**SPL-1 Project Report**

**[Abstract Syntax Tree Generator for Python]**

Submitted by

***Rifah Tashfiha Faria***

**BSSE Roll No: 1213**

**BSSE Session: 2019-2020**

Submitted to

***Kishan Kumar Ganguly  
Assistant Professor,  
Institute of Information Technology***

***University of Dhaka***

****

**Institute of Information Technology**

**University of Dhaka**

**[30-05-2022]**

# Table of Contents

# Page no.

# Introduction 3

# Background of the Project 3-4

# 2.1 Lexical Analysis 3

# 2.2 Parsing/ Syntax Analysis 3

# 2.3 Abstract Syntax Tree 3

# 2.4 Semantic Analysis 3-4

# 3. Description of the project 4-6

# 3.1 Lexical Analysis 4

# 3.2 Parsing/ Syntax Analysis 4-5

# 3.3 Formation Abstract Syntax Tree 5-6

# 3.4 Semantic Analysis 6

# 4. Implementation and testing 6

# 5. User interface 6-8

# 6. Challenges faced 8

# 7. Conclusion 9

# 8. References

# Index of Figures

# Page no.

# 3.1 The part which was used for counting line and indentation 5

# 3.2 Example of Parsing 5

# 5.1 Sample Input 6

# 5.2 Part 1 of Output 7

# 5.3 Part 2 of Output 7

# 5.4 Part 3 of Output 8

# Introduction

Abstract Syntax Tree Generator for Python is a project which generates AST on simple snippets of codes written in Python. Sometimes, codes become tough to understand by looking at it. In these cases, Abstract Syntax Tree helps to get more clear understanding regarding the written code. It is called abstract because it does not represent every detail of the code, rather only the structural details.

An Abstract Syntax Tree basically gives the workflow of a program. In my SPL-1, I will make AST for python which is simple and does not have any function or method.

# Background of the Project

Before implementing this project, I had to study about some topics.

**2.1:**

**Lexical Analysis:**

At first, I had to learn about lexical analysis. In lexical analysis, we take code as input and break down the code into small meaningful parts which are known as lexemes. After breaking down the code into lexemes, tokens are generated from these lexemes. Tokens are lexemes with their type values. (i.e. integer/ operator/keyword)

**2.2**

**Parsing/Syntax Analysis:**

Syntax analysis is the process where syntactical structure is checked. It is always done after lexical analysis. Syntax analysis should be done before creating an AST.

**2.3**

**Abstract Syntax Tree:**

AST is a tree representation of code. I had to study how it works, what is the purpose of it and why it is needed.

**2.4**

**Semantic Analysis:**

Semantic analysis helps to understand if the program is meaningful or not.

It checks-

* If a variable is unknown or undeclared
* If types are mismatched
* If there is any reserved identifier misuse

# Description of the Project

My project can be divided into three parts.

Those are-

* + - * 1. Lexical Analysis / Tokenization
        2. Parsing / Syntax Analysis
        3. Formation of Abstract Syntax Tree
        4. Semantic analysis for unknown variable

**3.1**

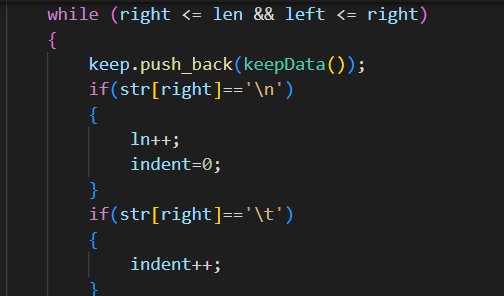
**Lexical Analysis/ Tokenization:**

After taking input from file, the input code is divided into multiple tokens. But before that all the comments (single line/ multi line) were removed so that tokenization becomes easier. For this process, conditions were made for all possible types of tokens. All the punctuation marks were skipped to ensure the proper tokenization. All the divided parts are tokenized and later syntax analysis was done on those tokens.

