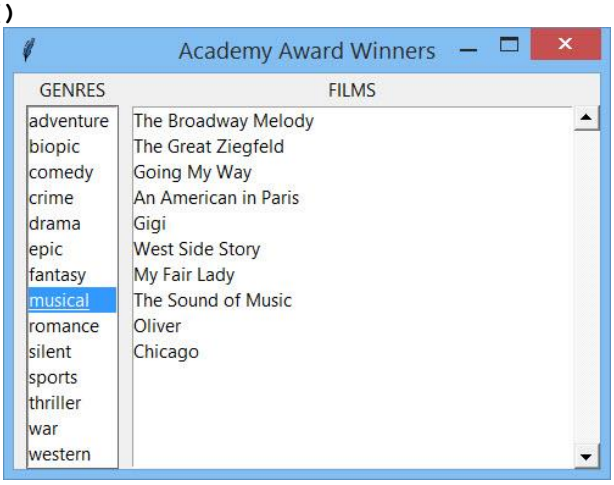


11. (Object-oriented style)
```
from tkinter import *
import pickle
class GreatLakes:
    def __init__(self):
        window = Tk()
        window.title("Great Lakes")
        global lakesDict
        lakesDict = {"Huron":23000, "Ontario":8000, "Michigan":22000,
                     "Erie":10000, "Superior":32000}
        self._lakeList = list((lakesDict).keys())
        self._lakeList.sort()
        self._conOFlstLakes = StringVar()
        global lstLakes
        lstLakes = Listbox(window, height=5, width=9,
                listvariable=self._conOFlstLakes)
        lstLakes.grid(row=0, column=0, padx=5, pady=5, rowspan=5,
                      sticky=NSEW)
        self._conOFlstLakes.set(tuple(self._lakeList))
        lstLakes.bind("<<ListboxSelect>>", self.displayData)
```

```
        Label(window, text="Area (sq. miles):").grid(row=2, column=1,
                        sticky=E)
        self._conOFentContinent = StringVar()
        self._conOFentArea = StringVar()
        entArea = Entry(window, width=7, state="readonly",
                        textvariable=self._conOFentArea)
        entArea.grid(row=2, column=2, padx=5)
        window.mainloop()
    def displayData(self, e):
        lake = lstLakes.get(lstLakes.curselection())
        self._conOFentArea.set("{0:,d}".format(lakesDict[lake]))
GreatLakes()
```

13.

```
from tkinter import *
def films(e):
    genre = lstGenres.get(lstGenres.curselection())
    F = [line.split(',')[0] for line in open("Oscars.txt", 'r') if
         line.split(',')[1].rstrip() == genre]
    conOFlstFilms.set(tuple(F))
window = Tk()
window.title("Academy Award Winners")
Label(window, text="GENRES").grid(row=0, column=0)
Label(window, text="FILMS").grid(row=0, column=1)
infile = open("Oscars.txt", 'r')
genreSet = {line.split(',')[1].rstrip() for line in infile}
infile.close()
L = list(genreSet)
L.sort()
conOFlstGenres = StringVar()
lstGenres = Listbox(window, width=9, height=len(L), listvariable=conOFlstGenres)
lstGenres.grid(row=1, column=0, padx=10, sticky=N)
conOFlstGenres.set(tuple(L))
lstGenres.bind("<<ListboxSelect>>", films)
yscroll = Scrollbar(window, orient=VERTICAL)
yscroll.grid(row=1, column=2, sticky=NS)
conOFlstFilms = StringVar()
lstFilms = Listbox(window, width=45, height=len(L),
                   listvariable=conOFlstFilms, yscrollcommand=yscroll.set)
lstFilms.grid(row=1, column=1, sticky=NSEW)
yscroll["command"] = lstFilms.yview
window.mainloop()
```



