2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)

Dynamic Formatting of Source Code in Editors

Rajesh Prabhu
BE Computer Engineering
VIVA Institute of Technology
Mumbai, India
rprabhuofficial@gmail.com

Ninad Phutane
BE Computer Engineering
VIVA Institute of Technology
Mumbai, India
ninadphutane44@gmail.com

Shiv Dhar
BE Computer Engineering
VIVA Institute of Technology
Mumbai, India
shivdhar@gmail.com

Siddhesh Doiphode
Asst-Prof (Comp. Engg.)
VIVA Institute of Technology
Mumbai, India
doiphodesiddhesh@gmail.com

***Abstract*—Source Code Editor, also simply known as code editor, is normally a text editor program or a set of programs that is specifically designed to edit, modify or change the source code of computer programs by software developers and programmers. It might either just be an independent standalone application or it might be consolidated into an integrated development environment (IDE). A code editor is an essential programming tool, as writing and altering source code is the most indispensable work of programmers. Source code editors contain a few components that are particularly intended to improve and in addition accelerate the contribution of source code; for example, syntax highlighting, code indentation, auto-completion of words, and bracket matching functionality. However, the existing systems do so statically; they analyze the entire source file in one go and reformat it when invoked. Static formatting or the batch-processing approach, is very inconvenient since the user must manually call the auto-formatter every time, and on the entire source file. The proposed system focuses on separating the content (source code) from the presentation (code style). The proposed system will provide all the following functionalities: word auto-completion, and auto-formatting which includes auto-indentation, auto-spacing, and auto-comments. The proposed project dynamically reformats the source code as per the programming language chosen by the user. The system will automatically format the source code as the user types it into the editor. This saves the programmer's time and help them to concentrate on writing the code which would otherwise have been wasted in manually reformatting the code, or invoking the automatic formatter simultaneously.**

***Index Terms*—Code Editor, Dynamic formatting, Auto-formatting, Auto-indentation, Auto-spacing, Auto-comments, Word auto-completion.**

I. Introduction

Programming style can be alluded to as an arrangement of rules or guidelines that are utilized while composing the source code for a computer program that will help programmers to understand and comprehend the program of a specific style, and help them to avoid any introduction of errors into it. Programming styles are generally designed for a specific programming language (or language family). However, the style often considered good for C source code may not be appropriate for Python source code, and so on. However, some rules can be commonly applied for multiple languages. For e.g., C++ and Java have vastly the same style and rules.

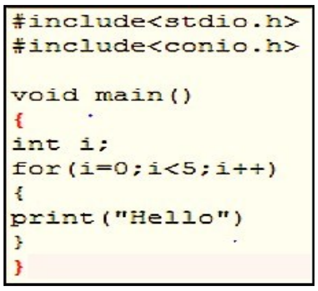
Source code Formatting includes: Auto-Indentation, Auto-Spacing and Auto-Comments. Generally, indentation and spacing do not affect function or semantic meaning, although logical and consistent indentation makes code more readable except for some languages like Python. Different programmers possess different styles of writing a programming language. With no specified set of rules or styles are followed while writing a program, it's quite hard to understand and define such rules.

Computer programming languages are rigid and formalized in structure, unlike natural languages which we use in day-to-day life, their lexical definitions are clear and well defined; indeed, for the language to be effectively parsed the lexical definitions must not leave any ambiguities. The grammars of most programming languages, however, do not define or recognize whitespace. This is done on purpose to keep the grammars simple. The price to pay for this is that the lexical phase the tokenizer must strip all whitespace.

A. Problem Definition

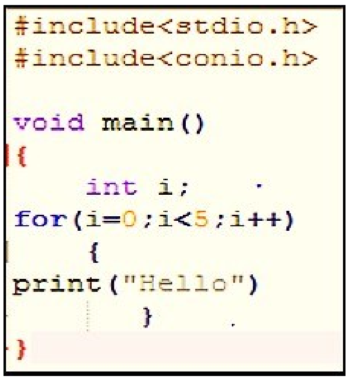
The current systems follow the 'What you see is What you get (WYSIWYG)' approach. They do not separate the content (source code) from the presentation (code style). The programmer must instead manually format the source code. This leads to many problems while writing the code. Large projects implement their own guidelines for code style to ensure uniformity in appearance. Following these style templates, is a tedious task and is a clear candidate for automation.

