SOLUTION CP1200 – Practical 8

Aims

1. Write and test programs using lists

Lists

1. Finishing/Testing Tutorial Problem

1.1 Storing A List of User-Supplied Integers

SEE TUTORIAL SOLUTION.

2. Problems For You To Solve

2.1. "Quick Pick" Lottery Ticket Generator **Version 2**

Write a program that asks the user how many "quick picks" they wish to generate. The program then generates that many lines of output. Each line of outputs consists of six random numbers between 1 and 45. **Each line should not contain any repeated number. Each line of numbers should be displayed in ascending order. (Look at the Part 1.1 and 2.1 in the tutorial.)**

Try to match the sample output:

```
How many quick picks? 5
 9 10 11 32 38 44
 2  9 12 14 28 30
 5 10 16 22 35 42
 1  2 16 22 37 40
 1 17 20 23 25 27
```

SOLUTION

```
import random

NUMBERS_PER_LINE = 6
MINIMUM = 1
MAXIMUM = 45

if NUMBERS_PER_LINE < (MAXIMUM - MINIMUM):
    quickPicks = int(input("How many quick picks? "))
    while quickPicks < 0:
        print("That makes no sense!")
        quickPicks = int(input("How many quick picks? "))

    for i in range(quickPicks):
        drawnNumbers = []
        for i in range(NUMBERS_PER_LINE):
            number = random.randint(MINIMUM, MAXIMUM)
            # number must not have been picked for this line
            while number in drawnNumbers:
                number = random.randint(MINIMUM, MAXIMUM)
            drawnNumbers.append(number)

        drawnNumbers.sort()
        for number in drawnNumbers:
            print(format(number, "2d"), end=' ')
        print()
```

2.2 Lists And Files

A. Write a function called **loadListFromFile** that takes a filename as a parameter, opens the file for reading and returns the list of lines as strings, with the newline characters removed. The function should open and close the file correctly.
E.g. **loadListFromFile**("jobs.txt") would return a list ["programmer", "designer", "butler"] from a file like

```
programmer
designer
butler
```

SOLUTION

```
def loadListFromFile(filename):
    result = []
    fileIn = open(filename, 'r')
    for line in fileIn:
        result.append(line.rstrip('\n'))
    fileIn.close()
    return result
```

B. Write a function called **loadFloatListFromFile** that takes a filename as a parameter. The function will assume that the file consists of some floating-point numbers, all on their own line. Return a list of those numbers (note: as **floats**, not strings - so [1.1, 2.2, 3.3] instead of ["1.1", "2.2", "3.3"]). How can **loadFloatListFromFile** make good use of **loadListFromFile**?
E.g. **loadListFromFile**("numbers.txt") would return a list [1.1, 2.2, 3.3] from a file like

```
1.1
2.2
3.3
```

SOLUTION

```
def loadFloatListFromFile(filename):
    result = []
    fileIn = open(filename, 'r')
    for line in fileIn:
        number = float(line)
        result.append(number)
    fileIn.close()
    return result
```

C. Write a function called **saveListToFile** that takes a filename and a list as parameters. The function should open the file for writing, and write each element in the list to the file as a string. Each element from the list should be on a separate line in the file.

E.g. **saveListToFile**(“data.txt”, data), where data = [“Harry”, 1980, “phoenix feather”] would produce “data.txt” as

```
Harry
1980
phoenix feather
```

SOLUTION

```
def saveListToFile(filename, data):
    fileOut = open(filename, 'w')
    for datum in data:
        fileOut.write(datum + '\n')
    fileOut.close()
```

Extension Work

1. Memberwise Addition

   In Python, the + operator concatenates lists, so [1, 2, 3] + [4, 5, 6] = [1, 2, 3, 4, 5, 6]. What if we want to add the elements together instead?

   Write a function, **`memberwiseAddition`**, that takes two lists, and returns the list that contains the sum of elements that are in the same index in the two lists. For example

   - memberwiseAddition([1, 2, 3], [4, 5, 6]) would return [5, 7, 9]
   - memberwiseAddition([1, 2, 3], [1, 2, 3, 4]) would return [2, 4, 6, 4]

SOLUTION

This is an interesting question, with multiple possible solutions.

You need to create a new list, so that implies starting with a blank list and creating it by adding the elements of the other two lists, but you could also start with one list as the result and add the other list elements to it.

Clearly you will need to use a loop – with definite iteration (for loop) – over each element in the list(s).

Part of the challenge is what to do with lists that are of different lengths. You don’t want to access elements that don’t exist.

Although the question doesn’t say so, we also need to avoid unwanted aliasing where adding the lists might modify one of the lists – it should just return a new list, leaving the others untouched.

So, these thoughts led us to the following option:

- check which list is longer and which is shorter, and create aliases so we can do it in one loop no matter which list is which.
- set the result list to a deep copy (by using slicing) of the longer list – this way, the elements that go beyond the length of the shorter list are already in place
- then just loop through the shorter list, adding its elements on to the values in the new result list.

```
def memberwiseAddition(list1, list2):
    if len(list1) > len(list2):
        longer = list1 # aliasing - OK
        shorter = list2
    else:   # it doesn't matter if they're the same length
        longer = list2
        shorter = list1
    result = longer[:] # slice (deep copy) to avoid unwanted aliasing
    for i in range(len(shorter)):
        result[i] += longer[i]
    return result

print(memberwiseAddition([1, 2, 3], [1, 2, 3, 4]))
```

2. Extend the first quick question so that the user can enter any number of numbers until a number less than zero is entered. Adjust the prompt so that it prints like "Number 1: " then "Number 2: "

See Python file

3. Write a program that asks the user for an indefinite set of strings (until an empty string is entered), then print all of the strings that were repeated.
   E.g. if the user entered: "hello", "world is good", "hello", "john", "good"… the program would print "Strings repeated: hello".
   If no repeated strings are entered, the program should print "No repeated strings entered".

See Python file