INSTRUCTOR SOLUTIONS MANUAL

to accompany
An Introduction to Programming Using Python

by David I. Schneider

CONTENTS

Answers

CHAPTER 2

EXERCISES 2.1

1. 12 **2.** 49 **3.** .125 **4.** **23** **5.** 8 **6.** -96 **7.** 2 **8.** 2

9. 1 **10.** 3 **11.** 1 **12.** 0 **13.** Not valid **14.** Not valid

15. Valid **16.** Not valid **17.** Not valid **18.** Not valid **19.** 10

20. 14 **21.** 16 **22.** 16 **23.** 9 **24.** 8

25.
```
print((7 * 8) + 5)
```

26.
```
(1 + (2 * 9)) **3
```

27.
```
print(.055 * 20)
```

28.
```
15 - (3 * (2 + (3 ** 4)))
```

29.
```
print(17 * (3 + 162))
```

30.
```
(4 + (1 / 2)) - (3 + (5 / 8))
```

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

32.

	bal	inter	withDr
bal = 100	100	does not exist	does not exist
inter = .05	100	.05	does not exist
withDr = 25	100	.05	25
bal += (inter * bal)	105	.05	25
bal = bal - withDr	80	.05	

33. `24` **34.** `1 8 9` **35.** `10` **36.** `225`

37. `2 15` **38.** `5 10` 39. The third line should read `c = a + b`.

40. 1,234 should not contain a comma; $100 should not have a dollar sign; Deposit should begin with a lowercase letter d.

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

49. -2 50. 2 **51.** 0 52. 1 **53.** 6 54. 1

55. `cost += 5` **56.** `sum *= 2` **57.** `cost /= 6` **58.** `sum -= 7`

59. `sum %= 2` **60.** `cost //= 3`

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

62. costPerShare = 25.625
    numberOfShares = 400
    amount = costPerShare * numberOfShares
    print(amount)

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

64. fixedCosts = 5000
    pricePerUnit = 8
    costPerUnit = 6
    breakEvenPoint = fixedCosts / (pricePerUnit - costPerUnit)
    print(breakEvenPoint)

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

66. balance = 100
    balance = ((1.05) * balance) + 100
    balance = ((1.05) * balance) + 100
    balance *= 1.05
    print(round(balance, 2))

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

68. purchasePrice = 10
    sellingPrice = 15
    percentProfit = 100 * ((sellingPrice - purchasePrice) / purchasePrice)
    print(percentProfit)

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

70. initialVelocity = 50
    initialHeight = 5
    t = 3
    height = (-16 * (t ** 2)) + (initialVelocity * t) + initialHeight
    print(height)
```

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

72. miles = 23695 - 23352
    gallonsUsed = 14
    milesPerGallon = miles / gallonsUsed
    print(milesPerGallon)

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

74. pizzasPerSecond = 350
    secondsInDay = 60 * 60 * 24
    numPerDay = pizzasPerSecond * secondsInDay
    print(numPerDay))

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

76. pop2000 = 281
    pop2050 = 404
    percentGrowth = round(100 * ((pop2050 - pop2000) / pop2000))
    print(round(percentGrowth))

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

78. cubicFeet = (5280 ** 3)
    caloriesPercubicFoot = 48600
    totalNumberOfCalories = cubicFeet * caloriesPercubicFoot
    print(totalNumberOfCalories))
```

EXERCISES 2.2

```
1. Python   2. Hello    3. Ernie    4. Bert     5. "o"          6. "o"

7. "h"      8. "n"      9. "Pyt"    10. []      11. "Py"        12. "Thon"

13. "h"     14. "ytho"  15. "th"    16. "th"    17. "Python"    19. 2

20. -1      21. -1      23. 10      24. 3       25. 2           26. 5

27. -1      28. -1      29. 3       30. "BRRR"  31. 8 ball      32. 4

33. "8 BALL"            35. "hon"   37. "The Artist"            39. 5
```

40. `"King Lear"` 41. `7` 42. `6` 43. `2` 45. `"King Kong"`

46. `1` 47.
```
12
MUNICIPALITY
City
6
```
48.
```
9
Microsoft
os
5
```

49. `flute` 50. `Acute` 51. `Your age is 21.` 52. `Fred has 2 children.`

53. `A ROSE IS A ROSE IS A ROSE` 54. `PYTHON` 55. `WALLAWALLA`

56. `murmur` 57. `goodbye` 58. `eighth` 59. `Mmmmmmm.`

60. `***YES***` 61. `a b` 62. `spamspamspamspam`

63. `76 trombones` 64. `5.5` 65. `17` 66. `8` 67. `8` 68. `8`

69. `The Great 9` 70. `The Dynamic Duo` 71. `s[:-1]` 72. `s[2:]`

73. `-8` 74. `7` 75. True 76. True 77. True 78. True

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

80. `I came to Casablanca for the waters.` should be surrounded by quotation marks.

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

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

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

84. Should be written `print('George "Babe" Ruth')`

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

86. `lower` should be changed to `lower()`

87. A string cannot be concatenated with a number.

88. The characters in a number cannot be indexed.

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

90. The `len` function can not be applied to numbers.

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

92. `show[9]` is not valid since the string "Spamalot" does not have a character of index 9.

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

94.
```
item = "ketchup"
regularPrice = 1.8
discount = 0.27
print(regularPrice - discount) + " is the sale price of " + item + "."
```

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

96.
```
prefix = "Fore"
print(prefix + "warned is " + prefix + "armed.")
```

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.
```

98.
```
## Calculate training heart rate.
age = float(input("Enter your age: "))
rhr = int(input("Enter your resting heart rate: "))
thr = .7 * (220 - age) + (.3 * rhr)
print("Training heart rate:", round(thr), "beats/minute.")
```

```
Enter your age: 20
Enter your resting heart rate: 70
Training heart rate: 161 beats/min.
```

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
```

100.
```
## Calculate cost of electricity.
wattage = int(input("Enter wattage: "))
hoursUsed = float(input("Enter number of hours used: "))
price = float(input("Enter price per kWh in cents: "))
cost = (wattage * hoursUsed) / (1000 * price)
print("Cost of electricity:", '$' + str(round(cost, 2)))
```

```
Enter wattage: 100
Enter number of hours used: 720
Enter price per kWh in cents: 11.76
Cost of electricity: $6.12
```

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.
```

102.
```
## Calculate price/earnings ratio.
earningsPerShare = float(input("Enter earnings per share: "))
pricePerShare = float(input("Enter price per share: "))
PEratio = pricePerShare / earningsPerShare
print("Price-to-Earnings ratio:", PEratio)
```

```
Enter earnings per share: 5.25
Enter price per share: 68.25
Price-to-Earnings ratio: 13.0
```

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
```

104.
```
## Convert a percent to a decimal.
percentage = input("Enter percentage: ")
percent = float(percentage[:-1]) / 100
print("Equivalent decimal:", percent)
```

```
Enter percentage: 125%
Equivalent decimal: 1.25
```

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.

106.
```
## Server's tip.
bill = float(input("Enter amount of  bill: "))
percentage = float(input("Enter percentage tip: "))
tip = (bill * percentage) / 100
print("Tip:", '$' + str(round(tip, 2)))
```

```
Enter amount of bill: 21.50
Enter percentage tip: 18
Tip: $3.87
```

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%
```

108.
```
## Marketing terms.
purchasePrice = float(input("Enter purchase price: "))
sellingPrice = float(input("Enter selling price: "))
markup = sellingPrice - purchasePrice
percentageMarkup = 100 * (markup / purchasePrice)
profitMargin = 100 * (markup / sellingPrice)
print("Markup:", '$' + str(round(markup, 2)))
print("Percentage markup:", str(round(percentageMarkup, 2)) + '%')
print("Profit margin:", str(round(profitMargin, 2)) + '%')
```

```
Enter purchase price: 215
Enter selling price: 645
Markup: $430.0
Percentage markup: 200.0%
Profit margin: 66.67%
```

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
```

110.
```
## Word replacement.
sentence = input("Enter a sentence: ")
word1 = input("Enter word to replace: ")
word2 = input("Enter replacement word: ")
location = sentence.find(word1)
newSentence = sentence[:location] + word2 + sentence[location + len(word1):]
print(newSentence)
```

```
Enter a sentence: Live long and prosper.
Enter word to replace: prosper
Enter replacement word: proper
Live long and proper.
```

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
```

112.
```
## Convert lengths.
numberOfInches = int(input("Enter number of inches: "))
feet = numberOfInches // 12
inches = numberOfInches % 12
print(numberOfInches, "inches equals", feet, "feet and", inches, "inches.")
```

```
Enter number of inches: 185
185 inches is 15 feet and 5 inches.
```

EXERCISES 2.3

1. Bon Voyage!

2. Price: $23.45

3. Portion: 90%

4. Python

5. 1 x 2 x 3

6. tic-tac-toe

7. father-in-law

8. father-in-law

9. T-shirt

10. spam and eggs

11. Python

12. on-site repair

13.
```
Hello
World!
```

14.
```
Hello
World!
```

15.
```
One     Two     Three    Four
```

16.
```
1       2       3
Detroit Lions
Indianapolis    Colts
```

17.
```
NUMBER  SQUARE
2       4
3       9
```

18.
```
COUNTRY LAND AREA
India   2.5 million sq km
China   9.6 million sq km
```

19.
```
Hello           World!
Hello           World!
```

20.
```
STATE          CAPITAL
North Dakota   Bismarck
South Dakota   Pierre
```

21.
```
01234567890
          A  B  C
```

```
22. 0123456789012345     23. 01234567890123456   24. 01234567890
      one  two  three           one  two  three          A  B  C

25. 0123456789     26. 0123456789
       12.30%              1,234
      123.0%             1,234
    1,230.00%          1,234

27. $1,234.57      28. 1,234         29. 1       30. #1,234.00

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%

32. Major          Percent of Students
    Biology              6.2%
    Psychology           5.4%
    Nursing              4.7%

33. Be yourself - everyone else is taken.

34. Plan first, code later

35. Always look on the bright side of life.

36. And now for something completely different.

37. The product of 3 and 4 is 12.

38. The chances of winning the Powerball Lottery are 1 in 175,223,510.

39. The square root of 2 is about 1.4142.

40. Pi is approximately 3.14159.

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

42. The cost of Alaska was about $10.86 per square mile.

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

44. 12% of the members of the U.S. Senate are from New England.

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

46. The area of Alaska is 17.5% of the area of the U.S.

47. abracadabra

48. When you have nothing to say, say nothing.

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

50. If you can dream it, you can do it. - Walt Disney
```

51. Yes **52.** 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
```

54.
```
## Calculate income.
revenue = eval(input("Enter revenue: "))
expenses = eval(input("Enter expenses: "))
netIncome = revenue - expenses
print("Net income: ${0:,.2f}".format(netIncome))
```

```
Enter revenue: 550000
Enter expenses: 410000
Net income: $140,000.00
```

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%
```

56.
```
## Calculte a change in salary.
beginningSalary = float(input("Enter beginning salary: "))
raisedSalary = 1.05 * 1.05 * 1.05 * beginningSalary
percentChange = (raisedSalary - beginningSalary) / beginningSalary
print("New salary: ${0:,.2f}".format(raisedSalary))
print("Change: {0:.2%}".format(percentChange))
```

```
Enter beginning salary: 35000
New salary: $40,516.88
Change: 15.76%
```

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
```

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

```
Enter future value: 10000
Enter interest rate (as %): 4
Enter number of years: 6
Present value: $7,903.15
```

EXERCISES 2.4

1. Pennsylvania Hawaii 2. New Jersey, Arizona 3. Alaska Hawaii

4. 50 5. Delaware Delaware 6. 0 7. 48 8. 22

9. Ohio 10. Hawaii Hawaii 11. DELAWARE 12. Puerto Rico

13. ['Puerto Rico'] 14. Georgia 15. United States 16. 48

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

18. ['Pennsylvania', 'New Jersey', 'Georgia']

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

20. ['New Mexico', 'Arizona', 'Alaska']

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

22. ['Delaware'] 23. ['Arizona', 'Alaska', 'Hawaii']

24. ['Alaska', 'Hawaii'] 25. [] 26. [] 27. Georgia

28. Arizona 29. ['Alaska', 'Hawaii'] 30. Massachusetts

31. New Mexico 32. New Jersey 33. 10 34. 30 35. 0 36. 50

37. 48 38. 46 39. ['Hawaii', 'Puerto Rico', 'Guam']

40. ['Alaska', 'Hawaii', ['Puerto Rico', 'Guam']]

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

42. ['Arizona', "Seward's Folly", 'Hawaii']

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

44. ['Delaware', 'Commonwealth of Pennsylvania', 'Pennsylvania']

45. ['New', 'Mexico'] 46. ['Jersey', 'New', 'Mexico']
    ['New', 'Jersey']

47. Pennsylvania,New Jersey,Georgia 48. ['Jersey', 'New', 'Mexico']

49. 8 50. 8 51. 100 52. 7 53. 0 54. 98

55. Largest Number: 8 56. Smallest Number: 0 57. Total: 16

58. Average 4.0

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

60. ['all', 'for', 'one'] 61. Babbage, Charles 62. Guido Rossum

63. Middle Name: van 64. Python 65. When in the course of human events

66. Less is more. 67. editor-in-chief 68. merry-go-round

69. e**pluribus**unum 70. ['around', 'the', 'clock']

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

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

74. a bcd 75. programmer

76. Live let live. 77. Follow your own star.

78.
```
Largest Number: 8
Length: 4
Total: 16
Number list: [6, 2, 8, 0]
```

79. 987-654-3219 80. Dairy 81. [3, 9, 6] 82. (-5, 17, 123)

83. each 84. (0, 2, 3) 85. ['soprano', 'tenor', 'alto', 'bass']

86. ['soprano', 'tenor', 'alto', 'bass'] 87. ['gold', 'silver', 'bronze']

88. ['gold', 'silver', 'bronze'] 89. murmur

90. [0, 0, 0, 0] 91. ('Happy', 'Sneezy', 'Bashful')

92. ['Nina', 'Pinta'] 93. 1 94. 2

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

96. The statement `word[1] = 'p'` is not valid since strings are immutable.

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

98. The tuple does not have an item of index 4.

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

100. Tuples do not support the *append* method.

```
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
```

```
102. ## Analyze a sentence
     sentence = input("Enter a sentence: ")
     L = sentence.split()
     print("First word:", L[0])
     print("Last word:", L[-1][:-1])
```

```
Enter a sentence: Reach for the stars.
First word: Reach
Last word: stars
```

```
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
```

```
104. ## Extract the middle name from a three-part name.
     name = input("Enter a 3-part Name: ")
     L = name.split()
     print("Middle Name:", L[1])
```

```
Enter a 3-part name: Augusta Ada Byron
Middle name: Ada
```

PROGRAMMING PROJECTS CHAPTER 2

```
1. ## Make change for an amount of less than one dollar.
   amount = int(input("Enter amount of change: "))
   remainder = amount
   quarters = remainder // 25
   remainder %= 25
   dimes = remainder // 10
   remainder %= 10
   nickels = remainder // 5
   remainder %= 5
   cents = remainder
   print("Quarters:", quarters, end="   ")
   print("\tDimes:", dimes)
   print("Nickels:", nickels, end="   ")
   print("\tCents:", cents)
```

```
Enter amount of change: 93
Quarters: 3       Dimes: 1
Nickels: 1        Cents: 3
```

2.
```
## Determine the monthly payment for a car loan.
loanAmount = float(input("Enter amount of loan: "))
interestRate = float(input("Enter interest rate (%): "))
numYears = float(input("Enter number of years: "))
i = interestRate / 1200
monthlyPayment = (i / (1 - ((1 + i) ** (-12 * numYears)))) * loanAmount
print("Monthly payment: ${0:,.2f}".format(monthlyPayment))
```

```
Enter amount of loan: 12000
Enter interest rate (%): 6.4
Enter number of years: 5
Monthly payment: $234.23
```

3.
```
faceValue = float(input("Enter face value of bond: "))
couponRate = float(input("Enter coupon interest rate: "))
interest = faceValue * couponRate
marketPrice = float(input("Enter current market price: "))
yrsUntilMaturity = float(input("Enter years until maturity: "))
a = (faceValue - marketPrice) / yrsUntilMaturity
b = (faceValue + marketPrice) / 2
ytm = (interest + a) / b
print("Approximate YTM: {0:.2%}".format(ytm))
```

```
Enter face value of bond: 1000
Enter coupon interest rate: .04
Enter current market price: 1180
Enter years until maturity: 15
Approximate YTM: 2.57%
```

4.
```
## Determine the unit price of a purchase.
price = float(input("Enter price of item: "))
print("Enter weight of item in pounds and ounces separately.")
pounds = float(input("Enter pounds: "))
ounces = float(input("Enter ounces: "))
weightInOunces = 16 * pounds + ounces
pricePerOunce = price / weightInOunces
print("Price per ounce: ${0:.2f}".format(pricePerOunce))
```

```
Enter price of item: 25.50
Enter weight of item in pounds and ounces separately.
Enter pounds: 1
Enter ounces: 9
Price per ounce: $1.02
```

5.
```
## Describe the distribution in a stock portfolio.
spy = float(input("Enter amount invested in SPY: "))
qqq = float(input("Enter amount invested in QQQ: "))
eem = float(input("Enter amount invested in EEM: "))
vxx = float(input("Enter amount invested in VXX: "))
total = spy + qqq + eem + vxx
print()
print("{0:6s}{1:>12s}".format("ETF", "PERCENTAGE"))
print("-" * 18)
```

```
print("{0:6s}{1:10.2%}".format("SPY", spy / total))
print("{0:6s}{1:10.2%}".format("QQQ", qqq / total))
print("{0:6s}{1:10.2%}".format("EEM", eem / total))
print("{0:6s}{1:10.2%}".format("VXX", vxx / total))
print()
print("{0:s}: ${1:,.2f}".format("TOTAL AMOUNT INVESTED", total))
```

```
Enter amount invested in SPY: 876543.21
Enter amount invested in QQQ: 234567.89
Enter amount invested in EEM: 345678.90
Enter amount invested in VXX: 123456.78

ETF     PERCENTAGE
------------------
SPY        55.47%
QQQ        14.84%
EEM        21.87%
VXX         7.81%

TOTAL AMOUNT INVESTED: $1,580,246.78
```

6.
```
## Convert a measurement from miles, yards, feet,
## and inches, to a metric one in meters, kilometers,
## and centimeters.
miles = float(input("Enter number of miles: "))
yards = float(input("Enter number of yards: "))
feet = float(input("Enter number of feet: "))
inches = float(input("Enter number of inches: "))
# Step #1: Add up given measurements into inches
totalInches = inches + 12 * feet + 36 * yards + 63360 * miles
# Step #2: Convert total inches into total meters
totalMeters = totalInches / 39.3700787
# Step #3: Compute kilometers, whole meters, and centimeters
# Step 3a: compute # of kilometers, subtract from meters
kilometers = int(totalMeters / 1000)
totalMeters = totalMeters - 1000 * kilometers
meters = int(totalMeters)
centimeters = 100 * (totalMeters - meters)
centimeters = round(centimeters, 1)
print("Metric length:")
print(" ", kilometers, "kilometers")
print(" ", meters, "meters")
print(" ", centimeters, "centimeters")
```

```
Enter number of miles: 5
Enter number of yards: 20
Enter number of feet: 2
Enter number of inches: 4
Metric length:
  8 kilometers
  65 meters
  73.5 centimeters
```

CHAPTER 3

EXERCISES 3.1

1. hi 2. C# 3. The letter before G is F. 4. B 5. Minimum: 3
Maximum: 17

6. Spread: 14 7. D is the 4th letter of the alphabet. 8. d 9. True

10. False 11. True 12. False 13. True 14. False 15. True

16. True 17. False 18. False 19. False 20. True 21. True

22. True 23. True 24. True 25. False 26. True 27. False

28. True 29. False 30.. true 31. False 32. False 33. False

34. False 35. True 36. True 37. False 38. False 39. False

40. False 41. True 42. False 43. False 44. True 45. Equivalent

46. Not equivalent 47. Not equivalent 48. Equivalent 49. Equivalent

50. Equivalent 51. Equivalent 52. Equivalent 53. Equivalent

54. Equivalent 55. `a <= b` 56. `(a != b) and (a != d)`

57. `(a >= b) or (c == d)` 58. `(a == b) or (a > b)`

59. `a > b` 60. 60. `(a == "") or (a >= b) or (len(a) >= 5)`

61. `ans in ['Y', 'y', "Yes", "yes"]`

62. `name in ["Athos", "Porthos", "Aramis"]` 63. `2010 <= year <= 2013`

64. `n in range(1, 7)` 65. `3 <= n < 9` 66. `1 < n <= 22` 67. `-20 < n <= 10`

68. `100 <= n <= 200` 69. True 70. False 71. True 72. True

73. True 74. True 75. True 76. True 77. True 78. False

79. False 80. False 81. False 82. False 83. False 84. False

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

EXERCISES 3.2

1. Less than ten. 2. Student 3. False 4. True

5. Remember, tomorrow is another day. 6. Your change contains 3 dollars.

7. 10 8. Cost of cloth is $15.50. 9. To be, or not to be.

10. `A is a vowel.` 11. `Hi` 12. `positive`

13. `A nonempty string is true.` 14. `An empty string is false.`

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"):
```

18. Syntax error: first line should be `if a == b:` 19. `a = 5` 20. `print("eleven")`

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

22.
```
if (state == "CA") and (city == "LA" or city == "SD"):
    print("Large city!")
```

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

24.
```
feet = eval(input("How tall (in feet) is the Statue of Liberty? "))
if (141 < feet < 161):
    print("Good")
else:
    print("Nope")
print("The statue is 151 feet tall from base to torch.")
```

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
```

26.
```
## Determine cost of bagels.
num = int(input("Enter number of bagels: "))
if num < 6:
    cost = 0.75 * num
else:
    cost = 0.6 * num
print("Cost is ${0:,.2f}.".format(cost))
```

```
Enter number of bagels: 12
Cost is $7.20.
```

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
```

28.
```
## Determine the cost of copies.
numberOfcopies = int(input("Enter number of copies: "))
if numberOfcopies < 100:
    cost = .05 * numberOfcopies
else:
    cost = 5 + 0.03 * (numberOfcopies - 100)
print("Cost is ${0:,.2f}.".format(cost))
```

```
Enter number of copies: 125
Cost is $5.75.
```

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.
```

30.
```
## Determine weekly pay (including overtime pay).
wage = float(input("Enter hourly wage: "))
hours = float(input("Enter number of hours worked: "))
if hours <= 40:
    grossPay = wage * hourse
else:
    grossPay = (wage * 40) + (1.5 * wage * (hours - 40))
print("Gross pay for week is ${0:,.2f}.".format(grossPay))
```

```
Enter hourly wage: 12.50
Enter number of hours worked: 47
Gross pay for week is $631.25.
```

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.
```

32.
```
## Convert a word to Pig Latin.
word = input("Enter word to translate: ")
word = word.lower()
firstLetter = word[0]
if firstLetter in "aeiou":
    word += "way"
else:
    listOfVowelPositions = []
    if 'a' in word:
        listOfVowelPositions.append(word.find('a'))
    if 'e' in word:
        listOfVowelPositions.append(word.find('e'))
    if 'i' in word:
        listOfVowelPositions.append(word.find('i'))
    if 'o' in word:
        listOfVowelPositions.append(word.find('o'))
    if 'u' in word:
        listOfVowelPositions.append(word.find('u'))
    positionOfFirstVowel = min(listOfVowelPositions)
    word = word[positionOfFirstVowel:] + word[:positionOfFirstVowel] + "ay"
print("The word in Pig Latin is " + word + ".")
```

```
Enter word to translate: chip
The word in Pig Latin is ipchay.
```

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.
```

34.
```
## Process a savings account withdrawal.
balance = float(input("Enter current balance: "))
amountOfWithdrawal = float(input("Enter amount of withdrawal: "))
if (balance >= amountOfWithdrawal):
    balance -= amountOfWithdrawal
    print("The new balance is ${0:,.2f}.".format(balance))
    if balance < 150:
        print("Balance below $150", "Warning")
else:
    print("Withdrawal denied.")
```

```
Enter current balance: 200
Enter amount of withdrawal: 25
The new balance is $175.00.
```

```
Enter current balance: 200
Enter amount of withdrawal: 225
Withdrawal denied.
```

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.
```

36.
```
## Determine if year is a leap year.
yr = int(input("Enter a year: "))
if (yr % 4 == 0) and ((yr % 100 != 0) or (yr % 400 == 0)):
    print(yr, "is a leap year.")
else:
    print(yr, "is not a leap year.")
```

```
Enter a year: 2016
2016 is a leap year.
```

```
Enter a year: 2018
2016 is not a leap year.
```

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.
```

38.
```
## A quiz
print("The four railroad properties")
print("are Reading, Pennsylvania,")
print("B & O, and Short Line.")
answer = input("Which is not a railroad? ")
if answer == "Short Line":
    print("Correct.")
    print(answer, "is a bus company.")
else:
    print("Incorrect.")
    print(answer, "is a railroad.")
```

```
The four railroad properties
are Reading, Pennsylvania,
B & O, and Short Line.
Which is not a railroad? Short Line
Correct.
Short Line is a bus company.
```

```
The four railroad properties
are Reading, Pennsylvania,
B & O, and Short Line.
Which is not a railroad? Reading
Incorrect.
Reading is a railroad.
```

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.
```

40.
```
## Bestow graduation honors.
# Request grade point average.
gpa = eval(input("Enter your gpa: "))
# Determine if honors are warranted.
if gpa >= 3.9:
    honors = " summa cum laude."
if 3.6 <= gpa < 3.9:
    honors = " magna cum laude."
if (3.3 <= gpa < 3.6):
    honors = " cum laude."
if gpa < 3.3:
    honors = "."
# Display conclusion.
print("You graduated" + honors)
```

```
Enter your gpa: 3.7
You graduated magna cum laude.
```

```
Enter your gpa: 3.2
You graduated.
```

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.
```

42.
```
## Second-Suit Half-Off Sale
cost1 = float(input("Enter cost of first suit: "))
cost2 = float(input("Enter cost of second suit: "))
twoCosts = [cost1,  cost2]
cost = max(twoCosts) + (.5 * min(twoCosts))
print("Cost of the two suits is ${0:.2f}".format(cost))
```

```
Enter cost of first suit: 378.50
Enter cost of second suit: 495.99
Cost of the two suits is $685.24
```

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 2. 18 3. 10 4. 10

5. 20 6. Atlantic, Pacific, Antarctic

7. a
   b
   c
   d

8. Later than 1950.
   Earlier than 1970.
   They appeared on the Ed Sullivan show in February 1964.
   You answered the question correctly in 3 tries.

9. Infinite loop

10. The colon at the end of the *while* header is missing.

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

12. In the 5th line, = should be ==. Also, an *IndexError* exception error will be since the loop will attempt to evaluate list1[4] before the break statement is reached.

