CSCI 2040: Introduction to Python **2024-2025 Term 1**

Lab Assignment 4

Instructor: Yim Pan, Chui Due: 23:59 on Wednesday, Nov. 20, 2024

Notes

1. You are allowed to form a group of two to do this lab assignment.
2. You are strongly recommended to bring your own laptop to the lab with Anaconda[1] and Pycharm[2] installed. You don't even have to attend the lab session if you know what you are required to do by reading this assignment.
3. Only **Python 3.x** is acceptable.
4. Passing the test scripts we have provided does not guarantee full marks for your question as our grade scripts will test for more cases.
5. Your code should only contain specified functions. Please delete all the debug statements (e.g. print) before submission.

Exercise 1 (30 marks)

(a) We want to get a RPN (Reverse Polish Notation) calculator by reusing the class `Stack, CalculatorEngine` as follows. Implement the `RPNCalculator` as the child class of `CalculatorEngine` (which has already been written for you). The new method `eval(line)` in `RPNCalculator` would return the evaluation result of the RPN expression in `line`. (*Hint: read the lecture notes*) **(20 marks)**

Remark: You may assume all inputs are valid, i.e., you do not need to consider scenarios like `'1␣2␣*␣*'` or `'1␣2␣3'`

For example, the expected value of x in the following is 5.

```
cal = RPNCalculator()
x = cal.eval('1␣2␣*␣3␣+')
```

(b) We now want to add a Modulo operator `%` to the RPN calculator. Do this by adding a new method `doModulo(self)` to the class `CalculatorEngine` and modifying other related code (*Hint: You may refer* ***`doDivision(self)`*** *to handle ZeroDivision exception in* ***`doModulo(self)`***). **(10 marks)**

```
class Stack(object):
    def __init__(self):
```

[1] An open data science platform powered by Python. `https://www.continuum.io/downloads`
[2] A powerful Python IDE. `https://www.jetbrains.com/pycharm/download/`

```
        self.storage = []

    def push(self, newValue):
        self.storage.append(newValue)

    def top(self):
        return self.storage[-1]

    def pop(self):
        result = self.top()
        self.storage.pop()
        return result

    def isEmpty(self):
        return len(self.storage) == 0


class CalculatorEngine(object):
    def __init__(self):
        self.dataStack = Stack()

    def pushOperand(self, value):
        self.dataStack.push(value)

    def currentOperand(self):
        return self.dataStack.top()

    def performBinaryOp(self, fun):
        right = self.dataStack.pop()
        left = self.dataStack.top()
        self.dataStack.push(fun(left, right))

    def doAddition(self):
        self.performBinaryOp(lambda x, y: x + y)

    def doSubtraction(self):
        self.performBinaryOp(lambda x, y: x - y)

    def doMultiplication(self):
        self.performBinaryOp(lambda x, y: x * y)
```

```
        def doDivision(self):
            try:
                self.performBinaryOp(lambda x, y: x / y)
            except ZeroDivisionError:
                print("division by 0!")
                exit(1)

        def doModulo(self):
            try:
                # your code here
            except # your code here
                print("divide by 0!")
                exit(1)

        def doTextOp(self, op):
            if (op == '+'):
                self.doAddition()
            elif (op == '-'):
                self.doSubtraction()
            elif (op == '*'):
                self.doMultiplication()
            elif (op == '/'):
                self.doDivision()


    class RPNCalculator(CalculatorEngine):
        def __init__(self):
            # your code here

        def eval(self, line):
            # your code here
```

Save your script for this exercise in `p1.py`

Exercise 2 (45 marks)

Create seven lists (**named `list1`, `list2`, `list3`, `list4`, `list5`, `list6`, `list7` respectively**) and one dictionary (**named `dict1`**) in the global scope as required below. Each of them should span **only one line of code** (`from functools import reduce` won't be counted).

(a) Using the function **`map` with a `lambda` argument**, write an expression that will produce each of the following from a list of values produced by `list0 = range(1, 12)`:

- A list of two to the power of the corresponding values in the `list0` (name it as `list1`). The expected output should be a list as follows, **(5 marks)**

  ```
  [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048]
  ```

- A list where each element is the modulo by three of the corresponding element in the `list0` (name it as `list2`); The expected output should be a list as follows, **(5 marks)**

  ```
  [1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2]
  ```

Note that you should use `lambda` arguments in this part and keep one line of code to generate `list1` and `list2`.

