Ministry of Education, Singapore
Computing Teachers’ Content Upgrading Course 2020

Practical Assessment 1 - Retest

20 Mar 2020

Time allowed: 3 hours

Instructions to candidates:

1. This is an **open-book** exam.
2. Answer all **five** questions.
3. You may complete your solutions in any IDE first before copying them into this Jupyter Notebook for submission.
4. Input validation is not required
5. Submit this Jupyter Notebook online before test ends. You may submit multiple times, but only the last submission before test end time will be accepted. https://driveuploader.com/upload/JDtXaQiUmX/ (https://driveuploader.com/upload/JDtXaQiUmX/)
6. Please note that the sample test cases may not be enough to test your program. Your programs will be tested with other inputs and they should exhibit the required behaviours to get full credit.

Name & Email

- Rename your jupyter notebook to "YourName" using menu `File > Rename`
- Enter your name and your email address in following cell

```
# YOUR NAME

# YOUR EMAIL
```

Question 1 (4 marks)

Implement a function `cumulative_sum(s)` where `"s"` a list of integers. It returns a list of its cumulative sum.

For example, when `s = [1,2,3,4,5]` , the function returns `[1,3,6,10,15]`

In [6]:
```
# WRITE YOUR CODE HERE
```

Test Case 1

- Input: `[1,2,3,4,5]`
- Expected output: `[1,3,6,10,15]`

In [7]:
```
s = cumulative_sum([1,2,3,4,5])
print(s)
```

```
[1, 3, 6, 10, 15]
```

Test Case 2

- Input: `[1,1,1,2,2,2]`
- Expected output: `[1,2,3,5,7,9]`

In [4]:
```
s = cumulative_sum([1,1,1,2,2,2])
print(s)
```

```
[1, 2, 3, 5, 7, 9]
```

Test Case 3

- Input: `[]`
- Expected output: `[]`

In [2]:
```
s = cumulative_sum([])
print(s)
```

```
[]
```

Question 2 (4 marks)

Implement a function `count_lengths(s)` which takes in a list of strings. It counts the length of each string in the list. It return a dictionary with string length as key and string count as value.

For example, the return value of `{3:4, 2:5, 6:2}` indicates there are 4 strings with length 3, 5 strings with length 2 and 2 strings with length 6.

In [5]:
```
# WRITE YOUR CODE HERE
```

Test Case 1

- Input: `['come', 'find', 'for', 'funny', 'go']`
- Expected output: `{4: 2, 3: 1, 5: 1, 2: 1}`
- This is because there are 2 strings of 4 characters, 1 string of 3 characters, 1 string of 5 characters, and 1 string of 2 characters.

In [6]:
```
s = count_lengths(['come', 'find', 'for', 'funny', 'go'])
print(s)
```

```
{4: 2, 3: 1, 5: 1, 2: 1}
```

Test Case 2

- Input: `['help', 'here', 'I']`
- Expected output: `{4: 2, 1: 1}`
- This is because there are 2 strings of 4 characters, and 1 string of 1 character.

In [7]:
```
s = count_lengths(['help', 'here', 'I'])
print(s)
```

```
{4: 2, 1: 1}
```

Test Case 3

- Input: `[]`
- Expected output: `{}`

In [8]:
```
s = count_lengths([])
print(s)
```

```
{}
```

Question 3 (4 marks)

Implement a function `gen_random()` which generates **5** distinct integers between 0 and 9 (inclusive).

To generate an integer between 0 and 9 (inclusive), use code:

```
import random
x = random.randrange(10)
```

In [1]:
```
import random

# WRITE YOUR CODE HERE
```

Test Case 1

Following code should generates 5 distinct integers between 0 and 9 (inclusive). If the returned list is correct, no Exception will be raised.

In [2]:
```
s = gen_randoms()
print(s)
assert(len(set(s)) == 5)  # Check if s contains 5 distinct integer
assert(all([x >=0 and x <= 9 for x in s]))
```

```
[6, 9, 4, 1, 5]
```

Question 4 (4 marks)

This question is to implement a simple checksum for a String message. The steps involved to create checksum on a string `s` is:

- **Step 1:** For each character in the string, find its ASCII value.
- **Step 2:** Multiply each ASCII value by a weight:
  - If the character is between **`a`** to **`z`**, multiple its ASCII value by 5.
  - If the character is between **`A`** to **`Z`**, multiple its ASCII value by 10.
  - Otherwise, multiple the ASCII value by 15.
- **Step 3:** Sum up all above values
- **Step 4:** Divide the sum by 13 and get its remainder;
- **Step 5:** Subtract the remainder from 13 to get the result;

Implement a function `checksum_num(s)` which takes in a string `s`, and returns an integer as the check digit using the above method.

- To find ASCII code of a character, use `ord()` method.
  - For example, `ord('A')` will return 65, `ord('Z')` will return 90.

In [13]:
```
# WRITE YOUR CODE HERE
```

Test Case 1

Input: `'HeLLo2'`
Expected output: `13`

$$\begin{aligned}
\textbf{Steps 1 and 2 and 3:}\quad & 72 * 10 + 101 * 5 + 76 * 10 + 76 * 10 + 111 * 5 + 50 * 15 = 4050 \\
\textbf{Step 4:}\quad & 4050 \div 13 \text{ gives remainder } 7 \\
\textbf{Step 5:}\quad & 13 - 7 = 6
\end{aligned}$$

In [14]:

```python
result = checksum_num('HeLLo2')
print(result)
```

```
6
```

Test Case 2

Input: `'abc@gmail.com'`

Expected output: `7`

In [15]:

```python
result = checksum_num('abc@gmail.com')
print(result)
```

```
7
```

Question 5 (4 marks)

Strings returned from web service's calls are commonly in JSON format. In Python, JSON strings are commonly parsed into Dictionaries.

```python
d = {
  "a": 1,
  "b": { "b1": 2, "b2": 3},
  "c": { "c1": 1,
         "c2": { "d1": 2, "d2": 3 }
       },
  "e": { "e1": 1, "e2": 2 }
}
```

Considering above dictionary whose value can be either an integer or another dictionary.

Implement a recursive function `sum_vals(d)` which takes in such a dictionary and returns sum of all values in `d`.

Non-recursive functions will also be accepted but with a 2 marks penalty.

In [16]:

```python
# WRITE YOUR CODE HERE
```

Test Case 1

- Input: `{"a": 1, "b": {"b1": 2, "b2":3}, "c": {"c1":1, "c2": {"d1":2, "d2":3}}, "e": {"e1":1, "e2":2}}`
- Expected output: `15`

In [17]:
```
d = {"a": 1, "b": {"b1": 2, "b2":3}, "c": {"c1":1, "c2": {"d1":2, "d2":3}}, "e":
result = sum_vals(d)
print(result)
```

```
15
```

Test Case 2

- Input: `{"c": {"c1":1, "c2": {"d1":2, "d2":3, "e": {"e1":1, "e2":2}}}}`
- Expected output: `9`

In [18]:
```
d = {"c": {"c1":1, "c2": {"d1":2, "d2":3, "e": {"e1":1, "e2":2}}}}
result = sum_vals(d)
print(result)
```

```
9
```

Test Case 3

- Input: `{"c": {"d": {"e": {"e1":1, "e2":2}}}}`
- Expected output: `3`

In [19]:
```
d = {"c": {"d": {"e": {"e1":1, "e2":2}}}}
result = sum_vals(d)
print(result)
```

```
3
```