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

Practical Assessment 1

26 Feb 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 `get_mean_median()` which takes in a list of integers. It returns a tuple containing the mean value followed by the median value of the list.

- When the list has an odd number of integers, its median value is the middle value of the sorted list.
- When the list has an even number of integers, its median value is average of the middle 2 values.

You get an extra bonus mark if your function does not modify the input parameter list.

In [1]:
```
# WRITE YOUR CODE HERE

def get_mean_median(s):
    s = s.copy()
    s.sort()
    n = len(s)
    mean = sum(s) / n
    if n % 2 == 1:
        median = s[ n // 2]
    else:
        median = (s[ n // 2 - 1] + s[ n // 2 ]) / 2
    return mean, median
```

Test Case 1

Input: `[1, 3]`
Expected output: `(2.0, 2.0)`
Note that because there is an even number of integers, the median is the average of the middle 2 sorted values, 1 and 3.

In [2]:
```
s = [1, 3]
get_mean_median(s)
```

Out[2]: `(2.0, 2.0)`

Test Case 2

Input: `[999, 1, 2]`
Expected output: `(334.0, 2)`
Note that because there is an odd number of integers, the median is just the middle sorted value, 2.

In [3]:
```
s = [999, 1, 2]
get_mean_median(s)
```

Out[3]: `(334.0, 2)`

Test Case 3

Input: `[8, 6, 4, 5, 18, 7]`
Expected output: `(8.0, 6.5)`

In [4]:
```
s = [8, 6, 4, 5, 18, 7]
get_mean_median(s)
```

Out[4]: `(8.0, 6.5)`

Question 2 (4 marks)

Implement a function `count_divisibles(s)` which takes in a list of integers; it counts the number of integers in the list that can be divided by 2, 3 and 5 respectively.

The result is returned in a dictionary. For example, `{2: 5, 3: 2, 5: 0}` indicates that 5 numbers can be divided by 2, 2 numbers can be divided by 3, and no numbers can be divided by 5.

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

def count_divisibles(s):
    result = {2:0, 3:0, 5:0}
    for i in s:
        if i % 2 == 0:
            result[2] = result[2] + 1
        if i % 3 == 0:
            result[3] = result[3] + 1
        if i % 5 == 0:
            result[5] = result[5] +1
    return result
```

Test Case 1

Input: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
Expected output: `{2: 5, 3: 3, 5: 2}`
This is because in the list there are 5 integers that can be divided by 2 (`2` , `4` , `6` , `8` , `10`), 3 integers that can be divided by 3 (`3` , `6` , `9`) and 2 integers that can be divided by 5 (`5` , `10`).

In [6]:
```
count_divisibles([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

Out[6]:
```
{2: 5, 3: 3, 5: 2}
```

Test Case 2

Input: `[11, 13]`
Expected output: `{2: 0, 3: 0, 5: 0}`
This is because neither `11` nor `13` are divisible by 2, 3 or 5.

In [7]:
```
count_divisibles([11, 13])
```

Out[7]:
```
{2: 0, 3: 0, 5: 0}
```

Question 3 (4 marks)

Implement a function `string_list()` which takes in a string and returns a list containing its first character, its first 2 characters, its first 3 characters, and so on.

Hint: Use string slicing

In [8]:
```
# WRITE YOUR CODE HERE

def string_list(s):
    result = []
    for i in range(1,len(s)+1):
        result.append(s[0:i])
    return result
```

Test Case 1

Input: `'ABCD'`
Expected output: `['A', 'AB', 'ABC', 'ABCD']`
The output is a list of the first character (`A`), the first 2 characters (`AB`), the first 3 characters (`ABC`) and finally the first 4 characters (`ABCD`). It is not possible to get 5 characters from the input, so the list stops there.

In [9]:
```
string_list('ABCD')
```

Out[9]:
```
['A', 'AB', 'ABC', 'ABCD']
```

Test Case 2

Input: `''`
Expected output: `[]`
An empty string doesn't even have a "first character", so the output is just an empty list.

