# Summer Fellowship Report

On

**Plagiarism Detection Tool for Python Programming Language**

Submitted by

**Ayan Banerjee**
**Tanmay Nandanwar**

Under the guidance of

**Prof. Kannan M. Moudgalya**
Chemical Engineering Department
IIT Bombay

July 16, 2018

# Acknowledgment

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We are greatly indebted to the authorities of FOSSEE, IIT Bombay for their guidance and constant supervision as well as providing us with the necessary information, facilities and also their support to successfully carry out this project work titled “Plagiarism Detection Tool”.

Firstly, we thank and express our solicit gratitude to Prof. Kannan M. Moudgalya and Prof. Prabhu Ramachandran for providing us an opportunity of doing summer fellowship under FOSSEE at IIT Bombay and also for keeping us motivated to successfully complete the project.

Secondly, we express our gratitude to Mr. Mahesh Gudi, who mentored us during our project and kept us encouraged throughout, which helped us in the successful completion of our project.

We would like to express our sincere gratitude to the authorities of IIT Bombay for providing the necessary infrastructure during our stay here.

Our thanks and appreciation also go to all our fellow colleagues who have willingly helped us with their knowledge and abilities.

Finally, we would like to express our heartfelt thanks to our parents who were very supportive both financially and mentally and for their encouragement to achieve our set goals.

Ayan Banerjee
Tanmay Nandanwar

# Contents

# Chapter 1

# Plagiarism Detector

## 1.1 Objective

To implement a Plagiarism Detection Tool compatible with Yaksh and integrate the same which can check for plagiarism of codes written in Python programming language.

## 1.2 Scope of Application

This is primarily made to integrate with Yaksh, but by tweaking the code a bit, it can be used for plagiarism detection almost anywhere. The user must keep in mind that the codes should be written in Python.

## 1.3 Technologies Used

### 1.3.1 Python

Python is an interpreted high-level programming language for general-purpose programming. Python is also Open Source and used in almost all field of Computer Science ranging from Scientific Computing, Data Science, Machine Learning to Desktop GUI Application, Operating Systems and Backend.

Since Django and Yaksh is written on Python, Python 3.6 is used for all of the backend coding.

### 1.3.2 Django

Django is a free and open-source web framework, written in Python, which follows the model-view-template architectural pattern. The complete backend is written on Django.

### 1.3.3 HTML, CSS, Bootstrap

HTML (Hypertext Markup Language)is the standard markup language for creating web pages and web applications.

CSS (Cascading Style Sheet) is used to style the HTML pages.

Bootstrap by Twitter is a UI framework written in CSS, JavaScript and HTML.

We also we Django's templating language which allows us to write Python code in HTML pages.

All of these are used to design the front-end part.

### 1.3.4 Git

Git is a version control system. GitLab is used as a remote repository for this purpose.

### 1.3.5 LaTeX

LaTeX is a high-quality typesetting system; it includes features designed for the production of technical and scientific documentation. LaTeX is used to write the report.

## 1.4 Possible Approaches

We considered several approaches and came up with the following. All of them share their own pros and cons.

### 1.4.1 Using Machine Learning

Using Machine Learning we can cluster the similar codes. This approach provides a one-time-run unlike other approaches but it takes a lot of time and memory. Although theoretically it should be more accurate, we did not use this owing to its high resource demands.
*Pros*: Quite accurate
*Cons*: Highly resource intensive

### 1.4.2 Abstract Syntax Tree (AST)

Abstract Syntax Tree is essentially a tree representation of the code. More on this has been discussed below.
*Pros*: More accurate than LCS.
*Cons*: Faster than Unsupervised Learning approach.

### 1.4.3 Longest Common Subsequence (LCS)

A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements. For example subsequences for the word Hello are: H, He, el, ll, lo,Ho, Hel etc. In Longest Common Subsequence, the length of the longest common subsequence is found in a set of sequences. For example, the longest common subsequence between *madam* and *maam* is maam. To check amount of plagiarism using this technique, we find the length of longest common subsequence between all the pairs of codes and check how much do they match.

We use the standard Dynamic Programming (DP) approach[**3**] to get the length of longest common subsequence. Note that codes store that codes submitted by users.

