Open LANSI – extending python as a language for (L)inux server automation scripting (A)nalytics (N)etworking (S)ecurity (I)ntegration of (L) (A) (N) (S).

Project -02

Expected time: 1 Project Cycle = 18 Days.

What is Open LANSI?

Open LANSI is an initiative for developing a Domain Specific Language (DSL), by extending python for following domains:

1. Linux server automation scripting.
2. Analytics
3. Networking
4. Security

Open LANSI will provide an integration platform for all 4 domains.

With Open LANSI, most complicated coding issues will be simplified, and everything will be simply offered as an object to the class of the extended python language.

With the intention to simplify the work for the global community we start developing this domain specific language, which not only focuses on solving the problem, but to make sure that solution proposed are more effective and worth the efforts, Open LANSI will also be training the developers, testers and leaders in technical as well as non-technical aspects. The training provided will be intensive and will test the participants not only in the technical aspects but also in terms of concentration, determination and self-confidence.

Open LANSI – is an open source project, in which all the people who are interested in their personal development can join. Forget about the technical skills you will learn, they are endless – but at the same time, you will also be learning some important life skills like – time management, setting up goals and priorities, which is way more take away then any technical skill set you will acquire.

We believe and trust people on their moral commitments. The more you invest yourself here, the more you will be progressive in the community as well as in your personal life as well.

Training rules:

As described below, the training will be intensive and focusing only on polishing the technical knowledge of the participants, but also their soft-skills and dedication.

Let's get on the same page with some basic rules:

1.) The training is INTENSIVE. The only way you can survive this is having PhD i.e. accepting that your knowledge is poor – leave the illusion that you know everything, show the hunger to gain more knowledge and be driven towards your goal.
2.) This will be a project based training, and specific time slots will be allocated for referring the concepts as well as for completing the projects.
3.) The projects will be posted on GitHub: https://github.com/yash7118/Open-LANSI/
4.) Each project must be completed in a single (PC – Project Cycle). If the project is large, it will be divided into separate project cycles from the very start itself. It will be made sure that a realistic deadline is given.
5.) Each PC = 18 days where day 1 = the release day of the project and day 18 is the 18^{th} day from the day of release.
6.) Training is self-based. Well, the final decision of where the hints will be given and where not to be given will be dependent on you corresponding mentor.
7.) It is an open-ended project, no questions asked.
8.) After the project is due and submitted on the deadline, you will have 9 days from the date of submission to improvise the project. This is how it will work:
    a. On a common platform, where the project is uploaded, you will have to download, run and test 3 other project-submission of the same batch.
    b. Comment the flaws or appreciate the work if you find anything better in their code. Check and analyze the code from quality point of view.
    c. If you get a comment on your code, either explain why the code was right from your point of view and settle the doubt and improvise the code.
    d. More the interaction, more will be the chances of you being promoted to the mentor and leader's quadrant. As with the community expanding, we will need more leaders and mentors to train the community.
9.) If the projects are not submitted within the given time-line, the community will not wait. There is a solution to every problem.

Project 00:

Before you start project 02 – you are expected to know the following basics:

1.) C/CPP/Java:
   a. Datatypes and memory requirements for each datatype.
   b. Variables and scope of variables.
   c. Decision making statements: if, if..else, if..elseif..else ladder, Switch Statements.
   d. Loop Constructs: for loops, while loops, do while loops.
      i. Time complexities involved with each loop constructs and their nested forms.
   e. Arrays – the primary Data Structure.
      i. Insert element/s in the array.
      ii. Delete element/s in the array.
      iii. Reversing an array.
      iv. Finding maximum and minimum of all array elements.
      v. Performing basic mathematical operations on the array.
      vi. Bi-dimensional & multi-dimensional arrays
   f. String – Very important structure of the language. (C, CPP preferred for this one, avoid the use of inbuilt functions. Try validating user input.)
      i. Reversing a string.
      ii. Checking if the string has a substring.
      iii. Slicing a string.
      iv. Merge String.
   g. Functions:
      i. Declaring, Defining and Calling functions.
   h. Pointers:
      i. Understanding how pointers work.
      ii. Referencing the pointers. – Call by values, call by reference.
      iii. Dynamic memory allocation.
      iv. Freeing the memory – equally important.
   i. File/IO
   j. Multi-threading

If you are not aware of any of there, get your hands dirty with them.

Project 02:

Project name: Expression Evaluation

Project statement:

Write a program in Cpp or Java to resolve an expressions. Following are some example of expressions:

```
3
Xyz
3-4*5
a-(b+A[B[2]])*d+3
A[2*(a+b)]
(varx + vary*varz[(vara+varb[(a+b)*33])])/55
```

To resolve the expression:

Here is the key instructions:

1.) Make sure the precedence of the brackets and operations are taken care.
   a. () -> Small brackets must be solved fist.
   b. [] -> Big brackets are used to represent only the array index.
2.) The variable values and the array values must be read from the file. To keep everyone on the same page, let me give an example of the file: The text file must have values such as:

```
a 2
b 3
c 4
d 8
A 5 (1,1) (2,2) (3,3) (4,4)
B 6 (1,1) (3,3) (4,4) (5,5)
CARR 2 (0,1) (1,2)
```

Here, variables have their values immediate as follows:
a 2 => (variable name) (variable's value)
Where as the array's value is as follows:
A 5 (1,2) (2,3) (5,4) which can be explained as:
(Array name) (Array Length) (Index, Value) (Index Value)…
If the index is missing in the file, means that the value to that index is zero.