In [10]:
```
string_list('')
```

Out[10]:
```
[]
```

Test Case 3

Input: `'Good day'`
Expected output: `['G', 'Go', 'Goo', 'Good', 'Good ', 'Good d', 'Good da', 'Good day']`

In [11]:
```
string_list('Good day')
```

Out[11]:
```
['G', 'Go', 'Goo', 'Good', 'Good ', 'Good d', 'Good da', 'Good day']
```

Question 4 (4 marks)

The Singapore NRIC number is made up of 7 digits and a letter behind it, known as the check digit. This letter is calculated from the first 7 digits using the modulus eleven method.

The steps involved to obtain the check digit is:

- **Step 1:** Multiply each digit in the NRIC number by its respective weight `[2, 7, 6, 5, 4, 3, 2]`;
- **Step 2:** Sum up above products;
- **Step 3:** Divide the sum by 11 and get its remainder;
- **Step 4:** Subtract the remainder from 11 to get the result;
- **Step 5:** Obtain the letter from the result where `1 = 'A', 2 = 'B', 3 = 'C', 4 = 'D', 5 = 'E', 6 = 'F', 7 = 'G', 8 = 'H', 9 = 'I', 10 = 'Z', 11 = 'J'`

Implement a function `get_check_letter()` which takes in a string of 7 digits, and returns a letter as the check digit using the above method.

- Return `'Invalid'` if input is invalid

In [12]:
```
# THE FOLLOWING DICTIONARY IS PROVIDED FOR YOUR CONVENIENCE.
LETTERS = {
    1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E',
    6: 'F', 7: 'G', 8: 'H', 9: 'I', 10: 'Z', 11: 'J'
}

# WRITE YOUR CODE HERE

def get_check_letter(n):
    if len(n) != 7 or (not n.isdigit()):
        return 'Invalid'
    s = list(n)
    weights = [2, 7, 6, 5, 4, 3, 2]
    sum = 0
    for i in range(len(weights)):
        sum = sum + int(s[i]) * weights[i]
    r = sum % 11
    c = 11 - r
    return LETTERS[c]
```

Test Case 1

Input: `'1234567'`
Expected output: `'D'`

Steps 1 and 2: $2(1) + 7(2) + 6(3) + 5(4) + 4(5) + 3(6) + 2(7) = 106$
Step 3: $106 \div 11$ gives remainder 7
Step 4: $11 - 7 = 4$
Step 5: $4 = D$

In [13]:
```
get_check_letter('1234567')
```

Out[13]: `'D'`

Test Case 2

Input: `'123abc'`
Expected output: `'Invalid'`

In [14]:
```
get_check_letter('123abc')
```

Out[14]: `'Invalid'`

Question 5 (4 marks)

Up to 5 integers are contained in a list structure. The structure may contain any combination of nested lists as shown in following examples.

- `[1,2,3,4,5]`
- `[4,[1],[[3,99],5]]`
- `[[[3,2,1],6],5]`

Implement a recursive function `sum_flexi(s)` which returns the sum of all integers in `s`, where `s` is a nested list structure containing integer values.

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

In [15]:
```
# WRITE YOUR CODE HERE

def sum_flexi(s):
    if len(s) == 0:
        return 0
    x = s.pop()
    if type(x) == int:
        return x + sum_flexi(s)
    else:
        return sum_flexi(x) + sum_flexi(s)
```

Test Case 1

Input: `[4,[1],[[3,99],5]]`
Expected output: `112`

In [16]:
```
s = [4,[1],[[3,99],5]]
sum_flexi(s)
```

Out[16]: `112`

Test Case 2

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

In [17]:
```
s = [1,2,3,4,5]
sum_flexi(s)
```

Out[17]: 15

Test Case 3

Input: `[[[3,2,1],6],5]`

Expected output: `17`

In [18]:
```
s = [[[3,2,1],6],5]
sum_flexi(s)
```

Out[18]: 17