# CSC488 Sprint 1

**Abstract Syntax Tree:**

To generate our AST, we are importing the Node class and visit function from pythonAST. We build our AST by creating a node and defining the node's children at each step in our parsing. For example, when parsing an if statement, we will create an "if_stmnt" node with the condition and following block of code as children. By doing this, we will have a complete AST when parsing the file is complete. We then present the AST using the visit() function.

**Language Structures Not Parsed:**

We are currently parsing all language structures as initially stated within our proposal.

**Example Inputs:**

Inside the tests folder, you will find the following tests:

- arithmetic
  - 4 lines of arithmetic code, mixing different operations in the same statement
- ast_method_test
  - Two function definitions, the first using the "->" indicator for return type, and the second using the ":" operator within the parameter definitions to define type of input. These are separated by multiple newlines with extra newlines at the end of the file to ensure correct parsing of multiple empty lines between code.
- deeply_nested_tree
  - Test file consisting of variable definitions and then six nested if statements with "elif" and "else" clauses. This will ensure the parser can handle deeply nested statements.
- full_ast_test
  - A general test case handling class definitions, for/while loops, deeply nested statements, floats, and lists.
- if_stmnt_test
  - Extremely simple two line program for if statements
- while_nested_if_test
  - An if statement nested in a while loop, checks for while loop parsing, correct indent/dedent parsing, and comparison expressions within the while loop condition
- input_testing_every_token
  - Larger, more complex test case. Includes at least one of almost every token. Tests for correct parsing on indentation, import statements, class definitions, method declarations, arithmetic, comparisons, and/or logic, elif/else statements, for loops using the range function.

**Running The Script:**

The script to run the test suite is named test_suite_script.py. This script is found in the sprint1 directory. To run this script, simply call “python3 test_suite_script.py”. This will run the parser on each test case found in the tests directory, and the output for each test will be found in the tests/output directory. Each output file will be named “*test_case_name*_output.txt”. To verify the output of a test case, open the it’s corresponding output text file and open the test case file. IT will help to view the output files in an IDE that can highlight the tabs, to clearly see the nesting. Ensure that for each complex language structure (function definitions, class definitions, if statements, loops, arithmetic expressions, etc.) that the root of the expression is correctly identified and all code encapsulated in the complex language structure is a child.

**Updated Grammar:**

Our grammar from sprint 0 to sprint 1 went through many changes. Essentially our grammar in sprint 0 was all theoretical and very abstract and condensed. This is because we hadn’t started the implementation.

When working on the lexer in sprint 1 we noticed our grammar was not easily implementable, and thus split our grammar into more specific sections, so our grammar became more detailed.

For example, one change we made is split *statement* into two definitions, *simple_statement* and *statement_with_block.* Where *simple_statement* are statements that are single lined (function calls, variable assignment), whereas *statement_with_block* are statements that have blocks (if-statement, loops).

We made many other similar changes to increase the ease of implementation of the grammar. Furthermore, we haven’t updated our grammar to reflect all the changes we made to it in the parser. For example in the parser there is *ids_one_or_more_in_commas_with_types* yet this doesn’t exist in our grammar file. We will update this in the next sprint.