HW5 Lexer GUI using python

Now it is time to merge our GUI design with our lexer logic together and create the Lexer GUI.

You will need the complete answers from the below works to form the answer for this HW.

* HW4: with 1 line of source code input, cut the line into tokens, and output a string with <type,token> list.
* --- HW3: get input from input window and print output out line-by-line to output window.

Example code that your Lexer should be able to process:

int A1=5

int BBB2 =1034

int cresult = A1 +BBB2 \* BBB2

if (cresult >10):

print(“TinyPie ” )

Please pay attention to this program, as you can see, user can choose to type any number of spaces between an identifier, literal and the = sign or + sign or > sign, etc.

**Required GUI functions for this HW**:

Exit

Next Line

Tokens

Current line number: 1

<keyword, int>

<identifier, A1>

<operator, =>

<literal, 5>

Source Code

int A1=5

int BBB2 =1034

int cresult = A1 +BBB2 \* BBB2

if (cresult >10):

print(“TinyPie ” )

Lexer for TinyPie

When a user presses the next line the first time, the <type, token> list belonging to the first line will be printed on the right side. In you last GUI homework, you printed out this line directly. This time, you need call the token finding function that you wrote in HW3 to find out the <type, token> list for a single line of code.

**Again, your lexer can have different appearance from this example. We check for functionalities when grading. Grading details on next page.**

IMPORTANT: Start coding early and ask for help early to get your Lexer work. **It is a lot harder to debug code with GUI in it**. The python online visualizer does not support GUI. IDLE does not have breakpoint debugging. You can use Pycharm to debug your code line by line.

**Submit only 1 python 3 .py file that we can run and check if your lexer works + a pdf file with all the screenshots showing your code is working**. You can get all the screenshots into a word document and save as pdf. This is solo work.

When user press “Next Line” button, the **second line gets processed** and the GUI looks like the following graph and gives out more <type, token> pairs.

Exit

Next Line

Tokens

Current line number: 2

<keyword, int>

<identifier, A1>

<operator, =>

<literal, 5>

<keyword, int>

<identifier, BBB2>

And so on…

Source Code

int A1=5

int BBB2 =1034

int cresult = A1 +BBB2 \* BBB2

if (cresult >10):

print(“TinyPie ” )

Lexer for TinyPie

**Grading:** (**If your submission does everything HW3 or HW4 wants, but nothing new from what is required from this HW, you get 2pt still. You can submit only 1 file though. This is just to encourage those whom do not have HW3 or HW4 working yet to get them to work.**)

NOTE: if your code only works when there are spaces between every token, please write that down in the submission comments section.

2pt – submission of working HW3 OR submission of working HW4.

3pt – Your code works only when there are spaces between every token.

2pt—You can get the tokens cut even when any amount of spaces is present/not present randomly between different tokens. (The testing code will follow the example code regular format, meaning “print()” under “if” would start with spaces but int define statement will not.)

2pt—Your outputs are in the correct format and order.

1pt—Your code can take any identifiers or literals that are legal under the tinyPie definition. (For example, we might test your code with print(“lalalalalala”).)

**Challenge question**:

**If you choose to do this part: Please note in your comment section that you did the challenge question so that I can keep track.**

**If you are already comfortable with the content we introduced so far and want to challenge yourself and learn more, make sure you try the challenge questions! It is for your own learning and FUN, not for a grade**.

**Challenge**: After confirming your lexer works, now you know how to do lexical analysis and tokenize code with the help of python regular expression. What if you are building a complier using lower level (but quicker) language that does not do regular expression? Referencing the following material and build your lexer without regular expression library, just common loop or if-else.

<http://katafrakt.me/2016/07/06/regular-expressions/>

<https://www.youtube.com/watch?v=GwsU2LPs85U>

<https://www.youtube.com/watch?v=MAQUNynQu_E>

<https://stackoverflow.com/questions/22366534/build-a-regular-expression-and-finite-automata>

Automaton theory is an SUPER (I mean SUPER!) useful tool to learn and apply to your event driven programs. Once you know it, it is every simple. It can organize your thoughts in complex event-driven programs and make things easier to program. State machine is everywhere in network programming, AI, gaming, etc. Also, automaton theory is the foundation of programming language theory research and creating new languages. Sadly, we do not have enough time in this class to go into this topic. But, try learning as much as you can about this topic starting with Java switch-case statement. And then google about how to use state machine in programming.

<https://www.youtube.com/watch?v=ShfT7HiXk3s>

<https://dev.to/karn/building-a-simple-state-machine-in-python>

<https://jaxenter.com/implement-switch-case-statement-python-138315.html>

<https://bytebaker.com/2008/11/03/switch-case-statement-in-python/>

and google how automaton is used in AI, gaming, PL creation etc.

Challenge questions are **NOT** required in this homework. You still get 100% if you can build the required lexer with the help of regular expression library.