List Solutions

13

Solution to Exercise 104: Sorted Order

```
##
#  Display the integers entered by the user in ascending order.
#

# Start with an empty list
data = []

# Read values, adding them to the list, until the user enters 0
num = int(input("Enter an integer (0 to quit): "))
while num != 0:
  data.append(num)
  num = int(input("Enter an integer (0 to quit): "))

# Sort the values
data.sort()
```

> Invoking the `sort` method on a list rearranges the elements in the list into sorted order. Using the `sort` method is appropriate for this problem because there is no need to keep a copy of the original list. The `sorted` function can be used to create a new copy of the list where the elements are in sorted order. Calling the `sorted` function does not modify the original list. As a result, it can be used in situations where the original list and the sorted list are needed simultaneously.

```
# Display the values in ascending order
print("The values, sorted into ascending order, are:")
for num in data:
  print(num)
```


B. Stephenson, *The Python Workbook*, DOI 10.1007/978-3-319-14240-1_13

Solution to Exercise 106: Remove Outliers

```
##
# Remove the outliers from a data set.
#

## Remove the outliers from a list of data
# @param data the list of data values to process
# @param num_outliers the number of smallest and largest values to remove
# @return a new copy of data where the values are sorted into ascending
#       order and the smallest and largest values have been removed
def removeOutliers(data, num_outliers):
  # Create a new copy of the list that is in sorted order
  retval = sorted(data)

  # Remove num_outliers largest values
  for i in range(num_outliers):
    retval.pop()

  # Remove num_outliers smallest values
  for i in range(num_outliers):
    retval.pop(0)

  # Return the result
  return retval

# Read data from the user, and remove the two largest and two smallest values
def main():
  # Read values from the user until a blank line is entered
  values = []
  s = input("Enter a value (blank line to quit): ")
  while s != "":
    num = float(s)
    values.append(num)
    s = input("Enter a value (blank line to quit): ")

  # Display the result or an appropriate error message
  if len(values) < 4:
    print("You didn't enter enough values.")
  else:
    print("With the outliers removed: ", removeOutliers(values, 2))
    print("The original data: ", values)

# Call the main function
main()
```

The smallest and largest outliers could be removed using the same loop. Two loops are used in this solution to make the steps more clear.

Solution to Exercise 107: Avoiding Duplicates

```
##
#  Read a collection of words entered by the user.  Display each word entered
#  by the user only once, in the same order that the words were entered.
#

# Begin reading words into a list
words = []
word = input("Enter a word (blank line to quit): ")
while word != "":
  # Only add the word to the list if
  # it is not already present in it
  if word not in words:
    words.append(word)

  # Read the next word from the user
  word = input("Enter a word (blank line to quit): ")

# Display the unique words
for word in words:
  print(word)
```

The expression `word not in words` is equivalent to `not (word in words)`.

Solution to Exercise 108: Negatives, Zeros and Positives

```
##
#  Read a collection of integers from the user.  Display all of the negative numbers,
#  followed by all of the zeros, followed by all of the positive numbers.
#

# Create three lists to store the negative, zero and
# positive values
negatives = []
zeros = []
positives = []

# Read all of the integers from the user, storing each
# integer in the correct list
line = input("Enter an integer (blank to quit): ")
while line != "":
  num = int(line)

  if num < 0:
    negatives.append(num)
  elif num > 0:
    positives.append(num)
  else:
    zeros.append(num)

  # Read the next line of input from the user
  line = input("Enter an integer (blank to quit): ")
```

This solution uses a list to keep track of the zeros that are entered. However, because all of the zeros are the same, it isn't actually necessary to save them. Instead, one could use an integer variable to count the number of zeros and then display that many zeros later in the program.

```
# Display all of the negative values, then all of the zeros, then all of the positive values
print("The numbers were: ")

for n in negatives:
  print(n)

for n in zeros:
  print(n)

for n in positives:
  print(n)
```

Solution to Exercise 110: Perfect Numbers

```
##
# A number, n, is a perfect number if the sum of the proper divisors of n is equal
# to n. This program displays all of the perfect numbers between 1 and LIMIT.
#
from proper_divisors import properDivisors

LIMIT = 10000

## Determine whether or not a number is perfect. A number is perfect if the
# sum of its proper divisors is equal to the number itself.
# @param n the number to check for perfection
# @return True if the number is perfect, False otherwise
def isPerfect(n):
  # Get a list of the proper divisors of n
  divisors = properDivisors(n)

  # Compute the total of all of the divisors
  total = 0
  for d in divisors:
    total = total + d

  # Return the appropriate result
  if total == n:
    return True
  return False

# Display all of the perfect numbers between 1 and LIMIT
def main():
  print("The perfect numbers between 1 and", LIMIT, "are:")
  for i in range(1, LIMIT + 1):
    if isPerfect(i):
      print(" ", i)

# Call the main function
main()
```

The total could also be computed using the `sum` function. This would reduce the calculation of the total to a single line.

