ECE 241: Data Structures and Algorithms

Project 1: FunWithNumbers- (Stack + Queue)

Description

This project is intended to familiarize you with operations on stacks and queues. The goal of the project is to design a RPN calculator for double (floating arithmetic) and implement various options including: (i) a basic interactive RPN calculator, (ii) a fancy interactive RPN calculator that also prints the cumulative infix and postfix notations, and (iii) a fancier interactive RPN calculator that allows symbolic operation with plotting option.

The project includes multiple files:

1. `Queue.py` file containing the class object Queue (provided)
2. `Stack.py` file containing the class object Stack (to complete)
3. `StackCalc.py` file containing extended operations on Stack (to complete)
4. `rpn` main program containing the RPN calculations (to complete)

At the execution the code `rpn` you will get:

```
================================================
================= Project 1 ====================
================================================
|                                              |
|         1-Simple  RPN calculator             |
|         2-Fancy   RPN calculator             |
|         3-Fancier RPN calculator             |
|                                              |
================================================

Your choice:
```

Before reviewing all three options, let us do some preliminary work on the class Stack.

Preliminary [15pts]

The `Stack.py` file is partially completed and contains a main method which is provided to you. It will help you implementing few new methods for this class.

While executing this file, the output should be (make sure you get the exact same output):

```
test __str__
0        10
1        20
2        30
test swap
0        10
1        30
2        20
test copy
0        10
1        30
2        20
3        20
test flush
0        11
```

How to proceed? You need to comment/uncomment the instructions in the `main` method in order to test step by step with the following implementation:

- Implement the `__str__` method that displays the Stack in reverse order with indexes (0 at the bottom). You can use the tab symbol `\t` to separate the index and value of the Stack.
- Implement the `__swap__` method that swaps the top item with the top-1 item
- Implement the `__copy__` method that duplicates the top item
- Implement the `__flush__` method that removes all items

1- Simple RPN calculator [40pts]

This is a live RPN calculator (like it can be found on old HP calculator), where the user can perform some calculations directly in postfix notation. As an example, if the user wants to calculate `(2+3)*4` (infix notation), it will first enter '2' then '3' then '+' then '4' then '*' (the postfix notation of the expression). Each argument is either push to the stack or manipulate arguments of the stack (if it is an operator such as '+' or '*'). The code is also printing the content of the stack after each entry. Here is the example for the expression above:

```
Your choice: 1
Welcome to the simple RPN calculator (enter 'quit' to quit)
-------------------------------------------------------

>2
-------------------------------------------------------
0 2.0
>3
-------------------------------------------------------
0 2.0
1 3.0
>+
-------------------------------------------------------
```

```
0 5.0
>4
------------------------------------------------------
0 5.0
1 4.0
>*
------------------------------------------------------
0 20.0
>
```

Remark: The Stack is displayed from bottom to top, the operations such as '+' and '-' can then access to the last two items printed (it is more intuitive to work as a calculator this way).

Here a list of arithmetic operations that must be defined:

- Basic operations: +,-,/,*,^ (power x^y)
- Basic Functions: sin, cos, exp, log, abs, sqrt (i.e. square root)
- Basic constant: pi, e

One more example: solving `(1+2*(3-4/(5+6)))`$^{1/2}$, postfix is: `1,2,3,4,5,6,+,/,-,*,+,sqrt`.

```
Welcome to the simple RPN calculator (enter 'quit' to quit)
------------------------------------------------------

>1
------------------------------------------------------
0 1.0
>2
------------------------------------------------------
0 1.0
1 2.0
>3
------------------------------------------------------
0 1.0
1 2.0
2 3.0
>4
------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 4.0
>5
------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 4.0
4 5.0
```

```
>6
------------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 4.0
4 5.0
5 6.0
>+
------------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 4.0
4 11.0
>/
------------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 0.36363636363636365
>-
------------------------------------------------------------
0 1.0
1 2.0
2 2.6363636363636362
>*
------------------------------------------------------------
0 1.0
1 5.2727272727272725
>+
------------------------------------------------------------
0 6.2727272727272725
>sqrt
------------------------------------------------------------
0 2.5045413298101655
>
```