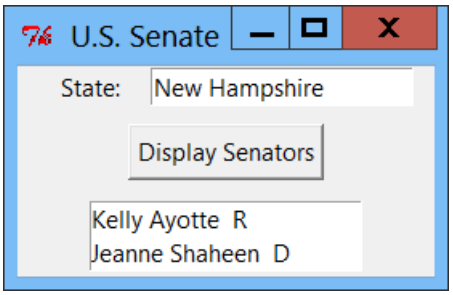
13. (Object-oriented style)

```
from tkinter import *
class Oscars:
    def __init__(self):
        window = Tk()
        window.title("Academy Award Winners")
        Label(window, text="GENRES").grid(row=0, column=0)
        Label(window, text="FILMS").grid(row=0, column=1)
        infile = open("Oscars.txt", 'r')
        self._genreSet = {line.split(',')[1].rstrip() \
                          for line in infile}
        infile.close()
        self._L = list(self._genreSet)
        self._L.sort()
        self._conOFlstGenres = StringVar()
        self._lstGenres = Listbox(window, width=9, height=len(self._L),
                                  listvariable=self._conOFlstGenres)
        self._lstGenres.grid(row=1, column=0, padx=10, sticky=N)
        self._conOFlstGenres.set(tuple(self._L))
        self._lstGenres.bind("<<ListboxSelect>>", self.films)
        yscroll = Scrollbar(window, orient=VERTICAL)
        yscroll.grid(row=1, column=2, sticky=NS)
        self._conOFlstFilms = StringVar()
        lstFilms = Listbox(window, width=45, height=len(self._L),
                           listvariable=self._conOFlstFilms,
                           yscrollcommand=yscroll.set)
        lstFilms.grid(row=1, column=1, sticky=NSEW)
        yscroll["command"] = lstFilms.yview
        window.mainloop()

    def films(self, e):
        genre = self._lstGenres.get(self._lstGenres.curselection())
        F = [line.split(',')[0] for line in open("Oscars.txt", 'r') \
             if line.split(',')[1].rstrip() == genre]
        self._conOFlstFilms.set(tuple(F))

Oscars()
```

15.

```
from tkinter import *

def clearBoxes(e):
    state.set("")
    listContents.set(tuple([]))

def senate():
    L = []
    result = state.get()
    infile = open("Senate114.txt", 'r')
    for line in infile:
        temp = line.split(',')
        if temp[1] == result:
            L.append(temp[0] + "  " + temp[2])
            listContents.set(tuple(L))
    infile.close()
```

```
window = Tk()
window.title("U.S. Senate")
Label(window, text="State:", width=5).grid(row=0, column=0, sticky=E)
state = StringVar()
entState = Entry(window, textvariable=state)
entState.grid(row=0, column=1, sticky=W)
entState.focus_set()
entState.bind("<Button-1>", clearBoxes) # to trigger event
                    # click on Entry box with left mouse button
btnDisplay = Button(text="Display Senators", command=senate)
btnDisplay.grid(row=1, columnspan=2, pady = 10)
L = []
listContents = StringVar()
listContents.set(tuple(L))
lstSenators = Listbox(window, height=2, width=21, listvariable=listContents)
lstSenators.grid(row=2,column=0, columnspan=2, padx=44, pady=2)
window.mainloop()
```



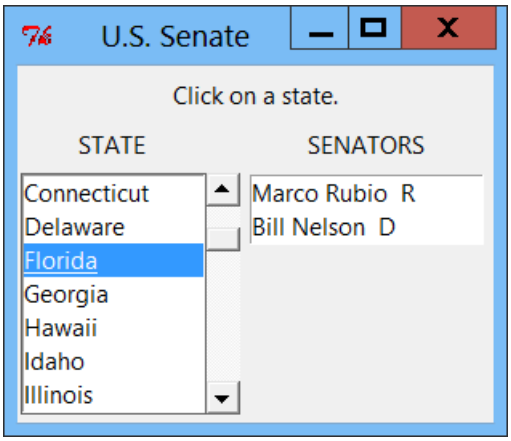
15. (Object-oriented style)

```
from tkinter import *

class Senators:
    def __init__(self):
        window = Tk()
        window.title("U.S. Senate")
        Label(window, text="State:", width=5).grid(row=0, column=0,
              sticky=E)
        self.state = StringVar()
        entState = Entry(window, textvariable=self.state)
        entState.grid(row=0, column=1, sticky=W)
        entState.focus_set()
        entState.bind("<Button-1>", self.clearBoxes) # to trigger event
                            # click on Entry box with left mouse button
        btnDisplay = Button(text="Display Senators", command=self.senate)
        btnDisplay.grid(row=1, columnspan=2, pady = 10)
        self.L = []
        self.listContents = StringVar()
        self.listContents.set(tuple(self.L))
        lstSenators = Listbox(window, height=2, width=21,
                              listvariable=self.listContents)
        lstSenators.grid(row=2,column=0, columnspan=2, padx=44, pady=2)
        window.mainloop()
```

```
        def clearBoxes(self, e):
            self.state.set("")
            self.listContents.set(tuple([]))

        def senate(self):
            self.L = []
            result = self.state.get()
            infile = open("Senate114.txt", 'r')
            for line in infile:
                temp = line.split(',')
                if temp[1] == result:
                    self.L.append(temp[0] + "  " + temp[2])
                    self.listContents.set(tuple(self.L))
            infile.close()

    Senators()
```

