CS1010E: Programming Methodology

PE1 (2019/2020 Sem 1)

Files

- `PE1.pdf`
- `Question1.py`
- `Question2.py`

Coursemology

- `Past PE1 > CS1010E 2019/20 Sem 1`

Questions

1. Happy Numbers
   - 1.1 Iterative SDS [*10 marks*]
   - 1.2 Recursive SDS [*10 marks*]
   - 1.3 Single Happy Numbers [*35 marks*]
   - 1.4 All Happy Numbers [*5 marks*]
2. Unique Sequence
   - 2.1 Simple Unique Sequence [*35 marks*]
   - 2.2 Complex Unique Sequence [*5 marks*]

Question 1: Happy Numbers

Believe it or not, Happy Number is a *real* definition in mathematics and not something we made up. This should not be confused with Harshad Number which means "numbers with great joy" in Sanskrit.

A number n is a Happy Number if it will eventually become 1 when we replace n with the sum of the square of each digit. For instance, 7 is a Happy Number because:

$$
\begin{aligned}
& 7 \\
\Rightarrow\ & 7^2 = 49 \\
\Rightarrow\ & 4^2 + 9^2 = 16 + 81 = 97 \\
\Rightarrow\ & 9^2 + 7^2 = 81 + 49 = 130 \\
\Rightarrow\ & 1^2 + 3^2 + 0^2 = 1 + 9 + 0 = 10 \\
\Rightarrow\ & 1^2 + 0^2 = 1
\end{aligned}
$$

Let us denote $\xrightarrow{SDS}$ as the operation to sum the square of each digit. Then, 836 is a Happy Number but 930 is not a happy number because:

- $836 \xrightarrow{SDS} 109 \xrightarrow{SDS} 82 \xrightarrow{SDS} 68 \xrightarrow{SDS} 100 \xrightarrow{SDS} 1$
- $930 \xrightarrow{SDS} 90 \xrightarrow{SDS} 81 \xrightarrow{SDS} 65 \xrightarrow{SDS} 61 \xrightarrow{SDS} 37 \xrightarrow{SDS} 58 \xrightarrow{SDS} 89 \xrightarrow{SDS} 145 \xrightarrow{SDS} 42 \xrightarrow{SDS} 20 \xrightarrow{SDS} 4 \xrightarrow{SDS} 16 \xrightarrow{SDS} 37$

Note how in the case of 930, the number 37 appears twice. Since the process is *deterministic*, if we start with 37, we will always eventually reach 37 again without reaching 1. Therefore, we will never reach 1.

General Restrictions

In this question, you MUST work with integers (`int`) and are NOT allowed to change the input number into any other data types including (*but not limited to*) float (`float`), string (`str`), tuple (`tuple`) or list (`list`).

continue on the next page...

1.1 Iterative SDS [10 marks]

Question

Write the *iterative* function `sum_digit_square_I(n)` that takes in a positive integer (`int`) `n` and returns a positive integer (`int`) corresponding to the sum of the square of digits of `n`.

Restrictions

- You may not use recursive function(s) to solve this.

Assumptions

- `n > 0`

Sample Run #1

```
>>> sum_digit_square_I(123456)
91
```

Sample Run #2

```
>>> sum_digit_square_I(987654321)
285
```

Sample Run #3

```
>>> sum_digit_square_I(99998888777766665555444433332222 1111)
1140
```

1.2 Recursive SDS [10 marks]

Question

Write the *recursive* function `sum_digit_square_R(n)` that takes in a positive integer (`int`) `n` and returns a positive integer (`int`) corresponding to the sum of the square of digits of `n`.

Restrictions

- You may not use iterative constructs (*e.g.*, loop, list comprehensions, *etc.*) to solve this.
- The function `sum_digit_square_R` must be *recursive* (*i.e.*, it calls itself). The use of any recursive helper functions will not be counted as being recursive.

Assumptions

- `n > 0`

continue on the next page...

Sample Run #1

```
>>> sum_digit_square_R(123456)
91
```

Sample Run #2

```
>>> sum_digit_square_R(987654321)
285
```

Sample Run #3

```
>>> sum_digit_square_R(99998888777766665555444433332222 1111)
1140
```

1.3 Single Happy Numbers [35 marks]

Before you start, there are some facts about Happy Numbers that may make your computation easier:

- For any number n such that $0 \leq n < 10$, there are only two Happy Numbers namely 1 and 7.
  - In other words, 2, 3, 4, 5, 6, 8 and 8 are NOT happy numbers as they will never reach 1 no matter how many times you apply SDS.
- For any number n such that $n \geq 10$, the cycle will always produce a number that is smaller than 10 eventually after some finite number of application of SDS.