Solution to Exercise 113: Formatting a List

```
##
#  Display a list of items so that they are separated by commas and the word
#  ''and'' appears between the final two items.
#

## Format a list of items so that they are separated by commas and ''and''
# @param items the list of items to format
# @return a string containing the items with the desired formatting
#
def formatList(items):
  # Handle lists of 0 and 1 items as special cases
  if len(items) == 0:
    return "<empty>"
  if len(items) == 1:
    return str(items[0])

  # Loop over all of the items in the list except the last two
  result = ""
  for i in range(0, len(items) - 2):
    result = result + str(items[i]) + ", "
```

Each item is explicitly cast to a string by calling the `str` function before it is added to the result. This allows `formatList` to format lists that contain numbers in addition to strings.

```
  # Add the second last and last items to the result, separated by ''and''
  result = result + str(items[len(items) - 2]) + " and "
  result = result + str(items[len(items) - 1])

  # Return the result
  return result

##
#  Read several items entered by the user and display them with nice formatting.
#
def main():
  # Read items from the user until a blank line is entered
  items = []
  line = input("Enter an item (blank to quit): ")
  while line != "":
    items.append(line)
    line = input("Enter an item (blank to quit): ")

  # Format and display the items
  print("The items are %s." % formatList(items))

# Call the main function
main()
```

Solution to Exercise 114: Random Lottery Numbers

```
##
# Compute random but distinct numbers for a lottery ticket.
#
from random import randrange

MIN_NUM = 1
MAX_NUM = 49
NUM_NUMS = 6

# Keep a list of the numbers for the ticket
ticket_nums = []

# Generate NUM_NUMS random but distinct numbers
for i in range(NUM_NUMS):
  # Generate a number that isn't already on the ticket
  rand = randrange(MIN_NUM, MAX_NUM + 1)
  while rand in ticket_nums:
    rand = randrange(MIN_NUM, MAX_NUM + 1)

  # Add the distinct number to the ticket
  ticket_nums.append(rand)

# Sort the numbers into ascending order and display them
ticket_nums.sort()
print("Your numbers are: ", end="")
for n in ticket_nums:
  print(n, end=" ")
print()
```

Using constants makes it easy to recongifure our program for other lotteries.

Solution to Exercise 118: Shuffling a Deck of Cards

```
##
# Create a deck of cards and shuffle it.
#
from random import randrange

# Construct a standard deck of cards with 4 suits and 13 values per suit
# @return a list of cards, with each card represented by two characters
def createDeck():
  # Create a list to store the cards in
  cards = []

  # For each suit and each value
  for suit in ["s", "h", "d", "c"]:
    for value in ["2", "3", "4", "5", "6", "7", "8", "9", \
                  "T", "J", "Q", "K", "A"]:
      # Construct the card and add it to the list
      cards.append(value + suit)
```

```
  # Return the complete deck of cards
  return cards

# Shuffle a deck of cards, modifying the deck of cards passed as a parameter
# @param cards the list of cards to shuffle
def shuffle(cards):
  # For each card
  for i in range(0, len(cards)):
    # Pick a random index
    other_pos = randrange(0, len(cards))

    # Swap the current card with the one at the random position
    temp = cards[i]
    cards[i] = cards[other_pos]
    cards[other_pos] = temp

# Display a deck of cards before and after it has been shuffled
def main():
  cards = createDeck()
  print("The original deck of cards is: ")
  print(cards)
  print()

  shuffle(cards)
  print("The shuffled deck of cards is: ")
  print(cards)

# Call the main program only if this file has not been imported
if __name__ == "__main__":
  main()
```

Solution to Exercise 121: Count the Elements

```
##
#  Count the number of elements in a list that are greater than or equal
#  to some minimum value and less than some maximum value.
#

# Determine how many elements in data are greater than or equal to mn and less than mx.
# @param data the list to process
# @param mn the minimum acceptable value
# @param mx the exclusive upper bound on acceptability
# @return the number of elements, e, such that mn <= e < mx
def countRange(data, mn, mx):
  # Count the number of elements within the acceptable range
  count = 0
  for e in data:
    # Check each element
    if mn <= e and e < mx:
      count = count + 1

  # Return the result
  return count
```

```
# Demonstrate the countRange function
def main():
  data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

  # Test a case where some elements are within the range
  print("Counting the elements in [1..10] that are between 5 and 7...")
  print("Result: %d  Expected: 2" % countRange(data, 5, 7))

  # Test a case where all elements are within the range
  print("Counting the elements in [1..10] that are between -5 and 77...")
  print("Result: %d  Expected: 10" % countRange(data, -5, 77))

  # Test a case where no elements are within the range
  print("Counting the elements in [1..10] that are between 12 and 17...")
  print("Result: %d  Expected: 0" % countRange(data, 12, 17))

  # Test a case where the list is empty
  print("Counting the elements in [] that are between 0 and 100...")
  print("Result: %d  Expected: 0" % countRange([], 0, 100))

  # Test a case with duplicate values
  data = [1, 2, 3, 4, 1, 2, 3, 4]
  print("Counting the elements in [1, 2, 3, 4, 1, 2, 3, 4] that are", \
        "between 2 and 4...")
  print("Result: %d  Expected: 4" % countRange(data, 2, 4))

# Call the main program
main()
```