Your RPN should also inherits from some Stack operations such as `copy`, `swap` and `flush`. As a result solving `cos(pi/5)^2+sin(pi/5)^2`, in postfix it could be (for example):

`5,pi,swap,/,copy,cos,2, ^,swap,2, ^,+`

```
Your choice: 1
Welcome to the simple RPN calculator (enter 'quit' to quit)
------------------------------------------------------------

>5
------------------------------------------------------------
0 5.0
>pi
```

```
------------------------------------------------------
0 5.0
1 3.141592653589793
>swap
------------------------------------------------------
0 3.141592653589793
1 5.0
>/
------------------------------------------------------
0 0.6283185307179586
>copy
------------------------------------------------------
0 0.6283185307179586
1 0.6283185307179586
>cos
------------------------------------------------------
0 0.6283185307179586
1 0.8090169943749475
>2
------------------------------------------------------
0 0.6283185307179586
1 0.8090169943749475
2 2.0
>^
------------------------------------------------------
0 0.6283185307179586
1 0.6545084971874737
>swap
------------------------------------------------------
0 0.6545084971874737
1 0.6283185307179586
>sin
------------------------------------------------------
0 0.6545084971874737
1 0.5877852522924731
>2
------------------------------------------------------
0 0.6545084971874737
1 0.5877852522924731
2 2.0
>^
------------------------------------------------------
0 0.6545084971874737
1 0.3454915028125263
>+
------------------------------------------------------
0 1.0
>
```

How to proceed?

- The code for the simple calculator is already provided in the `rpn` file.

- You need to implement the method `rpnCommand` in `StackCalc` that accepts a string as argument and perform various actions. For example if ``+'' is in argument, you may want to call a method `add` that you will need to implement, and so on. You would need some `if` conditions to redirect all the actions. There is no need to handle all input errors, the user must be alert and stay within the boundaries of the proposed options (example: if the user enters something which is no defined, just let the code crash). If the string input argument is a number ``2.41'' you can just push it (after converting to a float) to the Stack.
- Some operations such addition, multiplication, etc. requires the stack to have at least 2 items to work (if not nothing should happen). Some other operations such as sin, cos, etc. needs the stack to not be empty to work.
- Some other commands such as ``swap'', ``copy'', ``flush'' must also be recognized (you must use the methods inherited from the class `Stack`)

2- Fancy RPN calculator [20pts]

Exactly the same than option 1, but the calculator is now printing both postfix (stored in queue) and infix notations. Example: solving `sqrt(1+2*(3-4/(5+6)))`

```
Your choice: 2
Welcome to the fancy RPN calculator (enter 'quit' to quit)
------------------------------------------------------------

>1
------------------------------------------------------------
0 1.0
Postfix: 1
Infix:  1
>2
------------------------------------------------------------
0 1.0
1 2.0
Postfix: 1 2
Infix:  2
>3
------------------------------------------------------------
0 1.0
1 2.0
2 3.0
Postfix: 1 2 3
Infix:  3
>4
------------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 4.0
Postfix: 1 2 3 4
Infix:  4
```

```
>5
------------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 4.0
4 5.0
Postfix: 1 2 3 4 5
Infix:  5
>6
------------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 4.0
4 5.0
5 6.0
Postfix: 1 2 3 4 5 6
Infix:  6
>+
------------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 4.0
4 11.0
Postfix: 1 2 3 4 5 6 +
Infix:  (5+6)
>/
------------------------------------------------------------
0 1.0
1 2.0
2 3.0
3 0.36363636363636365
Postfix: 1 2 3 4 5 6 + /
Infix:  (4/(5+6))
>-
------------------------------------------------------------
0 1.0
1 2.0
2 2.6363636363636362
Postfix: 1 2 3 4 5 6 + / -
Infix:  (3-(4/(5+6)))
>*
------------------------------------------------------------
0 1.0
1 5.2727272727272725
Postfix: 1 2 3 4 5 6 + / - *
Infix:  (2*(3-(4/(5+6))))
>+
------------------------------------------------------------
0 6.2727272727272725
Postfix: 1 2 3 4 5 6 + / - * +
```

```
Infix:  (1+(2*(3-(4/(5+6)))))
>sqrt
------------------------------------------------------------
0 2.5045413298101655
Postfix: 1 2 3 4 5 6 + / - * + sqrt
Infix:  sqrt((1+(2*(3-(4/(5+6))))))
>
```

