Function Solutions

12

Solution to Exercise 84: Median of Three Values

```
##
# Compute and display the median of three values entered by the user.  This
# program includes two implementations of the median function that
# demonstrate different techniques for computing the median of three values.
#

## Compute the median of three values using if statements
# @param a the first value
# @param b the second value
# @param c the third value
# @return the median of values a, b and c
#
def median(a, b, c):
  if a < b and b < c or a > b and b > c:
    return b
  if b < a and a < c or b > a and a > c:
    return a
  if c < a and b < c or c > a and b > c:
    return c

## Compute the median of three values using the min and max functions
# and a little bit of arithmetic
# @param a the first value
# @param b the second value
# @param c the third value
# @return the median of values a, b and c
#
def alternateMedian(a, b, c):
  return a + b + c - min(a, b, c) - max(a, b, c)
```

Each function that you write should begin with a comment. Lines beginning with `@param` are used to describe the function's parameters. The value returned by the function is describe by a line that begins with `@return`.

The median of three values is the sum of the values, less the smallest, less the largest.

B. Stephenson, *The Python Workbook*, DOI 10.1007/978-3-319-14240-1_12

```
# Display the median of 3 values entered by the user
def main():
  x = float(input("Enter the first value: "))
  y = float(input("Enter the second value: "))
  z = float(input("Enter the third value: "))

  print("The median value is:", median(x, y, z))
  print("Using the alternative method, it is:", alternateMedian(x, y, z))

# Call the main function
main()
```

Solution to Exercise 86: The Twelve days of Christmas

```
##
# Generate the complete lyrics for the song The Twelve Days of Christmas.
#
from int_ordinal import intToOrdinal
```

The function that was written for the previous exercise is imported into this program so that the code for converting from an integer to its ordinal number does not have to be duplicated here.

```
## Generate and display one verse of The Twelve Days of Christmas
# @param n the verse to generate
# @return (none)
def displayVerse(n):
  print("One the", intToOrdinal(n), "day of Christmas")
  print("my true love sent to me:")

  if n >= 12:
    print("Twelve drummers drumming,")
  if n >= 11:
    print("Eleven pipers piping,")
  if n >= 10:
    print("Ten lords a leaping,")
  if n >= 9:
    print("Nine ladies dancing,")
  if n >= 8:
    print("Eight maids a milking,")
  if n >= 7:
    print("Seven swans a swimming,")
  if n >= 6:
    print("Six geese a laying,")
  if n >= 5:
    print("Five golden rings,")
  if n >= 4:
    print("Four calling birds,")
  if n >= 3:
    print("Three French hens,")
```

```
  if n >= 2:
      print("Two turtle doves,")
  if n == 1:
    print("A", end=" ")
  else:
    print("And a", end=" ")
  print("partridge in a pear tree.")
  print()

# Display all 12 verses of the song
def main():
  for verse in range(1, 13):
    displayVerse(verse)

# Call the main function
main()
```

Solution to Exercise 87: Center a String in the Terminal

```
##
# Center a string of characters within a certain width.
#
WIDTH = 80

## Create a new string that will be centered within a given width when it is printed.
# @param s the string that will be centered
# @param width the width in which the string will be centered
# @return a new copy of s that contains the leading spaces needed so that
#       s will appear centered when it is printed.
def center(s, width):
  # If the string is too long to center, then the original string is returned
  if width < len(s):
    return s

  # Compute the number of spaces needed and generate the result
  spaces = (width - len(s)) // 2
  result = " " * spaces + s

  return result

# Demonstrate the center function
def main():
  print(center("A Famous Story", WIDTH))
  print(center("by:", WIDTH))
  print(center("Someone Famous", WIDTH))
  print()
  print("Once upon a time...")

# Call the main function
main()
```

The // operator is used so that the result of the division will be truncated to an integer. This is necessary because a string can only be replicated an integer number of times.