For example, in Fig 1, the program is syntactically correct, but with no indentation, the format is not appealing. In Fig.2, there is improper indentation, which is also aesthetically unappealing.



```
#include<stdio.h>
#include<conio.h>

void main()
{
int i;
for(i=0;i<5;i++)
{
print("Hello")
}
}
```

Fig. 1. No indentation.



```
#include<stdio.h>
#include<conio.h>

void main()
{
     int i;
for(i=0;i<5;i++)
     {
print("Hello")
          }
}
```

Fig. 2. Improper indentation.

B. Problem Definition

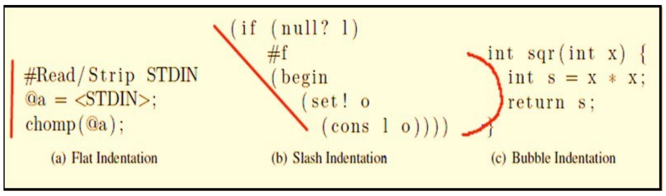
The scope of the project for the proposed system is as follows:

- The proposed system focuses on separating the content (source code) from the presentation (code style).
- The proposed system provides following functionalities: auto-completion, and auto-formatting that includes auto-spacing, auto-comments and auto-indentation.
- The proposed project dynamically reformats the source code as per the programming language chosen by the user.
- The system will automatically format the source code as the user types it into the editor. This saves the programmer's time and lets him/her concentrate on writing the code that would otherwise have been wasted in manually reformatting the code, or calling the auto- formatter repeatedly, every few minutes.
- However, the proposed system is not an Integrated Development Environment (IDE); it is just a standalone code editor application. It doesn't have an inbuilt compiler and debugger associated with it.

II. LITERATURE REVIEW

Following is some of the search which has been reviewed for the proposed system.

A. Hindle et al. proposed "From Indentation Shapes to Code Structures" studied a characterization of the underlying distributions of code structures represented by various indentation shapes with its variations. They studied that the indentation shapes patterns found in the different revisions such as Flat indentation, Slash Indentation and Bubble Indentation as shown in Fig. 3 below [1].



```
#Read/Strip STDIN
@a = <STDIN>;
chomp(@a);
```

```
(if (null? l)
    #f
    (begin
       (set! o
         (cons l o))))
```

```
int sqr(int x) {
  int s = x * x;
  return s;
}
```

Fig. 3. Types of Indentation.

R. Miara et al. proposed "Program Indentation and Comprehensibility" studied about the distinguishes in programming community. The test had two styles of indentation that were used and observed, blocked and un-blocked along with the addition of four possible levels of indentation (0, 2, 4, 6 spaces). The test was performed with two subjects- novice and experienced. After the study was performed, it concluded that 2 and 4 spaces had the highest mean score for program comprehension [2].

Anuj Vaijapurkar et al. proposed "An Auto-completion Algorithm Using Conditional Probability". It provides an auto-completion algorithm that works on user-input by providing the suggestions of the highly probabilistic words to the user. This algorithm uses a data structure called as trie Data structure. Thus, every time when a user enters a word, the algorithm implements it efficiently and modify its structure quickly and correctly [3].

Thomas E. Kesler et al. proposed "The effect of indentation on program comprehension". The paper gives an investigation of various strategies for indentation that have an effect on the capacity of programmers to understand and comprehend programs. Numerous understudies were chosen and each understudy got one of the three usage of a short Pascal program. There were different methods of source code indentation observed such as no indentation, excessive indentation, etc. [4].

