CS1010E-PE1-2019 - PE1

Programming Methodology (National University of Singapore)

CS1010E Practical Exam I

Grading Guidelines:

- No mark will be given *if your code cannot run*, namely any syntax error or crashes.
  - Your tutors will just grade what you submitted and they will **not** fix your code.
  - Hint: Comment out any part that you do not want.
- Workable code that can produce correct answers will only give you *partial* marks. Only good and efficient code will give you full marks
- Marks will be deducted if your code is unnecessarily long or hard-coded.
- You must use the same function names as those in the skeleton given
- You ***cannot import*** any additional packages or functions. For **Problem 4**, you cannot even use any built-in functions other than "`len()`" and "`range()`".
- Of course, other operations/operators such as =, "`if`", "`for`", "`while`", +,*, /, or slicing, etc. are allowed.

Problem Dependency

There are four problems in this PE.

- Problems 1, 2 and 4 are independent.
- However, **Problem 3** is depending on either 1 or 2. You may find it difficult if you start with Problem 3 without solving Problems 1 and 2 first. Also, when you are solving Problem 3, you cannot "assume" the code for Problem 1 or 2 are provided.

Problem 1 Sum of Digit Squares (Iterative) (10 marks)

Given a non-negative integer n, your task is to **sum the square of the digits** (**SDS**) of n. For example, if n = 2324,

$$SDS(2324) = 2^2+3^2+2^2+4^2 = 33$$

Your Task

Write an ***iterative*** version of the function `sumDigitSqrVerI(n)`. You can assume that your input is any non-negative integers. Your outputs have to be integers. Here are some sample output:

```
>>> sumDigitSqrVerI(123456)
91

>>> sumDigitSqrVerI(987654321)
285

>>> sumDigitSqrVerI(99998888777766665555444433332222111)
1140
```

Problem 2 Sum of Digit Squares (Recursive) (10 marks)

Your Task

Write a ***recursive*** version of the function `sumDigitSqrVerR(n)` with the same functionality in Problem 1.

Problem 3 Happy Numbers (35+5 marks)

Let n = 7, the sum of digit squares (SDS) of n is 49. And the SDS of 49 is 97, and the SDS of 97 is 130, and repeat…

$$7 \xrightarrow{SDS} 49 \xrightarrow{SDS} 97 \xrightarrow{SDS} 130 \xrightarrow{SDS} 10 \xrightarrow{SDS} 1$$

We call a number n a ***happy number**** if it becomes 1 after some iteration(s)! The number 836 is a happy number and 930 is not because,

$$836 \xrightarrow{SDS} 109 \xrightarrow{SDS} 82 \xrightarrow{SDS} 68 \xrightarrow{SDS} 100 \xrightarrow{SDS} 1$$

$$930 \xrightarrow{SDS} 90 \xrightarrow{SDS} 81 \xrightarrow{SDS} 65 \xrightarrow{SDS} 61 \xrightarrow{SDS} 37 \xrightarrow{SDS} 58 \xrightarrow{SDS} 89 \xrightarrow{SDS} 145 \xrightarrow{SDS} 42 \xrightarrow{SDS} 20 \xrightarrow{SDS} 2$$

Some facts about the happy numbers:

- For any number n for $0 \le n < 10$, only 1 and 7 are happy numbers.
- Therefore, the numbers 2, 3, 4, 5, 6, 8 and 9 will never reach 1 (or 7) no matter how many times you apply SDS onto any one of them.
- Also, you can assume that, for any number $n > 9$, the cycle will produce a number that is smaller than 10 eventually if you keep applying SDS repeatedly.

Your Task (35 marks)

Write a function `isHappyNumber(n)` to check if n is a happy number. Return `True` if n is a happy number and `False` otherwise. You can assume the number n is greater than 0. Here are some sample outputs:

```
>>> print(isHappyNumber(83))
False

>>> print(isHappyNumber(849))
False

>>> print(isHappyNumber(10888))
True
```

```
>>> print(isHappyNumber(100093))
True

>>> print(isHappyNumber(97931**10000))
False

>>> print(isHappyNumber(97931**10000+1))
True
```

*The *happy number* is a **real** definition in mathematics (not something we made up!). There are other definitions like *Harshad numbers*, which means "numbers with great joy" in Sanskrit (a classical Indian language).