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

14.
```
L = [2, 4, 6, 8
print(sum(L))
```

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
```

16.
```
## Drop a ball and find number of bounces and total distance traveled.
coefOfRestitution = float(input("Enter coefficient of restitution: "))
height = float(input("Enter initial height in meters: "))
height *= 100   # convert to centimeters
distanceTraveled = 0
bounces = 1     # first bounce
distanceTraveled = height
while height * coefOfRestitution >= 10:
    bounces += 1
    height = coefOfRestitution * height
    distanceTraveled += 2 * height  # up then down again
distanceTraveled /= 100             # convert back to meters
print("Number if bounces:", bounces)
print("Meters traveled: {0:,.2f}".format(distanceTraveled))
```

```
Enter coefficient of restitution: .7
Enter initial height in meters: 8
Number of bounces: 13
Meters traveled: 44.82
```

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
```

18.
```
## Prime factorization
lstFactors = []
n = int(input("Enter a positive integer (> 1): "))
f = 2
while n > 1:
    if n // f == n / f:    # true if f divides n
        lstFactors.append(str(f))
        n = n // f
    else:
        f += 1
result = "  ".join(lstFactors)
print("Prime factors are", result)
```

```
Enter a positive integer (> 1): 2345
Prime factors are 5  7  67
```

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.
```

20.
```
## Determine the year that the world population will exceed
## 8 billion, assuming a 1.1% rate of increase.
yr = 2011       # start at 2011
pop = 7         # population of 7 billion
while pop <= 8:
    pop = (1.011) * pop
    yr += 1
print("World population will be \n8 billion in the year", str(yr) + ".")
```

```
World population will be
8 billion in the year 2025.
```

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.
```

22.
```
## Determine when Consumer Price Index will double.
cpiIn2014 = 238.35
cpi = cpiIn2014
years = 0
while cpi <=  2 * cpiIn2014:
    cpi = 1.025 * cpi
    years += 1
print("Consumer prices will")
print("double in", years, "years.")
```

```
Consumer prices will
double in 29 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.
```

24.
```
## Determine value of an increasing annuity.
months = 0
balance = 0
while balance <= 3000:
    balance = (1.0025 * balance) + 100
    months += 1
print("Annuity will be worth")
print("$3000 after", months, "months.")
```

```
Annuity will be worth
$3000 after 29 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.
```

26.
```
## Determine the half-life of Carbon-14.
amount = 1
years = 0
while amount >= .5:
    amount -= .00012 * amount
    years += 1
print("Carbon-14 has a half-life")
print("of", years, "years.")
```

```
Carbon-14 has a half-life
of 5776 years.
```

27.
```
## Determine the 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.
```

28.
```
## Values of a decreasing annuity.
balance = float(input("Enter amount of deposit: "))
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))
```

```
Enter amount of deposit: 10000
Balance will be $73.19
after 17 months.
```

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.
```

30.
```
## Newton's Law of Cooling.
temperature = 212
count = 0
while temperature > 150:
    count += 1
    temperature -= (temperature - 70) * 0.079
print("The coffee will cool to below")
print("150 degrees in", count, "minutes.")
```

```
The coffee will cool to below
150 degrees in 7 minutes.
```

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

2. -11, -10, -9, -8

3. 2, 5, 8, 11

4. 2010, 2015, 2020, 2025

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

6. 0

7. 11, 10, 9, 8

8. 12, 7

9. `range(4, 20, 5)`

10. `range(4)`

11. `range(-21, -17)`

12. `range(4, 0, -1)`

13. `range(20, 13, -3)`

14. `range(7, 8)`

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

26. 3

27. 15

28. code

29. n

30. P

31. 3 20

32. 13

33. The shortest word has length 5

34. 4

35. Three

36.
```
Leaf 1: sunshine
Leaf 2: rain
Leaf 3: the roses that bloom in the lane
Leaf 4: somebody I adore
```

37. 18

38. 30

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

40. Hawaii

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

42. A string cannot be concatenated with a number.

43. The print function call is missing parentheses.

44. An individual item in a list can only be altered if referenced by its index.

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.

46. An individual item in a list can only be altered if referenced by its index.

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

48.
```
for num in range(4):
    print("hello")
```

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

50.
```
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.
```

52.
```
## Remove dashes from a phone number.
phoneNum = input("Enter a telephone number: ")
numWithoutDashes = ""
for ch in phoneNum:
    if ch != '-':
        numWithoutDashes += ch
print("Number without dashes is", numWithoutDashes + '.')
```

```
Enter a telephone number: 982-876-5432
Number without dashes is 9828765432.
```

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.
```

54.
```
## Find largest of three numbers.
largest = eval(input("Enter a number: "))
for i in range(2):
    num = eval(input("Enter a number: "))
    if num > largest:
        largest = num
print("Largest number:", largest)
```

```
Enter a number: 3.4
Enter a number: 9.3
Enter a number: 5.5
Largest number: 9.3
```

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.
```

56.
```
## Calculate sum of first 100 positive integers.
sum = 0
for i in range(1, 101):
    sum += i
print("The sum 1 + 2 + ... + 100")
print("is", str(sum) + '.')
```

```
The sum 1 + 2 + ... + 100
is 5050.
```

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.
```

58.
```
## Determine if a word contains every vowel.
word = input("Enter a word: ")
word = word.upper()
vowels = "AEIOU"
isVowelWord = True
for letter in vowels:
    if letter not in word:
        isVowelWord = False
        break
if isVowelWord:
    print(word, "is a vowel word.")
else:
    print(word, "is a not a vowel word.")
```

```
Enter a word: education
EDUCATION is a vowel word.
```

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.
```

60.
```
## Compare simple interest and compound interest.
print("  {0}  {1}".format("Simple Interest", "Compound Interest"))
amount = 1000
simple = amount
compound = amount
for i in range(1, 5):
    simple += .05 * amount
    compound = 1.05 * compound
    print("{0} ${1:,.2f}        ${2:,.2f}".format(i, simple, compound))
```

```
  Simple Interest  Compound Interest
1 $1,050.00        $1,050.00
2 $1,100.00        $1,102.50
3 $1,150.00        $1,157.62
4 $1,200.00        $1,215.51
```

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
```

62.
```
## Calculate balances in an increasing annuity.
print("         BALANCE AT")
print("YEAR    ", "END OF YEAR")
balance = 0
year = 2014
for i in range(1, 61):
    balance = 1.0025 * balance + 100
    if i % 12 == 0:
        print(year, "    ${0:,.2f}".format(balance))
        year += 1
```

```
          BALANCE AT
YEAR      END OF YEAR
2014      $1,216.64
2015      $2,470.28
2016      $3,762.06
2017      $5,093.12
2018      $6,464.67
```

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
```

64.
```
## Depreciation of an automobile.
value = 20000
##print("{0}     ${1:7,.2f}".format(0, value))
for i in range(1, 5):
    value = .85 * value
    print("{0}     ${1:7,.2f}".format(i, value))
```

```
1     $17,000.00
2     $14,450.00
3     $12,282.50
4     $10,440.12
```

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
```

66.
```
## Calculate a median.
howMany = int(input("How many numbers would you like to enter? "))
listOfNumbers = []
for i in range(howMany):
    num = int(input("Enter a number: "))
    listOfNumbers.append(num)
listOfNumbers.sort()
if howMany % 2 == 1:
    median = listOfNumbers[int(howMany / 2)]
else:
    m = int(howMany / 2)
    median = (listOfNumbers[m - 1] + listOfNumbers[m]) / 2
print("Median:", median)
```

```
How many numbers do you want to enter? 4
Enter a number: 9
Enter a number: 3
Enter a number: 6
Enter a number: 5
Median: 5.5
```

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.
```

68.
```
## Calculate value of stock at end of year.
value = 10000
for i in range(6):
    value -= .16 * value
for i in range(6):
    value += .18 * value
print("The value of the stock at the")
print("end of the year was ${0:,.2f}.".format(value))
```

```
The value of the stock at the
end of the year was $9,483.48.
```

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.
```

70.
```
## Determine when the Steelers first won a Super Bowl game.
num = 0
teams = open("SBWinners.txt", 'r')
for team in teams:
    num += 1
    if team.strip() == "Steelers":
        break
teams.close()
print("The Steelers first won the")
print("Super Bowl in game #" + str(num) + '.')
```

```
The Steelers first won the
Super Bowl in game #9.
```

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%
```

72.
```
## Calculate an average grade. Drop two lowest grades.
grades = []
for i in range(5):
    grade = int(input("Enter one of five grades: "))
    grades.append(grade)
grades.sort()
grades = grades[2:]
average = sum(grades) / len(grades)
print("Average grade: {0:.2f}".format(average))
```

```
Enter one of five grades: 84
Enter one of five grades: 96
Enter one of five grades: 88
Enter one of five grades: 77
Enter one of five grades: 90
Average grade: 91.33
```

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
```

74.
```
## Big Cross-Out Swindle
startingWord = "NAISNIENLGELTETWEORRSD"
crossedOutLetters = ""
remainingLetters = ""
oddLetter = True
for ch in startingWord:
    if oddLetter:
        crossedOutLetters += ch
    else:
        remainingLetters += ch
    oddLetter = not oddLetter
print("Starting word:", startingWord)
spreadoutWord = ""
for ch in crossedOutLetters:
    spreadoutWord += ch + " "
crossedOutLetters = spreadoutWord.rstrip()
spreadoutWord = ""
for ch in remainingLetters:
    spreadoutWord += ch + " "
remainingLetters = spreadoutWord.rstrip()
print("Crossed-out letters:", crossedOutLetters)
print("Remaining letters:", remainingLetters)
```

```
Starting word: NAISNIENLGELTETWEORRSD
Crossed out letters: N I N E L E T T E R S
Remaining letters: A S I N G L E W O R D
```

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
```

76.
```
## Display 13 original states in alphabetical order.
infile = open("States.txt", 'r')
states = [line.rstrip() for line in infile]
infile.close()
originalStates = states[:13]
originalStates.sort()
for state in originalStates:
    print(state)
```

```
Connecticut
Delaware
Georgia
Maryland
Massachusetts
New Hampshire
New Jersey
New York
North Carolina
Pennsylvania
Rhode Island
South Carolina
Virginia
```

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.
```

78.
```
## Find a special number.
for num in range(1000, 10000):
    list1 = list(str(num))
    list1.reverse()
    revNum = int("".join(list1))
    if revNum == 4 * num:
        break
print("Since 4 times", num, "is", str(revNum) + ',')
print("the special number is", str(num) + '.')
```

```
Since 4 times 2178 is 8712,
the special number is 2178.
```

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.
```

80.
```
## Determine 34th president.
infile = open("USPres.txt", 'r')
num = 0
for pres in infile:
    num += 1
    if num == 34:
        print("The 34th president was")
        print(pres.strip() + '.')
        break
infile.close()
```

```
The 34th president was
Dwight Eisenhower.
```

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.
```

82.
```
## Count the sum of the digits in the first million positive integers.
sum = 0
for i in range(1, 1000001):
    strNum = str(i)
    for j in range(len(strNum)):
        sum += int(strNum[j])
print("The sum of the digits in the numbers")
print("from 1 to one million is {0:,d}.".format(sum))
```

```
The sum of the digits in the numbers
from 1 to one million is 27,000,001.
```

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
```

PROGRAMMING PROJECTS CHAPTER 3

1.
```
## Analyze a car loan.
p = eval(input("Enter the amount of the loan: "))
a = eval(input("Enter the interest rate: "))
n = int(input("Enter the duration in months: "))
r = a / 1200
monthlyPayment = (p * r) / (1 - (1 + r) ** (-n))
monthlyPayment = round(monthlyPayment, 2)
print("Monthly Payment: ${0:,.2f}".format(monthlyPayment))
totalInterest = n * monthlyPayment - p
print("totalInterestPaid: ${0:,.2f}".format(totalInterest))
```

```
Enter the amount of the loan: 18000
Enter the interest rate: 5.25
Enter the duration in months: 60
Monthly payment: $341.75
Total interest paid: $2,504.86
```

2.
```
## Determine the real roots of a quadratic equation
## of the form ax**2 + bx +c = 0.
a = float(input("Enter a: "))
b = float(input("Enter b: "))
c = float(input("Enter c: "))
# Test that first coefficient is nonzero.
if a == 0:
    print("a must be non-zero.")
else:
    # Determine solution
    delta = b ** 2 - (4 * a * c)
    if delta < 0:     # no real solutions
        print("No real solutions")
    elif delta == 0:  # one real solution
        sol = -b / (2 * a)
        if int(sol) == sol:
            print("Solution: {0:,.0f}".format(sol))
        else:
            print("Solution: {0:,.4f}".format(sol))
    else:             # two real solutions
        sol1 = (-b + (delta ** 0.5)) / (2 * a)
        sol2 = (-b - (delta ** 0.5)) / (2 * a)
        if int(sol1) == sol1 and int(sol2) == sol2:
            print("Solutions: {0:,.0f} and {1:,.0f}".format(sol1, sol2))
        else:
            print("Solutions: {0:,.4f} and {1:,.4f}".format(sol1, sol2))
```

```
Enter a: 1
Enter b: -11
Enter c: 28
Solutions: 7 and 4
```

```
Enter a: 1
Enter b: -10
Enter c: 25
Solutions: 5
```

```
Enter a: 1
Enter b: 2
Enter c: 3
No real solutions
```

3.
```
## Analyze caffeine absorption.
amount = 130
hrs = 0
print("CAFFEINE VALUES")
while amount > (130 / 2):
    amount = 0.87 * amount
    hrs += 1
print("One cup:", "less than 65 mg. will remain after", hrs, "hours.")
amount = 130
for i in range(24):
    amount = 0.87 * amount
print("One cup: {0:.2f} mg. will remain after 24 hours.".format(amount))
amount = 0
for i in range(25):
    amount = 0.87 * amount + 130
print("One cup: {0:.2f} mg. will remain after 24 hours.".format(amount))
```

```
CAFFEINE VALUES
One cup: less than 65 mg. will remain after 5 hours.
One cup: 4.60 mg. will remain after 24 hours.
Hourly cups: 969.24 mg. will remain after 24 hours.
```

4.
```
## Analyze Rule of 72.
print("\t\tRule of 72")
print("Interest\tDoubling Time\tActual Doubling")
print("Rate\t\t(in years)\tTime (in years)")
for i in range(1,21):
    amount = 100
    years = 0
    while amount < 200:
        amount *= 1 + (i / 100)
        years += 1
    print(str(i) + '%' + "\t\t" +  str(72 // i) + "\t\t"+ str(years))
```

```
                Rule of 72
Interest        Doubling Time   Actual Doubling
Rate            (in years)      Time (in years)
1%              72          ... 70
2%              36              36
3%              24              24
4%              18              18
5%              14              15
6%              12              12
7%              10              11
8%              9               10
9%              8               9
10%             7               8
11%             6               7
12%             6               7
13%             5               6
14%             5               6
15%             4               5
16%             4               5
17%             4               5
18%             4               5
19%             3               4
20%             3               4
```

5.
```
## Compare IRA balances when starting early vs. starting late.
earl = 0
larry = 0
for year in range(15):
    #first 15 years between 2015-2063
    earl = 1.04 * earl + 5000
for year in range(33):
    #remaining 33 years between 2015-2063
    larry = 1.04 * larry + 5000
    earl *= 1.04
earlDeposited = 15 * 5000
larryDeposited = 33 * 5000
print("AMOUNTS DEPOSITED".center(45))
print("Earl: ${:<10,.2f}\t    Larry: ${:10,.2f}"
      .format(earlDeposited, larryDeposited))
print("AMOUNTS IN IRA UPON RETIREMENT".center(45))
print("Earl: ${:10,.2f}\t    Larry: ${:10,.2f}".format(earl, larry))
```

```
                AMOUNTS DEPOSITED
Earl: $75,000.00            Larry: $165,000.00
        AMOUNTS IN IRA UPON RETIREMENT
Earl: $365,268.39           Larry: $331,047.64
```

6.
```
## Encode words by sound with the Soundex System.
word = input("Enter a word to code: ")
code = word[0]
lastChar = ""
for ch in word[1:].lower():
    if ch in "bfpv" and lastChar != '1':
        code += '1'
        lastChar = '1'
    elif ch in "cgjkqsxz" and lastChar != '2':
        code += '2'
        lastChar = '2'
    elif ch in "dt" and lastChar != '3':
        code += '3'
        lastChar = '3'
    elif ch == 'l' and lastChar != '4':
        code += '4'
        lastChar = '4'
    elif ch in "mn" and lastChar != '5':
        code += '5'
        lastChar = '5'
    elif ch == 'r' and lastChar != '6':
        code += '6'
        lastChar = '6'
# Make the code 4 characters long.
extraZeros = 4 - len(code)
if extraZeros > 0:
    code += '0' * extraZeros
else:
    code = code[:4]
print("The coded word is {0}.".format(code))
```

```
Enter a word to code: Robert
The coded word is R163.
```

7.
```
## Validate a credit card number.
num = input("Enter a credit card number: ")
evenSum = 0
oddSum = 0
for i in range(0, len(num), 2):
    digit = int(num[i]) * 2
    if digit >= 10:
        digit -= 9
    evenSum += digit
for i in range(1, len(num) + 1, 2):
    oddSum += int(num[i])
if (evenSum + oddSum) % 10 == 0 and len(num) == 14:
    print("The number is valid.")
else:
    print("The number is not valid.")
```

```
Enter a credit card number: 58667936100244
The number is valid.
```

8.
```
## Determine if a word or phrase is a palindrome.
phrase = input("Enter a word or phrase: ")
phrase = phrase.upper()
strippedPhrase = ""
for char in phrase:
    if (48 <= ord(char) <= 57) or (65 <= ord(char) <= 90):
        strippedPhrase += char
flag = True
n = len(strippedPhrase)
for j in range(int(n / 2)):
    if strippedPhrase[j] != strippedPhrase[n - j - 1]:
        flag = False
        break
if flag:
    print(phrase, "is a palindrome.")
else:
    print(phrase, "is not a palindrome.")
```

```
Enter a word or phrase: A man, a plan, a canal: Panama.
A MAN, A PLAN, A CANAL: PANAMA. is a palindrome.
```

CHAPTER 4

EXERCISES 4.1

1.
```
H
w
```

2. You can park around 500 cars on a five-acre lot.

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

4. 27 is an odd number.

5. Your income tax is $499.00

6. There are 100 U.S. senators.

```
7. Why do clocks run clockwise?

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

8. It was the best of times.          9. 168 hours in a wee
   It was the worst of times.            76 trombones in the big parade

10. divorced      11. President Bush is a graduate of Yale.
    beheaded          President Obama is a graduate of Columbia.
    died
    divorced
    beheaded
    survived

12. George Washington was president number 1

13. 7       14. 5       15. Fredrick      16. SPAM
    5           5

17. Total cost: $106.00       18. You owe $900,000.00 in estate taxes.  19. 5

20. brag    21. When in the course of human events    22. 90
    garb

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

24.
```

```
Enter a quotation: You miss 100% of the shots you never take.—Wayne Gretsky

MENU
1. Count number of vowels in the quotation.
2. Count number of uppercase letters in the quotation.
Select 1 or 2 from menu: 1
Number of vowels: 15
```

```
Enter a quotation: You miss 100% of the shots you never take.—Wayne Gretsky

MENU
1. Count number of vowels in the quotation.
2. Count number of uppercase letters in the quotation.
Select 1 or 2 from menu: 2
Number of uppercase letters: 3
```

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

26.
```
def howMany(s1, s2):
    ## Count the number of nonoverlapping occurrances of s2 in s1
    if s2 != "":
        n = 0   # number of nonoverlapping occurrances
        i = 0
        while i < len(s1):
            if s1[i:].startswith(s2):
                n += 1
                i = i + len(s2)
            else:
                i += 1
        return n
    else:
        return len(s1) + 1
```

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.
```

28.
```
def main():
    ## Calculate a factorial.
    n = getN()
    print(str(n) + '!', "is", fact(n))

def getN():
    while True:
        n = eval(input("Enter a positive integer: "))
        if isinstance(n, int) and (n > 0):
            return n
        else:
            print("The number you entered is not a positve integer.")
```

```
def fact(n):
    product = 1
    for i in range(2, n + 1):
        product *= i
    return product

main()
```

```
Enter a positive whole number: 5
5! is 120
```

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.
```

30.
```
def main():
    ## Calculate a pay raise.
    firstName = getFirstName()
    lastName = getLastName()
    currentSalary = getCurrentSalary()
    newSalary = calculateNewSalary(currentSalary)
    displayResult(firstName, lastName, newSalary)

def getFirstName():
    firstName = input("Enter first name: ")
    return firstName

def getLastName():
    lastName = input("Enter last name: ")
    return lastName

def getCurrentSalary():
    currentSalary = float(input("Enter current salary: "))
    return currentSalary

def calculateNewSalary(currentSalary):
    if currentSalary < 40000:
        return (currentSalary * 1.05)
    else:
        return 2000 + currentSalary + (.02 * (currentSalary - 40000))

def displayResult(firstName, lastName, newSalary):
    print("New salary for {0} {1}: ${2:,.2f}"
          .format(firstName, lastName, newSalary))

main()
```

```
Enter first name: John
Enter last name: Doe
Enter current salary: 48000
New salary for John Doe: $50,160.00
```

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
```

32.
```
months = []

def main():
    ## display months containing the letter r.
    global months
    fillList()
    months = deleteNoRs()
    displayMonths()

def fillList():
    global months
    infile = open("Months.txt", 'r')
    months = [line.rstrip() for line in infile]
    infile.close

def deleteNoRs():
    reducedList = []
    for i in range(12):
        if 'r' in months[i].lower():
            reducedList.append(months[i])
    return reducedList
```

```
    def displayMonths():
        print("The R months are:")
        print((", ").join(months))

    main()
```

```
The R months are:
January, February, March, April, September, October, November, December
```

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
         for color in open("Colors.txt", 'r'):
             if color.startswith(letter):
                 colors.append(color.rstrip())

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

    main()
```

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

34.
```
    def main():
        ## Calculate the amount of a pension.
        age = getAge()
        monthsOfService = getMonthsOfService()
        salary1 = getFirstSalary()
        salary2 = getSecondSalary()
        salary3 = getThirdSalary()
        pension = calculatePension(age, monthsOfService, salary1,
                                   salary2, salary3)
        displayPension(pension)

    def getAge():
        age = eval(input("Enter your age: "))
        return age
```

```
def getMonthsOfService():
    monthsOfService = int(input("Enter number of months of service: "))
    return monthsOfService

def getFirstSalary():
    salary1 = eval(input("Enter first of three highest salaries: "))
    return salary1

def getSecondSalary():
    salary2 = eval(input("Enter second of three highest salaries: "))
    return salary2

def getThirdSalary():
    salary3 = eval(input("Enter third of three highest salaries: "))
    return salary3

def calculatePension(age, monthsOfService, salary1, salary2, salary3):
    ave = round((salary1 + salary2 + salary3) / 3, 2)
    yrs = monthsOfService / 12
    percentage = 36.25 + (2 * (yrs - 20))
    if percentage > 80:
        percentage = 80
    pension = ave * (percentage / 100)
    return pension

def displayPension(pension):
    print("Annual pension: ${0:,.2f}".format(pension))

main()
```

```
Enter your age: 65
Enter number of months of service: 448
Enter first of three highest salaries: 123456.78
Enter second of three highest salaries: 119876.55
Enter third of three highest salaries: 107546.45
Annual pension: $82,944.08
```

EXERCISES 4.2

1. 24 blackbirds baked in a pie.

2. Keep cool, but don't freeze.
   Source: A jar of mayonnaise.

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

4. Enter your numeric grade: 92
   You passed with a grade of 92.

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

6. Enter a name: Fred
   Enter a year of birth: 1995
   Fred will be 25 years old in 2020.

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

8. ['pear', 'apple', 'Banana']
   ['Banana', 'apple', 'pear']

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

10.
```
['wink']
['wink', 'wink']
['wink', 'wink', 'wink']
```

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

12.
```
Enter first integer: 4
Enter second integer: 25
Sum: 29
Product: 100
```

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

14.
```
Johann Sebastian Bach
Franz Joseph Haydn
Wolfgang Amadeus Mozart
Ralph Vaughan Williams
```

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

16.
```
['e', 'unum', 'pluribus']
['a', 'l', 'M', 'o', 'S', 't']
```

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

18.
```
C C++ Java PHP Python Ruby VB
```

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

20.
```
C VB C++ PHP Java Ruby Python
```

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

22.
```
-2 -3 4 5 6
```

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

24.
```
VB Python Ruby
```

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

26.
```
['Democratic', 'Sequoia', 'Equals', 'Brrr', 'Break', 'Two']
```

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

28.
```
['democratic', 'sequoia']
```

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

30.
```
['se', 'br', 'tw']
```

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

32.
```
outcome: [3.0, 2.0, 1.0]
```

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

34. A list consisting of the 50 states in alphabetical order.

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

36. The states with four-letter names

37. Valid 38. Valid 39. Valid 40. Not valid 41. Not valid 42. Not valid

43. Valid 44. Valid 45. Not valid 46. Valid 47. **Almost** 48. **sponge**

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
```

50.
```
def main():
    ## Determine semester grade.
    grade = getAverageGrade()
    semesterGrade = calculateLetterGrade(grade)
    print("Semester grade:", semesterGrade)

def getAverageGrade():
    midtermGrade = int(input("Enter grade on midterm exam: "))
    finalExamGrade = int(input("Enter grade on final exam: "))
    return ceil((midtermGrade + 2 * finalExamGrade) / 3)

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

def calculateLetterGrade(grade):
    if grade >= 90:
        return "A"
    elif grade >= 80:
        return "B"
    elif grade >= 70:
        return "C"
    elif grade >= 60:
        return "D"
    else:
        return "F"

main()
```

```
Enter grade on midterm: 88
Enter grade of final exam: 91
Semester Grade: A
```

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.
```

52.
```
def main():
    ## Determine semester grade.
    grades = []
    for i in range(1, 6):
        grade = eval(input("Enter grade " + str(i) + ": "))
        grades.append(grade)
    grades.sort()
    grades = grades[2:]
    (rng, ave) = analyzeGrades(grades)
    print("Range:", rng)
    print("Average:", ave)

def analyzeGrades(grades):
    rng = grades[-1] - grades[0]
    ave = sum(grades) / len(grades)
    return (rng, ave)

main()
```

```
Enter grade 1: 90
Enter grade 2: 75
Enter grade 3: 85
Enter grade 4: 72
Enter grade 5: 80
Range: 10
Average: 85
```

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
```

54.
```
def main():
    ## Sort states by population in descending order.
    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=sortByPopulation, reverse=True)
    print("Sorted by population in descending order:")
    for state in NE:
        print(state[0], " ", end="")

def sortByPopulation(state):
    return state[2]

main()
```

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

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
```

56.
```
def main():
    ## Sort New England states by length of name.
    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=sortByLengthOfName)
    print("Sorted by length of name in ascending order:")
    for state in NE:
        print(state[0], " ", end="")

def sortByLengthOfName(state):
    return len(state[0])

main()
```

```
Sorted by length of name in ascending order:
Maine  Vermont  Connecticut  Rhode Island  New Hampshire  Massachusetts
```

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
```

58.
```
def main():
    ## Sort numbers by the sum of their digits in ascending order.
    numbers = [865, 1169, 1208, 1243, 290]
    numbers.sort(key=sumOfDigits)
    print("Sorted by sum of digits:")
    print(numbers)

def sumOfDigits(num):
    listNums = list(str(num))
    for i in range(len(listNums)):
        listNums[i] = int(listNums[i])
    return sum(listNums)

main()
```

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

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]
```

60.
```
def main():
    ## Sort numbers in descending order by their last digit.
    numbers = [865, 1169, 1208, 1243, 290]
    numbers.sort(key=lastDigit, reverse=True)
    print("Sorted by last digit:")
    print(numbers)

def lastDigit(num):
    return str(num)[-1]

main()
```

```
Sorted by last digit:
[1169, 1208, 865, 1243, 290]
```

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]
```

62.
```
def main():
    ## Sort U.S. presidents alphabetically by last name.
    infile = open("USPres.txt", 'r')
    listPres = [pres.rstrip() for pres in infile]
    infile.close()
    listPres.sort(key=sortByLastName)
    for i in range(6):
        print(listPres[i])

def sortByLastName(pres):
    return pres.split()[-1]

main()
```

```
John Adams
John Q. Adams
Chester Arthur
James Buchanan
George H. W. Bush
George W. Bush
```