Solution to Exercise 89: Capitalize it

```
##
# Improve the capitalization of a string by replacing " i " with " I " and by
# capitalizing the first letter of each sentence.
#

## Capitalize the appropriate characters in a string
#  @param s the string that needs capitalization
#  @return a new string with the capitalization improved
def capitalize(s):
  # Correct the capitalization for i
  result = s.replace(" i ", " I ")

  # Capitalize the first character of the string
  if len(s) > 0:
    result = result[0].upper() + \
             result[1 : len(result)]

  # Capitalize the first letter that follows a ".", "!" or "?"
  pos = 0
  while pos < len(s):
    if result[pos] == "." or result[pos] == "!" or result[pos] == "?":
      # Move past the ".", "!" or "?"
      pos = pos + 1

      # Move past any spaces
      while pos < len(s) and result[pos] == " ":
        pos = pos + 1

      # If we haven't reached the end of the string then replace
      # the current character with its uppercase equivalent
      if pos < len(s):
        result = result[0 : pos] + \
                 result[pos].upper() + \
                 result[pos + 1 : len(result)]

    # Move to the next character
    pos = pos + 1

  return result

# Demonstrate the capitalize function
def main():
  s = input("Enter some text: ")
  capitalized = capitalize(s)
  print("It is capitalized as:", capitalized)

# Call the main function
main()
```

The `replace` method replaces all occurences of its first parameter with its second parameter in the string on which it is invoked.

Using a colon inside of square brackets retrieves a portion of a string. The characters that are retrieved start at the position that appears to the left of the colon, going up to (but not including) the position that appears to the right of the colon.

Solution to Exercise 90: Does a String Represent an Integer?

```
##
# Determine whether or not a string entered by the user is an integer.
#

## Determine if a string contains a valid representation of an integer
# @param s the string to check
# @return True if s represents an integer. False otherwise.
#
def isInteger(s):
  # Remove whitespace from the beginning and end of the string
  s = s.strip()

  # Determine if the remaining characters form a valid integer
  if (s[0] == "+" or s[0] == "-") and s[1:].isdigit():
    return True
  if s.isdigit():
    return True
  return False

# Demonstrate the isInteger function
def main():
  s = input("Enter a string: ")
  if isInteger(s):
    print("That string represents an integer.")
  else:
    print("That string does not represent an integer.")

# Only call the main function when this file has not been imported
if __name__ == "__main__":
  main()
```

The `isdigit` method returns true if and only if the string is at least one character in length and all of the characters in the string are digits.

The `__name__` variable is automatically assigned a value by Python when the program starts running. It contains `"__main__"` when the file is executed directly by Python. It contains the name of the module when the file is imported into another program.

Solution to Exercise 92: Is a Number Prime?

```
##
# Determine if a number entered by the user is prime.
#

## Determine whether or not a number is prime
# @param n the integer to test
# @return True if the number is prime, False otherwise
def isPrime(n):
  if n <= 1:
    return False
```

```
  # Check each number from 2 up to but not including n to see if it divides evenly into n
  for i in range(2, n):
    if n % i == 0:
      return False
  return True

# Determine if a number entered by the user is prime
def main():
  value = int(input("Enter an integer: "))
  if isPrime(value):
    print(value,"is prime.")
  else:
    print(value,"is not prime.")

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

If `n % i == 0` then `n` is evenly divisible by `i`, indicating that `n` is not prime.

Solution to Exercise 94: Random Password

```
##
# Generate and display a random password containing between 7 and 10 characters.
#
from random import randint

SHORTEST = 7
LONGEST = 10
MIN_ASCII = 33
MAX_ASCII = 126

## Generate a random password
# @return a string containing a random password
def randomPassword():
  # Select a random length for the password
  randomLength = randint(SHORTEST, LONGEST)

  # Generate an appropriate number of random characters, adding each one to the end of result
  result = ""
  for i in range(randomLength):
    randomChar = chr(randint(MIN_ASCII, MAX_ASCII))
    result = result + randomChar

  # Return the random password
  return result

# Generate and display a random password
def main():
  print("Your random password is:", randomPassword())

# Call the main function only if the module is not imported
if __name__ == "__main__":
  main()
```

The `chr` function takes an ASCII code as its parameter. It returns a string containing the character with that ASCII code as its result.

Solution to Exercise 96: Check a Password

```
##
# Check whether or not a password is good.
#

## Check whether or not a password is good. A good password is at least 8 characters
# long and contains an uppercase letter, a lowercase letter and a number.
# @param password the password to check
# @return True if the password is good, False otherwise
def checkPassword(password):
  has_upper = False
  has_lower = False
  has_num = False

  # Check each character in the password and see which requirement it meets
  for ch in password:
    if ch >= "A" and ch <= "Z":
      has_upper = True
    elif ch >= "a" and ch <= "z":
      has_lower = True
    elif ch >= "0" and ch <= "9":
      has_num = True

  # If the password has all 4 properties
  if len(password) >= 8 and has_upper and has_lower and has_num:
    return True

  # The password is missing at least on property
  return False

# Demonstrate the password checking function
def main():
  p = input("Enter a password: ")
  if checkPassword(p):
    print("That's a good password.")
  else:
    print("That isn't a good password.")

