HW03: 24 Game
CSE2050 Fall 2017
Instructors: Jeffrey Meunier & Wei Wei
TAs: Jenny Blessing, Param Bidja, Yamuna Rajan & Zigeng Wang

1. Introduction

In this exercise, we will complete a Python class and additional functions to find all possible solutions for a given 4-card combination and print each solution in postfix notation.

2. Objectives

The purpose of this assignment is to give you experience in:

- Using mathematical calculations in Python.
- Implementing and using the basic operations of a stack.
- Using postfix notation and different ways of calculating mathematical expressions.

***Note**: Before you start, if you are not familiar with the basic operations of a stack data structure and concepts of functions and classes in Python, it is recommended that you review the material covered in lecture first.*

3. Background

3.1. 24 Game

In this game, 4 cards are randomly picked from a standard deck of 52 cards. We will use the values of these 4 cards and a combination of addition, subtraction, multiplication, division, and parentheses to produce the value 24. Note that each of the 4 cards can only be used once.

For the purposes of this assignment, we will consider the following to be the face values of each card in a suit:

"A": 1

"2": 2

"3": 3

"4": 4

"5": 5

"6": 6

"7": 7

"8": 8

"9": 9

"10": 10

"J": 11

"Q": 12

"K": 13

For example, if the 4 cards chosen are "A 2 3 Q," we can produce the value 24 by doing the following operation:

$$(3 - 2 + 1) * 12 = 24$$

Note that there are many possible solutions for a given 4-card combination. For instance the cards "A 2 3 Q" can produce 33 different solutions which produce 24. Read the following link for more information about the game: https://en.wikipedia.org/wiki/24_Game.

3.2. Postfix Notation

To solve this problem, you will also need to be familiar with **postfix notation**, which is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if "(exp_1) **op** (exp_2)" is a normal, fully parenthesized expression whose operation is **op**, the postfix version of this is "$pexp_1$ $pexp_2$ **op**," where $pexp_1$ is the postfix version of exp_1 and $pexp_2$ is the postfix version of exp_2. The postfix version of a single number or variable is just that number or variable.

For example, the postfix version of "((5 + 2) * (8 – 3)) / 4" is "5 2 + 8 3 - * 4 /".

3.3. Fractions in Python

Note that the very first line in the skeleton code imports the smaller module 'Fraction' from module 'fractions'. A module is simply a file containing Python definitions and statements.

To use the module when you are doing division, you will use the following statement:

```
Fraction(n1, n2)
```

which will return an instance of Fraction. If you want to know the value of $\frac{n1}{n2}$, you can use the value returned by Fraction(n1, n2).

4. Assignment

In the skeleton zip file, you will find a skeleton for your .py file named *twenty_four.py*. All you need to do is to complete the skeleton code based on the instructions and submit it to the Mimir system.

4.1. ListStack Implementation

Now, open the skeleton code. In this assignment, you will be using a stack to evaluate an expression in postfix notation. To do this, a ListStack class has already been created for you. What you need to do is implement each of the functions within the stack.

The implementations for each of these functions should be very short. If you're writing more than one or two lines for one of these functions, think conceptually about what it is the function should be doing.

4.2. evaluate() Function

The evaluate() function will take in a list containing a postfix expression and make use of the ListStack class to evaluate it. The stack variable has already been created for you.

In this function, as long as the size of the stack is not 0, you will pop the next element from the stack. If the element is an operation (i.e. is either '+', '-', '*', or '/') you will pop the next two elements from the stack, which will always be card values. Try playing around with different postfix notations to see why this is true. Evaluate the two numbers using the operation. If the operation is division and the denominator is 0, you can simply return 0 to the function and terminate the evaluation, because this means that it will not evaluate to 24 anyway.

If the element is a card value, then just push the value back onto the stack.

Once the function finishes running, the final remaining element on the stack will be the result of evaluating the postfix expression.

4.3. evaluate_all_ops() Function

This function will find all valid postfix notations for each 4 card combination and evaluate them to see if they are equal to 24. You will call your evaluate() function inside of this function to evaluate the expression. The function takes the same postfix expression s as an argument.

There are five different possible forms of a valid postfix notation for 4 cards, where 'C' denotes a card value and 'X' denotes an operation:

1. CCXCCXX
2. CCCCXXX
3. CCCXCXX
4. CCXCXCX
5. CCCXXCX

Every valid possible postfix expression will take one of these five forms. Note that when you have 4 values, it is only possible to have 3 operations in the postfix expression for it to be a valid expression.

You will need to use three nested loops iterating over the ops list created for you at the top of the file in order to generate every possible postfix notation. Once you have generated all of these notations, evaluate them, and if an expression evaluates to 24, add it to the set of results.

4.4. Running your program

Once you have written the stack functions in 4.1 and both evaluation functions in 4.2 and 4.3, to run your program you will input the 4 cards as a single string, with no spaces. For example, if the cards selected are A, A, A, and J, the input you will type in is "AAAJ".

The output should look like this:

```
Input the cards:
AAAJ
['A', 'A', 'A', 'J']
['A', 'A', '+', 'J', 'A', '+', '*']
['J', 'A', '+', 'A', 'A', '+', '*']
['A', 'A', '+', 'A', 'J', '+', '*']
['A', 'J', '+', 'A', 'A', '+', '*']
4  solutions.
```

Note that the first list displayed after the "AAAJ" input is just a list containing the card values themselves. All following lists printed out are some combination of four cards and three operations which form a valid 24 Game solution in postfix notation.

This part of the program is already done for you; see the skeleton code for comments on how the code works.

5. Submit your work to Mimir

Submit your code to Mimir after you complete your code. Mimir will automatically check the output of your code and grade your submission. You can submit your code to Mimir up to **30 times** to refresh your existing score before the submission deadline.

6. Due date

This lab assignment is worth **4 points** in the final grade. It will be due by **11:59pm on October 8th, 2017**. A penalty of **10% per day** will be deducted from your grade, starting at 12:00am.

7. Getting help

Start your project early, because you will probably not be able to get timely help in the last few hours before the assignment is due.

- Go to the office hours of instructors and TAs.
    - Prof. Wei Wei: Mon. 2:30 - 3:15pm, Wed. 2:30 - 3:15pm, Thurs. 2:30 - 3:15pm, Fri. 2:30 - 3:15pm @ITE258
    - Jenny Blessing: Fri. 12pm - 2pm @ITE140
    - Param Bidja: Tues. 2pm - 3pm @ITE140
    - Yamuna Rajan: Tues. 11am - 12pm, Wed. 9:30am – 10:30am @ITE140
    - Zigeng Wang: Mon. 3pm - 4pm @ITE140
- Post your questions on Piazza. TAs and many of your classmates may answer your questions.
- Search the answer of your unknown questions on the Internet. Many questions asked by you might have been asked and answered many times online already.