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
```

64.
```
def main():
    ## Sort states by length of name in descending order.
    infile = open("States.txt", 'r')
    listStates = [state.rstrip() for state in infile]
    infile.close()
    listStates.sort(key=sortByLengthOfName, reverse=True)
    for i in range(6):
        print(listStates[i])

def sortByLengthOfName(state):
    return len(state)

main()
```

```
South Carolina
North Carolina
Massachusetts
New Hampshire
West Virginia
Pennsylvania
```

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
```

66.
```
def main():
    ## Calculate pay raise.
    (firstName, lastName, currentSalary) = getNameAndCurrentSalary()
    newSalary = calculateNewSalary(currentSalary)
    displayResult(firstName, lastName, newSalary)

def getNameAndCurrentSalary():
    firstName = input("Enter first name: ")
    lastName = input("Enter second name: ")
    currentSalary = float(input("Enter current salary: "))
    return (firstName, lastName, currentSalary)

def calculateNewSalary(currentSalary):
    if currentSalary < 40000:
        return (currentSalary * 1.05)
    else:
        return 2000 + currentSalary + (.02 * (currentSalary - 40000))

def displayResult(firstName, lastName, newSalary):
    print("New salary for {0} {1}: ${2:,.2f}"
          .format(firstName, lastName, newSalary))

main()
```

```
Enter first name: John
Enter last name: Doe
Enter current salary: 48000
New salary for John Doe: $50,160.00.
```

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
```

68.
```
def main():
    ## Analyze monthly payment of mortgage.
    annualRateOfInterest, monthlyPayment, begBalance = inputData()
    (intForMonth, redOfPrincipal, endBalance)= \
        calculateValues(annualRateOfInterest, monthlyPayment, begBalance)
    displayOutput(intForMonth, redOfPrincipal, endBalance)

def inputData():
    annualRateOfInterest = eval(input("Enter annual rate of interest: "))
    monthlyPayment = eval(input("Enter monthly payment: "))
    begBalance = eval(input("Enter beg. of month balance: "))
    return (annualRateOfInterest, monthlyPayment, begBalance)

def calculateValues(annualRateOfInterest, monthlyPayment, begBalance):
    intForMonth = (annualRateOfInterest / 1200) * begBalance
    redOfPrincipal = monthlyPayment - intForMonth
    endBalance = begBalance - redOfPrincipal
    return (intForMonth, redOfPrincipal, endBalance)

def displayOutput(intForMonth, redOfPrincipal, endBalance):
    print("Interest paid for the month: ${0:,.2f}".format(intForMonth))
    print("Reduction of principal: ${0:,.2f}".format(redOfPrincipal))
    print("End of month balance: ${0:,.2f}".format(endBalance))

main()
```

```
Enter annual rate of interest: 5
Enter monthly payment: 1932.56
Enter beg. of month balance: 357819.11
Interest paid for the month: $1,490.91
Reduction of principal: $441.56
End of month balance: $357,377.46
```

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
```

70.
```
def main():
    ## Use Wilson's Theorem to determine if a number is prime.
    n = int(input("Enter an integer greater than 1: "))
    if isPrime(n):
        print(n, "is a prime number.")
    else:
        print(n, "is not a prime number.")

def isPrime(n):
    if (factorial(n - 1) + 1) % n:
        return False
    else:
        return True

def factorial(n):
    value = 1
    for i in range(2, n + 1):
        value *= i
    return value

main()
```

```
Enter an integer greater than 1: 37
37 is a prime number.
```

PROGRAMMING PROJECTS CHAPTER 4

1.
```
def main():
    ## Analyze projectile motion.
    h0, v0 = getInput()
    print("The maximum height of the ball is " + \
          "{0:.2f} feet.".format(calculateMaximumHeight(h0, v0)))
    print("The ball will hit the ground after approximately " + \
          "{0:.2f} seconds.".format(timeToHitGround(h0, v0)))

def getInput():
    # Input the initial height and velocity of the ball
    h0 = eval(input("Enter the initial height of the ball: "))
    v0 = eval(input("Enter the initial velocity of the ball: "))
    return h0, v0

def heightOfBall(h0, v0, t):
    # Return height of ball after t seconds
    return h0 + v0 * t - 16 * t * t

def calculateMaximumHeight(h0, v0):
    return heightOfBall(h0, v0, v0 / 32)

def timeToHitGround(h0, v0):
    t = .01
    while heightOfBall(h0, v0, t) >= 0:
        t += .01
    return t

main()
```

```
Enter the initial height of the ball: 5
Enter the initial velocity of the ball: 34
The maximum height of the ball is 23.06 feet.
The ball will hit the ground after approximately 2.27 seconds.
```

2.
```
def main():
    ## Determine largest and smallest prime factors of a number.
    n = int(input("Enter a positive integer: "))
    print("Largest prime factor:", extremeFactors(n)[0])
    print("Smallest prime factor:", extremeFactors(n)[1])

def extremeFactors(n):
    listOfPrimeFactors = []
    f = 2
    while n > 1:
        if n // f == n / f:    # true if f divides n
            listOfPrimeFactors.append(f)
            n = n // f
        else:
            f += 1
    largestPrimeFactor = max(listOfPrimeFactors)
    smallestPrimeFactor = min(listOfPrimeFactors)
    return (largestPrimeFactor, smallestPrimeFactor)

main()
```

```
Enter a positive integer > 1: 2345
Largest prime factor: 67
Smallest prime factor: 5
```

3.
```
def main():
    ## Verbalize a number.
    number = int(input("Enter a positive integer: "))
    describeNumber(number)

def describeNumber(number):
    number = "{0:,d}".format(number)
    descriptors = ["", "thousand", "million", "billion", "trillion",
                   "quadrillion", "quintillion", "sextillion", "septillion"]
    numOfCommas = number.count(',')
    descriptors = descriptors[:numOfCommas + 1]
    loc = 3   # in case loop doesn't get entered; that is, no commas
    for i in range(numOfCommas, 0, -1):
        loc = number.find(',')
        front = number[:loc]
        front = front.strip('0')
        if front == "":
            front = '0'
        print("{0:>3} {1:s}".format(front, descriptors[i]))
        number = number[loc + 1:]
    front = number[:loc]
    front = front.strip('0')
    if front != "":
        print("{0:>3} {1:s}".format(front, descriptors[0]))
```

[[Output when number is 123,000,004,056,777,888,999,012,345]]

```
123 septillion
  0 sextillion
  4 quintillion
 56 quadrillion
777 trillion
888 billion
999 million
 12 thousand
345
```

4.
```
def main():
    ## Depreciation
    (item, purchYear, cost, numYears, methodOfDepreciation) = inputData()
    showDepSchedule(item, purchYear, cost, numYears, methodOfDepreciation)

def inputData():
    item = input("Enter name of item purchased: ")
    purchYear = int(input("Enter year purchased: "))
    cost = float(input("Enter cost of item: "))
    numYears = int(input("Enter estimated life of item (in years): "))
    methodOfDepreciation = input("Enter method of depreciation (SL or DDB): ")
    return (item, purchYear, cost, numYears, methodOfDepreciation)

def showDepSchedule(item, purchYear, cost, numYears, methodOfDepreciation):
    showHeading(item, purchYear, cost, numYears, methodOfDepreciation)
    if methodOfDepreciation == "SL":
        showSLtable(purchYear, cost, numYears)
    else:
        showDDBtable(purchYear, cost, numYears)

def showHeading(item, purchYear, cost, numYears, methodOfDepreciation):
    print()
    print("Description:", item)
    print("Year of purchase:", purchYear)
    print("Cost: ${0:,.2f}".format(cost))
    print("Estimated life:", numYears, "years")
    if methodOfDepreciation.upper() == "SL":
        method = "straight-line"
        print("Method of depreciation:", method)
    elif methodOfDepreciation.upper() == "DDB":
        method = "double-declining balance"
        print("Method of depreciation:",method)
    print()
    print("{0:5s} {1:>12s} {2:>15s} {3:>20s}".format("", "Value at",
                                    "Amount Deprec", "Total Depreciation"))
    print("{0:5s} {1:>12s} {2:>15s} {3:>20s}".format("", "Beg of Yr.",
                                       "During Year", "to End of Year"))

def showSLtable(purchYear, cost, numYears):
    straightLineDep = (1 / numYears) * cost
    value = cost
    totalDeprec = 0
    for i in range(numYears):
        depDuringYear = straightLineDep
        totalDeprec += depDuringYear
        print("{0:<5d} {1:12,.2f} {2:15,.2f} {3:20,.2f}".format(purchYear + i,
                                            value, depDuringYear, totalDeprec))
        value -=  straightLineDep
```

```
def showDDBtable(purchYear, cost, numYears):
    value = cost
    totalDeprec = 0
    for i in range(numYears - 1):
        depDuringYear = (2 / numYears) * value
        totalDeprec += depDuringYear
        print("{0:<5d} {1:12,.2f} {2:15,.2f} {3:20,.2f}".format(purchYear + i,
                                          value, depDuringYear, totalDeprec))
        value -=  depDuringYear
    print("{0:<5d} {1:12,.2f} {2:15,.2f} {3:20,.2f}".format(purchYear + i + 1,
                                            value, value, totalDeprec + value))

main()
```

```
Enter name of item purchased: computer
Enter year purchased: 2012
Enter cost of item: 2000
Enter estimated life of item (in years): 5
Enter method of depreciation (SL or DDB): DDB

Description: computer
Year of purchase: 2012
Cost: $2,000.00
Estimated life: 5 years
Method of depreciation: double-declining balance

          Value at   Amount Deprec   Total Depreciation
        Beg of Yr.     During Year       to End of Year
2012      2,000.00          800.00               800.00
2013      1,200.00          480.00             1,280.00
2014        720.00          288.00             1,568.00
2015        432.00          172.80             1,740.80
2016        259.20          259.20             2,000.00
```

5.
```
def main():
    ## Determine if word has 3 consecutive letters in alphabetical order.
    word = input("Enter a word: ")
    word = word.upper()
    if isTripleConsecutive(word):
        print(word, "contains three successive letters")
    else:
        print(word, "does not contain three successive letters")
    print("in consecutive alphabetical order.")

def isTripleConsecutive(word):
    n = len(word)
    for i in range(n - 2):
        threeLetters = word[i:i+3]
        if (ord(threeLetters[0:1]) + 1 ==
            ord(threeLetters[1:2]) and
            ord(threeLetters[1:2]) + 1 ==
            ord(threeLetters[2:3])):
            return True
    return False

main()
```

```
Enter a word: HIJACK
HIJACK contains three successive letters
in consecutive alphabetical order.
```

6.
```
def main():
    ## Calidate a ten-character ISBN number.
    isbn = input("Enter ten-character ISBN number: ")
    isbn = stripDashes(isbn)
    if checkFormat(isbn):
        if isValidISBN(isbn):
            print("The number is valid.")
        else:
            print("The number is not valid.")
    else:
        print("ISBN is not properly formatted.")

def stripDashes(isbn):
    noDashes = ""
    for ch in isbn:
        if ch != '-':
            noDashes += ch
    return noDashes

def checkFormat(isbn):
    if (len(isbn) == 10) and isbn[:-1].isdigit() and \
       (isbn[-1].isdigit() or isbn[-1] == 'X'):
        return True
    else:
        return False

def isValidISBN(isbn):
    L = list(isbn)
    if L[-1] == 'X':
        L[-1] = 10
    total = 0
    for i in range(10):
        total += (10 - i) * int(L[i])
        if total % 11:
            return False
        else:
            return True

main()
```

```
Enter an ISBN: 0-13-030657-6
The number is valid.
```

CHAPTER 5

EXERCISES 5.1

1. `Aloha`
2. `Hello`
3. `Hello`
`Aloha`
4. `Hello`
`Aloha`
`HelloAloha`
5. `6`
6. `[1, 3, 4]`
7. `[4, 1, 0, 1, 4]`
8. `[0, 1, 4]`
9. `Believe in yourself.`
10. `Never give up.`
11. `['a', 'c', 't']`
12. `['z', 'o', 'n', 'e', 'd']`
13. `ABC.txt` should be open for reading, not for writing.
14. `ABC.txt` should be surrounded by quotation marks.
15. `close()` should be called on the file object, *infile*, not on `ABC.txt`. That is, the last line should read `infile.close()`.
16. `ABC.txt` should be open for
17. The argument for write() must be a string, not an integer.
18. `len(outfile)` is not a valid function.
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.
20. `outfile.write((len(word))` is not valid since only a string can be written to a text file.
21. The file is cannot be read since it has been closed.
22. A set cannot have a list as one of its elements
23. The file `ABC.txt` is created. Nothing is displayed on the monitor.
24. The statement `File already exists` is displayed.

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

26.
```
def findItemsinBoth(list1, list2)
    s = set(list1).intersection(set(list2))
    return list(s)
```

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

28.
```
names = ["Donald Shell", "Harlan Mills", "Donald Knuth", "Alan Kay"]
setLN = {name.split()[-1] for name in names}
print(setLN)
```

29.
```
## Count the words in the Gettysburg Address.
infile = open("Gettysburg.txt")
originalLine = infile.readline()
infile.close()
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.
```

30. The new file will contain the names of the people who subscribe to either the New York Times or the Wall Street Journal (or both).

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

32. The new file will contain the names of the people who subscribe to the New York Times, but do not subscribe to 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()
```

34.
```
def main():
    ## Count the number of numbers in the file Numbers.txt.
    count = getCount("Numbers.txt")
    print("The file Numbers.txt \ncontains", count, "numbers.")

def getCount(fileName):
    infile = open("Numbers.txt", 'r')
    count = 0
    for line in infile:
        count += 1
    infile.close()
    return count

main()
```

```
The file Numbers.txt
contains 6 numbers.
```

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.
```

36.
```
def main():
    ## Display the smallest number in the file Numbers.txt.
    min = getMin("Numbers.txt")
    print("The smallest number in the \nfile Numbers.txt is",
          str(min) + ".")

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

main()
```

```
The smallest number in the
file Numbers.txt is 2.
```

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.
```

38.
```
def main():
    ## Display the average of the numbers in the file Numbers.txt.
    average = getAverage("Numbers.txt")
    print("The average of the numbers in \nthe file Numbers.txt is {0:,.1f}."
          .format(average))

def getAverage(fileName):
    infile = open("Numbers.txt", 'r')
    sum = 0
    quantity = 0
    for line in infile:
        sum += int(line)
        quantity += 1
    infile.close()
    return sum / quantity

main()
```

```
The average of the numbers in
the file Numbers.txt is 5.0.
```

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.
```

40.
```
import os

def main():
    ## Delete months that do not contain the letter r.
    infile = open("SomeMonths.txt", 'r')
    outfile = open("Temp.txt", 'w')
    for month in infile:
        if 'r' not in month.lower():
            outfile.write(month)
    infile.close()
    outfile.close()
    os.remove("SomeMonths.txt")
    os.rename("Temp.txt", "SomeMonths.txt")

main()
```

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")
```

42.
```
import os

def main():
    ## Delete states that do not begin with a vowel.
    infile = open("SomeStates.txt", 'r')
    outfile = open("Temp.txt", 'w')
    for state in infile:
        if state[:1] not in "AEIOU":
            outfile.write(state)
    infile.close()
    outfile.close()
    os.remove("SomeStates.txt")
    os.rename("Temp.txt", "SomeStates.txt")

main()
```

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')
    firstStates = {state.rstrip() for state in infile}
    infile.close()
    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()
```

44.
```
## Determine number of states that have produced presidents.
infile = open("PresStates.txt", 'r')
statesSet = {state.rstrip() for state in infile}
infile.close()
print(len(statesSet), "different states have")
print("produced presidents of the \nUnited States.")
```

```
18 different states have
produced presidents of the
United States.
```

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
```

```
46. ## Insert name into file.
    name = input("Enter name to be inserted into file: ")
    infile = open("Names.txt", 'r')
    setOfNames = {name for name in infile}
    infile.close()
    setOfNames.add(name + "\n")
    listOfNames = list(setOfNames)
    listOfNames.sort()
    outfile = open("Names.txt", 'w')
    outfile.writelines(listOfNames)
    outfile.close()
```

EXERCISES 5.2

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

2. Afghanistan is in Asia.
   Albania is in Europe.

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

4. Afghanistan's pop. density is 126.30 people per sq. mile.
   Albania's pop. density is 270.27 people per sq. mile.
```

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`.

6. 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 `Algeria,Africa` and `Angola,Africa`.

```
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
```

8.
```
def main():
    ## Determine best and worst performing stocks in the DOW.
    stockList = placeDataIntoList("DOW.txt")
    stockList.sort(key=byPercentGrowth)
    increase = (float(stockList[-1][5]) - float(stockList[-1][4])) / \
                                                 float(stockList[-1][4])
    print("Best performing stock: {0:1}  {1:0,.2f}%".
                              format(stockList[-1][0], 100 * increase))
    increase = (float(stockList[0][5]) - float(stockList[0][4])) / \
                                                  float(stockList[0][4])
    print("Worst performing stock: {0:1}  {1:0,.2f}%".
                               format(stockList[0][0], 100 * increase))

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 byPercentGrowth(stock):
    percentIncrease = (float(stock[5]) - float(stock[4])) / float(stock[4])
    return percentIncrease

main()
```

```
Best performing stock: Boeing  81.12%
Worst performing stock: International Business Machines  -2.08%
```

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%
```

10.
```
def main():
    ## Determine the Small Dogs of the DOW.
    stockList = placeDataIntoList("DOW.txt")
    stockList.sort(key=byEndOfYearPrice)
    displaySmallDogs(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 byEndOfYearPrice(stock):
        return stock[5]

    def displaySmallDogs(listOfStocks):
        print("{0:20} {1:8}   {2:s}".format("Company", "Symbol",
                                            "Price on 12/31/2013"))
        for i in range(5):
            print("{0:20} {1:8}   ${2:0.2f}".format(listOfStocks[i][0],
                              listOfStocks[i][1], listOfStocks[i][5]))

    main()
```

```
Company              Symbol   Price on 12/31/2013
Cisco Systems        CSCO     $22.26
Intel                INTC     $25.95
General Electric     GE       $28.03
Pfizer               PFE      $30.63
AT&T                 T        $35.16
```

11.
```
def main():
    ## Display justices appointed by a specified 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
```

12.
```
## Makeup of Supreme Court in January 2015
infile = open("Justices.txt", 'r')
justices = [line for line in infile if (int(line.split(',')[5]) == 0)]
justices.sort(key=lambda x: int(x.split(',')[4]))
print("Current Justices")
for justice in justices:
    print(justice.split(',')[0], justice.split(',')[1])
```

```
Current Justices:
Antonin Scalia
Anthony Kennedy
Clarence Thomas
Ruth Ginsburg
Stephen Breyer
John Roberts
Samuel Alito
Sonia Sotomayor
Elena Kagen
```

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
```

14.
```
def main():
    ## Display justices from a specified state.
    state = input("Enter a state abbreviation: ")
    justices = getJusticesByState(state)
    fixCurrentJustices(justices)
    justices.sort(key=lambda justice: justice[5] - justice[4],
                  reverse=True)
    print("\n{0:18} {1:20}  {2}".format("Justice", "Appointing Pres",
                                       "Yrs Served"))
    for justice in justices:
        print("{0:18} {1:20}  {2}".format(justice[0] + " " + justice[1],
                    justice[2].split(" ")[-1], justice[5] - justice[4]))

def getJusticesByState(state):
    infile = open("Justices.txt", 'r')
    listOfRecords = [line for line in infile if line.split(',')[3] == state]
    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 a state abbreviation: NH

Justice            Appointing Pres   Yrs Served
David Souter       Bush              19
Levi Woodbury      Polk              6
```

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
```

16.
```
    def main():
        ## Determine accomplishments of computer pioneers.
        displayNames("Pioneers.txt")
        print('\n')
        name = input("Enter the name of a computer pioneer: ")
        displayAccomplishment("Pioneers.txt", name)

    def displayNames(fileName):
        infile = open(fileName, 'r')
        for line in infile:
            lineList = line.split(',')
        print((lineList[0] + '\t').expandtabs(20), end="")
        infile.close()

    def displayAccomplishment(fileName, name):
        infile = open(fileName, 'r')
        for line in infile:
            lineList = line.split(',')
            if lineList[0] == name:
                print(name, lineList[1].rstrip() + '.')
                infile.close()
                break

    main()
```

```
Charles Babbage     Augusta Ada Byron   Alan Turing         John V. Atanasoff
Grace M. Hopper     John Mauchley       J. Presper Eckert   John von Neumann
John Backus         Reynold B. Johnson  Harlan B. Mills     Donald E. Knuth
Ted Hoff            Stan Mazer          Robert Noyce        Federico Faggin
Douglas Engelbart   Bill Gates          Paul Allen          Stephen Wozniak
Stephen Jobs        Dennis Ritchie      Ken Thompson        Alan Kay
Tim Berners-Lee     Charles Simonyi     Bjarne Stroustrup   Richard M. Stallman
Marc Andreessen     James Gosling       Linus Torvalds      Guido van Rossum

Enter the name of a computer pioneer: Alan Turing
Alan Turing was a computer science theorist.
```

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
```

18.
```
def main():
    ## Determine the last college founded in a given state before 1800.
    abbrev = input("Enter a state abbreviation: ")
    colleges = getOrderedListOfColleges(abbrev)
    displayResult(colleges, abbrev)

def getOrderedListOfColleges(abbrev):
    # Colleges will be ordered by date founded.
    infile = open("Colleges.txt", 'r')
    colleges = [line for line in infile if line.split(',')[1] == abbrev]
    colleges.sort(key=lambda x: int(x.split(',')[2]))
    return colleges

def displayResult(colleges, abbrev):
    if len(colleges) > 0:
        print("Last college in", abbrev, "founded before 1800:")
        print(colleges[-1].split(',')[0])
    else:
        print(abbrev, "had no colleges before 1800.")

main()
```

```
Enter a state abbreviation: PA
Last college in PA founded before 1800:
University of Pittsburgh
```

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
```

20.
```
## Display data about a requested state.
state = input("Enter the name of a state: ")
infile = open("StatesANC.txt", 'r')
found = False
state_data = infile.readline()
while (found == False) and (state_data != ""):
    data = state_data.split(",")
    if data[0] == state:
        print("Abbreviation:", data[1])
        print("Nickname:", data[2])
        print("Capital:", data[3].rstrip())
        found = True
    state_data = infile.readline()
infile.close()
```

```
Enter the name of a state: Ohio
Abbreviation: OH
Nickname: Buckeye State
Capital: Columbus
```

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
```

22.
```
## Determine Oscar winning film for a given year.
films = open("Oscars.txt", 'r')
incorrect = True
while incorrect:
    year = int(input("Enter a year from 1928-2013: "))
    if (year >= 1928) and (year <= 2013):
        incorrect = False
        infile = open("Oscars.txt", 'r')
        flicks = [film.rstrip() for film in infile]
        infile.close()
        film = flicks[year - 1928]
        data = film.split(',')
        print("Best File:", data[0])
        print("Genre:", data[1]
        films.close()
    else:
        print("Year must be between 1928 and 2013.\n")
        incorrect = True
```

```
Enter year from 1928-2013: 2012
Best Film: Argo
Genre: drama
```

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()
```

24.
```
def main():
    ## Markdown the price of a saddle by 20% and
    ## store the new price list into Cowboy2.txt.
    infile = open("Cowboy.txt", 'r')
    outfile = open("Cowboy2.txt", 'w')
    for line in infile:
        data = line.split(',')
        if data[0] == "Saddle":
            newPrice = round(0.8 * eval(data[1]), 2)
            outfile.write("Saddle," + str(newPrice) + "\n")
        else:
            outfile.write(line)
    outfile.close()
    infile.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()
```

```
3 Colt Peacemaker: $36.60
2 Holster: $4.00
10 Levi Strauss jeans: $13.50
1 Saddle: $40.00
4 Stetson: $40.00
TOTAL: $134.10
```

26.
```
def main():
    ## Add an article to the end of the file Cowboy.txt.
    outfile = open("Cowboy.txt", 'a')
    outfile.write("Winchester Rifle,20.50\n")
    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 2. 6 3. ['NH', 'CT', 'ME', 'VT', 'MA', 'RI']

4. [6.5, 0.6, 1.5, 3.6, 1.3, 1.1]

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

6. True 7. absent 8. 1.1 9. VT 10. CT 11. 2

12. 5 13. 2 14. {} 15. VT CT MA RI ME NH

16. CT MA ME NH RI VT 17. 14.6 18. 14.6 19. 5 20. 6 21. 2

22. 755 23. False 24. CT MA ME NH RI VT 25. Aaron 26. Bonds

27. ['Aaron', 'Bonds'] 28. False 29. [755, 762] 30. ['Aaron', 'Bonds']

31. 762 32. NA 33. {'Aaron': 755}

34. {'Ruth': 714, 'Bonds': 762, 'Aaron': 755}

35. 0 36. {'Bonds': 763, 'Aaron': 755} 37. Bonds
Aaron 38. Aaron
Bonds

39. 762
755 40. Aaron
Bonds 41. 762 42. 750

43. {'Bonds': 761, 'Aaron': 755, 'Ruth': 714} 44. 755

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."))
```

46.
```
def determineRank(years):
    rank = {1:"Freshman", 2:"Sophmore", 3:"Junior"}
    return rank.get(years, "Senior")
```

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
```

48.
```
{'Ruth': {'hits': 2873, 'atBats': 8399}}
```

```
del topHitters[max(topHitters)]
del topHitters[min(topHitters)]
print(topHitters)
```
code that generates answer

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.
```

50.
```
topHitters = {"Gehrig":{"atBats":8061, "hits":2721},
              "Ruth":{"atBats":8399, "hits":2873},
              "Williams":{"atBats":7706, "hits":2654}}
hitList = []
for hitter in topHitters:
    hitList.append(topHitters[hitter]["hits"])
print("The most hits by one of the")
print("baseball players was ", max(hitList), '.', sep="")
```

```
The most hits by one of the
baseball players was 2873.
```

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
```

52.
```
import pickle

def main():
    justicesDict = createDictFromFile("JusticesDict.dat")
    displayJusticesFromState(justicesDict)
```

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

def displayJusticesFromState(dictionaryName):
    state = input("Enter a state abbreviation: ")
    for x in dictionaryName:
        if dictionaryName[x]["state"] == state:
            print("  {0:19}{1}".format(x, str(dictionaryName[x]["yrAppt"])))

main()
```

```
Enter a state abbreviation: NH
  David Souter     1990
  Levi Woodbury    1845
```

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.
```

54.
```
import pickle

def main():
    justicesDict = createDictFromFile("JusticesDict.dat")
    displayStatesWithJustices(justicesDict)

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

def displayStatesWithJustices(dictionaryName):
    states = {}
    for justice in dictionaryName:
        states[(dictionaryName[justice]["state"])] = 0
    print(len(states), "states have produced justices.")
    for justice in dictionaryName:
        states[(dictionaryName[justice]["state"])] += 1
    for state in sorted(states):
        print("  " +  state + ': ' + str(states[state]))

main()
```

```
31 states have produced justices.
  AL: 3
  AZ: 2
  CA: 5
  CO: 1
  CT: 3
  GA: 3
  IA: 2
  IL: 4
  IN: 1
  KS: 1
  KY: 5
```
first 11 states in list

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
```

56.
```
def main():
    ## display winningest Rose Bowl winners.
    roseBowlDict = createDictionaryFromTextFile("Rosebowl.txt")
    displayTopTenTeams(roseBowlDict)

def createDictionaryFromTextFile(fileName):
    # each item of the list will be a line of the file, without \n
    infile = open(fileName, 'r')
    roseBowlList = [line.rstrip() for line in infile]
    infile.close()
    aSet = set(roseBowlList)
    infile.close()
    roseBowlDict = {}
    for x in aSet:
        roseBowlDict[x] = 0
    for x in roseBowlList:
        roseBowlDict[x] += 1
    return roseBowlDict

def displayTopTenTeams(dictionaryName):
    dictionaryList = list(dictionaryName.items())
    dictionaryList.sort(key=f, reverse=True)
    print("Teams with four or more")
    print("Rose Bowl wins as of 2014.")
    for x in dictionaryList:
        if x[1] > 3:
            print("  " + x[0] + ':', x[1])

def f(k):
    return k[1]

main()
```

```
Teams with four or more
Rose Bowl wins as of 2014:
  USC: 24
  Washington: 8
  Michigan: 8
  Ohio State: 7
  Stanford: 6
  UCLA: 5
  Alabama: 4
  Michigan State: 4
```

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
```

