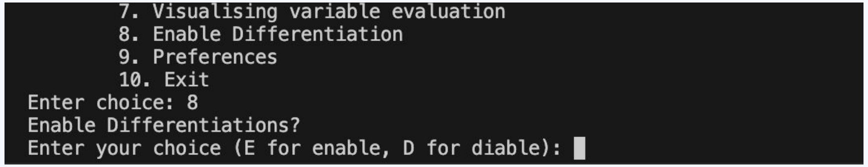
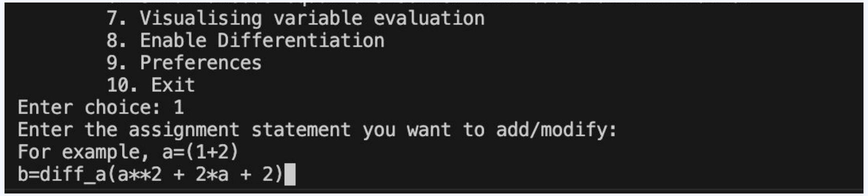
Data Structures & Algorithms CA2 (Ind)

Jeyakumar Sriram
2214618
DAAA/FT/2B/01

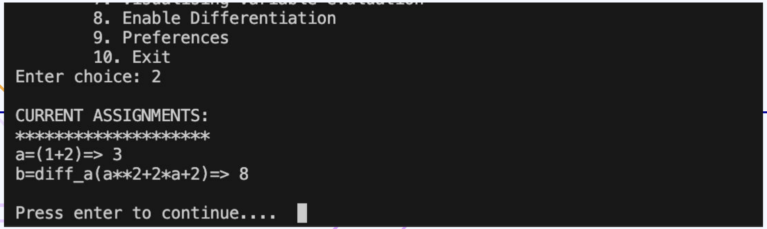
Feature 1 (Usage) - Differentiation



```
         7. Visualising variable evaluation
         8. Enable Differentiation
         9. Preferences
         10. Exit
 Enter choice: 8
 Enable Differentiations?
 Enter your choice (E for enable, D for diable): █
```



```
        7. Visualising variable evaluation
        8. Enable Differentiation
        9. Preferences
        10. Exit
Enter choice: 1
Enter the assignment statement you want to add/modify:
For example, a=(1+2)
b=diff_a(a**2 + 2*a + 2)█
```



```
        8. Enable Differentiation
        9. Preferences
        10. Exit
Enter choice: 2

CURRENT ASSIGNMENTS:
*******************
a=(1+2)=> 3
b=diff_a(a**2+2*a+2)=> 8

Press enter to continue.... █
```

This option allows the user to perform differentiation of a statement in relation to a user-defined variable. To so so, user will need to go to option 8 and enable the feature. (user_choice will be persistent)

After enabling user and tell the program to perform differentiation by using **diff_x(<statement>)** when assigning a value. In the "diff_x", x is the variable they want to differentiate in reference to.

Rules:

- diff_x must be at the start of the statement
- Only "+**-/*" are supported so no trigo or log

Feature 1 (Theory) - Differentiation

$$\frac{d}{dx}(u \cdot v) = u' \cdot v + u \cdot v'$$

Product Rule for "*" operator

$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{u'v - uv'}{v^2}$$

Quotient Rule for "/" operator

$$\left(b \cdot \left(\frac{\frac{d}{dx}a}{a}\right) + \left(\frac{d}{dx}b\right) \cdot \ln(a)\right) \cdot a^b$$

Derived Generalised Power rule for "^" operator. This one I did the maths and **derived myself** to accommodate 2^x, x^2, x^x . Basically every scenario where the base or exponent has the target variable

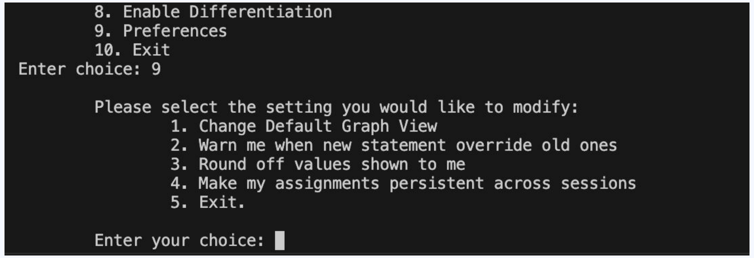
I recursively implemented the following formulas. Whenever there is a "x" symbol, I will use the product rule with its right and left children **recursively**. Thereby slowly replacing all the operators in the expressed with their differentiated ones.

Since the function only differentiates each node once, we can say this has O(n) time complexity.

$$\frac{d}{dx}\ln(x) = \frac{1}{x}$$

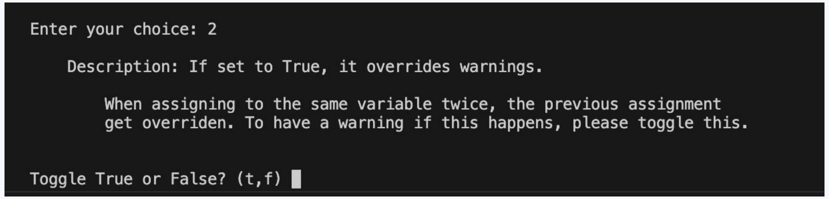
Since, I have to use natural logarithm in my generalised power rule, I will have to accomodate log differentiation as well.