The pseudocode for the algorithm is given below:

Listing 1.1: Longest Common Subsequence

```
function LCSLength(X[1..m], Y[1..n])
   C = array(0..m, 0..n)
   for i := 0..m
      C[i,0] = 0
   for j := 0..n
      C[0,j] = 0
   for i := 1..m
      for j := 1..n
         if X[i] = Y[j]
            C[i,j] := C[i-1,j-1] + 1
         else
```

```
            C[i,j] := max(C[i,j-1], C[i-1,j])
    return C[m,n]

for i := 0..n:
   for j := i..n:
      lcs\_length := LCSLength(codes[i], codes[j])
      # code to see if it crosses the threshold value
      if lcs_length\len(codes[i]) >= threshold_value or
              lcs_length\len(codes[j]) >= threshold_value:
          # i and j have copied
```

The pros and cons of this approach is:
*Pros*: Fastest among all approaches, the time complexity is
$\mathcal{O}(n^2 * code\_of\_maximum\_length^2)$.
*Cons*: As seen from the discussion the algorithm is least accurate and very easy to fool. Students can easily escape by changing the name of variables and order of if-else or functions.

From the discussion, it is clear that AST approach gives us a nice balance between accuracy and time, and hence we decided to go with this approach.

## 1.5 How the code works

### 1.5.1 AST

An abstract syntax tree (AST), or just syntax tree, is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code.

For example for the code (calculating GCD)[**2**], AST is shown.

Listing 1.2: Code to calculate GCD

```
    while b != 0
        if a > b
            a := a - b
        else
            b := b - a
        return a
```

Most popular methods to avoid plagiarism detector are:

1. Changing the name of variables,

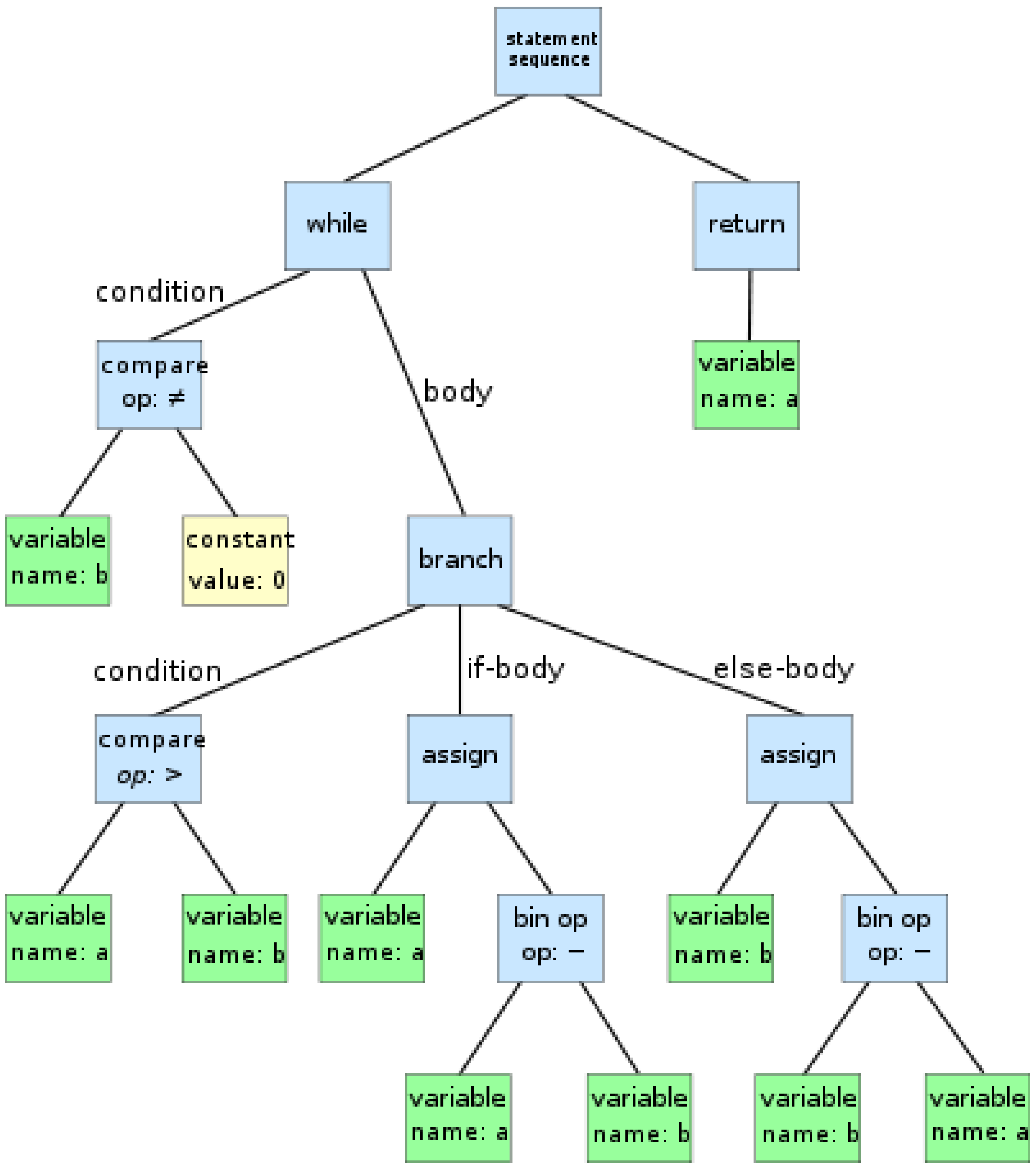

Figure 1.1: AST Representation of the Listing 2.2

2. Changing the order of functions, if-else etc.

It is quite evident that the AST structure cannot be changed using those techniques.

Python inbuilt ast module is used to get the ast representation of the code and it is used in the pycode_similar package in order to get the similarity between codes.

### 1.5.2 pycode_similar[1]

The code uses Python's inbuilt ast module in order to get the AST representations of the codes. The basic idea is to normalize Python AST representation and use difflib to get the modifications between codes. The package offers two different diff methods, namely UnifiedDiff and TreeDiff. We have used UnifiedDiff throughout owing to its better performance.

#### Our Contribution to the Package

The package used to accept only a list of the strings of the codes. We changed the code to accept the dictionary as well where the codes are sent as user_name: code format and it returns the names of the plagiarised codes, corresponding usernames and the respective plagiarism amount.

### 1.5.3 Integration with Yaksh and its implementation

It is integrated with Yaksh and during integration we used the following logic.

In the function *check_plagiarism* in views, the whole task is performed.

At first, each user's last correct submission to a problem is retrieved and it is stored in a Python dictionary.

Then the codes are sent to a function *sort_plagiarised_files* inside *plagiarism* module.

As obvious from the discussion the codes need at least one function to execute properly, so then the codes are sent to another function named *format_codes* inside plagiarism module which formats the code and add a function to the code if there isn't any.

Then inside the *sort_plagiarised_files* function the codes are checked for plagiarism using *pycode_similar*.

The plagiarised codes are further represented as groups, where each group contains a reference code and rest are candidate codes. Then the plagiarised codes are represented question wise using *plagiarism_questionwise* template.

At first the moderator is given the option if he/she wants to check for plagiarism for the particular questions. If he/she wants to check for plagiarism he/she has to set a threshold percentage, above which the level of similarity is considered as plagiarised codes. The default value is set to 70%.

Then he/she is shown all the courses and the corresponding quizzes of the course on *localhost:port_no/exam/manage/check_plagiarism*.

The moderator should select one quiz to check plagiarism by clicking on *Click for Plagiarism* button.

Then the moderator is shown with a list of students who have plagiarised sorted by questions.

The take action button redirects the moderator to the *Grade User* page where he/she can take actions (like reducing marks).

## 1.6 Possible Improvements

### 1.6.1 Language Dependency of the Package Designed

The Plagiarism Detection Tool is designed such that it can only check codes which are scripted in Python. So if some answers of any quiz or assignment are written in any other languages like C, C++, Java, etc. The code will skip all the answers written in these languages and search only for the codes written in Python.

### 1.6.2 Requirement of Functions in the Codes

The package used for designing this Tool i.e., *pycode_similar* works properly only if there are functions present in the codes otherwise it raises an error "NoFuncException". So we had to make sure that all the codes were either functional codes or object oriented codes. For which we had to check each and every code, and in case we find any procedural code we had to encapsulate the code inside a function so that *pycode_similar* doesn't raise an error and terminate the execution. There definitely must be a better and a cleaner way to do this such that we can get the same results, rather more accurate results than those which we are getting now regardless of the fact that the code is procedural or otherwise.

### 1.6.3 No support for cross language checking

As mentioned earlier, the tool can check codes which are written in Python only, it doesn't support any other language. So in case of a question where the students have the liberty of selecting any language for solving the question. If they copy from an answer but write their codes in different languages, then all those who have used languages other than Python could easily escape.

# Bibliography

[1] pycode_similar in GitHub
`https://github.com/fyrestone/pycode_similar`

[2] AST Article in WikiPedia
`https://en.wikipedia.org/wiki/Abstract_syntax_tree`

[3] Longest Common Subsequence Article in WikiPedia
`https://en.wikipedia.org/wiki/Longest_common_subsequence_problem`