Table I, shown below, provides and compares the features of different code editors for various platforms. The proposed system will include these features along with other features such as Auto-indentation, Auto-Completion and Syntax Highlighting [5].

TABLE I. COMPARISON OF DIFFERENT CODE EDITORS

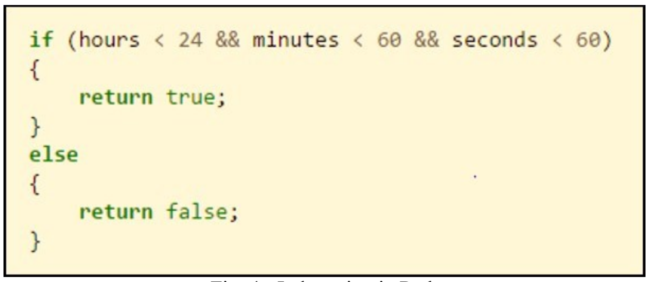
Code Editors	**Auto-Indentation**	**Auto-Spacing**	**Auto-Comments**	**Auto-Completion**
Notepad++	✓	✗	✗	✓
Text-Wrangler	✓	✗	✗	✓
jEdit	✓	✗	✗	✓
Crimson Editor	✓	✗	✗	✗

III. PROPOSED SYSTEM

There are various types of functionalities of varying importance included in the code editor.

A. Code Auto-Indentation

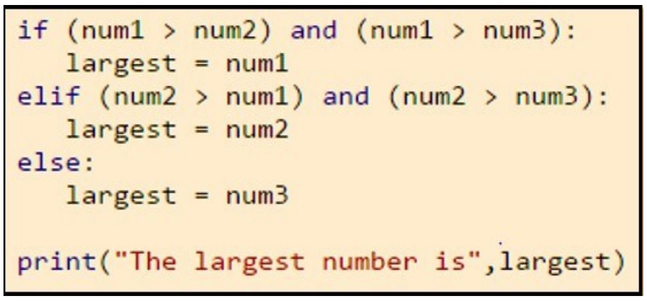
In computer programming, indentation style is more likely a tradition or a convention that oversees the indentation of the blocks of code to depict the structure of a program of any language. Indent style is, however, just one aspect of programming style. For example, Fig. 4 shows indentation for Java which is given below. Indentation, however, is generally not a requirement of most programming languages such as Java.



```
if (hours < 24 && minutes < 60 && seconds < 60)
{
    return true;
}
else
{
    return false;
}
```

Fig. 4. Indentation in Python.

Most programmers and developers like to indent their code to better convey the structure of their programs to other human readers. In any case, some programming languages, for example, Python code uses indentation in the code to decide the structure as opposed to utilizing braces or keywords, which is called off-side rule [9]. Here, indentation is not just meaningful to the reader but also to the compiler/interpreter. For example, Fig. 5 shows indentation for Python given below.



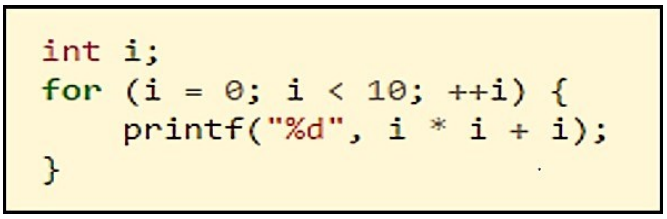
```
if (num1 > num2) and (num1 > num3):
   largest = num1
elif (num2 > num1) and (num2 > num3):
   largest = num2
else:
   largest = num3

print("The largest number is",largest)
```

Fig. 5. Indentation in Python.

B. Code Auto-Spacing

Spacing is included as the second important part in code formatting. If properly maintained, the code looks much more readable to everyone. So, programmers should follow proper spacing throughout the code along with its consistency.