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

Solution to Exercise 99: Arbitrary Base Conversions

```
##
# Convert a number from one base to another. Both the source base and the
# destination base must be between 2 and 16.
#
from hex_digit import *
```

The `hex_digit` module contains the functions `hex2int` and `int2hex` which were developed while solving Exercise 98. Using `import *` imports all of the functions from that module.

```
## Convert a number from base 10 to base n
#  @param num the base 10 number to convert
#  @param new_base the base to convert to
#  @return the string of digits in new_base
def dec2n(num, new_base):
  # Generate the representation of num in base new_base, storing it in result
  result = ""
  q = num

  # Perform the body of the loop once
  r = q % new_base
  result = int2hex(r) + result
  q = q // new_base

  # Continue looping until q == 0
  while q > 0:
    r = q % new_base
    result = int2hex(r) + result
    q = q // new_base

  # Return the result
  return result
```

```
## Convert a number from base b to base 10
#  @param num the base b number, stored in a string
#  @param b the base of the number to convert
#  @return the base 10 number
def n2dec(num, b):
  decimal = 0
  power = 0

  # Process each digit in the base b number
  for i in range(len(num)):
    decimal = decimal * b
    decimal = decimal + hex2int(num[i])

  # Return the result
  return decimal
```

The base *b* number must be stored in a string because it may contain letters that represent digits in bases larger than 10.

```
# Convert a number between two arbitrary bases
def main():
  # Read the number from the user
  from_base = int(input("Enter the base to convert from: "))
  from_num = input("Enter a sequence of digits in that base: ")
```

```
# Convert to base 10 and display the result
dec = n2dec(from_num, from_base)
print("That's %d in base 10." % dec)

# Convert to a new base and display the result
to_base = int(input("Enter the base to convert to: "))
to_num = dec2n(dec, to_base)
print("That's %s in base %d." % (to_num, to_base))
```

Solution to Exercise 101: Reduce a Fraction to Lowest Terms

```
##
# Reduce a fraction to its lowest terms.
#

## Compute the greatest common divisor of two integers.
# @param n the first integer under consideration (must be non-zero)
# @param m the second integer under consideration (must be non-zero)
# @return the greatest common divisor of the integers
def gcd(n, m):
  # Initialize d to the smaller of n and m
  d = min(n, m)

  # Use a while loop to find the greatest common divisor of n and m
  while n % d != 0 or m % d != 0:
    d = d - 1

  return d
```

This function used a loop to achieve its goal. There is also an elegant solution for finding the greatest common divisor of two integers that uses recursion. The recursive solution is explored in a later exercise.

```
## Reduce a fraction to lowest terms.
# @param the integer numerator of the fraction
# @param the integer denominator of the fraction (must be non-zero)
# @return the numerator and denominator of the reduced fraction
def reduce(num, den):
  # If the numerator is 0 then the reduced fraction is 0 / 1
  if num == 0:
    return (0, 1)

  # Compute the greatest common divisor of the numerator and denominator
  g = gcd(num, den)
```

```
  # Divide both the numerator and denominator
  # by the gcd and return the result
  return (num // g, den // g)

# Read the fraction from the user and display the reduced fraction
def main():
  # Read the input from the user
  num = int(input("Enter the numerator: "))
  den = int(input("Enter the denominator: "))

  # Compute the reduced fraction
  (n, d) = reduce(num, den)

  # Display the result
  print("%d/%d can be reduced to %d/%d." % (num, den, n, d))

# Call the main function
main()
```

We have used the integer division operator, `//`, so that we return a result like `(1, 3)` instead of `(1.0, 3.0)`.