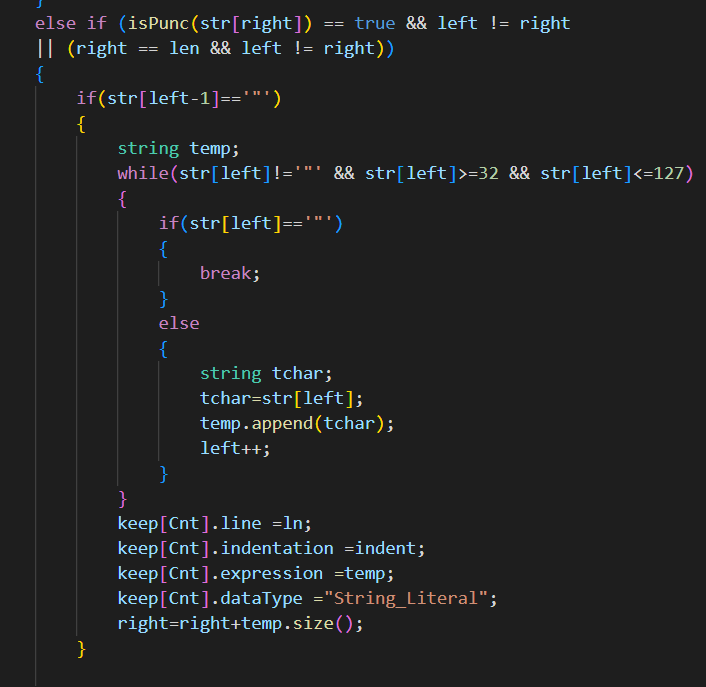
**3.2**

**Parsing/ Syntax Analysis:**

The created tokens from tokenization process were analyzed in this part of the project and categorized into their types (keywords, variables etc.) and additional information (such as line number, indentation). After tokenizing, the tokens were stored in a file. For this I had to compare all the tokens with the pre-defined list of operators, keywords and numbers. In this stage, I counted indentation of every line by comparing characters with tab. For some cases, four space are counted as one tab but they are expressed by different characters. Therefore, in the initial stage of code I converted 4 consecutive spaces into one tab.



**Fig. 3.1-The part which was used for counting line and indentation**

****

**Fig. 3.2-This is how parsing was done**

**3.3**

**Formation of Abstract Syntax Tree:**

Abstract Syntax Tree was made following the abstract grammar of Python. This was the most difficult part of my SPL-1 project. I made the tree by traversing every line linearly.

# 3.4

# Semantic analysis for unknown variable:

# There is a part of code where it is checked whether a variable is declared or not. It is a part of semantic analysis.

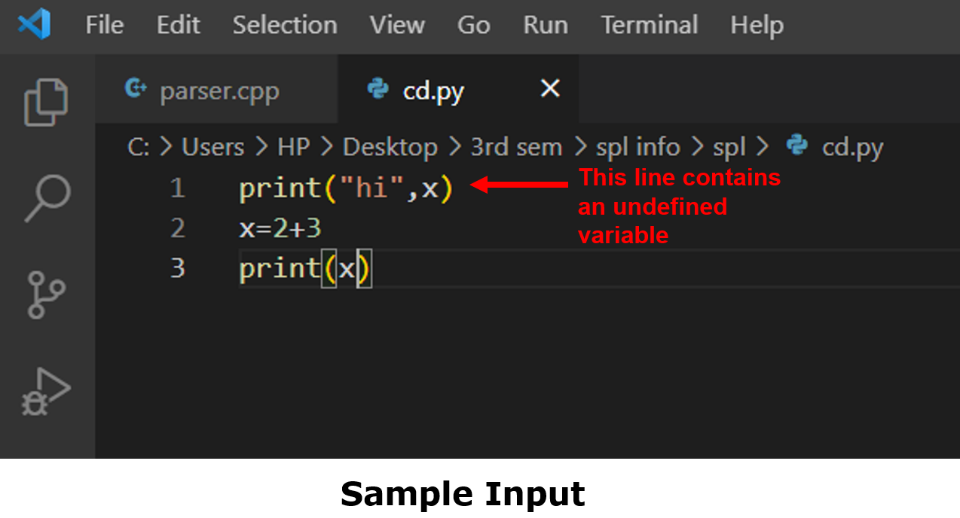
# Implementation and Testing

Abstract Syntax Tree itself is an algorithm. Besides, I have implemented the abstract grammar for python.

After taking the input, I have removed comments by using linear search. So that, the tokenization process becomes comparatively easier. After that I have divided the code into small fragments and then I have started the tokenization process. In tokenization process, the comparison was done by linear searching. Then parsing is done on the tokens to see what type of data is it. I have used structure to store different details of the tokens (such as line number, indentation number, data type and expression). After that, every line was traversed linearly and using recursion the tree was made.

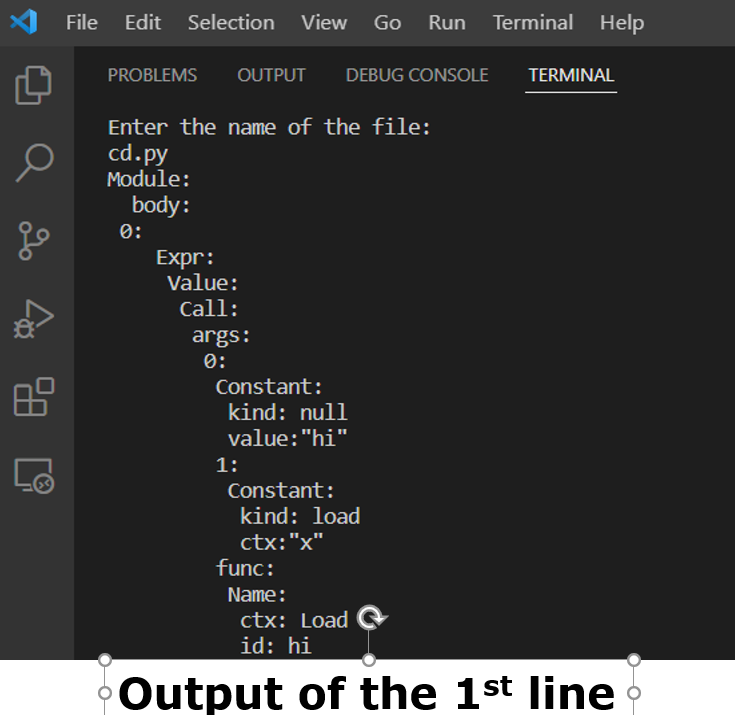
# User Interface

The input of the project is taken through the console and the output is also shown in the console.