(b) Write a **list comprehension** that will produce each of the following from a list of values produced by `range(1, 20)`:

- A list contains values in the original list that are odd numbers (name it as `list3`); The expected output should be a list as follows: **(5 marks)**

  ```
  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
  ```

- A list contains values that are the square numbers (name it as `list4`). The expected output should be a list as follows, **(5 marks)**

  ```
  [1, 4, 9, 16]
  ```

(c) Write a **list comprehension** that will produce a list of values from a list produced by `range(-5, 6)` (name it as `list5`). The expected output is as follows: **(5 marks)**

```
[0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5]
```

(d) Using the function **`reduce` with a `lambda` argument**, write an expression to create a list (name it as `list6`) of factorial values for numbers from 1 to 11. Each element in the list should correspond to the factorial of the index +1 (i.e., the forth element to the `list6` is $24 = 1 \times 2 \times 3 \times 4$). The expected output should be a list as follows, **(5 marks)**

```
[1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800]
```

(e) Given dictionary `student_scores` which catalogs students and their test scores out of 100:

```
student_scores = {
    "Alice": 92,
    "Bob": 85,
```

```
    "Charlie": 77,
    "Diana": 88,
    "Ethan": 95,
    "Fiona": 55,
    "George": 68,
    "Hannah": 99,
}
```

Use **list comprehension** to construct a list of the names of students in BLOCK LETTERS whose scores exceed 80 (name it as `list7`). The expected output should be a list as follows: **(7 marks)**

```
['ALICE', 'BOB', 'DIANA', 'ETHAN', 'HANNAH']
```

(f) Use **list comprehension** and **dictionary comprehension** to construct a dictionary of multiplication tables (name it as `dict1`). The expected output should be a list as follows: **(8 marks)**

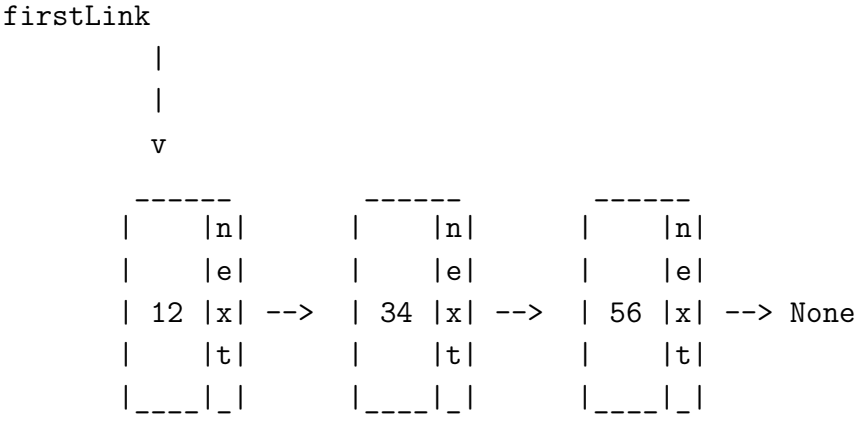
```
{
1: '1*1=1',
2: '1*2=2 2*2=4',
3: '1*3=3 2*3=6 3*3=9',
4: '1*4=4 2*4=8 3*4=12 4*4=16',
5: '1*5=5 2*5=10 3*5=15 4*5=20 5*5=25',
6: '1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36',
7: '1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49',
8: '1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64',
9: '1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81'
}
```

Hint:
i. You may need to use the ***`join`*** *method of* ***`str`*** *class. Each line of the table is the result of joining a list that contains the equations of this line. For example,* `'1*2=2 2*2=4'` *can be obtained from* `['1*2=2', '2*2=4']`.
ii. You can implement dictionary comprehension similar to list comprehension. For example, `{1:'1',2:'2',3:'3',4:'4'}` *can be obtained from* `{i:str(i) for i in range(1,5)}`.

Note that in this exercise, you are required to write **only one line of code for each expression**. We will manually check the correctness of your answer.
Save your script for this exercise in `p2.py`

Exercise 3 (25 marks)

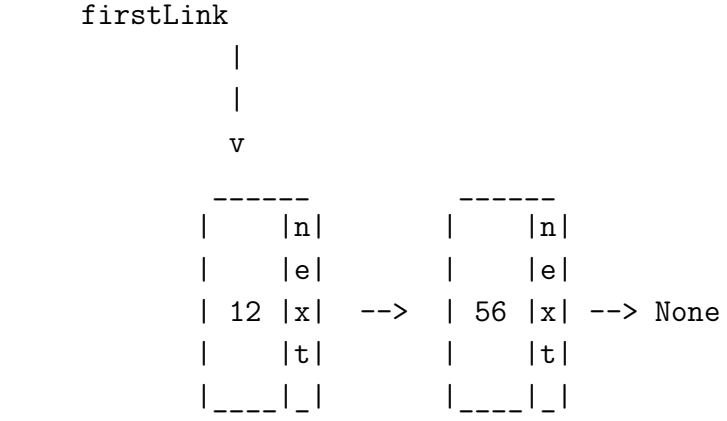
Linked List. A linked list could be developed using a pair of classes. The class `Node` is used to store an individual **linked node** (All values inside this linked list are assumed to be of integer type):



Complete the `LinkedList` class by implementing `test()`, `remove()`, `len()` and `Lprint()`.

