Loop Solutions

11

Solution to Exercise 64: No more Pennies

```
##
#  Compute the total due when several items are purchased.  The amount
#  payable for cash transactions is rounded to the closest nickel because
#  pennies have been phased out in Canada.
#
PENNIES_PER_NICKEL = 5
NICKEL = 0.05
```

> While it is highly unlikely that the number of pennies in a nickel will ever change, it is possible (even likely) that we will need to update our program at some point in the future so that it rounds to the closest dime. Using constants will make it easier to perform that update when it is needed.

```
# Track the total of all the items
total = 0.00

# Read the price of the first item as a string
line = input("Enter the price of the item (blank to quit): ")

# Continue reading items until a blank line is entered
while line != "":
  # Add the cost of the item to the total (after converting it to a float)
  total = total + float(line)

  # Read the cost of the next item
  line = input("Enter the price of the item (blank to quit): ")

# Display the exact total payable
print("The exact amount payable is %.02f" % total)

# Compute the number of pennies that would be left if the total was paid
# only using nickels
rounding_indicator = total * 100 % PENNIES_PER_NICKEL
```


B. Stephenson, *The Python Workbook*, DOI 10.1007/978-3-319-14240-1_11

```
if rounding_indicator < PENNIES_PER_NICKEL / 2:
  # If the number of pennies left is less than 2.5 then we round down by
  # subtracting that number of pennies from the total
  cash_total = total - rounding_indicator / 100
else:
  # Otherwise we add a nickel and then subtract the number of pennies
  cash_total = total + NICKEL - rounding_indicator / 100

# Display the cash amount payable
print("The cash amount payable is %.02f" % cash_total)
```

Solution to Exercise 65: Computer the Perimeter of a Polygon

```
##
# Compute the perimeter of a polygon. The user will enter a blank line
# for the x-coordinate to indicate that all of the points have been entered.
#
from math import sqrt

# Store the perimeter of the polygon
perimeter = 0

# Read the coordinate of the first point
first_x = float(input("Enter the x part of the coordiante: "))
first_y = float(input("Enter the y part of the coordinate: "))

# Provide initial values for prev_x and prev_y
prev_x = first_x
prev_y = first_y

# Read the remaining coordinates
line = input("Enter the x part of the coordinate (blank to quit): ")
while line != "":
  # Convert the x part to a number and read the y part
  x = float(line)
  y = float(input("Enter the y part of the coordinate: "))

  # Compute the distance to the previous point and add it to the perimeter
  dist = sqrt((prev_x - x) ** 2 + (prev_y - y) ** 2)
  perimeter = perimeter + dist

  # Set up prev_x and prev_y for the next loop iteration
  prev_x = x
  prev_y = y

  # Read the x part of the next coordinate
  line = input("Enter the x part of the coordinate (blank to quit): ")

# Compute the distance from the last point to the first point and add it to the perimeter
dist = sqrt((first_x - x) ** 2 + (first_y - y) ** 2)
perimeter = perimeter + dist
```

The distance between the points is computed using Pythagorean theorem.

```
# Display the result
print("The perimeter of that polygon is", perimeter)
```

Solution to Exercise 67: Admission Price

```
##
#  Compute the admission price for a group visiting the zoo.
#

# Store the admission prices as constants
BABY_PRICE = 0.00
CHILD_PRICE = 14.00
ADULT_PRICE = 23.00
SENIOR_PRICE = 18.00

# Store the age limits as constants
BABY_LIMIT = 2
CHILD_LIMIT = 12
ADULT_LIMIT = 64

# Create a variable to hold the total admission cost for all guests
total = 0

# Keep on reading ages until the user enters a blank line
line = input("Enter the age of the guest (blank to finish): ")
while line != "":
  age = int(line)

  # Add the correct amount to the total
  if age <= BABY_LIMIT:
    total = total + BABY_PRICE
  elif age <= CHILD_LIMIT:
    total = total + CHILD_PRICE
  elif age <= ADULT_LIMIT:
    total = total + ADULT_PRICE
  else:
    total = total + SENIOR_PRICE

  # Read the next line from the user
  line = input("Enter the age of the guest (blank to finish): ")

# Display the total due for the group, formatted using two decimal places
print("The total for that group is $%.2f" % total)
```

The first condition is not necessary with the current admission prices. However, including it makes it easy to start charging for babbies in the future.

Solution to Exercise 68: Parity Bits

```
##
#  Compute even parity for sets of 8 bits entered by the user.
#
```

```
# Read the first line of input
line = input("Enter 8 bits: ")

# Continue looping until a blank line is entered
while line != "":
  # Ensure that the line has a total of 8 zeros and ones and exactly 8 characters
  if line.count("0") + line.count("1") != 8 or len(line) != 8:
    # Display an appropriate error message
    print("That wasn't 8 bits... Try again.")
  else:
    # Count the number of ones
    ones = line.count("1")

    # Display the parity bit
    if ones % 2 == 0:
      print("The parity bit should be 0.")
    else:
      print("The parity bit should be 1.")

  # Read the next line of input
  line = input("Enter 8 bits: ")
```

The `count` method returns the number of times that the string provided as a parameter occurs in the string on which the method is invoked.