Question

Write the function `is_happy_number(n)` that takes in a positive integer (`int`) `n` and returns a Boolean (`bool`) `True` if the number `n` is a Happy Number and `False` otherwise.

Assumptions

- `n > 0`

Assumptions

- If you encounter infinite loop, press `CTRL + c` to stop the execution of your code on IDLE.
- We limit the execution on Coursemology to 2 seconds per test cases.

Sample Run #1

```
>>> is_happy_number(83)
False
```

continue on the next page...

Sample Run #2

```
>>> is_happy_number(849)
False
```

Sample Run #3

```
>>> is_happy_number(10888)
True
```

Sample Run #4

```
>>> is_happy_number(100093)
True
```

1.4 All Happy Numbers [5 marks]

Question

Write the function `all_happy_number(n,m)` that takes in two positive integers (`int`) `n` and `m`. The function returns a list (`list`) of all Happy Numbers between `n` (*inclusive*) and `m` (*inclusive*). Your list must be sorted in ascending order.

Assumptions

- `n > 0`

Note

- If you encounter infinite loop, press `CTRL + c` to stop the execution of your code on IDLE.
- We limit the execution on Coursemology to 2 seconds per test cases.
- You will only get the mark if your answer to Question 1.3 is completely correct.

Sample Run #1

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

Question 2: Unique Sequence

Consider a sequence which may be a string (`str`), a list (`list`) or a tuple (`tuple`). We want to check if the sequence contains a duplicate. A duplicate on the sequence `seq` is defined as two elements on two different index `i` and `j` such that `seq[i] == seq[j]` and `i != j`. In other words, two elements on two different index being equal.

The sequence may contain another sequence. For instance, we may have `[1, 2, [1], 3]` where the list `[1]` is nested inside the outer list. For simplicity, the checking only need to be a *shallow* checking. In other words, we do not need to check if the element `1` from the outer list is equal to the element inside `[1]`. In this case, all the elements inside `[1, 2, [1], 3]` are unique.

General Restrictions

You are NOT allowed to use:

- Any built-in functions or imported packages. This includes (*but not limited to*): `copy`, `count`, `sort`, `sorted`, `set`, `list`, `tuple`, `str`.
- Any `set` functionalities in Python. You are NOT even allowed to create `set` and/or `dict`.
- The `in` operator, except if it appears in a loop (*e.g.*, `for x in e:`).

Two ***exceptions***: you are allowed to use the function `len` and `range`. As usual, you are allowed (*and even encouraged*) to define any additional functions you may need.

2.1 Simple Unique Sequence [35 marks]

Question

Write a function `is_unique(seq)` that accepts a sequence `seq` that can be one of three types above and returns a Boolean (`bool`) `True` if all the elements inside `seq` are *unique* (*i.e.*, no duplicate). Otherwise, the function returns `False`.

Assumptions

- `seq` will be either `str`, `tuple` or `list`

Sample Run #1

```
>>> is_unique('minions')
False
```

Note: `'i'` and `'n'` are duplicated.

Sample Run #2

```
>>> is_unique('abcdefghijklmnopqrstuvwxyz')
True
```

Sample Run #3

```
>>> is_unique([1,2,3,4,5,6,7,8])
True
```

Sample Run #4

```
>>> is_unique(['a', 'b', 3, True, 999, 'a'])
False
```

Note: `'a'` is duplicated.

Sample Run #5

```
>>> is_unique((1, 2, 999, 4, 0, 6, (1, 2), 999))
False
```

Note: `999` is duplicated. Neither `1` nor `2` causes the output to be `False`.

Sample Run #6

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

Note: `1` is not duplicated and we do not need to check the duplicate `1` inside `(1,1)`.

2.2 Complex Unique Sequence [5 marks]

What we are going to do is we are going to complicate this a little by not allowing both `len` and `range`. You will get this mark automatically if your code for Question 2.1 is correct AND without `len` and `range`.

– End of Paper –