The name ‘auto-spacing’ is an appropriate reference to the whitespace characters in programming languages. The whitespace interpreter overlooks any of the non-whitespace characters not at all like the clear majority of the programming languages. Only spaces, tabs and linefeeds contain meaning. The readability of many lexical elements like the operators and variables can be enhanced with some appropriate spacing. Style related to white space is commonly used to enhance readability as shown in Fig. 6 below.



```
int i;
for (i = 0; i < 10; ++i) {
    printf("%d", i * i + i);
}
```

Fig. 6. Auto-Spacing Example.

C. Auto-Comments

In computer programming terminology, a comment is generally an explanation or annotation for certain blocks or modules in a computer program which are human-readable. Comments are used to define some blocks of code or to explain modules for the future reference or others to understand it. Compilers/interpreters generally ignore such comments which have different syntax for various programming languages.

Comments are very much an integral part of programming style guides which themselves are often unused. They provide a wide degree of variability as well as provides flexibility. Comments can be written or formatted either as block comments (stream comments) or as line comments (inline comments).

There are different kinds of comments supported by various programming languages that may or may not be alike. For e.g.: in Java, comments start with '//' or /* whereas in Python, comments start with a # or '''. The Java language supports three types of comments as shown below in Table II.

TABLE II. TYPES OF COMMENTS IN JAVA

Sr. No.	Comment Type	Description
1.	/*text*/	The compiler ignores everything from /* to */
2.	//text	The compiler ignores everything from // to the end of the line.
3.	/**documentation*/	This indicates documentation comment (doc comment, for short). The compiler ignores this kind of comment. The JDK Javadoc tool uses doc comments when preparing automatically generated documentation

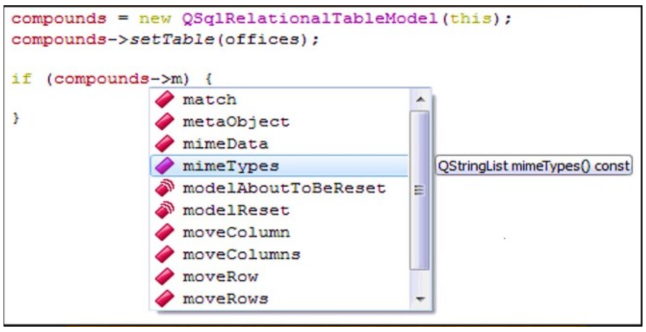
The Python language supports two types of comments as shown below in Table III.

TABLE III. TYPES OF COMMENTS IN PYTHON

Sr. No.	Comment Type	Description
1.	#text	The compiler ignores everything from # on the same line. It is called as single-line comment
2.	’’’ text…. ’’’	The compiler ignores everything from ’’’ to ’’’. It is called as multi-line comment

D. Word Auto-Completion

Autocomplete, or word completion, is a feature or a component in which an application predicts whatever is left of a word whenever a programmer is typing. It is a way to save programmers’ time by helping them to complete words (keyword) which otherwise would be wasted in typing every word. Figure 7 shows an example of word Auto-Completion.



```
compounds = new QSqlRelationalTableModel(this);
compounds->setTable(offices);

if (compounds->m) {
                  match
}                 metaObject
                  mimeData
                  mimeTypes          QStringList mimeTypes() const
                  modelAboutToBeReset
                  modelReset
                  moveColumn
                  moveColumns
                  moveRow
                  moveRows
```

Fig. 7. Word Auto-Completion Example.

IV. Implementation

A. Code Auto-Indentation

It governs the indentation of the blocks of code to convey the structure of a program. Figure 8 shows the flowchart for Auto-Indentation of the code.