17.

```
    from tkinter import *

    def senate(e):
        L = []
        state = lstStates.get(lstStates.curselection())
        infile = open("Senate114.txt", 'r')
        for line in infile:
            temp = line.split(',')
            if temp[1] == state:
                L.append(temp[0] + "  " + temp[2])
        infile.close()
        conOFlstSenators.set(tuple(L))

    window = Tk()
    window.title("U.S. Senate")
    instruction = "Click on a state."
    Label(window, text=instruction).grid(row=0, column=0, columnspan=3, pady=5)
    Label(window, text="STATE", width=14).grid(row=1, column=0)
    Label(window, text="SENATORS").grid(row=1, column=2)
    yscroll = Scrollbar(window, orient=VERTICAL)
    yscroll.grid(row=2, column=1, pady=5, sticky=NS)
    stateSet = {line.split(',')[1] for line in open("Senate114.txt", 'r')}
    stateList = list(stateSet)
    stateList.sort()
    conOFlstStates = StringVar()
    lstStates = Listbox(window, width=14, height=7, listvariable=conOFlstStates,
                        yscrollcommand=yscroll.set)
    lstStates.grid(row=2, column=0, pady=5, sticky=E)
    lstStates.bind("<<ListboxSelect>>", senate)
    conOFlstStates.set(tuple(stateList))
    conOFlstSenators = StringVar()
    lstSenators = Listbox(window, width=18, height=2,listvariable=conOFlstSenators)
    lstSenators.grid(row=2, column=2, padx=8, pady=5, sticky=N)
    yscroll["command"] = lstStates.yview
    window.mainloop()
```



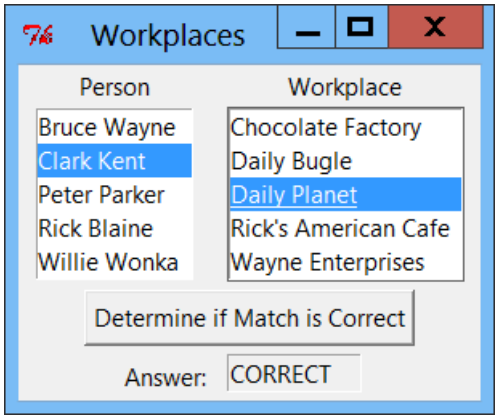
17. (Object-oriented style)

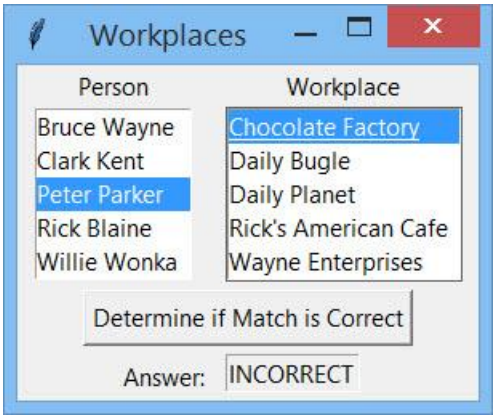
```
from tkinter import *

class Senators:
    def __init__(self):
        window = Tk()
        window.title("U.S. Senate")
        instruction = "Click on a state."
        Label(window, text=instruction).grid(row=0, column=0,
                                             columnspan=3, pady=5)
        Label(window, text="STATE", width=14).grid(row=1, column=0)
        Label(window, text="SENATORS").grid(row=1, column=2)
        yscroll = Scrollbar(window, orient=VERTICAL)
        yscroll.grid(row=2, column=1, pady=5, sticky=NS)
        infile = open("Senate114.txt", 'r')
        stateSet = {line.split(',')[1] for line in infile}
        infile.close()
        stateList = list(stateSet)
        stateList.sort()
        conOFlstStates = StringVar()
        self._lstStates = Listbox(window, width=14, height=7,
                    listvariable=conOFlstStates,
                    yscrollcommand=yscroll.set)
        self._lstStates.grid(row=2, column=0, pady=5, sticky=E)
        self._lstStates.bind("<<ListboxSelect>>", self.senate)
        conOFlstStates.set(tuple(stateList))
        self._conOFlstSenators = StringVar()
        self._lstSenators = Listbox(window, width=18, height=2,
                                    listvariable=self._conOFlstSenators)
        self._lstSenators.grid(row=2, column=2, padx=8, pady=5, sticky=N)
        yscroll["command"] = self._lstStates.yview
        window.mainloop()
```

```
    def senate(self, e):
        self.L = []
        state = self._lstStates.get(self._lstStates.curselection())
        infile = open("Senate114.txt", 'r')
        for line in infile:
            temp = line.split(',')
            if temp[1] == state:
                self.L.append(temp[0] + "  " + temp[2])
        infile.close()
        self._conOFlstSenators.set(tuple(self.L))

Senators()
```

