HW 3 – Single line lexer (no GUI)

We are building a compiler for a language called TinyPie. It has only the following tokens:

Keywords: if else int float

Operators: = + > \*

Separators: ( ) : “ “ ;

Identifiers: letters, or letters followed by digits

Int\_literal: only integers

Float\_literal: only float

String\_literal: only strings

Based on this, you can use our futuristic TinyPie to write programs like this:

int A1=5

float BBB2 =1034.2

float cresult = A1 +BBB2 \* BBB2

if (cresult >10):

print(“TinyPie ” )

**In this HW, you will use regular expression to cut 1 line of code into tokens and print the tokens out with <type, token> format.**

**Build a lexer for TinyPie (hints & worksheet):**

**Step 1**

Let us define the rules of different tokens so that we can use regular expression to find them. Pay special attention on how we can distinguish: keyword **if** and identifier **ifAA**, what to do with space.

Lexer scans through a line of code “int A1=5” and try to match a regular expression to “cut out” the token start from the beginning of the line. The first one to cut out would be <keyword, int>, then the lexer tries to match the second token starting from “A” in “int A1=5” and should find out <identifier, A1>, and so on.

Find the regular expression for the tokens.

**Operator is defined as:**

**Its regular expression should be:**

**Keywords:**

**Its regular expression should be:**

**Separators:**

**Its regular expression should be:**

**Identifiers:**

**Its regular expression should be:**

**Literals:**

**Its regular expression should be:**

Test them out in the interactive regex building website we use.

**Step 2**

Think about how you are going to organize them in your if-else statements. Which one do you want to test first? Does it matter?

**Orders of regex match testing for if statements:**

**Step 3**

Define 1 function in python that will take in a line of code and generate the token list.

**Def CutOneLineTokens ( one line of TinyPie like “int A1=5”):**

**Output list starting from empty list**

**Your lexer logic, find tokens using regular expression**

**add its type and format into <type, token> pair as a string and save it into the output list**

**remove/cut the first token you found from the line of code, and continue finding/cutting the next token**

**Print your output list, look like this: [<key,int>, <id,A1>, <op,=>, <lit,5>]**

**Step 4**

Test your function with different source code input and make sure it works. Call this function with every line of code in the sample code from page 1 one by one. You can also use your own example to test your code. Note that the testing code used during grading might be different than the sample code, but would be very similar.

# Output format of your single line lexer (no GUI):

Test input string: int A1=5

Output <type, token> list: [<key,int>, <id,A1>, <op,=>, <lit,5>]

Test input string: …

# Submission:

Solo work.

Please submit only the .py file containing your code (not project) + a screenshot showing your code together with your output. If your code is longer than 1 page, just partial code is sufficient in the screenshot.

Still, we require you submit code that can run. If you only finished part of the requirement, just print out a statement saying what works, what part of code has bug, but you finished coding, etc.

# Grading:

10pt total

2pt - Submission of code that cannot run.

2pt - Submission of code that can run + processes less than 1 complete line of sample code (like your code can break down a line looks like “ int A”, “if(cresult >” ) + print out statements saying what works, what part of code has bug, but you finished coding, etc.

5pt – Correct processing of each line of the sample code (or similar code) gets 1pt

1pt – Output format correct + screenshot submitted.