Another Example: Solving `cos(pi/5)`^{2}`+sin(pi/5)`^{2}.

```
Your choice: 2
Welcome to the fancy RPN calculator (enter 'quit' to quit)
------------------------------------------------------------

>pi
------------------------------------------------------------
0 3.141592653589793
Postfix: pi
Infix:  pi
>5
------------------------------------------------------------
0 3.141592653589793
1 5.0
Postfix: pi 5
Infix:  5
>/
------------------------------------------------------------
0 0.6283185307179586
Postfix: pi 5 /
Infix:  (pi/5)
>copy
------------------------------------------------------------
0 0.6283185307179586
1 0.6283185307179586
Postfix: pi 5 / copy
Infix:  (pi/5)
>cos
------------------------------------------------------------
0 0.6283185307179586
1 0.8090169943749475
Postfix: pi 5 / copy cos
Infix:  cos((pi/5))
>2
------------------------------------------------------------
0 0.6283185307179586
1 0.8090169943749475
2 2.0
Postfix: pi 5 / copy cos 2
Infix:  2
>^
```

```
------------------------------------------------------------
0 0.6283185307179586
1 0.6545084971874737
Postfix: pi 5 / copy cos 2 ^
Infix:  (cos((pi/5)))**2
>swap
------------------------------------------------------------
0 0.6545084971874737
1 0.6283185307179586
Postfix: pi 5 / copy cos 2 ^ swap
Infix:  (pi/5)
>sin
------------------------------------------------------------
0 0.6545084971874737
1 0.5877852522924731
Postfix: pi 5 / copy cos 2 ^ swap sin
Infix:  sin((pi/5))
>2
------------------------------------------------------------
0 0.6545084971874737
1 0.5877852522924731
2 2.0
Postfix: pi 5 / copy cos 2 ^ swap sin 2
Infix:  2
>^
------------------------------------------------------------
0 0.6545084971874737
1 0.3454915028125263
Postfix: pi 5 / copy cos 2 ^ swap sin 2 ^
Infix:  (sin((pi/5)))**2
>+
------------------------------------------------------------
0 1.0
Postfix: pi 5 / copy cos 2 ^ swap sin 2 ^ +
Infix:  ((cos((pi/5)))**2+(sin((pi/5)))**2)
>
```

How to proceed?

- The code for the fancy calculator is already provided in the `rpn` file.
- You need to implement the static method `postfix2infix` in `StackCalc` that accepts the postfix notation stored as a queue. It uses a stack to evaluate symbolically the expression (to become infix) and return the current item at the top of the stack (index 0). Remark: this stack is independent of the RPN stack that performs the calculations.
- The queue that is passed in argument is a mutable object and it must then be copied, since we need to operate on it. We use the function copy of the module copy to do so.
- Hint: You can follow the Postfix to Infix conversion (brief) pseudo-code provided below.

```
1. While the queue is not empty, repeat steps 2-3-4
```

```
2. Dequeue the next symbol from the queue.
3. If the symbol is an operand (value): Push it onto a stack.
4. Otherwise, the symbol is an operator.
                Perform then the operation, for example for +, -, etc. that needs 2 values:
  Pop the top 2 values from the stack.
  Put the operator, within these two values and form a string.
  Encapsulate the resulted string with parenthesis.
  Push the resulted string back to stack.
                You need to manage all particular cases (sin, cos, etc. operators)
                Be careful at special cases such as: swap, copy, pi, e , x (for option 3) etc.
                Also the "^" symbol, should return (x)**y
5. the infix notation you want to return should be the one at the top of the Stack
(last item that you can "peek")
```

Remark: To test if a given queue (String) item is a float, you may need to use the try-except python instructions while trying to convert this item into a float.