58.
```
import pickle

def main():
    ## Display presidents with a specified first name.
    presDict = createDictFromBinaryFile("USpresStatesDict.dat")
    firstName = input("Enter a first name: ")
    displayOutput(presDict, firstName)

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

def displayOutput(dictName, name):
    foundFlag = False
    for k in dictName:
        x = k[1].split()
        if x[0] == name:
            print("  ", k[1], k[0])
            foundFlag = True
    if not foundFlag:
        print("No president has the first name", name + '.')

main()
```

```
Enter a first name: John
  John Adams
  John Q. Adams
  John Kennedy
  John Tyler
```

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
```

60.
```
import pickle

def main():
    ## Display the large cities in a specified state.
    largeCities = createDictionaryFromBinaryFile("LargeCitiesDict.dat")
    state = input("Enter the name of a state: ")
    getCities(state, largeCities)

def createDictionaryFromBinaryFile(fileName):
    # Assume pickle module has been imported.
    infile = open(fileName, 'rb')
    dictionaryName = pickle.load(infile)
    infile.close()
    return dictionaryName

def getCities(state, dictionaryName):
    if dictionaryName[state] != []:
        print("Large cities:", "  ".join(dictionaryName[state]))
    else:
        print("There are no large cities in", state + '.')

main()
```

```
Enter the name of a state: Maryland
Large cities: Baltimore
```

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
```

PROGRAMMING PROJECTS CHAPTER 5

1.
```
def main():
    ## Convert units of length.
    feet = populateDictionary("Units.txt")
    displayUnits(feet)
    orig, dest, length = getInput()
    ans = length * feet[orig] / feet[dest]
    print("Length in {0}: {1:,.4f}".format(dest, ans))


def populateDictionary(fileName):
    infile = open(fileName, 'r')
    feet = {}
    for line in infile:
        temp = line.split(',')
        feet[temp[0]] = eval(temp[1])
    return feet

def displayUnits(feet):
    print("UNITS OF LENGTH")
    i = 0
    for key in feet:
        print((key + '\t').expandtabs(12), end="")
        i += 1
        if i % 3 == 0:
            print()
    print()

def getInput():
    orig = input("Units to convert from: ")
    dest = input("Units to convert to: ")
    length = eval(input("Enter length in " + orig + ": "))
    return orig, dest, length

main()
```

```
UNITS OF LENGTH
inches      furlongs    yards
rods        miles       fathoms
meters      kilometers  feet

Units to convert from: yards
Units to convert to: miles
Enter length in yards: 555
Length in miles: 0.3153
```

2.
```
average = 0
stDev = 0

def main():
    ## Curve grades.
    global average
    global stDev
    infile = open("Scores.txt")
    scoresList = [eval(line) for line in infile]
    infile.close()
    numberOfScores = len(scoresList)
    average = sum(scoresList) / numberOfScores
    stDev = calculateStandardDeviation(scoresList, average)
    print("Number of scores:", numberOfScores)
    print("Average score:", average)
    print("Standard deviation of scores: {0:.2f}".format(stDev))
    # Curve the grades.
    for i in range(len(scoresList)):
        scoresList[i] = f(scoresList[i])
    dic = {'A':0, 'B':0,'C':0,'D':0,'F':0}
    for score in curvedScoresList:
        dic[score] += 1
    print("GRADE DISTRIBUTION AFTER CURVING GRADES.")
    for key in sorted(dic):
        print(key + ':', dic[key], end="    ")

def calculateStandardDeviation(listOfNumbers, m):
    n = len(listOfNumbers)
    listOfSquaresOfDeviations = [0] * n
    for i in range(n):
        listOfSquaresOfDeviations[i] = (listOfNumbers[i] - m) ** 2
    standardDeviation = (sum(listOfSquaresOfDeviations) / n) ** .5
    return standardDeviation

def f(x):
    ## Curve grade.
    if x >= average + (1.5 * stDev):
        return 'A'
    elif x >= average + (.5 * stDev):
        return 'B'
    elif x >= average - (.5 * stDev):
        return 'C'
    elif x >= average - (1.5 * stDev):
        return 'D'
    else:
        return 'F'

main()
```

```
Number of scores: 14
Average score: 71.0
Standard deviation of scores: 14.42
GRADE DISTRIBUTION AFTER CURVING GRADES.
A: 2    B: 1    C: 6    D: 4    F: 1
```

```
3. def main():
       ## Sort teams by percentage of games won.
       teams = placeRecordsIntoList("ALE.txt")
       teams.sort(key=lambda team: team[1]/team[2], reverse=True)
       createNewFile(teams)  # Create file of teams and their wins and losses.

   def placeRecordsIntoList(fileName):
       infile = open(fileName, 'r')
       listOfRecords = [line.rstrip() for line in infile]
       infile.close()
       for i in range(len(listOfRecords)):
           listOfRecords[i] = listOfRecords[i].split(',')
           listOfRecords[i][1] = eval(listOfRecords[i][1])  # won
           listOfRecords[i][2] = eval(listOfRecords[i][2])  # lost
       return listOfRecords

   def createNewFile(teams):
        outfile = open("OrderedALE.txt", 'w')
        for team in teams:
            outfile.write(team[0] + ',' + str(team[1]) + ',' +
                      str(team[2]) + ',' + str(round(team[1]/162, 3)) + "\n")
        outfile.close()

   main()
```

```
Baltimore,96,66,0.593
New York,84,78,0.519
Toronto,83,79,0.512
Tampa Bay,77,85,0.475
Boston,71,91,0.438
```

contents of file OrderedALE.txt

```
4(a). def main():
          ## Create file containing 114th Senate.
          infile = open("Senate113.txt", 'r')
          set1 = {line.rstrip() + "\n" for line in infile}
          infile.close()
          infile = open("RetiredSen.txt", 'r')
          set2 = {line.rstrip() + "\n" for line in infile}
          infile.close()
          infile = open("NewSen.txt", 'r')
          set3 = {line.rstrip() + "\n" for line in infile}
          infile.close()
          set1 = set1.difference(set2)
          set1 = set1.union(set3)
          listx = list(set1)
          listx.sort(key=lambda x: x.split(',')[1])  # sort by state
          outfile = open("Senate114.txt", 'w')
          outfile.writelines(listx)
          outfile.close()

      main()
```

4(b).
```
def main():
    ## Count the three affiliations.
    republicans = 0
    democrats = 0
    independents = 0
    infile = open("Senate114.txt", 'r')
    for line in infile:
        party = (line.split(','))[2]
        if party == 'R\n':
            republicans += 1
        elif party == 'D\n':
            democrats += 1
        else:
            independents += 1
    infile.close()
    print("Party Affiliations:")
    print(" ", "Republicans:", republicans)
    print(" ", "Democrats:", democrats)
    print(" ", "Independents:", independents)

main()
```

```
Party Affiliations:
  Republicans: 54
  Democrats: 44
  Independents: 2
```

4(c).
```
def main():
    ## Determine number of states having both senators from the same party.
    sameParty = 0
    infile = open("Senate114.txt", 'r')
    for i in range(25):
        line1 = infile.readline()
        line2 = infile.readline()
        if line1.split(',')[2] == line2.split(',')[2]:
            sameParty += 1
    print("In", sameParty, "states both senators ")
    print("are from the same party.")

main()
```

```
In 19 states both senators
are from the same party.
```

```
4(d). def main():
          ## Identify senators from a specified state.
          state = input("Enter the name of a state: ")
          infile = open("Senate114.txt", 'r')
          for line in infile:
              if line.split(',')[1] == state:
                  print(" ", line.split(',')[0])
                  line2 = infile.readline()
                  print(" ", line2.split(',')[0])
                  break

      main()
```

```
Enter the name of a state: Illinois
  Richard Durbin
  Mark Kirk
```

```
5. import pickle

   def main():
       ## Display tables containing information about bachelor degrees earned.
       degreesDict = createDictFromFile("DegreesDict.dat")
       print("1: Display bachelor degrees conferred.")
       print("2: Display percentage change.")
       print("3: Display histogram.")
       print("4: Quit.")
       while True:
           print()
           choice = int(input("Enter one of the options: "))
           if choice == 1:
               print()
               displayBachelorDegreesConferred(degreesDict)
           elif choice == 2:
               print()
               displayPercentageChange(degreesDict)
           elif choice == 3:
               print()
               displayHistogram(degreesDict)
           else:
               break
```

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

def displayBachelorDegreesConferred(degreesDict):
    print("Bachelor degrees conferred in certain fields.\n")
    print("{0:37}     {1}             {2}".
          format("Field of Study", 1981, 2010))
    for field in sorted(degreesDict):
        print("{0:37}{1:10,d}       {2:10,d}".
           format(field,degreesDict[field][0], degreesDict[field][1]))

def displayPercentageChange(degreesDict):
    print("Percentage change in bachelor degrees conferred.\n")
    print("{0:37}{1}".format("Field of Study", "% Change (1981-2010)"))
    for field in sorted(degreesDict, key=lambda k: f(degreesDict[k][1],
                degreesDict[k][0]), reverse=True):
        print("{0:42}{1:>7.1%}".format(field, f(degreesDict[field][1],
                degreesDict[field][0])))

def f(x, y):
    ## return percentage change
    return (x - y) / y

def displayHistogram(degreesDict):
    print("Bachelor degrees conferred in 2010 in certain fields.\n")
    for field in sorted(degreesDict, key=lambda k: degreesDict[k][1]):
        asterisks = '*' * round(degreesDict[field][1] / 10000)
        print("{0:>27} {1} {2:,d}".
              format(field, asterisks, degreesDict[field][1]))

main()
```

```
1: Display bachelor degrees conferred.
2: Display percentage change.
3: Display histogram.
4: Quit.

Enter one of the options: 1

Bachelor degrees conferred in certain fields.

Field of Study                          1981            2010
Business                              200,521         358,293
Computer and info. science             15,121          39,589
Education                             108,074         101,265
Engineering                            63,642          72,654
Social sciences and history           100,513         172,780

Enter one of the options: 2

Percentage change in bachelor degrees conferred.

Field of Study                     % Change (1981-2010)
Computer and info. science               161.8%
Business                                  78.7%
Social sciences and history               71.9%
Engineering                               14.2%
Education                                 -6.3%

Enter one of the options: 3

Bachelor degrees conferred in 2010 in certain fields.

 Computer and info. science **** 39,589
               Engineering ******* 72,654
                 Education ********** 101,265
Social sciences and history ***************** 172,780
                  Business ********************************** 358,293

Enter one of the options: 4
```

6.
```
## Display cars sorted by average miles per gallon.
infile = open("Mileage.txt",'r')
data = [line.rstrip() for line in infile]
infile.close()
data = [s.split(',') for s in data]
cars = {}
# Place data into dictionary.
for line in data:
    model = line[0]
    gal = float(line[1])
    # if we haven't come across this model yet
    if not model in cars:
        cars[model] = (1, gal)
    # if we have come across this model before
    else:
        cars[model] = (cars[model][0] + 1, cars[model][1] + gal)
# Convert dictionary into list with average mpg.
lst = [[model, tup[0] * 100 / tup[1]] for (model, tup) in
                                          list(cars.items())]
print("Model\t  MPG")
for car in sorted(lst, key=lambda k: k[1], reverse=True):
    print("{0}\t  {1:.2f}".format(car[0], car[1]))
```

```
Model       MPG
Prius       45.45
Camry       25.00
Sebring     23.81
Accord      23.44
Mustang     19.05
```

7.
```
def main():
    cities = placeRecordsIntoList("Cities.txt")
    # sort by percent population growth
    cities.sort(key=lambda city: (city[3] - city[2])/city[2], reverse=True)
    createNewFile(cities)  # Create file of cities and their % growth.

def placeRecordsIntoList(fileName):
    infile = open(fileName, 'r')
    listOfRecords = [line.rstrip() for line in infile]
    infile.close()
    for i in range(len(listOfRecords)):
        listOfRecords[i] = listOfRecords[i].split(',')
        listOfRecords[i][2] = eval(listOfRecords[i][2])  # population in 2000
        listOfRecords[i][3] = eval(listOfRecords[i][3])  # population in 2010
    return listOfRecords
```

```
    def createNewFile(cities):
        outfile = open("OrderedCities.txt", 'w')
        for city in cities:
            outfile.write(city[0] + ',' +
                   str(round(100*((city[3] - city[2])/city[2]),1)) + "\n")
        outfile.close()

    main()
```

8(a).
```
def main():
    ## Display name of currency and exchange rate for requested country.
    country = input("Enter the name of a country: ")
    infile = open("Exchrate.txt", 'r')
    foundFlag = False
    for line in infile:
        line = line.rstrip()
        data = line.split(',')
        if data[0] == country:
            foundFlag = True
            print("Currency:", data[1])
            print("Exchange rate:", data[2])
            break
    if not foundFlag:
        print(country, "is not in the file.")
main()
```

```
Enter the name of a country: Chile
Currency: Peso
Exchange rate: 591.408
```

8(b).
```
def main():
    ## Sort countries by number of units of their currency
    ## that can be purchased for one American dollar.
    infile = open("Exchrate.txt", 'r')
    countryList = [line.rstrip() for line in infile]
    infile.close()
    for i in range(len(countryList)):
        countryList[i] = countryList[i].split(',')
    countryList.sort(key=lambda x: x[2])
    for i in range(3):
        print(countryList[i][0])

main()
```

```
Kuwait
United Kingdom
Austria
```

8(c).
```
def main():
    ## Convert currency.
    country1 = input("Enter the name of first country: ")
    country2 = input("Enter the name of second country: ")
    amount = eval(input("Enter amount of money to convert: "))
    infile = open("Exchrate.txt", 'r')
    countryList = [line.rstrip() for line in infile]
    infile.close()
    d = {}
    for i in range(len(countryList)):
        countryList[i] = countryList[i].split(',')
        d[countryList[i][0]] = (countryList[i][1], eval(countryList[i][2]))
    print("{0} {1}s from {2} equals {3:,.2f} {4}s from {5}".
        format(amount, d[country1][0].lower(), country1, amount *
        d[country2][1] / d[country1][1], d[country2][0].lower(), country2))

main()
```

```
Enter the name of first country: America
Enter the name of second country: Chile
Enter amount of money to convert: 100
100 dollars from America equals 59,140.77 pesos from Chile
```

CHAPTER 6

EXERCISES 6.1

1. f 2. k 3. l 4. j 5. B 6. j 7. i 8. c 9. s 10. e 11. o

12. a 13. g 14. g 15. n 16. t 17. d 18. q 19. h 20. m 21. r 22. p

23. `You must enter a number.`

24. `Error occurred.`

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

26. `File Ages.txt contains an invalid age.`

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

28. `File Ages.txt not found.`

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
```

30.
```
## Remove a requested state capital from a list of state capitals.
while True:
    try:
        state = input("Enter a state capital to delete: ")
        stateCapitals.remove(state)
        print("Capital deleted.")
        break
    except ValueError:
        print("Not a state capital.")
```

```
Enter a state capital to delete: Chicago
Not a state capital.
Enter a state capital to delete: Springfield
Capital deleted.
```

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.

2. Toss a coin.

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

4. Roll a pair of dice.

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

6. Randomly choose the flavors for a three-scoop ice cream cone.

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

8. Randomly choose the officers for a club.

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))
```

10.
```
## Randomly select a perfect square integer from 1 to 10,000.
list1 = [n ** 2 for n in range(1, 101)]
x = random.choice(list1)
print(x)
```

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])
```

12.
```
## Select a vowel at random.
vowels = ['a', 'e', 'i', 'o', 'u']
x = random.choice(vowels)
print(x)
```

13.
```
import random
## 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))
```

14.
```
## Approximate the probability of obtaining 7 when rolling a pair of dice.
list1 = [1, 2, 3, 4, 5, 6]
list2 = [1, 2, 3, 4, 5, 6]
numberOfSevens = 0
for i in range(100000):
    if random.choice(list1) + random.choice(list2) == 7:
        numberOfSevens += 1
print(100 * numberOfSevens / 100000, '%')
```

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.

16.
```
import random

## Create a file containing the states in a random order.
infile = open("StatesAlpha.txt", 'r')
states = [line for line in infile]
infile.close()
random.shuffle(states)
outfile = open("RandomStates.txt", 'w')
outfile.writelines(states)
outfile.close()
```

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.
```

18.
```
import random

plays = ("rock", "paper", "scissors")
p1 = random.choice(plays)  # Player 1
p2 = random.choice(plays)  # Player 1
print("Player 1:", p1)
print("Player 2:", p2)
winner = ""
if ((p1 == "rock") and (p2 == "scissors") or
    (p1 == "paper") and (p2 == "rock") or
    (p1 == "scissors") and (p2 == "paper")):
    print("Player 1 wins.")
elif p1 == p2:
    print("TIE")
else:
    print("Player 2 wins.")
```

```
Player 1: paper
Player 2: scissors
Player 2 wins.
```

```
Player 1: rock
Player 2: rock
TIE
```

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
```

20.
```
import random

def main():
    total = 0
    for trial in range(100000):
        L = [n for n in range(1, 60)]
        numbers = random.sample(L, 5)
        # Can replace above two lines with
        # numbers = random.sample(range(1,60), 5)
        numbers.sort()
        if two_consecutive(numbers):
            total += 1
    sentence = " of the time there were at least \ntwo consecutive numbers" + \
               "in the set \nof five numbers."
    print(("{0:.0%}" + sentence).format(total / 100000))

def two_consecutive(x):
    for index in range(0, 4):
        if x[index] + 1 == x[index + 1]:
            return True

main()
```

```
31% of the time there were at least
two consecutive numbers in the set
of five numbers.
```

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
```

22.
```
import random

## Think of the cards as being numbered 1 through 52.
cards = [n for n in range(1, 53)]
total = 0
for i in range(100000):
    aceLocations = random.sample(cards, 4)
    n = min(aceLocations)
    total += n
print("The average number of cards \nturned up was {0:.2f}".
      format(total / 100000))
```

```
The average number of cards
turned up was 10.61.
```

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
```

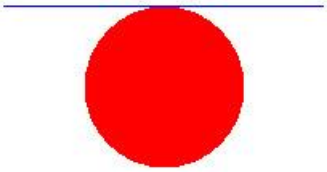
24.
```
import random

numbers_used = []    # will prevent a number from being repeated
numbers_missed = []  # numbers of questions missed
infile = open("StatesANC.txt", 'r')
state_data = [line.rstrip() for line in infile]
infile.close()
# Get five different randomly selected numbers.
for num in range(5):
    num = random.randint(0, 49)
    while num in numbers_used:
        num = random.randint(0, 49)
    numbers_used.append(num)
    item = state_data[num].split(',')
    capital = input("What is the capital of " + item[0] + "? ")
    if capital != item[3]:
        numbers_missed.append(num)
# give score and corrections
print()
total_number_missed = len(numbers_missed)
if total_number_missed == 0:
    print("Congratulations. You answered every question correctly.")
else:
    if total_number_missed == 1:
        print("You missed 1 question.")
    else:
        print("You missed", total_number_missed, "questions.")
    for number in numbers_missed:
        item = state_data[number].split(',')
        print(item[3], "is the capital of", item[0])
```

```
What is the capital of Minnesota? Saint Paul
What is the capital of California? Sacramento
What is the capital of Illinois? Chicago
What is the capital of Alabama? Montgomery
What is the capital of Massachusetts? Boston

You missed 1 question.
Springfield is the capital of Illinois.
```

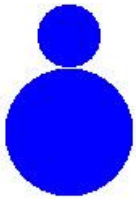
EXERCISES 6.3

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)
```

2.
```
import turtle
t = turtle.Turtle()
t.hideturtle()
t.dot(80, "blue")
t.up()
t.goto(0, 60)
t.dot(40, "blue")
```

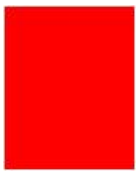


3.
```
import turtle
t = turtle.Turtle()
t.hideturtle()
t.dot(80, "blue")
t.up()
t.goto(0, 60)
t.dot(40, "blue")
```



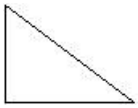
4.
```
import turtle
t = turtle.Turtle()
t.hideturtle()
t.dot(80, "blue")
t.up()
t.goto(0, 60)
t.dot(40, "blue")
```

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()
```

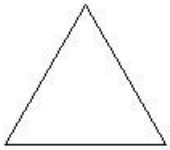


6.
```
import turtle
t = turtle.Turtle()
t.pencolor("red")
t.fillcolor("orange")
t.hideturtle()
t.up()
t.goto(-40, -40)
t.down()
t.begin_fill()
t.goto(40, -40)
t.goto(40, 40)
t.goto(-40, 40)
t.end_fill()
```

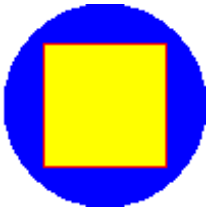
7.
```
import turtle
t = turtle.Turtle()
t.hideturtle()
t.goto(0, 60)
t.goto(80, 0)
t.goto(0, 0)
```



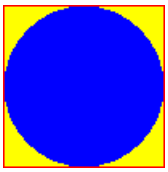
8.
```
import turtle
t = turtle.Turtle()
t.hideturtle()
t.forward(100)
for i in range(2):
    t.left(120)
    t.forward(100)
```



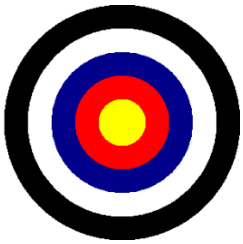
9.
```
import turtle

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()
```



10.
```
import turtle

def main():
    ## Draw a dot inside a square.
    t = turtle.Turtle()
    t.hideturtle()
    drawFilledRectangle(t, 0, 0, 100, 100,"red", "yellow")
    drawDot(t, 50, 50, 100, "blue")

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) having 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()
```



12.
```
import turtle

def main():
    t = turtle.Turtle()
    t.hideturtle()
    drawDot(t, 0, 0, 300, "blue")
    drawDot(t, 0, 0, 200, "white")
    drawDot(t, 0, 0, 100, "blue")

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

main()
```

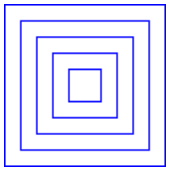
13.
```
import turtle

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()
```

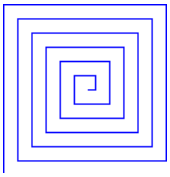
14.
```
    import turtle

    def main():
        t = turtle.Turtle()
        t.hideturtle()
        t.up()
        drawDot(t, 0,0, 200, "red")
        drawDot(t, -120,120, 200, "white")

    def drawDot(t, x, y, diameter, colorP):
        # draw 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()
```



16.
```
    import turtle

    def main():
        t = turtle.Turtle()
        t.speed(10)
        t.hideturtle()
        t.pencolor("blue")
        for i in range(1, 25):
            t.forward(5 * i)
            t.left(90)

    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"))
```

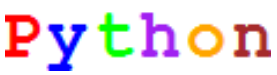
```
def drawFilledRectangle(t, x, y, w, h, colorP="black", colorF="black"):
    ## Draw a filled rectangle with bottom-left corner (x, y),
    ## width w, height h, pen color colorP, and fill color colorF.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x, y)          # Start at 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()
```

18.
```
import turtle

def main():
    t = turtle.Turtle()
    t.hideturtle()
    drawRectangle(t, 0, 0, 200, 40, "black", "black")
    drawRectangle(t, 5, 5, 190, 30, "yellow", "yellow")
    t.up()
    t.goto(100,0)
    t.pencolor("red")
    t.write("PYTHON", align="center", font=("Ariel", 20, "bold"))
```

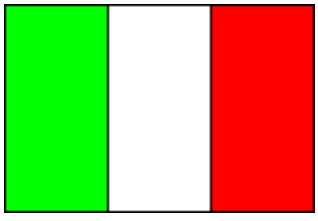
```
def drawRectangle(t, x, y, w, h, colorP="black", colorF="white"):
    # Draw a rectangle with bottom-left corner (x, y),
    # width w, height h, pencolor colorP, and fill color colorF.
    originalPenColor = t.pencolor()
    originalFillColor = t.fillcolor()
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x , y)          # bottom-left corner of rectangle
    t.down()
    t.begin_fill()
    t.goto(x + w, y)       # bottom-right corner of rectangle
    t.goto(x + w, y + h)   # top-right corner of rectangle
    t.goto(x, y + h)       # top-left corner of rectangle
    t.goto(x , y)          # bottom-left corner of rectangle
    t.end_fill()
    t.pencolor(originalPenColor)
    t.fillcolor(originalFillColor)

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()
```

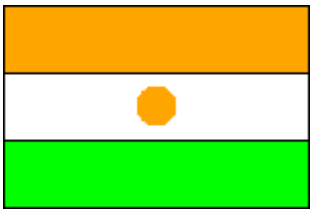
20.
```
import turtle

def main():
    t = turtle.Turtle()
    t.hideturtle()
    t.up()
    colors = ["red", "blue", "green", "purple",
              "orange", "brown", "black", "yellow"]
    word = "Python"
    for i in range(len(word)):
        t.color(colors[i])
        t.goto(20 * i, 0)
        t.write(word[i], font=("Courier New", 18, "bold"))

main()
```



21.
```
import turtle

def main():
    ## Draw flag of Italy.
    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()
```

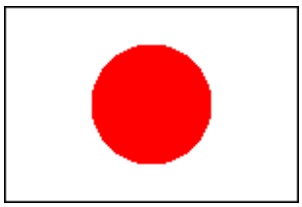


22.
```
import turtle

def main():
    # Draw flag of Niger.
    t = turtle.Turtle()
    t.hideturtle()
    drawRectangle2(t, (0,0), 150, 33, "green")
    drawRectangle2(t, (0,33), 150, 33, "white")
    drawRectangle2(t, (0,66), 150, 33, "orange")
    t.up()
    t.goto(75,50)
    t.color("orange")
    t.dot(20)

def drawRectangle2(t, startPoint, width, height, color):
    t.up()
    t.setheading(0)
    (x, y) = startPoint
    t.fillcolor(color)
    #t.color(color)   # replacement for above line for Swiss flag
    t.begin_fill()
    t.goto(x, y)
    t.down()
    t.forward(width)
    t.left(90)
    t.forward(height)
    t.left(90)
    t.forward(width)
    t.left(90)
    t.forward(height)
    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)
```

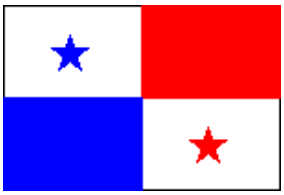
```
def drawFilledRectangle(t, x, y, w, h, colorP="black", colorF="white"):
    # Draw a filled rectangle with bottom-left corner (x, y),
    # width w, height h, pencolor colorP, and fill color colorF.
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()                  # Disable drawing of lines.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

main()
```



