1/ As we know there are 6 phases in a compiler.
An abstract syntax tree is created during the period of syntax analysis.
Syntax analysis phase is the second phase which takes the input from a lexical analysis in the form of token streams. It analyzes the source code against the written programming language rules to detect any errors in the code.
The output of this phase is a parse tree which is the base for all next phases

2/ The syntax tree generated using the PLY tool:

- Import libraries and phase 1
- Using rules written in phase 2
- Create class ASTnode
- Create node name where necessary
- Write child elements for node if applicable
- Handling the errors
- Build the program

3/ a) Variable definitions
In my program, I wrote function for variable definition which consists of sheet_definition, range_definition and scalar_definition.
Almost same rules for each three. Two child node of name and init. For example-

```
├──funcs_vars[0]: sheet_definition
│  ├──name: SHEET_IDENT (BARN)
│  └──init: sheet_init_list
│     └──rows[0]: col_init_list
│        ├──cols[0]: decimal_number (4.0)
│        ├──cols[1]: decimal_number (2.0)
│        └──cols[2]: decimal_number (1.0)
```

b) For loop:
I have concrete for loop into different functions to get every children node of ranges and stmts and generate a tree like-

```
├──stmt_list[0]: for_stmt
│  ├──ranges[0]: range
│  │  ├──coord1: cell_ref
│   │   │   ├──name: SHEET_IDENT (S)
│   │   │   └──coord: coord (A1)
│   │   └──coord2: cell_ref
│   │       ├──name: SHEET_IDENT (S)
│   │       └──coord: coord (A12)
│   └──stmts[0]: if_stmt
│       ├──condition: oper <
│       │   ├──left: loopvar
│       │   │   └──range_selector: NONE
│       │   └──right: scalar_ref
│       │       └──name: IDENT (min)
│       ├──then_stmts[0]: scalar_assign
│       │   ├──var: scalar_ref
│       │   │   └──name: IDENT (min)
│       │   └──expr: loopvar
│       │       └──range_selector: NONE
│       └──else_stmts[0]: if_stmt
│           ├──condition[0]: oper >
│           │   ├──left: loopvar
│           │   │   └──range_selector: NONE
│           │   └──right: scalar_ref
│           │       └──name: IDENT (max)
│           ├──then_stmts[0]: scalar_assign
│           │   ├──var: scalar_ref
│           │   │   └──name: IDENT (max)
│           │   └──expr: loopvar
│           │       └──range_selector: NONE
│           └──else_stmts[EMPTY]: NONE
```

c) Function call:

I made it simple way just calling the function get the name of function and variables and the tree is like –

```
└──expr: function_call
    ├──name: FUNC_IDENT (Total_legs)
    └──args[0]: sheet_ref
      └──name: SHEET_IDENT (BARN)
```

4/ a. If we put the children name but never assign a value is remains empty and when we create children array but never append value into it. It remains empty.

b. I did simplify from phase 2. I omitted some unnecessary rules to generate the tree like I removed statement_list.

5/ I implemented all function and subroutine definitions, function and subroutine calls in my program.

6/ This assignment is interesting but also, I found it difficult. At first, I didn't understand how to do it, then I looked into public examples and understand the simple expressions very well. Then I try a simple line to generate the tree. Then step by step I finished with given sample inputs file of – animals.ss, minmax.ss, simple1.ss, simple2.ss and simple_function. In other files it may produce errors.