Solution to Exercise 70: Caesar Cipher

```
##
#  Implement a Caesar cipher that shifts all of the letters in a message by an amount
#  provided by the user.  Use a negative shift value to decode a message.
#

# Read the message and shift amount from the user
message = input("Enter the message: ")
shift = int(input("Enter the shift value: "))

# Process each character, constructing a new message
new_message = ""
for ch in message:
  if ch >= "a" and ch <= "z":
    # Process a lowercase letter by determining its
    # position in the alphabet (0 - 25), computing its
    # new position, and adding it to the new message
    pos = ord(ch) - ord("a")
    pos = (pos + shift) % 26
    new_char = chr(pos + ord("a"))
    new_message = new_message + new_char
  elif ch >= "A" and ch <= "Z":
    # Process an uppercase letter by determining its position in the alphabet
    # (0 - 25), computing its new position, and adding it to the new message
    pos = ord(ch) - ord("A")
    pos = (pos + shift) % 26
    new_char = chr(pos + ord("A"))
    new_message = new_message + new_char
```

The `ord` function converts a character to its integer position within the ASCII table. The `chr` function returns the character from the ASCII table in the position provided.

```
  else:
    # If the character is not a letter then copy it into the new message
    new_message = new_message + ch

# Display the shifted message
print("The shifted message is", new_message)
```

Solution to Exercise 72: Is a String a Palindrome?

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

# Read the input from the user
line = input("Enter a string: ")

# Assume that it is a palindrome until we can prove otherwise
is_palindrome = True

# Check the characters, starting from the ends until
# the middle is reached
for i in range(0, len(line) // 2):
  # If the characters don't match then mark
  # that the string is not a palindrome
  if line[i] != line[len(line) - i - 1]:
    is_palindrome = False

# Display a meaningful output message
if is_palindrome:
  print(line, "is a palindrome")
else:
  print(line, "is not a palindrome")
```

All of the parameters to the range function must be integers. The // operator is used so that the result of the division is truncated to an integer.

Solution to Exercise 74: Multiplication Table

```
##
# Display a multiplication table for 1 times 1 through 10 times 10.
#
MIN = 1
MAX = 10

# Display the top row of labels
print("    ", end="")
for i in range(MIN, MAX + 1):
  print("%4d" % i, end="")
print()
```

Including `end=""` as the final parameter to print prevents it from moving down to the next line after displaying the value.

```
# Display the table
for i in range(MIN, MAX + 1):
  print("%4d" % i, end="")
  for j in range(MIN, MAX + 1):
    print("%4d" % (i * j), end="")
  print()
```

Solution to Exercise 75: Greatest Common Divisor

```
##
#  Compute the greatest common divisor of two positive integers using a while loop.
#

# Read two positive integers from the user
n = int(input("Enter a positive integer: "))
m = int(input("Enter a positive integer: "))

# 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

# Report the result
print("The greatest common divisor of", n, "and", m, "is", d)
```

Solution to Exercise 78: Decimal to Binary

```
##
#  Convert a number from Decimal (base 10) to Binary (base 2)
#
NEW_BASE = 2

# Read the number to convert from the user
num = int(input("Enter a non-negative integer: "))

# Generate the binary representation of num,
# storing it in result
result = ""
q = num

# Perform the body of the loop once
r = q % NEW_BASE
result = str(r) + result
q = q // NEW_BASE
```

The algorithm provided for this question is expressed using a repeat-until loop. However, this structure isn't available in Python. As a result, the algorithm has to be adjusted so that it generates the same result using a `while` loop. This is achieved by duplicating the loop body and placing it ahead of the `while` loop.

```
# Keep on looping until q == 0
while q > 0:
  r = q % NEW_BASE
  result = str(r) + result
  q = q // NEW_BASE

# Display the result
print(num, "in Decimal is", result, "in Binary.")
```

Solution to Exercise 79: Maximum Integer

```
##
#  Find the maximum of 100 random integers, counting the number of times the
#  maximum value is updated during the process
#
from random import randrange

NUM_ITEMS = 100

# Generate the first number and display it
mx_value = randrange(1, NUM_ITEMS + 1)
print(mx_value)

# Count of the number of updates
num_updates = 0

# For each of the remaining numbers
for i in range(1, NUM_ITEMS):
  # Generate a new random number
  current = randrange(1, NUM_ITEMS + 1)

  # If the generated number is the largest one we have seen so far
  if current > mx_value:
    # Update the maximum and count the update
    mx_value = current
    num_updates = num_updates + 1
    # Display the number, noting that an update occurred
    print(current, "<== Update")
  else:
    # Display the number
    print(current)

# Display the summary results
print("The maximum value found was", mx_value)
print("The maximum value was updated", num_updates, "times")
```