Concepts to be used:

a. Stack
b. Linked list wherever required.

The main idea is to enter the expression as input and read the relevant values from the file if the expression contains variables or array index and then evaluate the expression.

When you implement the evaluate method, you may want to test as you go, implementing code for and testing simple expressions, then building up to more complex expressions. The following is an example sequence of the kinds of expressions you may want to build with:

- ☑ 3
- ☑ a
- ☑ 3+4
- ☑ a+b
- ☑ 3+4*5
- ☑ a+b*c
- ☑ Then introduce parentheses
- ☑ Then try nested parentheses
- ☑ Then introduce array subscripts, but no parentheses
- ☑ Then try nested subscripts, but no parentheses
- ☑ Then try using parentheses as well as array subscripts
- ☑ Then try mixing arrays within parentheses, parentheses within array subscripts, etc.

Sample outputs with required files:

No variables

```
Enter the expression, or hit return to quit => 3
Enter variable values file name, or hit return if no variables =>
Value of expression = 3.0

Enter the expression, or hit return to quit => 3-4*5
Enter variable values file name, or hit return if no variables =>
Value of expression = -17.0

Enter the expression, or hit return to quit =>
```

Variables, values loaded from file

```
Enter the expression, or hit return to quit => a
Enter variable values file name, or hit return if no variables => etest1.txt
Value of expression = 3.0

Enter the expression, or hit return to quit =>
```

Since the expression has a variable, a, the evaluator needs to be supplied with a file that has a value for it. Here's what etest1.txt looks like:

```
a 3
b 2
A 5 (2,3) (4,5)
B 3 (2,1)
d 56
```

If the value in array index is not explicitly listed, it is set to 0 by default.
So, in the example above, A = [0,0,3,0,5] and B = [0,0,1]

Some more examples w.r.t etest1.txt file:

```
Enter the expression, or hit return to quit => (a + A[a*2-b])
Enter variable values file name, or hit return if no variables => etest1.txt
Value of expression = 8.0

Enter the expression, or hit return to quit => a - (b+A[B[2]])*d + 3
Enter variable values file name, or hit return if no variables => etest1.txt
Value of expression = -106.0

Enter the expression, or hit return to quit =>
```

Reference to etest2.txt

```
varx 6
vary 5
arrayA 10 (3,5) (8,12) (9,1)
```

Note: To avoid errors, please make sure there are no "null" lines at the end of the file. It will leave you in middle of no-where while troubleshooting.

Sample:
Enter the expression, or hit return to quit => arrayA[arrayA[9]*(arrayA[3]+2)+1]-varx
Enter variable values file name, or hit return if no variables => etest2.txt
Value of expression = 6.0

Enter the expression, or hit return to quit =>

How about testing this:

(varx + vary*varz[(vara+varb[(a+b)*33])])/55

If the code works fine for the above given value, it should probably satisfy all the test cases.

Important note: Make sure, after evaluation of array index value, that specific array index exists in the array and is less than the length of the array. Eg: If A[a+b*c] -> is evaluated to A[21] and array A's length is less than 21 then it is a problem! Make sure your array-index is in bound always!

Test cases:

```
1   3
2   3+4
3   3-4-5
4   3+4*5
5   b
6   a/c
7   a-b+c
8   a-b*d/a+c-4
9   (a)
10  a-(b-c)
11  a-((b-c))
12  a-(b-c)*(2-d)+c
13  A[1]
14  A[b-2]
15  A[(d-c)]
16  A[3]+B[4]
17  A[B[0]]
18  d*(c-B[a+CARR[0]-A[1]+b])
```

Following file can be used:

```
a 2
b 3
c 4
d 8
A 5 (1,1) (2,2) (3,3) (4,4)
B 6 (1,1) (3,3) (4,4) (5,5)
CARR 2 (0,1) (1,2)
```

What to submit:

1.) Your "well-commented ingenious" code – either .cpp or .java file.
2.) The code must compile on any environment.
3.) Mention the compiler version and the environment you are using. Try using the latest one. Also mention the LANSI-ID you are using.
4.) The Open LANSI community appreciates if the Code is developed as per standard nomenclature.
5.) The test-case files that you have used.
6.) A READ-ME file if you feel is required.

When to submit:

1.) Time allocated to the project is 1 PC = 18days.
2.) The project must be submitted by 18^{th} day before midnight.
3.) If you are not familiar with Project 00, please take the required time to get hands dirty on project 00. Estimated time for project 00 is 3 PC = =54 days.
4.) If you are in leader's quadrant, the deadline is half the deadline for the developers which makes it's 0.5 PC = 9 days.

Maintaining the project integrity:

1.) You can work "with" your group but not "in" group. As in you can discuss with whoever you want, take help from any online resource you wish to use, and get it done. But make sure when you submit the code, you submit your own original "self-developed" code.
2.) The purpose of the project is to make sure you develop not only the coding skills but also time management skills as well.

3.) Make your own logic, the community will not ask any question or judge you for any flawed logic or copied material. Remember – Your work here will be a direct impression of who you are. So, face it. Improve it && grow.

Logging:

1.) Log the number of hours you work on the project.
2.) Logging hours must be classified as follows:
    a. Learning hours – where you have learnt the basic concepts.
    b. Earning hours – where you productively implemented the project.

Happy Coding!

Your code is your reflection of your inner-personality.