(a) `test`: see if `value` is in link list. Return `True` if the value is in the linked list, otherwise return `False`. **(7 marks)**

(b) `remove`: remove the first occurence of `value` from linked list. Return `True` if the value is in the linked list and the node containing this value is removed; otherwise return `False`. For example, if you remove `34` from the above linked list, it will become:



Hint: Suppose the node you find that contains the value is ***target****. And if the node before and after it are* ***prev*** *and* ***succ*** *respectively, we have the typology as* ***prev -> target -> succ****. To remove* ***target****, you can connect* ***prev*** *and* ***succ*** *directly so that the resulting typology is* ***prev -> succ****. The arrow here is essentially abstracted by the* ***next*** *attribute of a node.* **(7 marks)**

(c) `len`: return the length of the linked list. The length is the number of nodes this linked list contains. **(6 marks)**

(d) `Lprint`: print the linked list. For example, for the linked list mentioned above, after removing `34`, the output is: **(5 marks)**

```
Current linked list: 12-->56-->none
```

```
class Node(object):
    def __init__(self, v, n):
        self.value = v
        self.next = n

class LinkedList(object):
    def __init__(self):
        self.first = None
        self.length = 0

    def add(self, value):
        # add will insert at the begining of the list
        self.first = Node(value, self.first)
        self.length = self.length + 1

    def test(self, value):
        tmp_node = self.first
        result = False
        # Iterate through nodes in the linked list
        while tmp_node:
            # Complete the function with the following hints:
            # If the current node's value matches the query value, set result
            # to True and break the loop.
            # Otherwise, move to the next node in the list.
        return result

    def remove(self, value):
        tmp_node = self.first
        prev_len = self.length
        prev_node = self.first
        # Iterate through nodes in the linked list
        while tmp_node:
            # Complete the function with the following hints:
            # Check if the current node's value matches the value to be removed.
            # If found and it is the first node, update the head of the list.
            # If found and it is not the first node, update the previous
            # node's next to skip the current node.
```

```
            # If the node is removed, decrement the length of the list.
        if self.length == prev_len:
            return False
        else:
            return True

    def len(self):
        # Complete the function with the following hint:
        # Return the current length of the linked list.

    def Lprint(self):
        print("Current linked list: ", end='')
        result = []
        tmp_node = self.first
        # Iterate through nodes in the linked list
        while tmp_node:
            # Complete the function with the following hints:
            # Append the current node's value to the result list.
            # Move to the next node.
        # Append "none" to indicate the end of the list.
        # Print the linked list values in the standard format via result list.
```

You can type the following sequences to test your program,

```
import p3
l = p3.LinkedList()
print('list length %d' % (l.len()))
l.add(20)
l.add(10)
l.add(15)
l.add(20)
l.Lprint()
print(l.test(10))
print(l.test(30))
l.add(30)
print('list length %d' % (l.len()))
l.Lprint()
l.remove(20)
l.Lprint()
```

should produce the following output

```
list length 0
Current linked list: 20-->15-->10-->20-->none
True
False
list length 5
Current linked list: 30-->20-->15-->10-->20-->none
Current linked list: 30-->15-->10-->20-->none
```

To ensure the correctness of your program, you can create more test cases. You can also use the test script to check the correctness of you code. When submitting the script file, **DO NOT** put the test code along with the class implementation.

Save your script for this exercise in `p3.py`

Submission rules

1. Please name the `functions` and script files with the **exact** names specified in this assignment and test all your scripts. Any script that has any wrong name or syntax error will not be marked.
2. For each group, please pack all your script files as a single archive named as
   `<student-id1>_<student-id2>_lab4.zip`
   For example, `1155012345_1155054321_lab4.zip`, i.e., just replace `<student-id1>` and `<student-id2>` with your own student IDs. If you are doing the assignment alone, just leave `<student-id2>` empty, e.g, `1155012345_lab4.zip`.
3. Upload the zip file to your blackboard (`https://blackboard.cuhk.edu.hk`),
   - Only one member of each group needs to upload the archive file.
   - Subject of your file should be `<student-id1>_<student-id2>_lab4` if you are in a two-person group or `<student-id1>_lab4` if not.
   - No later than 23:59 on Wednesday, Nov. 20, 2024
4. Students in the same group would get the same marks. Marks will be deducted if you do not follow the submission rules. Anyone/Any group caught with plagiarizing would get 0 score!