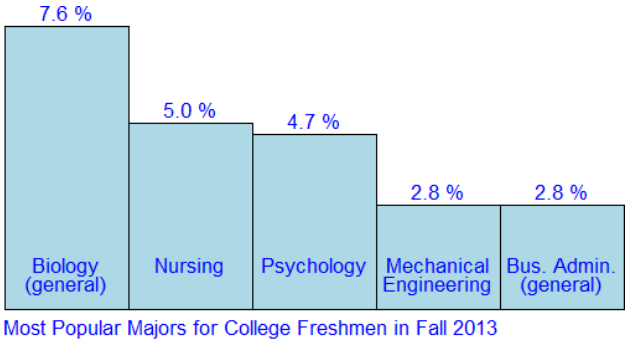
24.
```
import turtle
def main():
    ## Draw flag of Switzerland.
    t = turtle.Turtle()
    t.hideturtle()
    drawRectangle2(t, (0,0), 100, 100, "red")
    drawRectangle2(t, (20,40), 60, 20, "white")
    drawRectangle2(t, (40,20), 20, 60, "white")

def drawRectangle2(t, startPoint, width, height, color):
    t.up()
    t.setheading(0)
    (x, y) = startPoint
    t.color(color)
    t.begin_fill()
    t.goto(x, y)
    t.down()
    t.forward(width)
    t.left(90)
    t.forward(height)
    t.left(90)
    t.forward(width)
    t.left(90)
    t.forward(height)
   t.end_fill()

main()
```

25.
```
import turtle

def main():
    ## Draw 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()
```

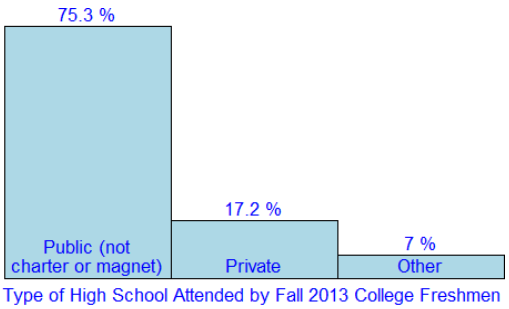
26.
```
import turtle
def main():
    ## Draw flag of Panama.
    t = turtle.Turtle()
    t.hideturtle()
    t.speed(10)
    t.down()
    drawRectangle(t, 0, 0, 150, 100, colorP="black")
    drawRectangle(t, 75, 50, 75, 50, colorP="red", colorF="red")
    drawRectangle(t, 0, 0, 75, 50, colorP="blue", colorF="blue")
    drawFivePointStar(t, 30, 65, 20, "blue", "blue")
    drawFivePointStar(t, 105, 15, 20, "red", "red")
def drawFivePointStar(t, x, y, lenthOfSide, colorP="black", colorF="white"):
    t.up()
    t.goto(x, y)
    t.setheading(0)
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.left(36)
    t.down()
    t.begin_fill()
    for i in range(6):
        t.forward(lenthOfSide)
        t.left(144)
    t.end_fill()
def drawRectangle(t, x, y, w, h, colorP="black", colorF="white"):
    # Draw a rectangle with bottom-left corner (x, y),
    # width w, height h, pencolor colorP, and fill color colorF.
    originalPenColor = t.pencolor()
    originalFillColor = t.fillcolor()
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x , y)          # bottom-left corner of rectangle
    t.down()
    t.begin_fill()
    t.goto(x + w, y)       # bottom-right corner of rectangle
    t.goto(x + w, y + h)   # top-right corner of rectangle
    t.goto(x, y + h)       # top-left corner of rectangle
    t.goto(x , y)          # bottom-left corner of rectangle
    t.end_fill()
    t.pencolor(originalPenColor)
    t.fillcolor(originalFillColor)
def drawDot(t, x, y, diameter, colorP):
    # draw dot with center (x, y) having color colorP
    t.up()
    t.goto(x, y)
    t.dot(diameter, colorP)

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 vertical 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()
```

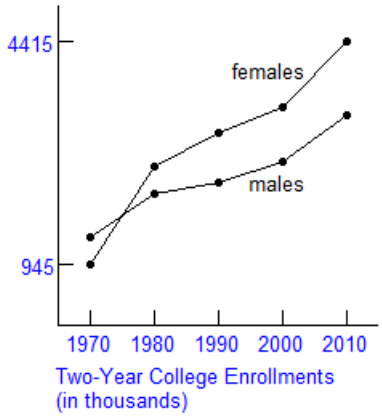


28.
```
import turtle
values = [75.3, 17.2, 7]

def main():
    ## Draw bar chart for type of high school attended.
    t = turtle.Turtle()
    t.speed(10)
    t.hideturtle()
    for i in range(3):
        height = 3 * values[i]
        drawFilledRectangle(t, (-250 + 150 * i), 0, 150, height, "black",
                            "light blue")
    insertText(t )

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

def insertText(t):
    t.up()
    for i in range(3):
        labels1 = ["Public (not", "", ""]
        labels2 = ["charter or magnet)", "Private", "Other"]
        t.pencolor("blue")
        t.goto(-175 + (150 * i), 3 * values[i])
        t.write(str(values[i]) + '%', align="center",
                font=("Ariel", 10, "normal"))
        t.goto(-175 + 150 * i, 18)
        t.write(labels1[i], align="center",font=("Ariel", 10, "normal"))
        t.goto(-175 + (150 * i), 1)
       t.write(labels2[i], align="center", font=("Ariel", 10, "normal"))
    t.goto(-250, -25)
   t.write("Type of High School Attended by Fall 2013 College Freshmen",
           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)
```

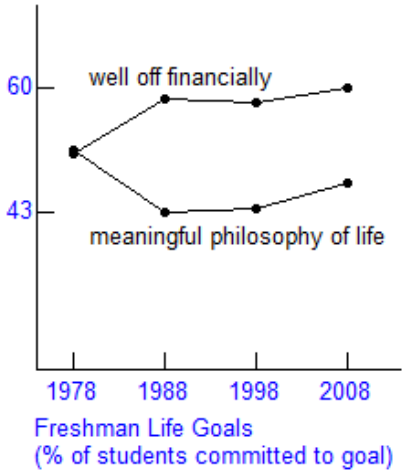
```
def insertText(t):
    t.up()
    t.pencolor("black")
    t.goto(110, 150)
    t.write("Females")
    t.goto(120, 80)
    t.write("Males")
    # Display greatest enrollment value.
    t.color("blue")
    t.goto(-30, (max(FEMALE_ENROLLMENTS)/25)-10)
    t.write(max(FEMALE_ENROLLMENTS))
    # Display 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()
```

30.
```
import turtle
wf = [59, 74, 73, 77]   # well off financially
mp = [60, 43, 44, 51]   # meaningful philosophy of life

def main():
    ## Draw line chart for Freshmen life goals.
    t = turtle.Turtle()
    t.hideturtle()
    drawLine(t, 0, 0, 200, 0)    # x-axis
    drawLine(t, 0, 0, 0, 200)    # y-axis
    for i in range(3):
        drawLineWithDots(t, 20 + (50 * i), 2 * wf[i], 70 + 50 * i,
                         2 * wf[i+1], "black")
    for i in range(3):
        drawLineWithDots(t, 20 + (50 * i), 2 * mp[i], 70 + 50 * i,
                         2 * mp[i+1], "black")
    drawTickMarks(t)
    insertText(t)

def drawLine(t, x1, y1, x2, y2, colorP="black"):
    t.up()
    t.goto(x1,y1)
    t.down()
    t.color(colorP)
    t.goto(x2,y2)
```

```
def drawLineWithDots(t, x1, y1, x2, y2, colorP="black"):
    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(4):
        drawLine(t, 20 + (50 * i), 0, 20 + 50 * i , 10)
    drawLine(t, 0, 2 * max(wf), 10, 2 * max(wf))
    drawLine(t, 0, 2 * min(mp), 10, 2 * min(mp))
def insertText(t):
    t.up()
    t.pencolor("black")
    t.goto(30, 152)
    t.write("Well off financially")
    t.pencolor("black")
    t.goto(30, 65)
    t.write("Meaningful philosophy of life")
    # Display greatest enrollment value.
    t.color("blue")
    t.goto(-15, 2 * max(wf) - 7)
    t.write(max(mp))
    # Display least enrollment value.
    t.goto(-15, 2 * min(mp) - 7)
    t.write(min(mp))
    # Display labels for tick marks on x-axis.
    t.goto(0, -20)
    x = 20
    for i in range(1978, 2009, 10):
        t.goto(x, -20)
        t.write(str(i), align="center")
        x += 50
    t.goto(0, -40)
    t.write("Freshman Life Goals")
    t.goto(0, -55)
    t.write("(% of students committed to goal)")
main()
```



EXERCISES 6.4

1. 15 2. 54321 3. ****** 4. 20 5. harpo

6. The function finds the smallest number (call it *s*) that divides *n* and then repeats the process with *n* / *s* until the quotient is 1.

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:])
```

8.
```
def displaySequenceOfNumbers2(m, n):
    ## Display the numbers from m to n, where m <= n.
    if m <= n:
        print(m)
        displaySequenceOfNumbers2(m + 1, n)
```

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
```

10.
```
def main():
    n = int(input("Enter a positive integer: "))
    print("Fibonacci number:", fib(n))

def fib(n):
    if n <= 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

main()
```

```
Enter a positive integer: 7
Fibonacci number: 13
```

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
```

12.
```
def main():
    ## Calculate the balance owed on a mortgage.
    p = float(input("Enter the principal: "))
    r = float(input("Enter the annual rate of interest: "))
    pmt = float(input("Enter the monthly payment: "))
    n = int(input("Enter the number of monthly payments made: "))
    print("The amount still owed is ${0:,.2f}.".format(balance(p,
          pmt, r, n)))

def balance(p, pmt, r, n):
    if n == 0:
        return p
    else:
        return (1 + r/1200) * balance(p, pmt, r, n - 1) - pmt

main()
```

```
Enter the principal: 204700
Enter the annual rate of interest: 4.8
Enter the monthly payment: 1073.99
Enter the number of monthly payments made: 300
The amount still owed is $57,188.74.
```

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
```

PROGRAMMING PROJECTS CHAPTER 6

1.
```
import random

numberOfTries = 1
n = random.randint(1, 100)
print("\nI've thought of a number from 1 through 100.")
while True:
    try:
        guess = int(input("Guess the number: "))
        break
    except ValueError:
        numberOfTries += 1
        print("You did not enter a number.")
while guess != n:
    numberOfTries += 1
    if (guess > 100) or (guess < 1):
        print("Number must be from 1 through 100.")
    elif guess > n:
        print("Too high")
    elif guess < n:
        print("Too low")
    while True:
        try:
            guess = int(input("Try again: "))
            break
        except ValueError:
            numberOfTries += 1
            print("You did not enter a number.")
print("Correct.", end=" ")
if numberOfTries == 1:
    print("You got it in one guess.")
else:
    print("You took", numberOfTries, "guesses.")
```

```
I've thought of a number from 1 through 100.
Guess the number: 50
Too low
Try again: 123
Number must be from 1 through 100.
Try again: sixty
You did not enter a number.
Try again: 60
Too high
Try again: 56
Correct. You took 5 guesses.
```

2.
```
import pickle
import random

def main():
    pokerHand = getHandOfCards(5)
    pokerHand.sort()
    displayPokerHand(pokerHand)
    analyzePokerHand(pokerHand)

def getHandOfCards(numberOfCards):
    deckOfCards = pickle.load(open("deckOfCardsList.dat", 'rb'))
    return random.sample(deckOfCards, 5)

def displayPokerHand(pokerHand):
    print(", ".join(pokerHand))

def analyzePokerHand(pH):
    ranks = {x[:-1] for x in pH}
    numberOfRanks = len(ranks)
    if numberOfRanks == 5:
        print("ranks-all-different")
    elif numberOfRanks == 4:
        print("one pair")
    elif numberOfRanks == 3:
        foundThree = False
        for i in range(2):
            if ((pH[i][0] == pH[i + 1][0]) and
                (pH[i + 1][0] == pH[i + 2][0])):
                print("three of a kind")
                foundThree = True
                break
        if foundThree == False:
            print("two pairs")
    else:  # two different ranks
        if ((pH[0][0] == pH[1][0]) and (pH[1][0] == pH[2][0]) \
             and (pH[2][0] == pH[3][0])) \
             or ((pH[1][0] == pH[2][0]) and (pH[2][0] == pH[3][0]) \
             and (pH[3][0] == pH[4][0])):
            print("four of a kind")
        else:
            print("full house")

main()
```

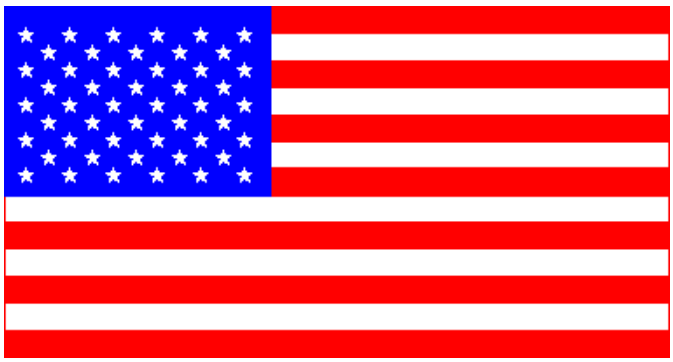
```
K♥, K♦, 2♦, K♣, 5♠
three-of-a-kind
```

3.
```
import pickle
import random

def main():
    ## Analyze a bridge hand.
    bridgeHand = getHandOfCards(13)
    displayBridgeHand(bridgeHand)
    analyzeBridgeHand(bridgeHand)

def getHandOfCards(numberOfCards):
    deckOfCards = pickle.load(open("deckOfCardsList.dat", 'rb'))
    return random.sample(deckOfCards, numberOfCards)

def displayBridgeHand(bridgeHand):
    print(", ".join(bridgeHand))

def analyzeBridgeHand(bridgeHand):
   suits = {x[-1] for x in bridgeHand}
   d = {suit:0 for suit in suits}  # distribution of cards into suits
   for card in bridgeHand:
       d[card[-1]] += 1
   t = tuple(d.items())
   tSorted = sorted(t)
   tSorted = sorted(t, key=lambda x: x[1], reverse=True)
   for k in tSorted:
       print("Number of", k[0], "is", k[1])

main()
```

```
10♥, 3♥, J♣, 2♣, 10♦, K♣, 2♥, 6♦, 6♣, 4♣, 7♦, 6♠, 4♦
Number of ♣ is 5
Number of ♦ is 4
Number of ♥ is 3
Number of ♠ is 1
```

4.
```
import turtle

def main():
    ## Draw an American flag.
    t = turtle.Turtle()
    t.hideturtle()
    t.speed(10)
    drawRectangle(t, -200, 0, 380, 200, "red", "red")
    for i in range(1, 13, 2):
        drawRectangle(t, -200, (200/13)*i , 380, (200/13), "red", "white")
    drawRectangle(t, -200, (200/13)*6, (2/5)*380, (200/13)*7, "blue", "blue")
    for i in range(5):
        y = 180 - (20 * i)
        for j in range(6):
            x = -190 + 25*j
            drawFivePointStar(t, x, y, 8, "white")
    for i in range(4):
        y = 170 - (20 * i)
        for j in range(5):
            x = -177 + 25*j
            drawFivePointStar(t, x, y, 8, "white")

def drawFivePointStar(t, x, y, lenthOfSide, colorP="black", colorF="white"):
    t.up()
    t.goto(x, y)
    t.setheading(0)
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.left(36)
    t.down()
    t.begin_fill()
    for i in range(6):
        t.forward(lenthOfSide)
        t.left(144)
    t.end_fill()
```

```
def drawRectangle(t, x, y, w, h, colorP="black", colorF="white"):
    # Draw a rectangle with bottom-left corner (x, y),
    # width w, height h, pencolor colorP, and fill color colorF.
    originalPenColor = t.pencolor()
    originalFillColor = t.fillcolor()
    t.pencolor(colorP)
    t.fillcolor(colorF)
    t.up()
    t.goto(x , y)          # bottom-left corner of rectangle
    t.down()
    t.begin_fill()
    t.goto(x + w, y)       # bottom-right corner of rectangle
    t.goto(x + w, y + h)   # top-right corner of rectangle
    t.goto(x, y + h)       # top-left corner of rectangle
    t.goto(x , y)          # bottom-left corner of rectangle
    t.end_fill()
    t.pencolor(originalPenColor)
    t.fillcolor(originalFillColor)

main()
```



5.
```
def main():
    ## Determine all the permutations of a word.
    w = input("Enter a word: ")
    print("  ".join(permutations(w)))

def permutations(w):
    # Construct a list consisting of all permutations of the string s
    if len(w) == 1:
        return w
    listOfPermuations = []         # resulting list
    for i in range(len(w)):
        restOfw = w[:i] + w[i+1:]  # w with ith character removed
        z = permutations(restOfw)  # permutations of remaining characters
        for t in z:
            listOfPermuations.append(w[i] + t)
    return listOfPermuations

main()
```

```
Enter a word: ear
ear  era  aer  are  rea  rae
```

6.
```
def main():
    ## Display line of Pascal's Triangle.
    n = int(input("Enter a nonnegative integer: "))
    line = [str(x) for x in pascal(n)]
    print("Row", str(n) + ':' ,"  ".join(line))

def pascal(n):
    # Display the nth line of Pascal's triangle.
    if n == 0:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])
        line += [1]
    return line

main()
```

```
Enter a nonnegative integer: 6
Row 6: 1  6  15  20  15  6  1
```

CHAPTER 7

EXERCISES 7.1

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

2. There should be a colon at the end of the line beginning with **def**.

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.

4. Second line must be written as **def _ _init_ _(self, altitude, base=1):**. Parameters that are passed to by position must precede those with default values.

5. **1** 6. **3.14** 7. **4** 8. **5**

9. **12.56** 10. **18.84** 11. **18.84** 12. **12.56**

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
```

14.
```
def main():
    ## Determine the distance between two points.
    x1 = float(input("Enter x-coordinate of first point: "))
    y1 = float(input("Enter y-coordinate of first point: "))
    x2 = float(input("Enter x-coordinate of second point: "))
    y2 = float(input("Enter y-coordinate of second point: "))
    p = Point(x1 - x2, y1 - y2)
    print("Distance between points: {0:,.2f}".format(p.distanceFromOrigin()))

class Point:
    def __init__(self, x, y):
        self._x = x
        self._y = y

    def distanceFromOrigin(self):
        return (self._x ** 2 + self._y ** 2) ** .5

main()
```

```
Enter x-coordinate of first point: 2
Enter y-coordinate of first point: 3
Enter x-coordinate of second point: 7
Enter y-coordinate of second point: 15
Distance between points: 13.00
```

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
```

16.
```
import pairOfDice

def main():
    ## Play a game of dice.
    dice1 = pairOfDice.PairOfDice()
    dice1.roll()
    print("Player 1:", dice1.sum())
    dice2 = pairOfDice.PairOfDice()
    dice2.roll()
    print("Player 2:", dice2.sum())
    if dice1.sum() == dice2.sum():
        print("TIE")
    elif dice1.sum() > dice2.sum():
        print("Player 1 wins.")
    else:
        print("Player 2 wins.")

main()
```

```
Player 1: 8
Player 2: 6
Player 1 wins.
```

```
Player 1: 7
Player 2: 7
TIE
```

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()
```

18.
```
import pairOfDice

def main():
    ## Estimate the Chevalier de Méré probability.
    numberOfSuccesses = 0
    for i in range(10000):
        if playGame():
            numberOfSuccesses += 1
    print(numberOfSuccesses / 10000)
```

```
    def playGame():
        doubleSixes = False
        dice = pairOfDice.PairOfDice()
        for i in range(24):
            dice.roll()
            if dice.sum() == 12:
                doubleSixes = True
        return doubleSixes

    main()
```

19. queen of hearts 20. queen 21. 10 of clubs

22. diamonds 23. 7 of hearts 24. 5 of clubs

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()
```

26.
```
    import pCard

    def main():
        c = pCard.PlayingCard()
        c.selectAtRandom()
        c.setSuit("diamonds")
        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)
```

28.
```
import fraction

def main():
    ## Reduce a specified fraction to lowest terms.
    f = fraction.Fraction()
    numerator = int(input("Enter numerator of fraction: "))
    f.setNumerator(numerator)
    denominator = int(input("Enter denominator of fraction: "))
    f.setDenominator(denominator)
    f.reduce()
    msg = "Reduction to lowest terms:"
    if f.getDenominator() != 1:
        print(msg, str(f.getNumerator()) + '/' + str(f.getDenominator()))
    else:
        print(msg, f.getNumerator())

main()
```

```
Enter numerator of fraction: 12
Enter denominator of fraction: 30
Reduction to lowest terms: 2/5
```

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
```

30.
```
import fraction

def main():
    ## Add two fractions.
    f1 = fraction.Fraction()
    numerator = int(input("Enter numerator of first fraction: "))
    f1.setNumerator(numerator)
    denominator = int(input("Enter denominator of first fraction: "))
    f1.setDenominator(denominator)
    f2 = fraction.Fraction()
    numerator = int(input("Enter numerator of second fraction: "))
    f2.setNumerator(numerator)
    denominator = int(input("Enter denominator of second fraction: "))
    f2.setDenominator(denominator)
    print("Sum:", calculateSum(f1, f2))

def calculateSum(f1, f2):
    # Note: a/b + c/d = (ad + bc)/bd
    sum = fraction.Fraction()
    sum.setDenominator(f1.getDenominator() * f2.getDenominator())
    sum.setNumerator((f1.getNumerator() * f2.getDenominator()) +
                     (f1.getDenominator() * f2.getNumerator()))
    sum.reduce()
    if sum.getDenominator() != 1:
        return  str(sum.getNumerator()) + '/' + str(sum.getDenominator())
    else:
        return sum.getNumerator()

main()
```

```
Enter numerator of first fraction: 1
Enter denominator of first fraction: 6
Enter numerator of second fraction: 3
Enter denominator of second fraction: 4
Sum: 11/12
```

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
```

32.
```
def main():
    ## Calculate an average.
    listOfGrades = []
    for i in range(6):
        quizGrade = float(input("Enter grade on quiz " + \
                                str(i + 1) + ": "))
        listOfGrades.append(quizGrade)
    q = Quizzes(listOfGrades)
    print(q)

class Quizzes:
    def __init__(self, listOfGrades):
        self._quizGrades = listOfGrades

    def average(self):
        self._quizGrades.sort()
        self._quizGrades = self._quizGrades[1:]  # Drop lowest quiz grade.
        return sum(self._quizGrades) / 5

    def __str__(self):
        return "Quiz average: " + str(self.average())

main()
```

```
Enter grade on quiz 1: 9
Enter grade on quiz 2: 10
Enter grade on quiz 3: 5
Enter grade on quiz 4: 8
Enter grade on quiz 5: 10
Enter grade on quiz 6: 10
Quiz average: 9.4
```

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
```

34.
```
import random
import pCard

def main():
    ## Display a bridge hand.
    deckOfCards = []
    ranks = ['2', '3', '4', '5', '6', '7', '8', '9',
             "10", "jack", "queen", "king", "ace"]
    suits = ["spades", "hearts","diamonds", "clubs"]
    for i in ranks:
        for j in suits:
            c = pCard.PlayingCard(i, j)
            deckOfCards.append(c)
    bridgeHand = random.sample(deckOfCards, 13)
    bridgeHand.sort(key=lambda x: x.getSuit(), reverse=True)
    for k in bridgeHand:
        print(k)

main()
```

```
4 of spades
7 of spades
     :
3 of hearts
queen of diamonds
jack of clubs
8 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
```

36.
```
def main():
    ## Simulate a toll booth cash register.
    device = Register()
    carryOn = 'Y'
    while carryOn.upper() == 'Y':
        vehicle = input("Enter type of vehicle (car/truck): ")
        if vehicle == "car":
            device.ProcessCar()
        else:
            device.ProcessTruck()
        print("Number of vehicles:", device.getCount())
        print("Money Collected: ${0:,.2f}".format(device.getTally()))
        carryOn = input("Do you want to enter more vehicles (Y/N)? ")
    print("Have a good day.")

class Register:
    def __init__(self, count=0, tally=0):
        self._count = count
        self._tally = tally

    def setCount(self, count):
        self._count = count

    def setTally(self, tally):
        self._tally = tally

    def getCount(self):
        return self._count

    def getTally(self):
        return self._tally

    def ProcessCar(self):
        self._count += 1
```

```
            self._tally += 1  # Cost is $1 per car.

        def ProcessTruck(self):
            self._count += 1
            self._tally += 2  # Cost is $2 per truck.

main()
```

```
Enter type of vehicle (car/truck): car
Number of vehicles: 1
Money Collected: $1.00
Do you want to enter more vehicles (Y/N)? Y
Enter type of vehicle (car/truck): truck
Number of vehicles: 2
Money Collected: $3.00
Do you want to enter more vehicles (Y/N)? N
Have a good day.
```

EXERCISES 7.2

1. 4 2. 0.433 3. 6.928 4. 2.999824

5. The rectangle has area 6.00. 6. The rectangle has area 30.00.

7. Howdy
   G'day mate

8. I have a backbone.
   I have jointed limbs and no backbone.

9. Change the 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())
```

10.
```
import student
def main():
    listOfStudents = obtainListOfStudents()  # students and grades
    displayResults(listOfStudents)

def obtainListOfStudents():
    listOfStudents = []
    carryOn = 'Y'
    while carryOn == 'Y':
        name = input("Enter student's name: ")
        midterm = float(input("Enter student's grade on midterm exam: "))
        final = float(input("Enter student's grade on final exam: "))
        category = input("Enter category (LG or PF): ")
        if category.upper() == "LG":
            st = student.LGstudent(name, midterm, final)
        else:
            st = student.PFstudent(name, midterm, final)
        listOfStudents.append(st)
        carryOn = input("Do you want to continue (Y/N)? ")
        carryOn = carryOn.upper()
    return listOfStudents
```

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

    main()
```

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
```

12.
```
def main():
    ## Calculate semester grades.
    listOfStudents = obtainListOfStudents()  # students and grades
    displayResults(listOfStudents)

def obtainListOfStudents():
    listOfStudents = []
    carryOn = 'Y'
    while carryOn == 'Y':
        name = input("Enter student's name: ")
        midterm = float(input("Enter student's grade on midterm exam: "))
        final = float(input("Enter student's grade on final exam: "))
        category = input("Enter category (LG or PF): ")
        if category.upper() == "LG":
            st = LGstudent(name, midterm, final)
        else:
            status = input("Are you a full-time student (Y/N)? ")
            if status.upper() == 'Y':
                fullTime = True
            else:
                fullTime = False
            st = PFstudent(name, midterm, final, fullTime)
        listOfStudents.append(st)
        carryOn = input("Do you want to continue (Y/N)? ")
        carryOn = carryOn.upper()
    return listOfStudents

def displayResults(listOfStudents):
    print("\nNAME\tGRADE\tSTATUS")
    listOfStudents.sort(key = lambda x: x.getName())
    for pupil in listOfStudents:
        print(pupil)

class Student:
    def __init__(self, name="", midterm=0, final=0):
        self._name = name
        self._midterm = midterm
        self._final = final
        self._semesterGrade = ""

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

    def setMidterm(self, midterm):
        self._midterm = midterm

    def setFinal(self, final):
        self._final = final

    def getName(self):
        return self._name

    def __str__(self):
        return self._name + "\t" + self.calcSemGrade()
```

```
class LGstudent(Student):

    def calcSemGrade(self):
        average = round((self._midterm + self._final) / 2)
        if average >= 90:
            return "A"
        elif average >= 80:
            return "B"
        elif average >= 70:
            return "C"
        elif average >= 60:
            return "D"
        else:
            return "F"

    def __str__(self):
        return (self._name + "\t" + self.calcSemGrade() +
                "\tFull-time student")

class PFstudent(Student):
    def __init__(self, name="", midterm=0, final=0, fullTime=True):
        super().__init__(name, midterm, final)
        self._fullTime = fullTime

    def setFullTime(self, fullTime):
        self._fullTime = fullTime

    def getFullTime(self):
        return self._fullTime

    def calcSemGrade(self):
        average = round((self._midterm + self._final) / 2)
        if average >= 60:
            return "Pass"
        else:
            return "Fail"

    def __str__(self):
        if self._fullTime:
            status = "Full-time student"
        else:
            status = "Part-time student"
        return (self._name + "\t" + self.calcSemGrade()
                + "\t" + status)

main()
```

```
Enter student's name: Bob
Enter student's grade on midterm exam: 79
Enter student's grade on final exam: 85
Enter category (LG or PF): LG
Do you want to continue (Y/N)? Y
Enter student's name: Alice
Enter student's grade on midterm exam: 92
Enter student's grade on final exam: 96
Enter category (LG or PF): PF
Are you a full-time student (Y/N)? N
Do you want to continue (Y/N)? N

NAME    GRADE   STATUS
Alice   Pass    Part-time student
Bob     B       Full-time student
```

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)
```

14.
```
import mortgage

def main():
    ## Calculate the monthly payment for a mortgage.
    principal = float(input("Enter amount of mortgage: "))
    interestRate = float(input("Enter percent interest rate: "))
    term = float(input("Enter duration of mortgage in years: "))
    mort = mortgage.Mortgage(principal, interestRate, term)
    print("Monthly payment: ${0:,.2f}".format(mort.calculateMonthlyPayment()))
```

```
Enter principal of mortgage: 350000
Enter percent interest rate: 5.25
Enter duration of mortgage in years: 30
Monthly payment: $1,932.71
```

15.
```
def main():
    ## Calculate 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
```

16.