Solution to Exercise 102: Reduce Measures

```
##
# Reduce an imperial measurement so that it is expressed using the largest possible units of
# measure. For example, 59 teaspoons is reduced to 1 cup, 3 tablespoons, 2 teaspoons.
#
TSP_PER_TBSP = 3
TSP_PER_CUP = 48

## Reduce an imperial measurement so that it is expressed using the largest possible
# units of measure.
# @param num the number of units that need to be reduced
# @param unit the unit of measure (cup, tablespoon or teaspoon)
# @return a string representing the measurement in reduced form
def reduceMeasure(num, unit):
  # Compute the number of teaspoons that the parameters represent
  unit = unit.lower()
```

The unit is converted to lowercase by invoking the `lower` method on `unit` and storing the result into the same variable. This allows the user to use any mixture of uppercase and lowercase letters when specifying the unit.

```
  if unit == "teaspoon" or unit == "teaspoons":
    teaspoons = num
  elif unit == "tablespoon" or unit == "tablespoons":
    teaspoons = num * TSP_PER_TBSP
  elif unit == "cup" or unit == "cups":
    teaspoons = num * TSP_PER_CUP
```

```
# Convert the number of teaspoons to the largest possible units of measure
cups = teaspoons // TSP_PER_CUP
teaspoons = teaspoons - cups * TSP_PER_CUP
tablespoons = teaspoons // TSP_PER_TBSP
teaspoons = teaspoons - tablespoons * TSP_PER_TBSP

# Generate the result string
result = ""

# Add the number of cups to the result string (if any)
if cups > 0:
  result = result + str(cups) + " cup"
  # Make cup plural if there is more than one
  if cups > 1:
    result = result + "s"

# Add the number of tablespoons to the result string (if any)
if tablespoons > 0:
  # Include a comma if there were some cups
  if result != "":
    result = result + ", "

  result = result + str(tablespoons) + " tablespoon"
  # Make tablespoon plural if there is more than one
  if tablespoons > 1:
    result = result + "s"

# Add the number of teaspoons to the result string (if any)
if teaspoons > 0:
  # Include a comma if there were some cups and/or tablespoons
  if result != "":
    result = result + ", "

  result = result + str(teaspoons) + " teaspoon"
  # Make teaspoons plural if there is more than one
  if teaspoons > 1:
    result = result + "s"

# Handle the case where the number of units was 0
if result == "":
  result = "0 teaspoons"


return result
```

Several test cases are included in this program. They exercise a variety of different combinations of zero, one and multiple occurences of the different units of measure. While these test cases are reasonably thorough, they do not guarantee that the program is bug free.

```
# Demonstrate the reduceMeasure function by performing several reductions
def main():
  print("59 teaspoons is %s." % reduceMeasure(59, "teaspoons"))
  print("59 tablespoons is %s." % reduceMeasure(59, "tablespoons"))
  print("1 teaspoon is %s." % reduceMeasure(1, "teaspoon"))
  print("1 tablespoon is %s." % reduceMeasure(1, "tablespoon"))
  print("1 cup is %s." % reduceMeasure(1, "cup"))
  print("4 cups is %s." % reduceMeasure(4, "cups"))
  print("3 teaspoons is %s." % reduceMeasure(3, "teaspoons"))
  print("6 teaspoons is %s." % reduceMeasure(6, "teaspoons"))
  print("95 teaspoons is %s." % reduceMeasure(95, "teaspoons"))
  print("96 teaspoons is %s." % reduceMeasure(96, "teaspoons"))
  print("97 teaspoons is %s." % reduceMeasure(97, "teaspoons"))
  print("98 teaspoons is %s." % reduceMeasure(98, "teaspoons"))
  print("99 teaspoons is %s." % reduceMeasure(99, "teaspoons"))

# Call the main function
main()
```

Solution to Exercise 103: Magic Dates

```
##
# Determine all of the magic dates in the 1900s
#
from days_in_month import daysInMonth

## Determine whether or not a date is ''magic''
# @param day the day portion of the date
# @param month the month portion of the date
# @param year the year portion of the date
# @return True if the date is magic, False otherwise
def isMagicDate(day, month, year):
  if day * month == year % 100:
    return True

  return False

# Find and display all of the magic dates in the 1900s
def main():
  for year in range(1900, 2000):
    for month in range(1, 13):
      for day in range(1, daysInMonth(month, year) + 1):
        if isMagicDate(day, month, year):
          print("%02d/%02d/%04d is a magic date." % (day, month, year))

# Call the main function
main()
```

The expression `year % 100` evaluates to the two digit year.