Green = Content stayed the same. Black = Completely new content.

For my final project, I would like to build upon the research survey I conducted for the midterm into automated bug detection using Machine Learning. For the midterm, my work assessed the current status quo in the field of bug detection by examining existing research into individual and aggregator ML methods each adapted for different contexts, bugs and languages. In summarizing the aforementioned research, I was able to gain an understanding into the three distinct spheres in which the recent advancements made in bug detection largely fall under: data acquisition, code representation and learning modifications. My final midterm paper presented these advancements and the results thereof as well as the results of a user survey on bug detection and programming preferences among software developers. I believe that work established a comprehensive base on which I can now attempt to build a new bug detection tool with improvements in the right directions - stimulated by both research and user perspectives.

The bug detection tool/system that I would like to build for the final project will thus have the following attributes:

- It will be written in Python with the primary purpose of examining Python code. Based on the tools used to analyze the code, it is possible that the tool proposed in the project could be extended to analyze code written in different languages. However, given 1) the growth in usage of the Python language and 2) the choice of prior ML-based bug detectors to analyze languages such as C and Java, having the system be built to primarily analyze Python code seemed like a natural first choice.
- Apart from merely creating a new tool to analyze code and detect bugs, my work will also show a comparison between different code representations and ML methods with regards to bug detection. Given that most current ML-based bug detectors (especially non-ensemble methods) are each built to only detect a single type of bug, my work will also examine the feasibility and accuracy of ML methods detecting multiple bug types. From the feedback generated through the user-survey previously conducted, it was obvious than an all-in-one solution for bug detection was a top priority for users for a multitude of reasons.
- The tool can be broken down into three main components:
    1. **Data Collection and Pre-processing**: The first task in any ML-based bug detection is to find data to train the detector on. As pointed out by countless prior research - and highlighted in my midterm paper, training a bug detector merely on non-buggy code will result in a high number of false-positives when the bug detector sees new code/coding styles. From my preliminary examination, there do not exist any datasets of substantial size in the Python language that contain labelled buggy code with corresponding non-buggy code of a similar style. If repeated examination confirms the same, then I will first set about scraping Github (and possibly similar code banks) for Python files keeping in mind the quality and diversity of the same. I will ensure that the reviews of these public repositories note that the code in the repositories work as intended with no detected bugs from multiple review sources having used the code in different ways. Once I have obtained a substantial amount of ‘non-buggy’, I will develop an input/processing pipeline that artificially injects bugs in the code being extracted thereby creating two datasets from a single one: ‘buggy’ and ‘not buggy’. Bugs such as argument swapping, argument count reduction and expansion, removal of function calls, removal of parenthesis, incorrect comparisons and incorrect operators among others will be introduced at every possible location to create a substantial amount of ‘buggy’ data - all labelled with the corresponding bug created. This approach is similar to the ones taken by DeepBug and Offside.
    2. **Code Representation/Vectorization**: Also highlighted in prior research is the importance of extracting the data correctly from the files to ensure that feature

vectorization techniques are able to prioritize the right classification features between buggy and non-buggy code. Once lines and their corresponding functions are extracted from the code, my first go-to will be to use Python's inbuilt *ast* module to extract the nodes that each function/line comprises of. This extraction will help draw importance to the defining features such as operators, conditionals etc. in the code by segregating them from magic numbers, syntax, comments etc. Once this is accomplished, the next task will be to convert this *ast* representation from string format to a numerical representation that can be consumed by ML methods. My preference is to use a form of representation learning (as done in prior research) or similar vectorization that is tailored to breaking down *ast* representations of nodes into numeric vectors. However, should this prove to be inaccurate or infeasible, I will resort to using the native vectorization tools in scikit-learn and tensorflow. Despite those tools being developed more for NLP purposes, they do provide the option to analyze a corpus of code as well by avoiding the pre-processing normally done on text corpuses. Using these tools will be beneficial as they have been developed to work with the ML methods that I will be using from the same modules. I will also attempt to compare the different vectorization tools used by holding a representative ML method constant and comparing its classification accuracy and time-taken when using each of the proposed vectorization tools to determine which representation resulted in both the greatest accuracy and the greatest usability.

3. **ML methods**: I will be using the ensemble methods from scikit-learn and neural network methods from tensorflow as the primary ML methods being compared; the best of which will be integrated into the final bug detector tool. The results of all the methods tried will also be presented however - to help readers understand how/why the final combination was chosen.