Solution to Exercise 122: Tokenizing a String

```
##
#  Tokenize a string containing a mathematical expression.
#

# Convert a mathematical expression into a list of tokens
# @param s the string to tokenize
# @return a list of the tokens in s, or an empty list if an error occurs
def tokenList(s) :
  # Remove all of the spaces from s
  s = s.replace(" ", "")
```

```
    # Loop through all of the characters in the string,
    # identifying the tokens and adding them to the list.
    tokens = []
    i = 0
    while i < len(s):
      # Handle the tokens that are always a single character: *, /, ^, ( and )
      if s[i] == "*" or s[i] == "/" or s[i] == "^" or \
         s[i] == "(" or s[i] == ")":
        tokens.append(s[i])
        i = i + 1

      # Handle + and -
      elif s[i] == "+" or s[i] == "-":
        # If there is a previous character and it is a number or close bracket
        # then the + or - is a token on its own
        if i > 0 and (s[i-1] >= "0" and s[i-1] <= "9" or s[i-1] == ")"):
          tokens.append(s[i])
          i = i + 1
        else:
          # The + or - is part of a number
          num = s[i]
          i = i + 1

          # Keep on adding characters to the token as long as they are digits
          while i < len(s) and s[i] >= "0" and s[i] <= "9":
            num = num + s[i]
            i = i + 1
          tokens.append(num)

      # Handle a number without a leading + or -
      elif s[i] >= "0" and s[i] <= "9":
        num = ""
        # Keep on adding characters to the token as long as they are digits
        while i < len(s) and s[i] >= "0" and s[i] <= "9":
          num = num + s[i]
          i = i + 1
        tokens.append(num)

      # Any other character means the expression is not valid.
      # Return an empty list to indicate that an error ocurred.
      else:
        return []
    return tokens

# Read an expression from the user and tokenize it, displaying the result
def main():
    exp = input("Enter a mathematical expression: ")
    tokens = tokenList(exp)
    print("The tokens are:", tokens)

# Call the main function only if the file hasn't been imported
if __name__ == "__main__":
    main()
```

Solution to Exercise 126: Generate All Sublists of a List

```
##
# Compute all sublists of a list
#

## Generate a list of of all of the sublists of a list
# @param data the list for which the sublists are generated
# @return a list containing all of the sublists of data
def allSublists(data):
  # Start out with the empty list as the only sublist of data
  sublists = [[]]

  # Generate all of the sublists of data from length 1 to len(data)
  for length in range(1, len(data) + 1):
    # Generate the sublists starting at each index
    for i in range(0, len(data) - length + 1):
      # Add the current sublist to the list of sublists
      sublists.append(data[i : i + length])

  # Return the result
  return sublists

# Demonstrate the allSublists function
def main():
  print("The sublists of [] are: ")
  print(allSublists([]))

  print("The sublists of [1] are: ")
  print(allSublists([1]))

  print("The sublists of [1, 2] are: ")
  print(allSublists([1, 2]))

  print("The sublists of [1, 2, 3] are: ")
  print(allSublists([1, 2, 3]))

  print("The sublists of [1, 2, 3, 4] are: ")
  print(allSublists([1, 2, 3, 4]))

# Call the main function
main()
```

A list containing an empty list is denoted by `[[]]`.

Solution to Exercise 127: The Sieve of Eratosthenes

```
##
# Determine all of the prime numbers from 2 to some limit entered
# by the user using the Sieve of Eratosthenes.
#

# Read the limit from the user
limit = int(input("Generate all primes up to what limit? "))
```

```
# Generate all of the numbers from 0 to limit
nums = []
for i in range(0, limit + 1):
  nums.append(i)

# ''Cross out'' 1 by replacing it with a 0
nums[1] = 0

# Cross out all of the multiples of each prime number that we discover
p = 2
while p < limit:
  # Cross out all multiples of p (but not p itself)
  for i in range(p*2, limit + 1, p):
    nums[i] = 0

  # Find the next number that is not crossed out
  p = p + 1
  while p < limit and nums[p] == 0:
    p = p + 1

# Display the result
print("The primes up to", limit, "are:")
for i in nums:
  if nums[i] != 0:
    print(i)
```