```
import mortgage

def main():
    ## Calculate values for a mortgage with points.
    principal = float(input("Enter principal amount of mortgage: "))
    interestRate = float(input("Enter percent interest rate: "))
    term = float(input("Enter duration of mortgage in years: "))
    numberOfPoints = float(input("Enter number of discount points: "))
    mort = MortgageWithPoints(principal, interestRate,
                              term, numberOfPoints)
    print("Cost of discount points: ${0:,.2f}".\
                    format(mort.calculateCostOfPoints()))
    print("Monthly payment: ${0:,.2f}".\
                    format(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 MortgageWithPoints(Mortgage):
    def __init__(self, principal, interestRate,
                 term, numberOfPoints):
        super().__init__(principal, interestRate, term)
        self._numberOfPoints = numberOfPoints

    def calculateCostOfPoints(self):
        return self._numberOfPoints * (self._principal / 100)

main()
```

```
Enter principal of mortgage: 350000
Enter percent interest rate: 5
Enter duration of mortgage in years: 30
Enter number of discount points: 2
Cost of discount points: $7,000.00
Monthly payment: $1,878.88
```

PROGRAMMING PROJECTS CHAPTER 7

1(a).
```
import pickle

def main():
    createDictionayOfNations()

def createDictionayOfNations():
    nationDict = {}
    for line in open("UN.txt", 'r'):
        data = line.split(',')
        country = Nation()
        country.setName(data[0])
        country.setContinent(data[1])
        country.setPopulation(float(data[2]))
        country.setArea(float(data[3].rstrip()))
        nationDict[country.getName()] = country
    # Save list as binary file.
    pickle.dump(nationDict, open("nationDict.dat", 'wb'))
    return nationDict

class Nation:
    def __init__(self):
        self_name = ""
        self_continent = ""
        self_population = 0.0
        self_area = 0

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

    def getName(self):
        return self._name

    def setContinent(self, continent):
        self._continent = continent

    def getContinent(self):
        return self._continent

    def setPopulation(self, population):
        self._population = population

    def getPopulation(self):
        return self._population

    def setArea(self, area):
        self._area = area
```

```
    def getArea(self):
        return self._area

    def popDensity(self):
        return self._population / self._area

main()
```

1(b).
```
import pickle
from nation import Nation

def main():
    ## Display information about a country.
    nationDict = pickle.load(open("nationDict.dat", "rb"))
    country = input("Enter a country: ")
    print("Continent:", nationDict[country].getContinent())
    print("Population: {0:,.0f}".
          format(1000000 * nationDict[country].getPopulation()))
    print("Area: {0:,.2f} square miles".
          format(nationDict[country].getArea()))

main()
```

```
Enter a country: Canada
Continent: North America
Population: 34,800,000
Area: 3,855,000.00 square miles
```

1(c).
```
import pickle
from nation import Nation

def main():
    ## Display the most density populated countries on a continent.
    nationDict = pickle.load(open("nationDict.dat", "rb"))
    nationList = list(nationDict.keys())
    continent = input("Enter a continent: ")
    nationsInContinent = [nation for nation in nationList if
             nationDict[nation].getContinent() == continent]
    nationsInContinent.sort(key=lambda x: nationDict[x].popDensity(),
                            reverse=True)
    for i in range(5):
        print(nationsInContinent[i])

main()
```

```
Enter a continent: South America
  Ecuador
  Colombia
  Venezuela
  Peru
  Brazil
```

2.
```
def main():
    acct = SavingsAccount()
    name = input("Enter person's name: ")
    acct.setName(name)
    print("D = Deposit, W = Withdrawal, Q = Quit")
    request = input("Enter D, W, or Q: ").upper()
    while True:
        if request == 'D':
            amount = float(input("Enter amount to deposit: "))
            acct.makeDeposit(amount)
            print("Balance: ${0:,.2f}".format(acct.getBalance()))
            request = input("Enter D, W, or Q: ").upper()
        elif request == 'W':
            amount = float(input("Enter amount to withdraw: "))
            acct.makeWithdrawal(amount)
            print("Balance: ${0:,.2f}".format(acct.getBalance()))
            request = input("Enter D, W, or Q: ").upper()
        elif request == 'Q':
            print("End of transactions. Have a good day",
                  acct.getName() + '.')
            break
        else:
            request = input("Enter D, W, or Q: ").upper()

class SavingsAccount:
    def __init__(self, name="", balance=0.0):
        self._name = name
        self._balance = balance
    def setName(self, name):
        self._name = name
    def getName(self):
        return self._name
    def setBalance(self, balance):
        self._balance = balance
    def getBalance(self):
        return self._balance
    def makeDeposit(self, amount):
        self._balance += amount
    def makeWithdrawal(self, amount):
        if amount <= self._balance:
            self._balance -= amount
        else:
            print("Insufficient funds, transaction denied.")
```

```
Enter person's name: Fred
D = Deposit, W = Withdrawal, Q = Quit
Enter D, W, or Q: D
Enter amount to deposit: 1000
Balance: $1,000.00
Enter D, W, or Q: W
Enter amount to withdraw: 4000
Insufficient funds, transaction denied.
Balance: $1,000.00
Enter D, W, or Q: W
Enter amount to withdraw: 400
Balance: $600.00
Enter D, W, or Q: Q
End of transactions. Have a good day Fred.
```

3.
```
def main():
    ## Create a payroll summary.
    listOfEmployees = createListOfEmployees()
    displayResults(listOfEmployees)

def createListOfEmployees():
    listOfEmployees = []
    carryOn = 'Y'
    while carryOn == 'Y':
        name = input("Enter employee's name: ")
        prompt = "Enter employee's classification (Salaried or Hourly): "
        classification = input(prompt)
        hours = float(input("Enter the number of hours worked: "))
        if classification.upper() == "SALARIED":
            rate = float(input("Enter weekly salary: "))
            person = SalariedEmployee(name, rate, hours)
        else:
            rate = float(input("Enter hourly wage: "))
            person = HourlyEmployee(name, rate, hours)
        listOfEmployees.append(person)
        carryOn = input("Do you want to continue (Y/N)? ")
        carryOn = carryOn.upper()
    return listOfEmployees

def displayResults(listOfEmployees):
    print()
    numberOfSalariedEmployees = 0
    totalPayroll = 0.0
    totalHoursWorked = 0.0
    for person in listOfEmployees:
        print("{0:s}: ${1:,.2f}".format(person.getName(),
                                        person.calculatePay()))
    for person in listOfEmployees:
        totalHoursWorked += person.getHoursWorked()
        if isinstance(person, SalariedEmployee):
            numberOfSalariedEmployees += 1
        totalPayroll += person.calculatePay()
    print("Number of employees:", len(listOfEmployees))
    print("Number of salaried employees:", numberOfSalariedEmployees)
    print("Total payroll: ${0:,.2f}".format(totalPayroll))
    average = "Average number of hours worked per employee: {0:.2f}"
    print(average.format(totalHoursWorked / len(listOfEmployees)))

class Employee:
    def __init__(self, name="", rate=0.0, hoursWorked=0.0):
        self._name = name
        self._rate = rate
        self._hoursWorked = hoursWorked

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

    def getName(self):
        return self._name

    def setRate(self, rate):
        self._rate = rate
```

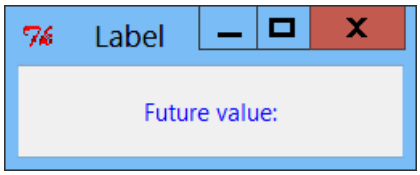
```
    def getRate(self):
        return self._rate

    def setHoursWorked(self, hoursWorked):
        self._hoursWorked = hoursWorked

    def getHoursWorked(self):
        return self._hoursWorked

class SalariedEmployee(Employee):

    def calculatePay(self):
        return self._rate

class HourlyEmployee(Employee):

    def calculatePay(self):
        return self._hoursWorked * self._rate

main()
```

```
Enter employee's name: Jane
Enter employee's classification (Salaried or Hourly): Salaried
Enter the number of hours worked: 40
Enter weekly salary: 1000
Do you want to continue (Y/N)? Y
Enter employee's name: Fred
Enter employee's classification (Salaried or Hourly): Hourly
Enter the number of hours worked: 10
Enter hourly wage: 25
Do you want to continue (Y/N)? N

Jane: $1,000.00
Fred: $250.00
Number of employees: 2
Number of salaried employees: 1
Total payroll: $1,250.00
Average number of hours worked per employee: 25.00
```

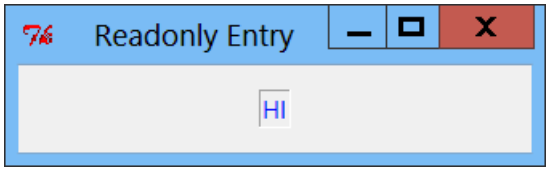
CHAPTER 8

EXERCISES 8.1

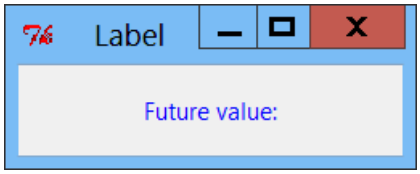
1.
```
from tkinter import *
window = Tk()
window.title("Label")
lblFV = Label(window, text="Future value:", fg="blue")
lblFV.grid(padx=75, pady=15)
window.mainloop()
```



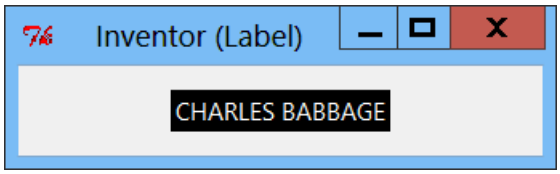
2.
```
from tkinter import *
window = Tk()
window.title("Readonly Entry")
conOFentQuote = StringVar()   # contents of the Entry widget
entQuote = Entry(window, fg="blue", state="readonly",
                 width=2, textvariable=conOFentQuote)
entQuote.grid(padx=150, pady=15)
conOFentQuote.set("HI")
window.mainloop()
```



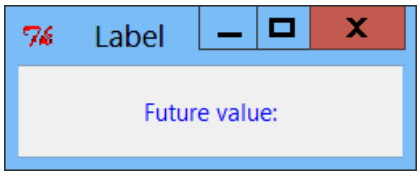
3.
```
from tkinter import *
window = Tk()
window.title("Quotation")
conOFentQuote = StringVar()   # contents of the Entry widget
entQuote = Entry(window, fg="blue", textvariable=conOFentQuote)
entQuote.grid(padx=40, pady=15)
conOFentQuote.set("Less is More")
window.mainloop()
```



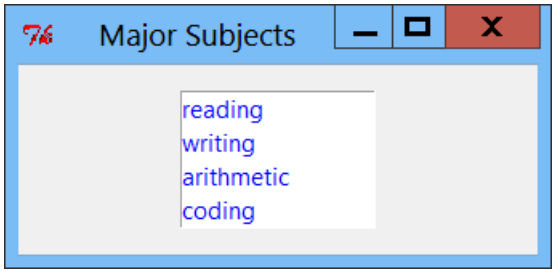
4.
```
from tkinter import *
window = Tk()
window.title("Inventor (Label)")
conOFlblInventor = StringVar()   # contents of the Entry widget
lblInventor = Label(window, fg="white", bg="blue",
                    textvariable=conOFlblInventor)
lblInventor.grid(padx=95, pady=15)
conOFlblInventor.set("CHARLES BABBAGE")
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()
```

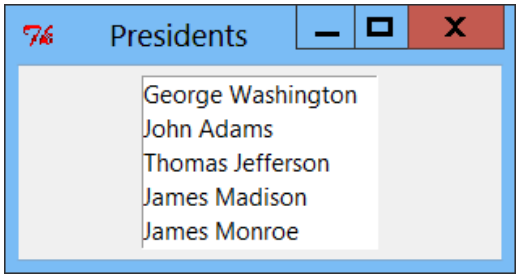


6.
```
from tkinter import *
window = Tk()
window.title("Major Subjects")
L = ["reading", "writing", "arithmetic", "coding"]
conOFlstSubjects = StringVar()
lstSubjects = Listbox(window, width=15, height=4, fg="blue",
                      listvariable=conOFlstSubjects)
lstSubjects.grid(padx=100, pady=15)
conOFlstSubjects.set(tuple(L))
window.mainloop()
```

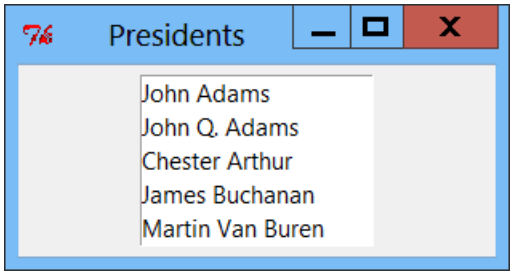


8. ipadx and ipady pad the horizontal and vertical space inside the button's borders.

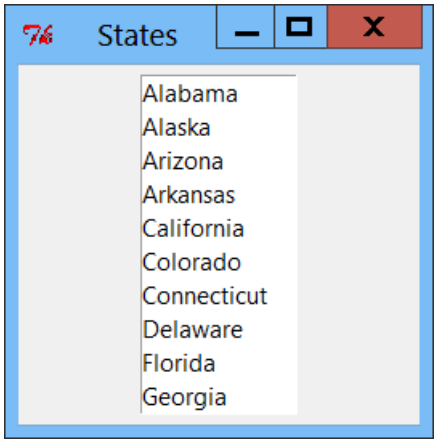
9.
```
from tkinter import *
window = Tk()
window.title("Presidents")
infile = open("USPres.txt", 'r')
listOfPresidents = [line.rstrip() for line in infile]
infile.close()
conOFlstPres = StringVar()
lstPres = Listbox(window, height=5, width=18,
                  listvariable=conOFlstPres)
lstPres.grid(padx=75,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



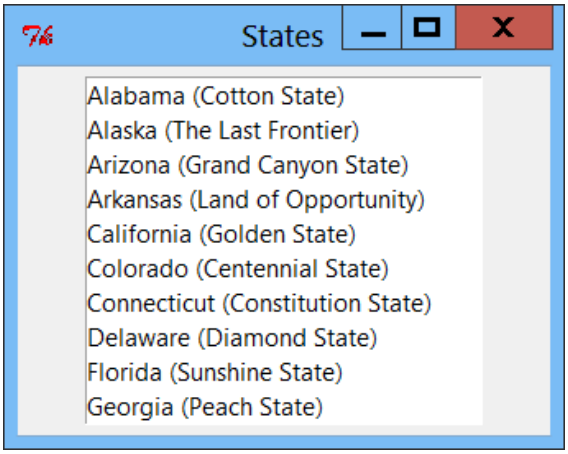
10.
```
from tkinter import *
window = Tk()
window.title("Presidents")
infile = open("USpres.txt", 'r')
listOfPresidents = [line for line in infile]
infile.close()
listOfPresidents.sort(key=lambda x: x.split(" ")[-1])
conOFlstPres = StringVar()
lstPres = Listbox(window, height=5, width=18,
listvariable=conOFlstPres)
lstPres.grid(padx=75, pady=5)
conOFlstPres.set(tuple(listOfPresidents))
window.mainloop()
```



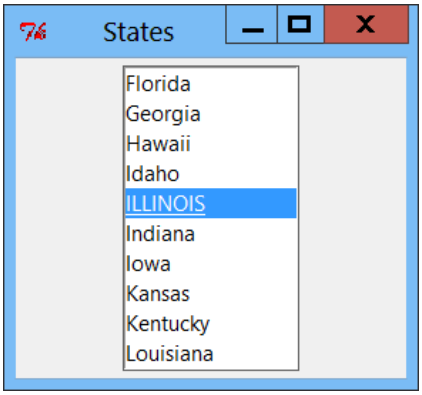
11.
```
from tkinter import *
window = Tk()
window.title("States")
infile = open("StatesANC.txt", 'r')
listOfStates = [line.split(',')[0] for line in infile]
infile.close()
conOFlstStates = StringVar()
lstStates = Listbox(window, height=10, width=12,
                    listvariable=conOFlstStates)
lstStates.grid(padx=75,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



12.
```
from tkinter import *
window = Tk()
window.title("States")
infile = open("StatesANC.txt", 'r')
listOfStates = [line.split(',')[0] + " (" + line.split(',')[2] + ")"\
                for line in infile]
infile.close()
conOFlstStates = StringVar()
lstStates = Listbox(window, height=10,
                    width=30, listvariable=conOFlstStates)
lstStates.grid(row=1, column=0, padx=40, pady=5)
conOFlstStates.set(tuple(listOfStates))
window.mainloop()
```

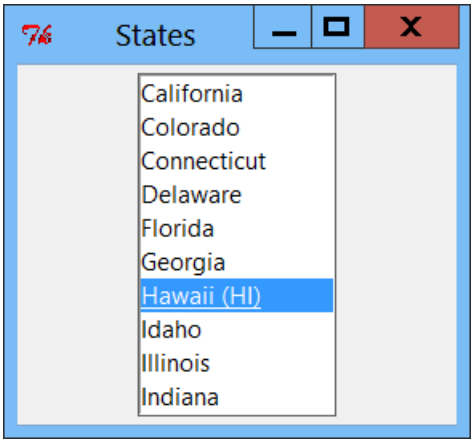


13.
```
def convertToUC(event):
    state = lstStates.get(lstStates.curselection())
    n = listOfStates.index(state)
    listOfStates.remove(state)
    listOfStates.insert(n, state.upper())
    conOFlstStates.set(tuple(listOfStates))

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()
```



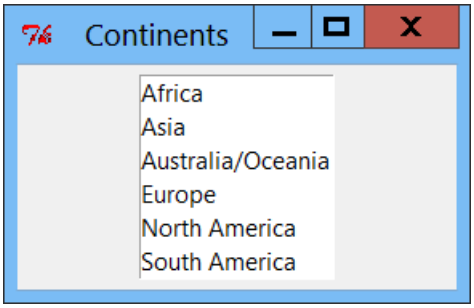
14.
```
def addAbbreviation(e):
    state = lstStates.get(lstStates.curselection())
    abbreviation = findAbbreviation(state)
    n = listOfStates.index(state)
    listOfStates.remove(state)
    listOfStates.insert(n, state + " (" + abbreviation + ")")
    conOFlstStates.set(tuple(listOfStates))

def findAbbreviation(state):
    infile = open("StatesANC.txt", 'r')
    for line in infile:
        if line.split(',')[0] == state:
            return line.split(',')[1]
    infile.close()
```

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

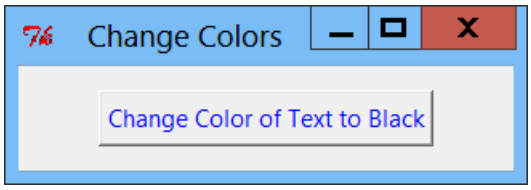


15.
```
from tkinter import *
window = Tk()
window.title("Nations")
infile = open("UN.txt", 'r')
listOfNations = [line.split(',')[0] for line in infile]
infile.close()
conOFlstNations = StringVar()
lstNations = Listbox(window, height=10,
                     width=38, listvariable=conOFlstNations)
lstNations.grid(padx=15,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

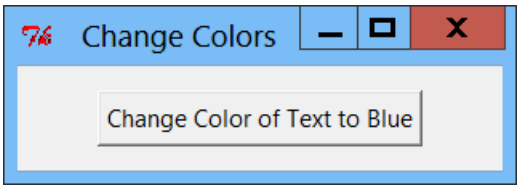
16.
```
from tkinter import *
window = Tk()
window.title("Continents")
infile = open("UN.txt", 'r')
setOfContinents = {line.split(',')[1] for line in infile}
infile.close()
listOfContinents = list(setOfContinents)
listOfContinents.sort()
conOFlstContinents = StringVar()
lstContinents = Listbox(window, height=len(setOfContinents),
                        width=15, listvariable=conOFlstContinents)
lstContinents.grid(row=1, column=0, padx=75, pady=5)
conOFlstContinents.set(tuple(listOfContinents))
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.

18.
```
from tkinter import *

def changeColorandText(e):
    if lblChange["fg"] == "blue":
        lblChange["fg"] = "black"
        lblChange["text"] = "Change Color of Text to Blue"
    else:
        lblChange["fg"] = "blue"
        lblChange["text"] = "Change Color of Text to Black"

window = Tk()
window.title("Change Colors")
lblChange = Label(window, text="Change Color of Text to Black", fg="blue")
lblChange.grid(padx=50, pady=15)
lblChange.bind("<Button-1>", changeColorandText)
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.

20.
```
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", width=12, command=changeText)
btnTest.grid(padx=100, pady=15)
window.mainloop()
```

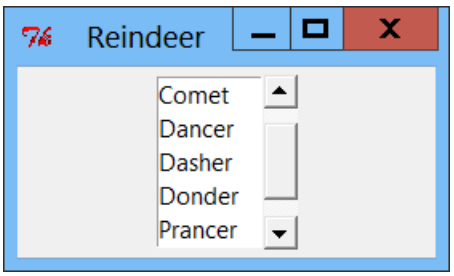
(a) Original display.

(b) Display after first left-click.

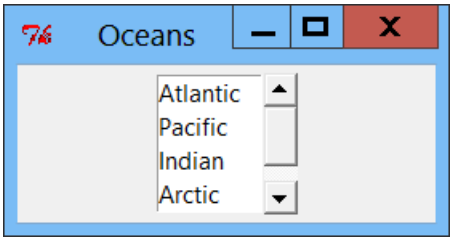
EXERCISES 8.2

1. D **2.** F **3.** B **4.** C **5.** A **6.** E

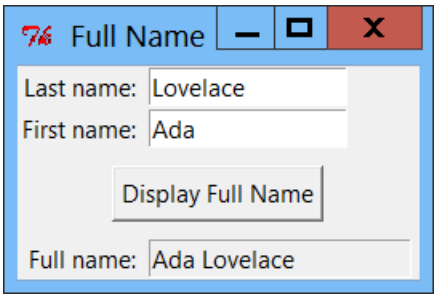
7.
```
from tkinter import *
window = Tk()
window.title("Reindeer")
Label(window, text="", width = 10).grid(row=0, column=0)
Label(window, text="", width = 10).grid(row=0, column=3)
yscroll = Scrollbar(window, orient=VERTICAL)
yscroll.grid(row=0, column=2, rowspan=9, pady=5, sticky=NS)
deerList = ["Blitzen", "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()
```



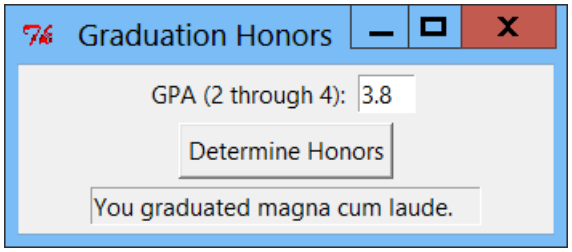
8.
```
from tkinter import *

window = Tk()
window.title("Oceans")
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)
oceanList = ["Atlantic", "Pacific", "Indian", "Arctic", "Antarctic"]
conOFlstOcean = StringVar()
lstOcean = Listbox(window, width=8, height=4, listvariable=conOFlstOcean,
                   yscrollcommand=yscroll.set)
lstOcean.grid(row=0, column=1, rowspan=4, pady=5, sticky=E)
conOFlstOcean.set(tuple(oceanList))
yscroll["command"] = lstOcean.yview
window.mainloop()
```



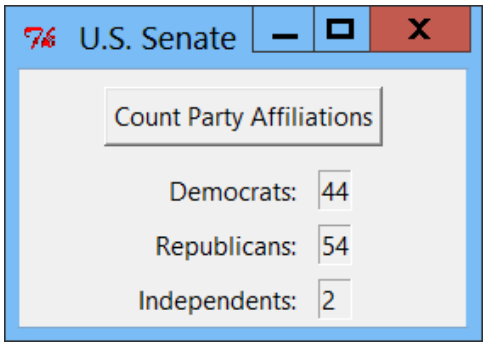
9.
```
from tkinter import *
window = Tk()
window.title("Full Name")
Label(window, text="Last name:").grid(row=0, column=0, sticky=E)
entLastName = Entry(window, width=15)
entLastName.grid(row=0, column=1, padx=5, sticky=W)
Label(window, text="First name:").grid(row=1, column=0, sticky=E)
entFirstName = 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()
```



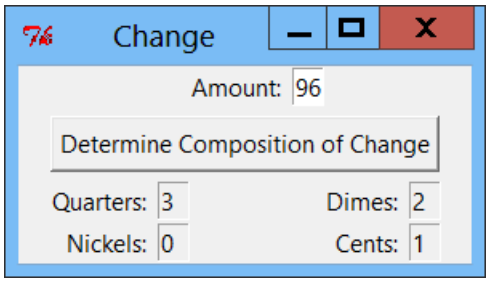
10.
```
from tkinter import *
window = Tk()
window.title("Graduation Honors")
caption = "GPA (2 through 4):"
Label(window, text=caption).grid(row=0, column=0, pady=5, sticky=E)
entGPA = Entry(window, width=4)
entGPA.grid(row=0, column=1, padx=5, sticky=W)
btnDisplay = Button(text="Determine Honors")
btnDisplay.grid(row=1, column=0, columnspan=2, padx=100)
entHonors = Entry(window, state="readonly", width=30)
entHonors.grid(row=2, column=0, columnspan=2, padx=5, pady=5)
window.mainloop()
```



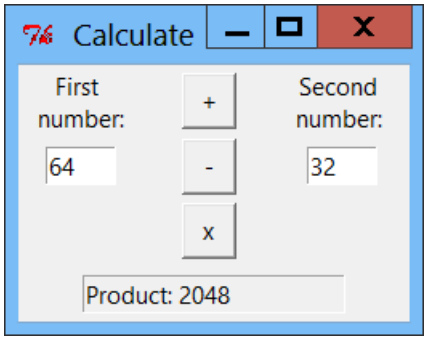
11.
```
from tkinter import *
window = Tk()
window.title("U.S. Senate")
lblDemocrats = Label(window, text="Democrats:")
lblRepublicans = Label(window, text="Republicans:")
lblIndependents = Label(window, text="Independents:")
entDemocrats = Entry(window, width=2, state="readonly)
entRepublicans = Entry(window, 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()
```



