Leo Yan - leoyan
Junli Shao - junlis

We mutated the assign operator, changing the Assign operator to AugAssign operator and vice versa. We did this to change an assignment from the program. We also negate the comparison operator type and swap the binary operators. Our comparison operator swap pairs including == and !=, < and >=, > and <=, in and not in, is and is not. Our binary operator mutation pairs include + and -, * and /. We also implemented if and while condition mutations. We made the if statement evaluation value to be true all the time and while loop check condition to be false all the time. When we first decided which operators we wanted to apply, we wanted to make as many changes as possible. However, we later discovered that it was time consuming and not necessary. Therefore, we briefly looked into the fuzzyWuzzy program and found some most common operators that we wanted to change. For example, there are more binary operators but since the fuzzyWuzzy program seems to not to contain them, we did not include them in our mutation program.

To be familiar with and utilize the AST module was definitely harder than we expected. We had to read through different documents many times and try out to understand the AST structure and how to modify them. In addition, in order to make high-quality test suites, we had to figure out a way to limit the number of mutation within one single test, so we finally find out a way by using ast.NodeVisitor to keep track of the nodes waiting to be mutated and ast.NodeTransformer to modify them.

After submitting our mutation testing tool, we compared and contrasted it with previous statement coverage tools. These two kinds of test case analysis tools both measure the robustness of test cases, the result generated by these tools can tell if a test case is well designed. When it comes to the difference between mutation testing analysis and statement coverage analysis, mutation analysis measures a test case in a different scope to statement coverage analysis. While statement coverage centers on how thoroughly the test case covers the source code, it doesn’t tell much about the effectiveness of the test case. The mutation analysis, on the other hand, uses created mutants to determine if a test case is effective in capturing the exact source code logic. A single high score for statement coverage or mutation analysis doesn’t mean the test is a good test, because the test can be created deliberately to satisfy statement coverage without correctness measurement or the source code has low tolerance for the variation. In addition, statement coverage analysis is easier to deploy than mutation analysis because creating effective mutants need a lot of thinking over source code so that the mutants will not be under-killed or over-killed.