3- Fancier RPN calculator [20pts]

The user needs to enter the postfix notation argument by argument (all arguments of the queue) like in option 2. In contrast to option 1 and 2, No calculation is performed and the postfix and infix notation appears interactively. Once done, the use can enter 'run' to compute the expression. Example with: `(2+3)*4`

```
Your choice: 3
Welcome to the fancier RPN calculator (enter 'quit' to quit)
-------------------------------------------------------------
>2
-------------------------------------------------------------
Postfix: 2
Infix:  2
>3
-------------------------------------------------------------
Postfix: 2 3
Infix:  3
>+
-------------------------------------------------------------
Postfix: 2 3 +
Infix:  (2+3)
>4
-------------------------------------------------------------
Postfix: 2 3 + 4
Infix:  4
>*
-------------------------------------------------------------
Postfix: 2 3 + 4 *
Infix:  ((2+3)*4)
>run
Solution using infix:   20
Solution using postfix: 20.0
-------------------------------------------------------------
```

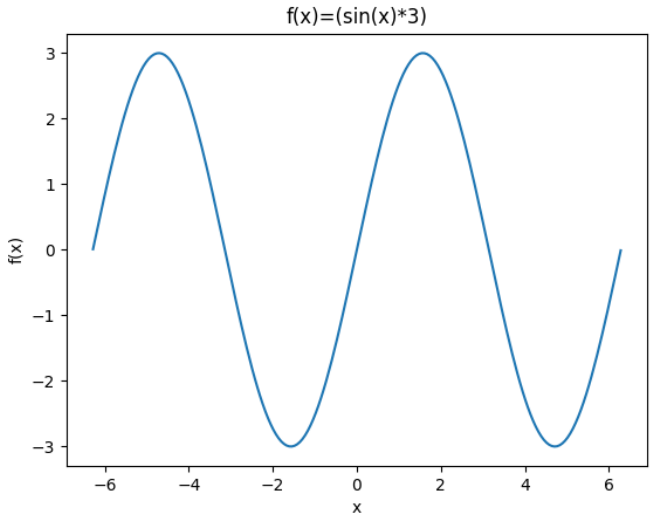
```
Postfix: 2 3 + 4 *
Infix:  ((2+3)*4)
>flush
------------------------------------------------------------
>
```

It is possible to also consider the unknown variable 'x' (symbolic calculation). Once the postfix notation entered, if 'x' is present in the queue, and 'run' is selected, the code will ask for the value of x before printing the result calculated both using the infix and postfix notations. Example with: `3*sin(x)`

```
>x
------------------------------------------------------------
Postfix: x
Infix:  x
>sin
------------------------------------------------------------
Postfix: x sin
Infix:  sin(x)
>3
------------------------------------------------------------
Postfix: x sin 3
Infix:  3
>*
------------------------------------------------------------
Postfix: x sin 3 *
Infix:  (sin(x)*3)
>run
Enter x value: 5
Solution using infix:   -2.8767728239894153
Solution using postfix: -2.8767728239894153
------------------------------------------------------------
Postfix: x sin 3 *
Infix:  (sin(x)*3)
>
```

In addition to the command 'run', you have also the possibility to 'plot' the symbolic expression (only if 'x' is present in the equation). The code will ask for the xmin and xmax and the number of grid points to use. Example:

```
------------------------------------------------------------
Postfix: x sin 3 *
Infix:  (sin(x)*3)
>plot
Enter values of xmin, xmax, nbp: -6.28 6.28 200
```



It should produce the following plot (using matplotlib):

How to proceed?

- The code for the fancier calculator must be implemented in the `rpn` file.
- To evaluate the infix notation, it is actually quite easy: just do `eval(infix)` where infix is the expression returned by the `postfix2infix` method (like in Option 2). If a `x` is present, you must assign its value before calling eval.
- To evaluate the postfix notation, you need to implement the static method `evaluate_postfix` in `StackCalc` that accepts the postfix notation stored as a queue, and a x variable if any. It must returns the solution of the operation. It uses a stack to evaluate the queue expression using successive calls to the method `rpnCommand` (like in option 1). The queue that is passed in argument is a mutable object and it must then be copied, since we need to operate on it. You need to use the function copy of the module copy to do so.
- Hint: to find if the symbol `''x''` belong to the queue, you can use the method `find` already implemented for you in the `Queue` class.
- Make sure that your plot has correct axis labels, title, etc.

Submissions

Submit a single zip file composed of: All your source files (*.py).

Grading Proposal

This project will be graded out of 100 points: Preliminary (15 points), Option 1 (40 points), Option 2 (20 points), Option 3 (20 points), Overall programming style: source code should have proper identification, and comments. (5 points)