12.
```
from tkinter import *
window = Tk()
window.title("Change")
caption = "Amount: "
Label(window, text=caption).grid(row=0, column=1, sticky=E)
entAmount = Entry(window, width=2)
entAmount.grid(row=0, column=2, sticky=W)
caption = "Determine Composition of Change"
btnDetermine = Button(window, text=caption)
btnDetermine.grid(row=1, column=0, columnspan=4, padx=20, pady=5)
Label(window, text="Quarters: ").grid(row=2, column=0, sticky=E)
Label(window, text="Nickels: ").grid(row=3, column=0, sticky=E)
Label(window, text="Dimes: ").grid(row=2, column=2, sticky=E)
Label(window, text="Cents: ").grid(row=3, column=2, sticky=E)
entQuarters = Entry(window, width=2, state="readonly")
entQuarters.grid(row=2, column=1, sticky=W)
entNickels = Entry(window, width=2, state="readonly")
entNickels.grid(row=3, column=1, sticky=W)
entDimes = Entry(window, width=2, state="readonly")
entDimes.grid(row=2, column=3, sticky=W)
entCents = Entry(window, width=2, state="readonly")
entCents.grid(row=3, column=3, sticky=W)
window.mainloop()
```

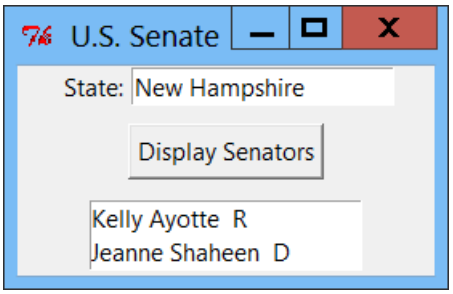


13.
```
from tkinter import *
window = Tk()
window.title("Calculate")
Label(window, text="First \nnumber:").grid(row=0, column=0)
Label(window, text="Second \nnumber: ").grid(row=0, column=2)
entFirst = Entry(window, width=5)
entFirst.grid(row=1, column=0)
entSecond = Entry(window, width=5)
entSecond.grid(row=1, 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()
```

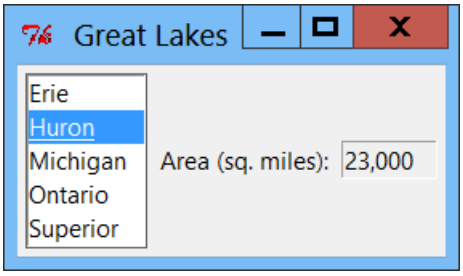


14.
```
from tkinter import *
window = Tk()
window.title("Best Picture")
Label(window, text="Academy Award (1928-2013):").grid(row=0, column=0,
                                    padx=(20,3), pady=5, columnspan=2)
entYear = Entry(window, width=6)
entYear.grid(row=0, column=2, pady=10, sticky=W)
btnFind = Button(window, text="Find Best Picture")
btnFind.grid(row=1, column=0, columnspan=3, pady=(0,8))
Label(window, text="Film:").grid(row=2, column=0, sticky=E)
entFilm = Entry(window, width=37, state="readonly")
entFilm.grid(row=2, column=1, columnspan=2, padx=5, sticky=W)
Label(window, text="Genre:").grid(row=3, column=0, pady=5, sticky=E)
entGenre = Entry(window, width=37, state="readonly")
entGenre.grid(row=3, column=1, columnspan=2, padx=5, sticky=W)
window.mainloop()
```

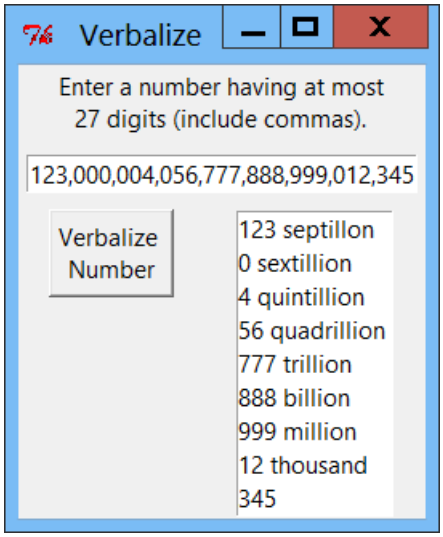
15.
```
from tkinter import *
window = Tk()
window.title("U.S. Senate")
Label(window, text="State:", width=5).grid(row=0, column=0, sticky=E)
state = StringVar()
entState = Entry(window, textvariable=state)
entState.grid(row=0, column=1, sticky=W)
btnDisplay = Button(text="Display Senators")
btnDisplay.grid(row=1, 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()
```



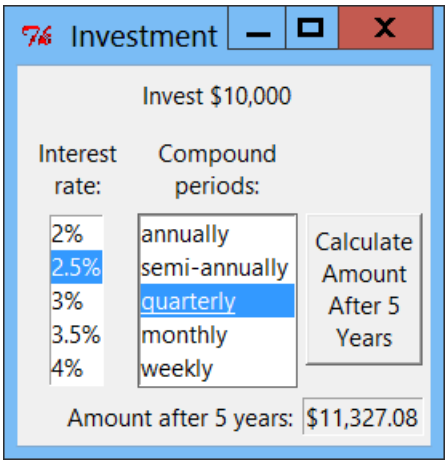
16.
```
from tkinter import *
import pickle
window = Tk()
window.title("Great Lakes")
global lakesDict
lstLakes = Listbox(window, height=5, width=9)
lstLakes.grid(row=0, column=0, padx=5, pady=5, rowspan=5, sticky=NSEW)
lstLakes.bind("<<ListboxSelect>>")
Label(window, text="Area (sq. miles):").grid(row=2, column=1, sticky=E)
entArea = Entry(window, width=7, state="readonly")
entArea.grid(row=2, column=2, padx=5)
window.mainloop()
```



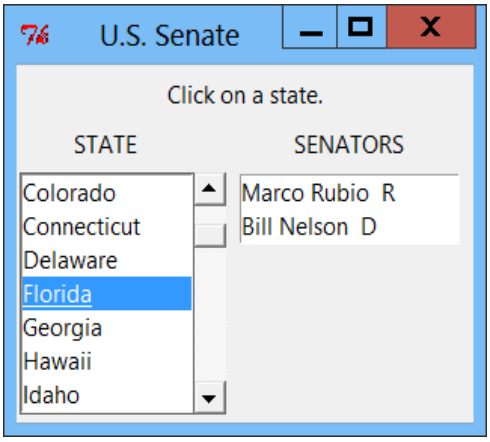
17.
```
from tkinter import *
window = Tk()
window.title("Verbalize")
instruction = "Enter a number having at most\n" + \
              "27 digits (include commas)."
Label(window, text=instruction).grid(row=0, column=0,
      columnspan=2, padx=15)
entNum = Entry(window, width=27)
entNum.grid(row=1, 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()
```



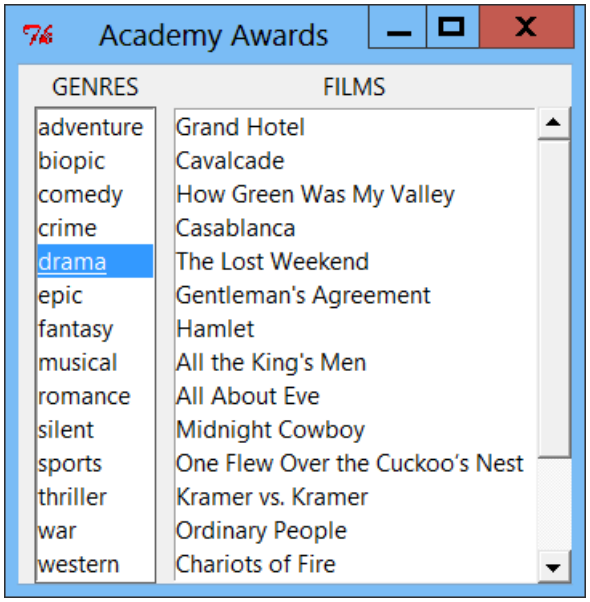
18.
```
from tkinter import *
window = Tk()
window.title("Investment")
Label(window, text="Invest $10,000").grid(row=0, column=1, pady=5)
Label(window, text="Interest\nrate:").grid(row=1, column=0, padx=10, pady=5)
Label(window, text="Compound\nperiods:").grid(row=1, column=1,
                                              padx=10, pady=5)
btnCalculate = Button(window, text="Calculate\nAmount\nAfter 5\nYears")
btnCalculate.grid(row=3, column=2, padx=5, sticky=N)
lstRates = Listbox(window, height=5, width=4)
lstRates.grid(row=3, column=0)
lstPeriods = Listbox(window, height=5, width=12)
lstPeriods.grid(row=3, column=1)
Label(window, text="Amount after 5 years:").grid(row=4, column=0,
                                        pady=5, columnspan=2, sticky=E)
entAmount = Entry(window,  width=9, state="readonly")
entAmount.grid(row=4, column=2, padx = 3, pady=5, sticky=W)
window.mainloop()
```



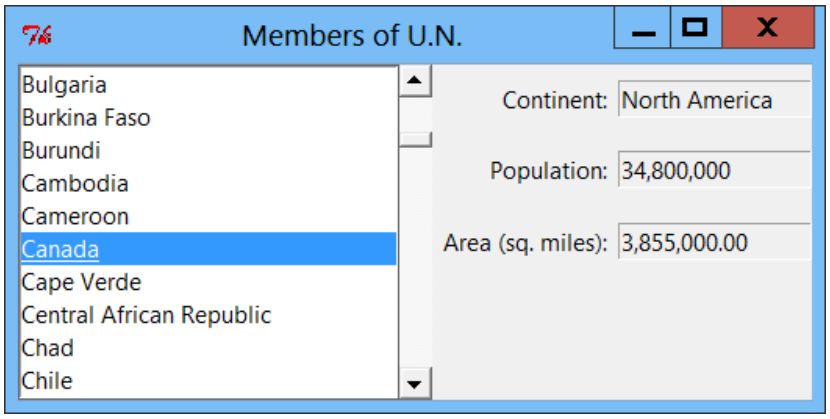
19.
```
from tkinter import *
window = Tk()
window.title("U.S. Senate")
instruction = "Click on a state."
Label(window, text=instruction).grid(row=0, column=0, columnspan=3, pady=5)
Label(window, text="STATE", width=14).grid(row=1, column=0)
Label(window, text="SENATORS").grid(row=1, column=2)
yscroll = 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()
```



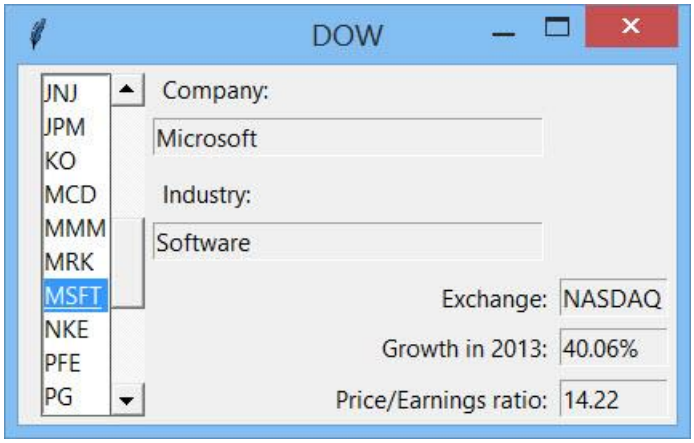
20.
```
from tkinter import *
window = Tk()
window.title("Academy Awards")
Label(window, text="GENRES").grid(row=0, column=0)
Label(window, text="FILMS").grid(row=0, column=1)
genreSet = {line.split(',')[1].rstrip() \
                  for line in open("Oscars.txt", 'r')}
L = list(genreSet)
lstGenres = Listbox(window, width=9, height=len(L))
lstGenres.grid(row=1, column=0, padx=10, sticky=N)
lstGenres.bind("<<ListboxSelect>>")
yscroll = Scrollbar(window, orient=VERTICAL)
yscroll.grid(row=1, column=2, sticky=NS)
lstFilms = Listbox(window, width=28, height=len(L),
                         yscrollcommand=yscroll.set)
lstFilms.grid(row=1, column=1, sticky=NSEW)
yscroll["command"] = lstFilms.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()
```



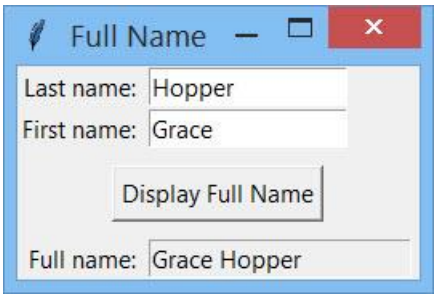
22.
```
from tkinter import *
window = Tk()
window.title("DOW")
Label(window, text="", width=1).grid(row=0, column=0)
Label(window, text="  Company:").grid(row=0, column=3,  sticky=W)
Label(window, text="  Industry:").grid(row=3, column=3, sticky=W)
Label(window, text="Exchange:").grid(row=6, column=4, sticky=E)
Label(window, text="Growth in 2013:").grid(row=7, column=4, sticky=E)
Label(window, text="Price/Earnings ratio:").grid(row=8, column=4, sticky=E)
yscroll = Scrollbar(window, orient=VERTICAL)
yscroll.grid(row=0, column=2, rowspan=9,pady=5, sticky=NS)
lstSymbols = Listbox(window, width=5, yscrollcommand=yscroll.set)
lstSymbols.grid(row=0, column=1, rowspan=9, pady=5, sticky=E)
lstSymbols.bind("<<ListboxSelect>>")
entCompany = Entry(window, state="readonly", width=30)
entCompany.grid(row=1, column=3, columnspan=2, padx=5, sticky=W)
entIndustry = Entry(window, state="readonly", width=30)
entIndustry.grid(row=4, column=3, columnspan=2, padx=5, sticky=W)
entExchange = Entry(window, width=8, state="readonly")
entExchange.grid(row=6, column=5, padx=5, sticky=W)
entGrowth = Entry(window, width=8, state="readonly")
entGrowth.grid(row=7, column=5, padx=5, sticky=W)
entPE = Entry(window, width=8, state="readonly")
entPE.grid(row=8, column=5, padx=5, sticky=W)
yscroll["command"] = lstSymbols.yview
window.mainloop()
```



EXERCISES 8.3

(Most of the programs are written both 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()
```



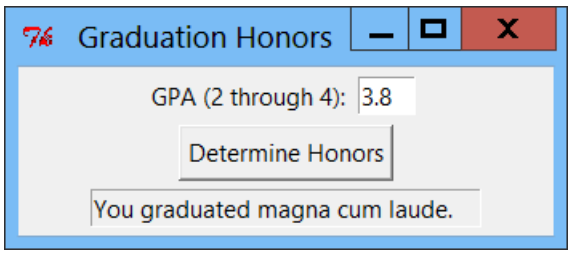
1. (Object-oriented style)

```
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()
```

2.

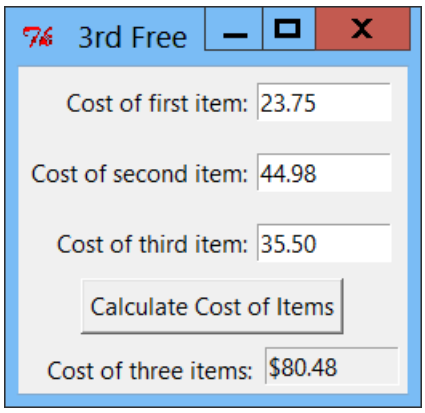
```
from tkinter import *

def honors():
    gpa = float(conOFentGPA.get())
    if gpa >= 3.9:
        honor = " summa cum laude."
    elif gpa >= 3.6:
        honor = " magna cum laude."
    elif gpa >= 3.3:
        honor = " cum laude."
    else:
        honor = "."
    # Display conclusion.
    conOFentHonors.set("You graduated" + honor)

window = Tk()
window.title("Graduation Honors")
caption = "GPA (2 through 4):"
Label(window, text=caption).grid(row=0, column=0, pady=5, sticky=E)
conOFentGPA = StringVar()
entGPA = Entry(window, width=4, textvariable=conOFentGPA)
entGPA.grid(row=0, column=1, padx=5, sticky=W)
btnDisplay = Button(text="Determine Honors", command=honors)
btnDisplay.grid(row=1, column=0, columnspan=2, padx=100)
conOFentHonors = StringVar()
entHonors = Entry(window, state="readonly", width=30,
                  textvariable=conOFentHonors)
entHonors.grid(row=2, column=0, columnspan=2, padx=5, pady=5)
window.mainloop()
```



2. (Object-oriented style)

```
from tkinter import *

class GPA:
    def __init__(self):
        window = Tk()
        window.title("Graduation Honors")
        caption = "GPA (2 through 4):"
        Label(window, text=caption).grid(row=0, column=0, pady=5, sticky=E)
        self.conOFentGPA = StringVar()
        entGPA = Entry(window, width=4, textvariable=self.conOFentGPA)
        entGPA.grid(row=0, column=1, padx=5, sticky=W)
        btnDisplay = Button(text="Determine Honors", command=self.honors)
        btnDisplay.grid(row=1, column=0, columnspan=2, padx=100)
        self.conOFentHonors = StringVar()
        self.entHonors = Entry(window, state="readonly", width=30,
                               textvariable=self.conOFentHonors)
        self.entHonors.grid(row=2, column=0, columnspan=2, padx=5, pady=5)
        window.mainloop()

    def honors(self):
        gpa = float(self.conOFentGPA.get())
        if gpa >= 3.9:
            honor = " summa cum laude."
        elif gpa >= 3.6:
            honor = " magna cum laude."
        elif gpa >= 3.3:
            honor = " cum laude."
        else:
            honor = "."
        # Display conclusion.
        self.conOFentHonors.set("You graduated" + honor)

GPA()
```

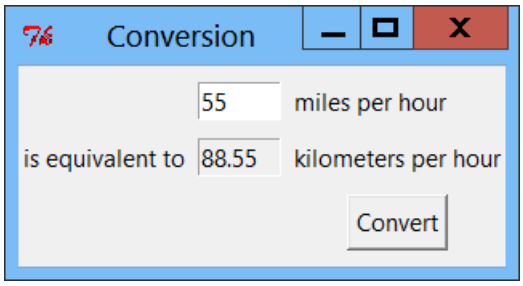
3.
```
from tkinter import *

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)

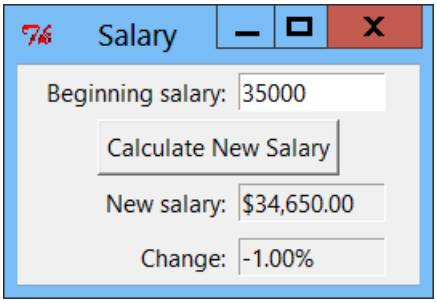
```
from tkinter import *

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()
```

4.
```
from tkinter import *

def convertUnits():
    kph = 1.61 * eval(conOFentMPH.get())
    conOFentKPH.set("{0:,.2f}".format(kph))

window = Tk()
window.title("Conversion")
conOFentMPH = StringVar()
entMPH = Entry(window, width=6, textvariable=conOFentMPH)
entMPH.grid(row=0, column=1, pady=10)
Label(window, text="miles per hour").grid(row=0, column=2, sticky=W)
Label(window, text="is equivalent to").grid(row=1, column=0)
conOFentKPH = StringVar()
entKPH = Entry(window, width=6, textvariable=conOFentKPH, state="readonly")
entKPH.grid(row=1, column=1, padx=5)
Label(window, text="kilometers per hour").grid(row=1, column=2)
btnCalculate = Button(window, text="Convert", command=convertUnits)
btnCalculate.grid(row=2, column=2,pady=10)
window.mainloop()
```



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)
```

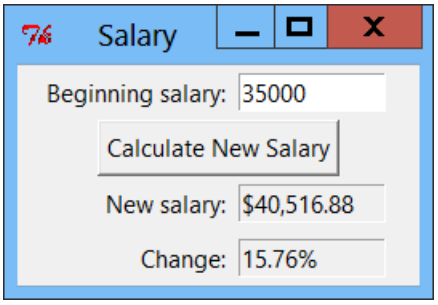
```
Label(window, text="Change:").grid(row=4, column=0, sticky=E)
conOFentChange = StringVar()
entChange = Entry(window, width=11, state="readonly",
                          textvariable=conOFentChange)
entChange.grid(row=4, column=1, padx=5, pady=5, sticky=W)
window.mainloop()
```



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()
```

6.
```
from tkinter import *

def newSalary():
    salary = eval(conOFentBegSalary.get())
    for i in range(3):
        salary += .05 * 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()
```



6. (Object-oriented style)

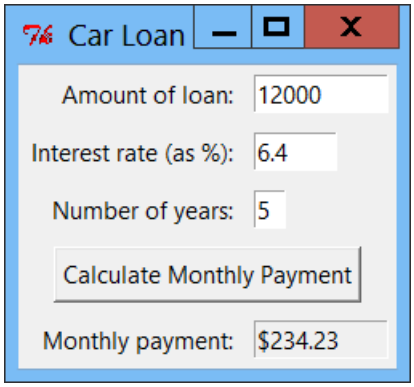
```
from tkinter import *

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):
        salary = eval(self.conOFentBegSalary.get())
        for i in range(3):
            salary += .05 * 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()
```

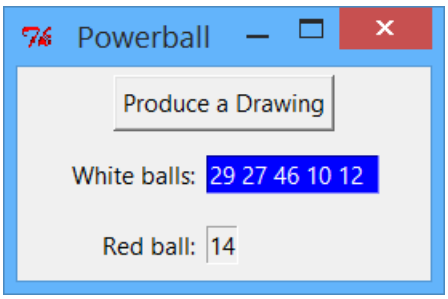


7. (Object-oriented style)

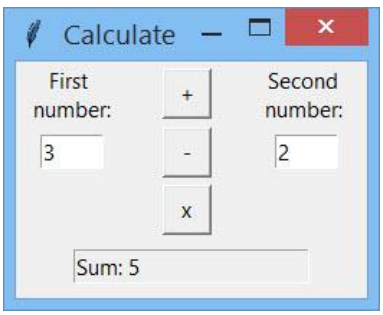
```
from tkinter import *

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()
```

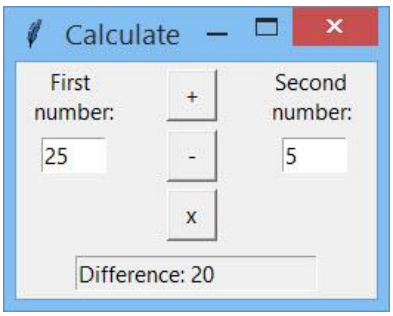
```
        def calculate(self):
            p = eval(self.principal.get())
            r = eval(self.interestRate.get())
            n = eval(self.numberOfYears.get())
            payment = (p*(r/1200)/(1 - (1 + (r/1200)) ** (-12*n)))
            payment = "${0:,.2f}".format(payment)
            self.monthlyPayment.set(payment)

    CarLoan()
```

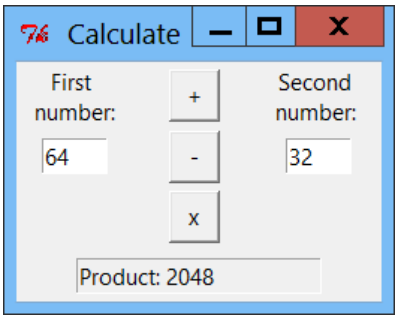
8.
```
    from tkinter import *
    import random

    def drawing():
        conOFentWhiteBalls.set("")
        nums = [x for x in range(1, 60)]
        five = random.sample(nums, 5)
        fiveString = [str(x) for x in five]
        conOFentWhiteBalls.set(" ".join(fiveString))
        num = random.choice(range(1, 36))
        conOFentRedBalls.set(str(num))

    window = Tk()
    window.title("Powerball")
    btnProduce = Button(window, text="Produce a Drawing", command=drawing)
    btnProduce.grid(row=0, column=0, columnspan=2, padx=60, pady=5)
    Label(window, text="White balls: ").grid(row=1, column=0, sticky=E)
    conOFentWhiteBalls = StringVar()
    entWhiteBalls = Entry(window, width=13, fg="white", bg="blue",
                          textvariable=conOFentWhiteBalls)
    entWhiteBalls.grid(row=1, column=1, pady=10, sticky=W)
    Label(window, text="Red ball: ").grid(row=2, column=0, sticky=E)
    conOFentRedBalls = StringVar()
    entRedBalls = Entry(window, width=2, fg="black", bg="white",
                        textvariable=conOFentRedBalls)
    entRedBalls.grid(row=2, column=1, pady=10, sticky=W)
    window.mainloop()
```



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()
```



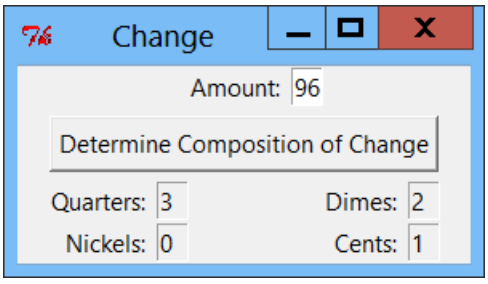




9. (Object-oriented style)

```
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()
```

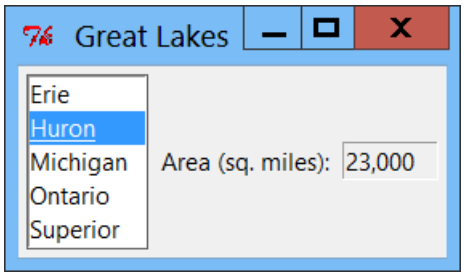
10.
```
from tkinter import *

def makeChange():
    amount = int(conOFentAmount.get())
    remainder = amount
    quarters = remainder // 25
    remainder %= 25
    dimes = remainder // 10
    remainder %= 10
    nickels = remainder // 5
    remainder %= 5
    cents = remainder
    conOFentQuarters.set(str(quarters))
    conOFentDimes.set(str(dimes))
    conOFentNickels.set(str(nickels))
    conOFentCents.set(str(cents))

window = Tk()
window.title("Change")
caption = "Amount: "
Label(window, text=caption).grid(row=0, column=1, sticky=E)
conOFentAmount = StringVar()
entAmount = Entry(window, width=2, textvariable=conOFentAmount)
entAmount.grid(row=0, column=2, sticky=W)
caption = "Determine Composition of Change"
btnDetermine = Button(window, text=caption, command=makeChange)
btnDetermine.grid(row=1, column=0, columnspan=4, padx=20, pady=5)
Label(window, text="Quarters: ").grid(row=2, column=0, sticky=E)
Label(window, text="Nickels: ").grid(row=3, column=0, sticky=E)
Label(window, text="Dimes: ").grid(row=2, column=2, sticky=E)
Label(window, text="Cents: ").grid(row=3, column=2, sticky=E)
conOFentQuarters = StringVar()
entQuarters = Entry(window, width=2, state="readonly",
                    textvariable=conOFentQuarters)
entQuarters.grid(row=2, column=1, sticky=W)
conOFentNickels = StringVar()
entNickels = Entry(window, width=2, state="readonly",
                    textvariable=conOFentNickels)
entNickels.grid(row=3, column=1, sticky=W)
conOFentDimes = StringVar()
entDimes = Entry(window, width=2, state="readonly",
                    textvariable=conOFentDimes)
entDimes.grid(row=2, column=3, sticky=W)
conOFentCents = StringVar()
entCents = Entry(window, width=2, state="readonly", textvariable=conOFentCents)
entCents.grid(row=3, column=3, sticky=W)
window.mainloop()
```



10. (Object-oriented style)