19.

```
from tkinter import *

def checkAnswer():
    m = people.index(lstPeople.get(lstPeople.curselection()))
    n = places.index(lstPlaces.get(lstPlaces.curselection()))
    if m == n:
        conOFentAnswer.set("CORRECT")
    else:
        conOFentAnswer.set("INCORRECT")

window = Tk()
window.title("Workplaces")
Label(window, text="Person").grid(row=0, column=0)
Label(window, text="Workplace").grid(row=0, column=1)
people = ["Bruce Wayne", "Clark Kent", "Peter Parker",
          "Rick Blaine", "Willie Wonka"]
places = ["Wayne Enterprises", "Daily Planet", "Daily Bugle",
          "Rick's American Cafe", "Chocolate Factory"]
placesSorted = list(places)
placesSorted.sort()
conOFlstPeople = StringVar()
lstPeople = Listbox(window, width=12, height=5, exportselection=0,
            listvariable=conOFlstPeople)
lstPeople.grid(row=1, column=0, padx=10)
conOFlstPeople.set(tuple(people))
conOFlstPlaces = StringVar()
lstPlaces = Listbox(window, width=18, height=5, exportselection=0,
                    listvariable=conOFlstPlaces)
lstPlaces.grid(row=1, column=1, padx=10)
conOFlstPlaces.set(tuple(placesSorted))
btnDetermine = Button(window, text="Determine if Match is Correct",
                      command=checkAnswer)
btnDetermine.grid(row=2, column=0, columnspan=2, pady=5)
Label(window, text="Answer:").grid(row=3, column=0, sticky=E)
conOFentAnswer = StringVar()
entAnswer = Entry(window, width=10, textvariable=conOFentAnswer,
                  state="readonly")
entAnswer.grid(row=3, column=1, padx=10, pady=(0,5), sticky=W)
window.mainloop()
```





19. (Object-oriented style)

```
from tkinter import *


class Workplaces:
    def __init__(self):
        window = Tk()
        window.title("Workplaces")
        Label(window, text="Person").grid(row=0, column=0)
        Label(window, text="Workplace").grid(row=0, column=1)
        self._people = ["Bruce Wayne", "Clark Kent", "Peter Parker",
                        "Rick Blaine", "Willie Wonka"]
        self._places = ["Wayne Enterprises", "Daily Planet",
              "Daily Bugle", "Rick's American Cafe", "Chocolate Factory"]
        self._placesSorted = list(self._places)
        self._placesSorted.sort()
        self._conOFlstPeople = StringVar()
        self._lstPeople = Listbox(window, width=12, height=5,
                    exportselection=0, listvariable=self._conOFlstPeople)
        self._lstPeople.grid(row=1, column=0, padx=10)
        self._conOFlstPeople.set(tuple(self._people))
        self._conOFlstPlaces = StringVar()
        self._lstPlaces = Listbox(window, width=18, height=5,
                    exportselection=0, listvariable=self._conOFlstPlaces)
        self._lstPlaces.grid(row=1, column=1, padx=10)
        self._conOFlstPlaces.set(tuple(self._placesSorted))
        self._btnDetermine = Button(window,
                                    text="Determine if Match is Correct",
                                    command=self.checkAnswer)
        self._btnDetermine.grid(row=2, column=0, columnspan=2, pady=5)
        Label(window, text="Answer:").grid(row=3, column=0, sticky=E)
        self._conOFentAnswer = StringVar()
        self._entAnswer = Entry(window, width=10,
                                textvariable=self._conOFentAnswer,
                                state="readonly")
        self._entAnswer.grid(row=3, column=1, padx=10, pady=(0,5),
                             sticky=W)
        window.mainloop()
```

```
    def checkAnswer(self):
        m = self._people.index(
                    self._lstPeople.get(self._lstPeople.curselection()))
        n = self._places.index(
                    self._lstPlaces.get(self._lstPlaces.curselection()))
        if m == n:
            self._conOFentAnswer.set("CORRECT")
        else:
            self._conOFentAnswer.set("INCORRECT")

Workplaces()
```