Feature 2 (Usage) - Settings

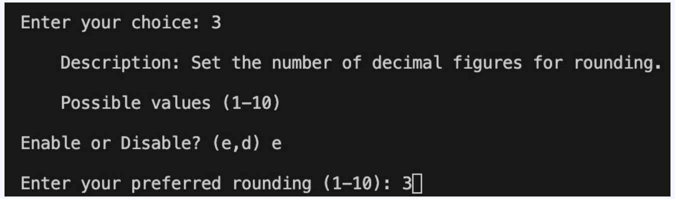


```
        8. Enable Differentiation
        9. Preferences
        10. Exit
Enter choice: 9

        Please select the setting you would like to modify:
                1. Change Default Graph View
                2. Warn me when new statement override old ones
                3. Round off values shown to me
                4. Make my assignments persistent across sessions
                5. Exit.

        Enter your choice:
```

User can change these settings which will be stored in a settings.json file after each session. And loaded at the start of each session. (Validation included). Along these choices, user choice for differentiation will also be stored.



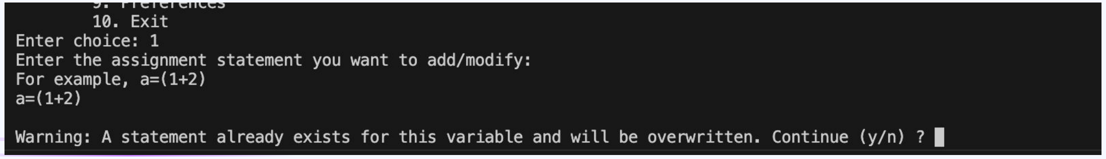
```
Enter your choice: 2

    Description: If set to True, it overrides warnings.

        When assigning to the same variable twice, the previous assignment
        get overriden. To have a warning if this happens, please toggle this.


Toggle True or False? (t,f)
```



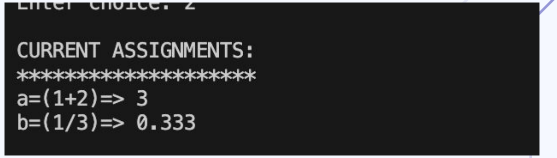
```
Enter your choice: 3

    Description: Set the number of decimal figures for rounding.

    Possible values (1-10)

Enable or Disable? (e,d) e

Enter your preferred rounding (1-10): 3
```

This feature will warn user if their assignment will override existing ones

This feature will round up values that are shown to the user.

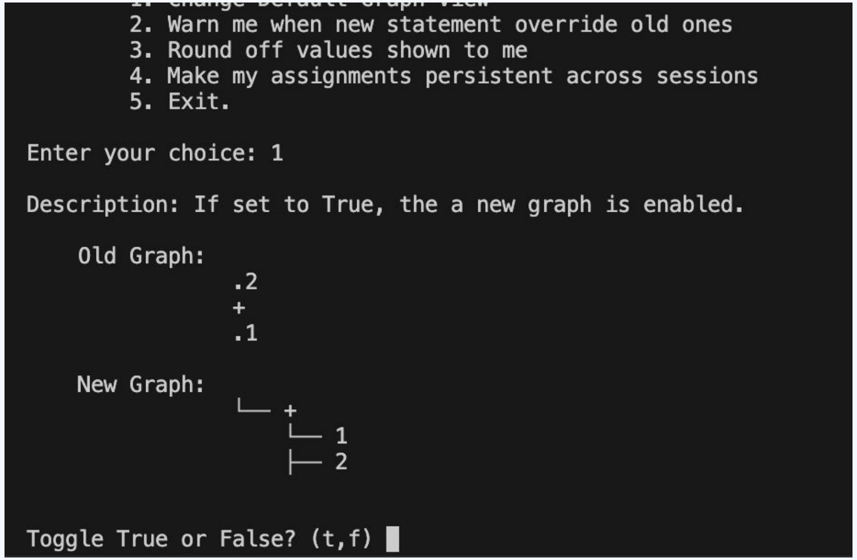


```
        9. Preferences
        10. Exit
Enter choice: 1
Enter the assignment statement you want to add/modify:
For example, a=(1+2)
a=(1+2)

Warning: A statement already exists for this variable and will be overwritten. Continue (y/n) ?
```



```
Enter choice: 2

CURRENT ASSIGNMENTS:
********************
a=(1+2)=> 3
b=(1/3)=> 0.333
```

Feature 2 (Usage) - Settings

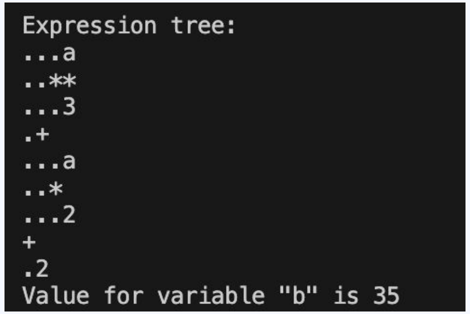


```
        1. Change Default Graph View
        2. Warn me when new statement override old ones
        3. Round off values shown to me
        4. Make my assignments persistent across sessions
        5. Exit.

Enter your choice: 1

Description: If set to True, the a new graph is enabled.

    Old Graph:
                .2
                +
                .1

    New Graph:
                └── +
                    └── 1
                    ├── 2


Toggle True or False? (t,f)
```

Setting 1: User can toggle graph view.