Bonus (5 marks)

Write a function `allHappyNumbers(m,n)` to return a list of all happy numbers between m and n inclusively. You can get this bonus ***only if*** your answer of `isHappyNumber()` is completely correct. And your list must be sorted in ascending order. Some sample output:

```
>>> print(allHappyNumbers(1,70))
[1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70]

>>> print(allHappyNumbers(123**5,123**5+20))
[28153056846, 28153056847, 28153056853, 28153056857]
```

Problem 4 Checking if All Elements are Unique in a Sequence (35+5 marks)

In this problem, you are **NOT allowed** to use:

- Any built-in functions or imported packages. Namely, you cannot use any `copy()`, `count()`, `sort()`, `sorted()`, `set()`, `list()`, `tuple()`, `str()` and other built-in or package functions.
- Any **`set`** functionalities in Python. (In any word, creating or using any sets.)

However, there is an exception that you are **allowed** to use the functions "**`len()`**" and "**`range()`**", or define extra function(s) by yourself for your own use, including calling `isElementUnique()` *if* you use recursion.

Your Task (35 marks)

Give a sequence **`seq`**, namely, a string, a list or a tuple (excluding dictionaries and sets), write a function `isElementUnique(seq)` to check if all the elements inside **`seq`** are ***unique*** (no duplicate). Here are some sample outputs:

```
>>> print(isElementUnique('minions'))
False

>>> print(isElementUnique('abcdefghijklmnopqrstuvwxyz'))
True

>>> print(isElementUnique([1,2,3,4,5,6,7]))
True

>>> print(isElementUnique(['a', 'b', 3 , True, 999, 'a']))
False

>>> print(isElementUnique((1, 2, 999, 4, 0, 6, (1,2,), 999)))
False

>>> print(isElementUnique(['aaa', 'bbb', (1,1), 1]))
True
```

However, if the input sequence is a list or a tuple, you only need to do a ***shallow*** check. Namely, you do *not* need to check if any elements **within** ONE element in the sequence are unique or not, see the last line of the above sample outputs for an example. The second last line outputs a False because of 999, not because of 1, 2 and (1,2,).

Bonus (5 marks)

If you can implement the `isElementUnique(seq)` correctly without ANY other built-in functions or packages, namely, even without using "`len()`" nor "`range()`", you can get this 5 marks bonus.

Template and Test Cases

(Copy and paste all the followings into ONE .py file. Replace all the "**pass**"'s with your own answers and add your own indentations and newlines.)

```
# Problems 1 and 2 Function Definitions
def sumDigitSqrVerI(n): pass
def sumDigitSqrVerR(n): pass

# Test cases for SDS
print(sumDigitSqrVerR(1230045697900))
print(sumDigitSqrVerI(1230045697900))
print(sumDigitSqrVerR(0))

# Problems 3 Function Definition
def isHappyNumber(n): pass

# Test cases for isHappyNumber()
print(isHappyNumber(836)) # True
print(isHappyNumber(930)) # False

#This following case is True
print(isHappyNumber(59465934639563464389643685439107410594659346395634643896436872 4))

#This following case is False
print(isHappyNumber(59465934639563464389643685439107410594659346395634643896436872 3))

#Function Definition for allHappyNumber
def allHappyNumbers(m,n): pass

# Test case fo allHappyNumber() and the sample output should be:
# [10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
print(allHappyNumbers(10,100))

# Problems 4 Function Definition
def isElementUnique(seq): pass

# Test cases for isElementUnique (), please refer the sample outputs above.
lst1 = [1,2,3,4,5,6,7]
tup1 = (1,2,3,4,5,6,7)
lst2 = ['a','b',3, True, 999]
lst3 = ['a','b',3, True, 999, 'a']
str1 = 'abcdefg'
str2 = 'abdsjljlj'

print(isElementUnique(lst1))
print(isElementUnique(tup1))
print(isElementUnique(lst2))
print(isElementUnique(lst3))
print(isElementUnique(str1))
print(isElementUnique(str2))
```