```
from tkinter import *

class Change:
    def __init__(self):
        window = Tk()
        window.title("Change")
        caption = "Amount: "
        Label(window, text=caption).grid(row=0, column=1,
                                         sticky=E)
        self._conOFentAmount = StringVar()
        self.entAmount = Entry(window, width=2,
                               textvariable=self._conOFentAmount)
        self.entAmount.grid(row=0, column=2, sticky=W)
        caption = "Determine Composition of Change"
        btnDetermine = Button(window, text=caption,
                                    command=self.makeChange)
        btnDetermine.grid(row=1, column=0, columnspan=4, padx=20, pady=5)
        Label(window, text="Quarters: ").grid(row=2, column=0, sticky=E)
        Label(window, text="Nickels: ").grid(row=3, column=0, sticky=E)
        Label(window, text="Dimes: ").grid(row=2, column=2, sticky=E)
        Label(window, text="Cents: ").grid(row=3, column=2, sticky=E)
        self._conOFentQuarters = StringVar()
        self.entQuarters = Entry(window, width=2, state="readonly",
                                 textvariable=self._conOFentQuarters)
        self.entQuarters.grid(row=2, column=1, sticky=W)
        self._conOFentNickels = StringVar()
        self.entNickels = Entry(window, width=2, state="readonly",
                                 textvariable=self._conOFentNickels)
        self.entNickels.grid(row=3, column=1, sticky=W)
        self._conOFentDimes = StringVar()
        self.entDimes = Entry(window, width=2, state="readonly",
                                 textvariable=self._conOFentDimes)
        self.entDimes.grid(row=2, column=3, sticky=W)
        self._conOFentCents = StringVar()
        self.entCents = Entry(window, width=2, state="readonly",
                                 textvariable=self._conOFentCents)
        self.entCents.grid(row=3, column=3, sticky=W)
        window.mainloop()

    def makeChange(self):
        amount = int(self._conOFentAmount.get())
        remainder = amount
        quarters = remainder // 25
        remainder %= 25
        dimes = remainder // 10
        remainder %= 10
        nickels = remainder // 5
        remainder %= 5
        cents = remainder
        self._conOFentQuarters.set(str(quarters))
        self._conOFentDimes.set(str(dimes))
        self._conOFentNickels.set(str(nickels))
        self._conOFentCents.set(str(cents))

Change()
```

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)

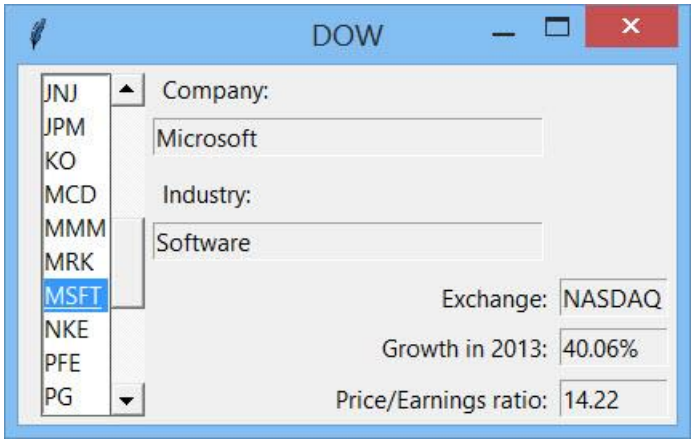
```
from tkinter import *

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()
```

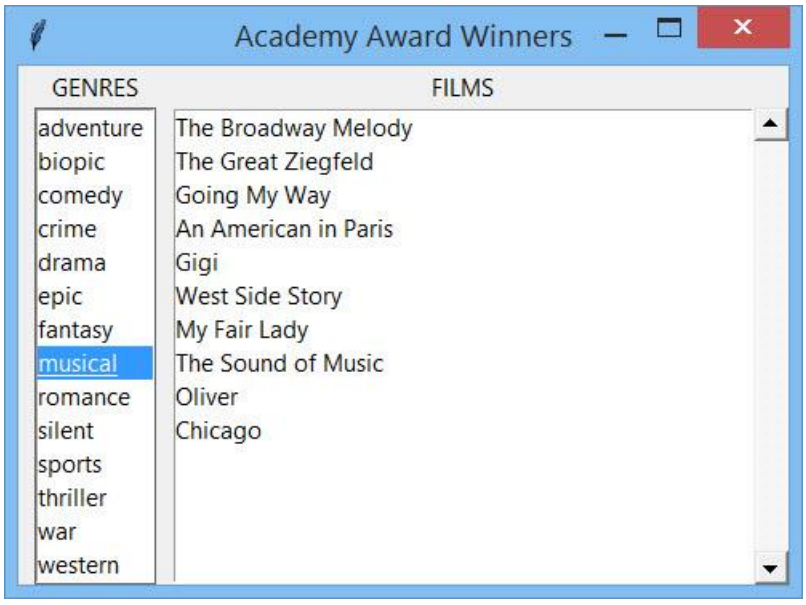
12.

```
    from tkinter import *

    class DOW:

        def __init__(self):
            window = Tk()
            window.title("DOW")
            Label(window, text="", width=1).grid(row=0, column=0)
            Label(window, text="  Company:").grid(row=0, column=3,  sticky=W)
            Label(window, text="  Industry:").grid(row=3, column=3, sticky=W)
            Label(window, text="Exchange:").grid(row=6, column=4, sticky=E)
            Label(window, text="Growth in 2013:").grid(row=7, column=4, sticky=E)
            Label(window, text="Price/Earnings ratio:").grid(row=8, column=4,
                                                             sticky=E)
            yscroll = Scrollbar(window, orient=VERTICAL)
            yscroll.grid(row=0, column=2, rowspan=9,pady=5, sticky=NS)
            infile = open("DOW.txt", 'r')
            symbolSet = {line.split(',')[1] for line in infile}
            infile.close()
            symbolList = list(symbolSet)
            symbolList.sort()
            self.conOFlstSymbols = StringVar()
            self._lstSymbols = Listbox(window, width=5,
                listvariable=self.conOFlstSymbols, yscrollcommand=yscroll.set)
            self._lstSymbols.grid(row=0, column=1, rowspan=9, pady=5, sticky=E)
            self._lstSymbols.bind("<<ListboxSelect>>", self.facts)
            self.conOFlstSymbols.set(tuple(symbolList))
            self.conOFentCompany = StringVar()
            self.entCompany = Entry(window, state="readonly", width=30,
                                    textvariable=self.conOFentCompany)
            self.entCompany.grid(row=1, column=3, columnspan=2, padx=5, sticky=W)
            self.conOFentIndustry = StringVar()
            self.entIndustry = Entry(window, state="readonly", width=30,
                                     textvariable=self.conOFentIndustry)
            self.entIndustry.grid(row=4, column=3, columnspan=2, padx=5, sticky=W)
            self.conOFentExchange = StringVar()
            self.entExchange = Entry(window, width=8, state="readonly",
                                     textvariable=self.conOFentExchange)
```

```
        self.entExchange.grid(row=6, column=5, padx=5, sticky=W)
        self.conOFentGrowth = StringVar()
        self.entGrowth = Entry(window, width=8, state="readonly",
                               textvariable=self.conOFentGrowth)
        self.entGrowth.grid(row=7, column=5, padx=5, sticky=W)
        self.conOFentPE = StringVar()
        self.entPE = Entry(window, width=8, state="readonly",
                           textvariable=self.conOFentPE)
        self.entPE.grid(row=8, column=5, padx=5, sticky=W)
        yscroll["command"] = self._lstSymbols.yview
        window.mainloop()

    def facts(self, e):
        ## Display information about a DOW stock.
        symbol = self._lstSymbols.get(self._lstSymbols.curselection())
        infile = open("DOW.txt", 'r')
        while True:
            line = infile.readline()
            lineList = line.split(',')
            if lineList[1] == symbol:
                break
        infile.close()
        self.conOFentCompany.set(lineList[0])
        self.conOFentIndustry.set(lineList[3])
        self.conOFentExchange.set(lineList[2])
        increase = (float(lineList[5]) - float(lineList[4])) /
                    float(lineList[4])
        self.conOFentGrowth.set("{0:.2%}".format(increase))
        priceEarningsRatio = float(lineList[5]) / float(lineList[6])
        self.conOFentPE.set("{0:.2f}".format(priceEarningsRatio))

DOW()
```



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()
```



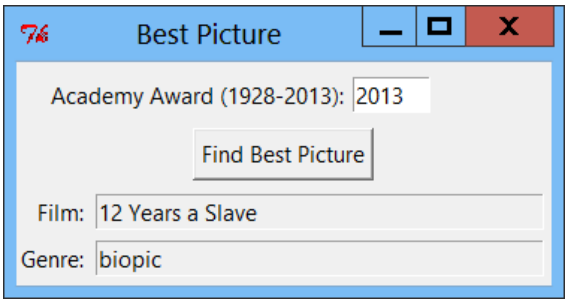
13. (Object-oriented style)

```
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()
```

14.

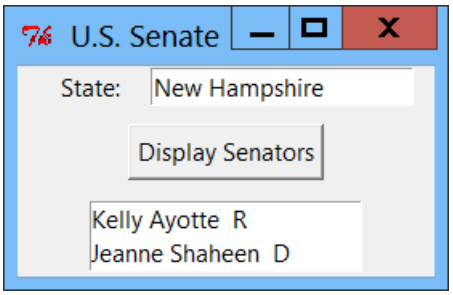
```
from tkinter import *
class Oscars:
    def __init__(self):
        window = Tk()
        window.title("Academy Awards")
        caption = "Year (1928-2013): "
        Label(window, text=caption).grid(row=0, column=0)
        self._conOFentYear = StringVar()
        self.entYear = Entry(window, width=4,
                             textvariable=self._conOFentYear)
        self.entYear.grid(row=0, column=1, sticky=W)
        caption = "Find Best Picture"
        btnFind = Button(window, text=caption, command=self.displayFilm)
        btnFind.grid(row=1, column=1, pady=2)
        Label(window, text="Film:").grid(row=2, column=0, sticky=E)
        Label(window, text="Genre:").grid(row=3, column=0, pady=5,
                                          sticky=E)
```

```
            self._conOFentFilm = StringVar()
            self.entFilm = Entry(window, width=30, state="readonly",
                                 textvariable=self._conOFentFilm)
            self.entFilm.grid(row=2, column=1, padx=5,sticky=W)
            self._conOFentGenre = StringVar()
            self.entGenre = Entry(window, width=30, state="readonly",
                                  textvariable=self._conOFentGenre)
            self.entGenre.grid(row=3, column=1, padx=5,pady=5, sticky=W)
            window.mainloop()

        def displayFilm(self):
            infile = open("Oscars.txt", 'r')
            for i in range(int(self._conOFentYear.get()) - 1928):
                infile.readline()
            line = infile.readline().rstrip()
            infile.close()
            data = line.split(',')
            self._conOFentFilm.set(data[0])
            self._conOFentGenre.set(data[1])

    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)

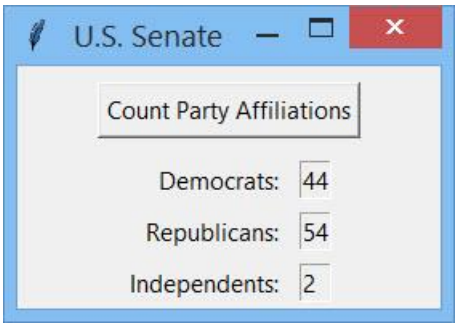
```
from tkinter import *

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()
```

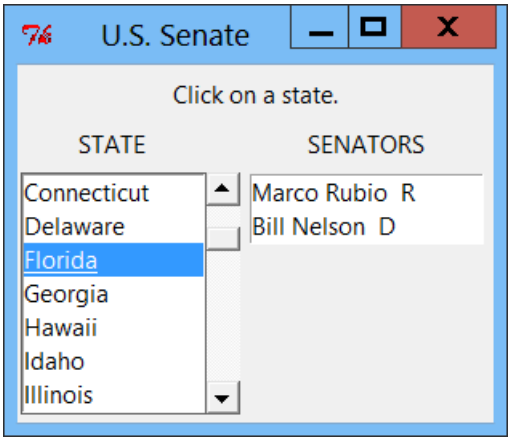
16.

```
    from tkinter import *

    class Affiliations:

        def __init__(self):
            window = Tk()
            window.title("U.S. Senate")
            lblDemocrats = Label(window, text="Democrats:")
            lblRepublicans = Label(window, text="Republicans:")
            lblIndependents = Label(window, text="Independents:")
            self._conOFentDemocrats = StringVar()
            self._conOFentRepublicans = StringVar()
            self._conOFentIndependents = StringVar()
            entDemocrats = Entry(window, width=2, state="readonly",
                                 textvariable=self._conOFentDemocrats)
            entRepublicans = Entry(window, width=2, state="readonly",
                                   textvariable=self._conOFentRepublicans)
            entIndependents = Entry(window, width=2, state="readonly",
                                    textvariable=self._conOFentIndependents)
            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",
                                command=self.count)
            btnDisplay.grid(row=0, columnspan=4, padx=50, pady=10)
            window.mainloop()
```

```
    def count(self):
        D = 0
        R = 0
        I = 0
        infile = open("Senate114.txt", 'r')
        for line in infile:
            lst = line.split(',')
            if lst[2] == "D\n":
                D += 1
            elif lst[2] == "R\n":
                R += 1
            else:
                I += 1
        infile.close()
        self._conOFentDemocrats.set(str(D))
        self._conOFentRepublicans.set(str(R))
        self._conOFentIndependents.set(str(I))

Affiliations()
```



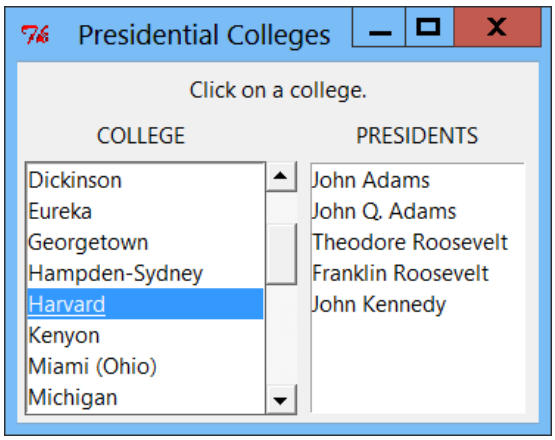
17.
```
from tkinter import *

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()
```



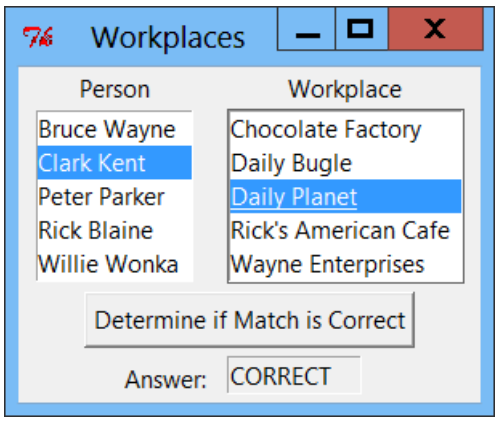
17. (Object-oriented style)

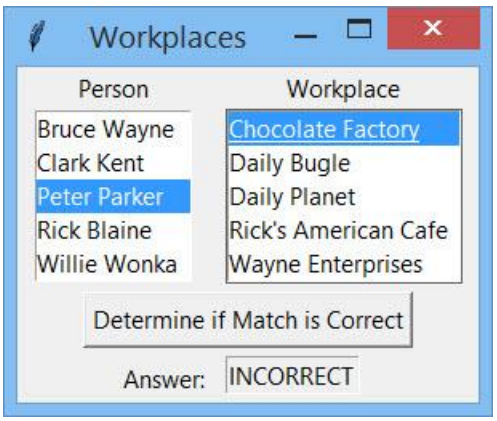
```
from tkinter import *

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())
        for line in open("Senate114.txt", 'r'):
            temp = line.split(',')
            if temp[1] == state:
                self.L.append(temp[0] + "  " + temp[2])
        self._conOFlstSenators.set(tuple(self.L))

Senators()
```

18.
```
from tkinter import *

class PresColleges:

    def __init__(self):
        window = Tk()
        window.title("Presidential Colleges")
        instruction = "Click on a college."
        Label(window, text=instruction).grid(row=0, column=0,
                                             columnspan=3, pady=5)
        Label(window, text="COLLEGE", width=14).grid(row=1, column=0)
        Label(window, text="PRESIDENTS").grid(row=1, column=2)
        yscroll = Scrollbar(window, orient=VERTICAL)
        yscroll.grid(row=2, column=1, pady=5, sticky=NS)
        infile = open("PresColl.txt", 'r')
        collegeSet = {line.split(',')[1].rstrip() for line in infile}
        infile.close()
        collegeList = list(collegeSet)
        collegeList.sort()
        conOFlstColleges = StringVar()
        self._lstColleges = Listbox(window, width=20, height=8,
              listvariable=conOFlstColleges, yscrollcommand=yscroll.set)
        self._lstColleges.grid(row=2, column=0, padx=(5,0), pady=5, sticky=E)
        self._lstColleges.bind("<<ListboxSelect>>", self.presidents)
        conOFlstColleges.set(tuple(collegeList))
        self._conOFlstPresidents = StringVar()
        self._lstPresidents = Listbox(window, width=18, height=8,
                              listvariable=self._conOFlstPresidents)
        self._lstPresidents.grid(row=2, column=2, padx=8, pady=5, sticky=N)
        yscroll["command"] = self._lstColleges.yview
        window.mainloop()

    def presidents(self, e):
        self.L = []
        college = self._lstColleges.get(self._lstColleges.curselection())
        for line in open("PresColl.txt", 'r'):
            temp = line.split(',')
        if temp[1].rstrip() == college:
            self.L.append(temp[0])
        self._conOFlstPresidents.set(tuple(self.L))

PresColleges()
```



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()
```



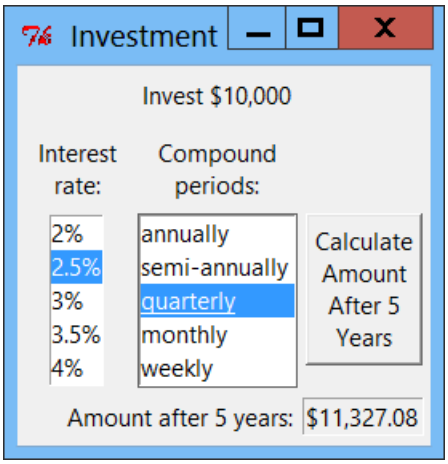


19. (Object-oriented style)

```
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()
```

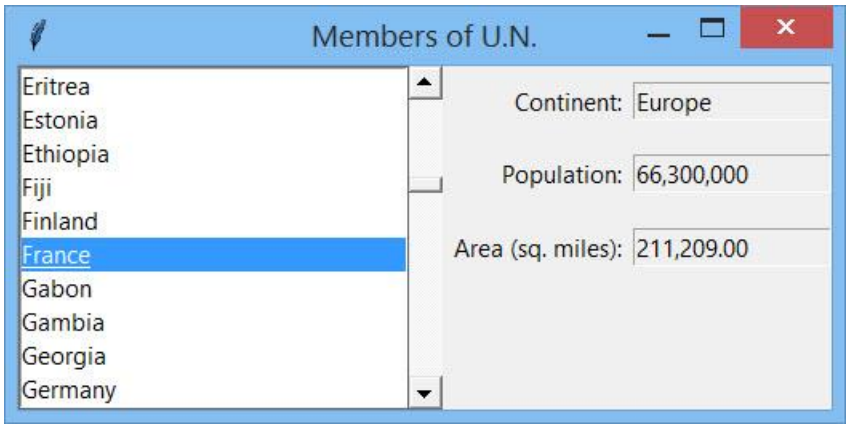
PROGRAMMING PROJECTS CHAPTER 8

1.
```
from tkinter import *
def calculate():
    rate = lstRates.get(lstRates.curselection())
    if rate == "2%":
        intRate = .02
    elif rate == "2.5%":
        intRate = .025
    elif rate == "3%":
        intRate = .03
    elif rate == "3.5%":
        intRate = .035
    elif rate == "4%":
        intRate = .04
    periods = lstPeriods.get(lstPeriods.curselection())
    if periods == "annually":
        n = 1
    elif periods == "semi-annually":
        n = 2
    elif periods == "quarterly":
        n = 4
    elif periods == "monthly":
        n = 12
    elif periods == "weekly":
        n = 52
    amount = 10000 * (1 +  intRate/n) ** (5*n)
    conOFentAmount.set("${0:,.2f}".format(amount))

window = Tk()
window.title("Investment")
Label(window, text="Invest $10,000").grid(row=0, column=1, pady=5)
Label(window, text="Interest\nrate:").grid(row=1, column=0, padx=10, pady=5)
Label(window, text="Compound\nperiods:").grid(row=1, column=1,
                                              padx=10, pady=5)
btnCalculate = Button(window, text="Calculate\nAmount\nAfter 5\nYears",
                      command=calculate)
btnCalculate.grid(row=3, column=2, padx=5, sticky=N)
conOFlstRates = StringVar()
lstRates = Listbox(window, height=5, width=4, exportselection=0,
                   listvariable=conOFlstRates)
lstRates.grid(row=3, column=0)
conOFlstPeriods = StringVar()
lstPeriods = Listbox(window, height=5, width=12, exportselection=0,
                     listvariable=conOFlstPeriods)
lstPeriods.grid(row=3, column=1)
Label(window, text="Amount after 5 years:").grid(row=4, column=0,
                                    pady=5, columnspan=2, sticky=E)
conOFentAmount = StringVar()
entAmount = Entry(window, textvariable=conOFentAmount,
                  width=9, state="readonly")
entAmount.grid(row=4, column=2, padx = 3, pady=5, sticky=W)
conOFlstRates.set(("2%", "2.5%","3%","3.5%","4%"))
conOFlstPeriods.set(("annually", "semi-annually",
                     "quarterly", "monthly", "weekly"))
window.mainloop()
```

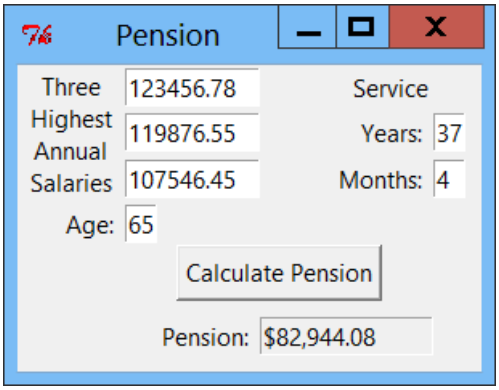


2.
```
from tkinter import *
import pickle

class Nations:

    def __init__(self):
        window = Tk()
        window.title("Members of U.N.")
        infile = open("UNdict.dat", 'rb')
        self._nationDict = pickle.load(infile)
        infile.close()
        self._nationList = list((self._nationDict).keys())
        self._nationList.sort()
        self._conOFlstNations = StringVar()
        yscroll = Scrollbar(window, orient=VERTICAL)
        yscroll.grid(row=0, column=1, rowspan=7, sticky=NS)
        self._lstNations = Listbox(window, height=10, width=30,
            listvariable=self._conOFlstNations, yscrollcommand=yscroll.set)
        self._lstNations.grid(row=0, column=0, rowspan=7, sticky=NSEW)
        self._conOFlstNations.set(tuple(self._nationList))
        self._lstNations.bind("<<ListboxSelect>>", self.displayData)
        yscroll["command"] = self._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)
        self._conOFentContinent = StringVar()
        entContinent = Entry(window, width=15, state="readonly",
                             textvariable=self._conOFentContinent)
        entContinent.grid(row=0, column=4, sticky=W)
        self._conOFentPopulation = StringVar()
        entPopulation = Entry(window, width=15, state="readonly",
                              textvariable=self._conOFentPopulation)
        entPopulation.grid(row=1, column=4,)
        self._conOFentArea = StringVar()
        entArea = Entry(window, width=15, state="readonly",
                            textvariable=self._conOFentArea)
        entArea.grid(row=2, column=4)
        window.mainloop()
```

```
    def displayData(self, e):
        nation = self._lstNations.get(self._lstNations.curselection())
        self._conOFentContinent.set(self._nationDict[nation]["cont"])
        self._conOFentPopulation.set("{0:,.0f}". \
             format(1000000 * float(self._nationDict[nation]["popl"])))
        self._conOFentArea.set("{0:,.2f}".\
                               format(self._nationDict[nation]["area"]))

Nations()
```



3.
```
from tkinter import *

def calculate():
    ave = (eval(conOFentSalary1.get()) + eval(conOFentSalary2.get()) +
           eval(conOFentSalary3.get())) / 3
    yrs = eval(conOFentYears.get())
    months = eval(conOFentMonths.get())
    yrs += months / 12
    percentage = 36.25 + (2 * (yrs - 20))
    if percentage > 80:
        percentage = 80
    pension = ave * (percentage / 100)
    conOFentPension.set("${0:,.2f}".format(pension))
window = Tk()
window.title("Pension")
Label(window, text="Three\nHighest\nAnnual\nSalaries").grid(row=0,
                                                column=0, rowspan=3, padx=5)
conOFentSalary1 = StringVar()
entSalary1 = Entry(window, width=10, textvariable=conOFentSalary1)
entSalary1.grid(row=0, column=1)
conOFentSalary2 = StringVar()
entSalary2 = Entry(window, width=10, textvariable=conOFentSalary2)
entSalary2.grid(row=1, column=1)
conOFentSalary3 = StringVar()
entSalary3 = Entry(window, width=10, textvariable=conOFentSalary3)
entSalary3.grid(row=2, column=1)
Label(window, text="Service").grid(row=0, column=2, sticky=E)
Label(window, text="Years: ").grid(row=1, column=2, sticky=E)
conOFentYears = StringVar()
entYears = Entry(window, width=2, textvariable=conOFentYears)
entYears.grid(row=1, column=3, padx=(0, 10), sticky=W)
Label(window, text="Months: ").grid(row=2, column=2, sticky=E)
conOFentMonths = StringVar()
entMonths = Entry(window, width=2, textvariable=conOFentMonths)
entMonths.grid(row=2, column=3, sticky=W)
Label(window, text="Age: ").grid(row=3, column=0, sticky=E)
conOFentAge = StringVar()
entAge = Entry(window, width=2, textvariable=conOFentAge)
entAge.grid(row=3, column=1, sticky=W)
btnCalculate = Button(window, text="Calculate Pension", command=calculate)
btnCalculate.grid(row=4, column=1, columnspan=2)
Label(window, text="Pension: ").grid(row=5, column=1, sticky=E)
conOFentPension = StringVar()
entPension = Entry(window, width=13,
                   textvariable=conOFentPension,state="readonly")
entPension.grid(row=5, column=2, pady=10)
window.mainloop()
```



4.
```
from tkinter import *

def stripOutLeadingZeros(front):
    if front == "000":
        front = "0"
    elif front[:2] == "00":
        front = front[2]
    elif front[0] == "0":
        front = front[1:]
    return front

def verbalize():
    L = ["", " thousand", " million", " billion", " trillion",
         " quadrillion", " quintillion", " sextillion", " septillion"]
    N = []
    number = conOFentNumber.get()
    numberOfCommas = number.count(',')
    L = L[:numberOfCommas + 1]
    for i in range(numberOfCommas + 1, 0, -1):
        loc = number.find(',')
        if loc == -1:
            number = stripOutLeadingZeros(number)
            N.append(number)
        else:
            front = number[:loc]
            front = stripOutLeadingZeros(front)
            N.append(front + L[-1])
        conOflstBox.set(tuple(N))
        number = number[loc + 1:]
        del L[-1]
```

```
window = Tk()
window.title("Verbalize")
instructions = "Enter a number having at most\n27 digits (include commas)."
lbl = Label(window, text=instructions)
lbl.grid(row=0, column=0, columnspan=2)
conOFentNumber = StringVar()
entNumber = Entry(window, width=30, textvariable=conOFentNumber)
entNumber.grid(row=1, column=0, columnspan=2,padx=5, pady=10)
conOflstBox = StringVar()
lstBox = Listbox(window, width=12, height=9, listvariable=conOflstBox)
lstBox.grid(row=2, column=1)
b = Button(window, text="Verbalize \nNumber", command=verbalize)
b.grid(row=2, column=0, sticky=N)
window.mainloop()
```