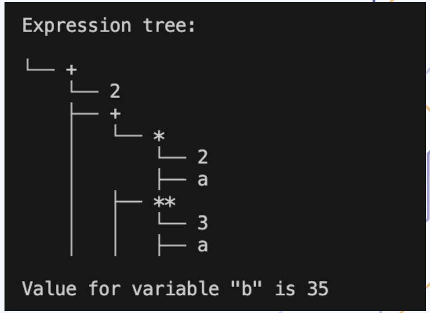
I created a new graph function similar to how “tree” in powershell works. It looks nicer. User can enable it as you can see on the side. It also uses a **recursive process** but also uses a **stack** to keep track of the nodes.

Old View



```
Expression tree:
...a
..**
...3
.+
...a
..*
...2
+
.2
Value for variable "b" is 35
```

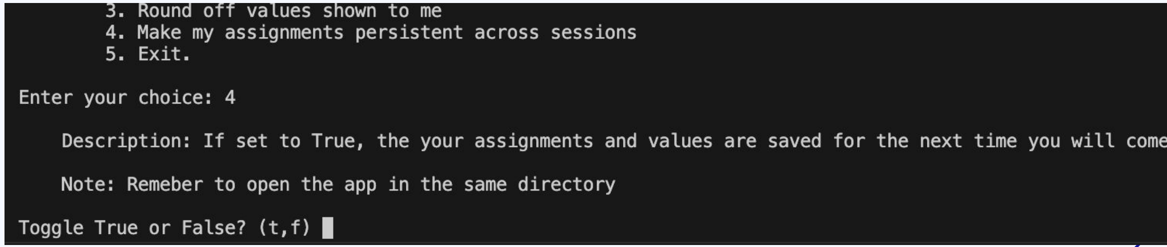
New View



```
Expression tree:

└── +
    └── 2
    ├── +
    │   └── *
    │       └── 2
    │       ├── a
    │   ├── **
    │   │   └── 3
    │   │   ├── a

Value for variable "b" is 35
```

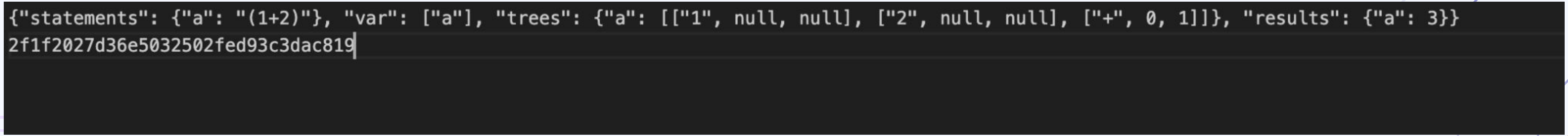
Feature 2 (Usage) - Settings

For this feature, I will save the user's current assignments, not just the statements but the **whole tree** and load it each time. This will be done using a **custom serialisation** and deserialisation function that I wrote.



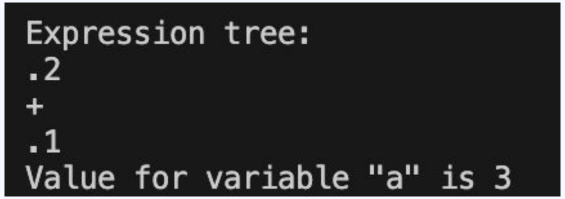
```
        3. Round off values shown to me
        4. Make my assignments persistent across sessions
        5. Exit.

Enter your choice: 4

    Description: If set to True, the your assignments and values are saved for the next time you will come

    Note: Remeber to open the app in the same directory

Toggle True or False? (t,f)
```

I will be saving the statements, variables, results, and trees in a json like format. I will also be saving a **checksum** which will be checked every time it loads. This is to make sure the user doesn't tamper with the file and crash the program.



```
{"statements": {"a": "(1+2)"}, "var": ["a"], "trees": {"a": [["1", null, null], ["2", null, null], ["+", 0, 1]]}, "results": {"a": 3}}
2f1f2027d36e5032502fed93c3dac819
```

Feature 2 (Theory) - Settings



```
Expression tree:
.2
+
.1
Value for variable "a" is 3
```

[["1", null, null], ["2", null, null], ["+", 0, 1]]

The Binary Parse Tree will be serialised using a recursive function that makes and **array of tuples** like **(value, left_index, right_index)**. So let's say there is ("+",4,5), This means this node is adding up the nodes present in the 4th and 5th position of the array.

Due to the nature of this recursive process, the headnode will **always be the last element** of this serial array.

During deserialization, I will start with the last element and follow the references to rebuild the tree. Essentially this code flattens a tree and its structure into a 1d array.