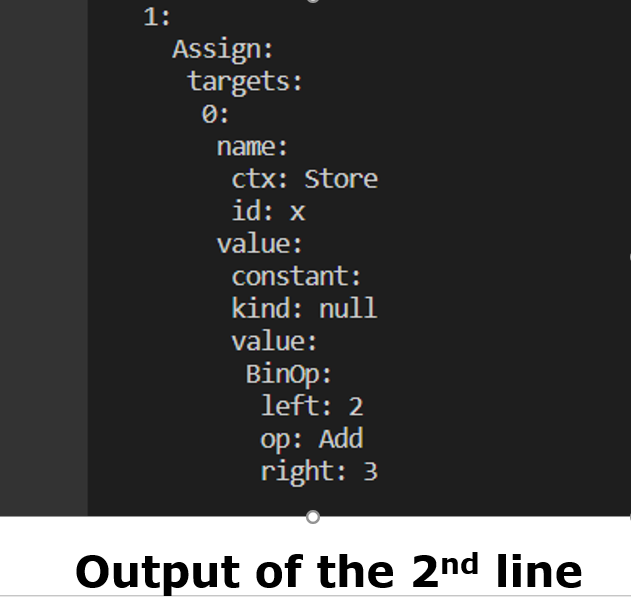


**Fig. 5.1**

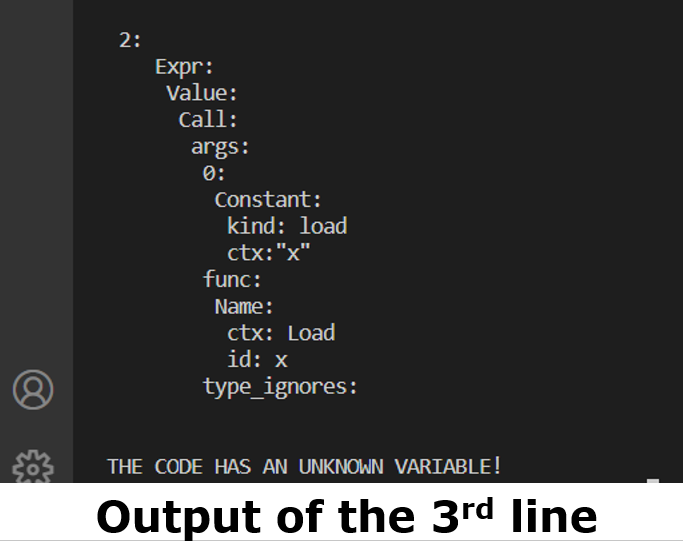
In this screenshot, a sample input is taken where we have an undefined variable.



**Fig. 5.2**



**Fig. 5.3**



**Fig. 5.4**

These screenshots denote that the main code has 3 child nodes (i.e. 3 independent parts)

# Challenges Faced

There were a lot of challenges I faced while working on the project as a lot of terms were new for me. Some of the challenges I faced during the implementation of my project are:

* Dealing with the input file
* Tokenizing the file with different types of symbols, characters and spaces.
* Removing comments
* Counting indentation
* Building header files
* Using multiple header files
* Printing the tree in the fixed format for python abstract syntax tree

# Conclusion

The whole project added a new experience and for me it was really helpful. I learned lot of new concepts and implementation. It taught me how to debug a code and how to deal with a large project. I learnt to make header files and adding those to a source code.

I want to work more on this project to improvise it so that it will be able to build syntax tree for any python code.

# Reference

1. <https://rosettacode.org/wiki/Compiler/lexical_analyzer#C.2B.2B> , compiler/lexical analyzer, last accessed on 12-02-2022
2. <https://ruslanspivak.com/lsbasi-part7/>, Abstract Syntax Tree, last accessed on 03-05-2022
3. <https://en.wikipedia.org/wiki/Abstract_syntax_tree#:~:text=Parse%20trees%20are%20typically%20built,analysis%20and%20program%20transformation%20systems>., Abstract Syntax Tree, last accessed on 21-04-2022
4. <https://docs.python.org/3/library/ast.html#:~:text=The%20ast%20module%20helps%20Python,the%20Python%20abstract%20syntax%20grammar.&text=The%20result%20will%20be%20a,built%2Din%20compile()%20function>., Abstract Grammar, last accessed on 26-05-2022
5. <https://en.cppreference.com/w/cpp/header> , Headers in C++, last accessed on 23-05-2022
6. <https://medium.com/basecs/leveling-up-ones-parsing-game-with-asts-d7a6fc2400ff> , Abstract Syntax Tree, last accessed on 26-05-2022