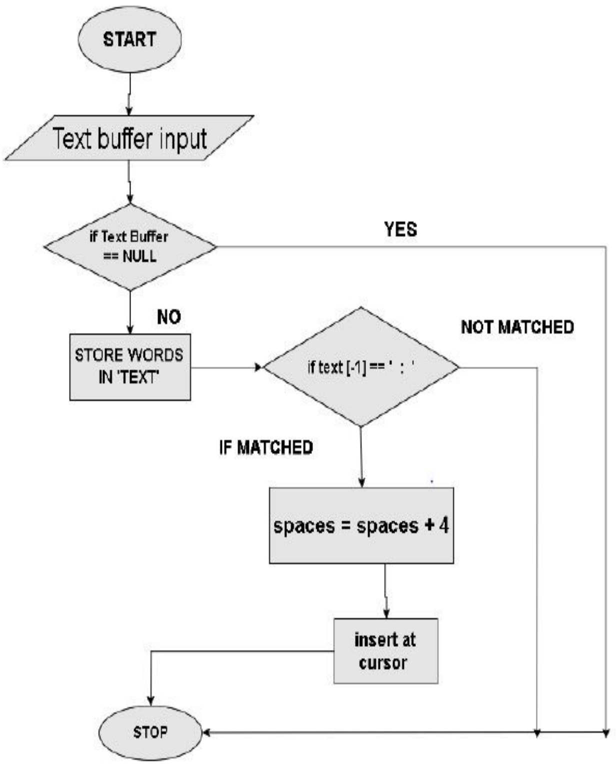


Fig. 8. Auto-Indentation for Python code

B. Code Auto-Spacing

Whitespace can set things off and reduce the strain on the reader's eyes. Since the compiler for the most part disregards the whitespace, the developer is free to utilize it and format it for comfortable viewing. If the programmer uses it properly, this can be a great help. Figure 9 shows the working of the Auto-spacing algorithm.

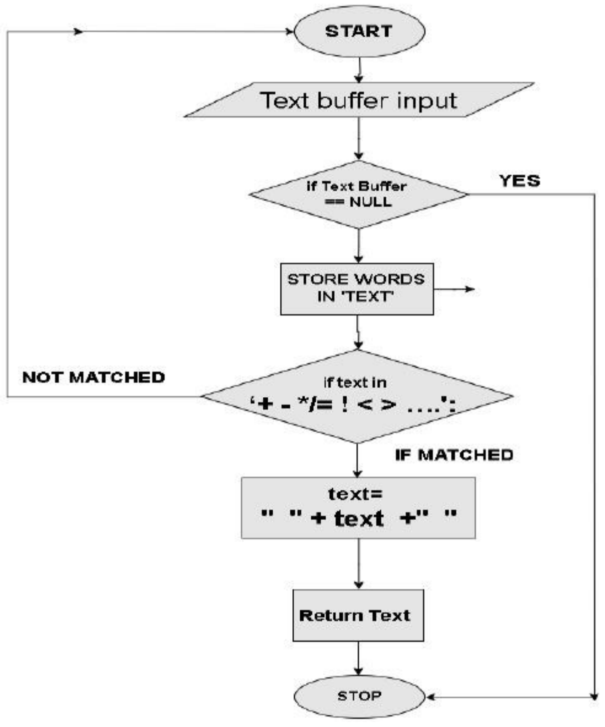


Fig. 9. Auto-Spacing algorithm flow

C. Auto-Comments

The system would remind the users to add comments whenever they add a block of code or a function. This will help the user understand the same block of code or any functions when they refer it in the future.

Compilers/interpreters ignore such comments which have different syntax for various programming languages. So, the programmers can be free to add whatever comments in that block. The algorithm for auto-comments is given below.

```
AUTO-COMMENT (text-buffer):
1.  Get input (text-buffer):
2.  While (end of line): (Pos. of cursor)
3.  If text-buffer = empty ()
4.      Return False
5.  Else text = get-text (text-buffer ())
6.      spaces = 0
7.      if text [-1] == ':'
8.          Insert at cursor ('spaces + 20 #add comments here:
        \n')
9.          spaces=0
10.         spaces = spaces + 4
11.         Insert at cursor(spaces)
```

D. Word Auto-Completion

Autocomplete, or word completion, is a functionality by which the words entered by users are predicted and then completed. In many interfaces, to scroll through the list use up-down arrow keys and the user presses the right arrow key to choose. Figure. 10 shows about the auto-completion flowchart.

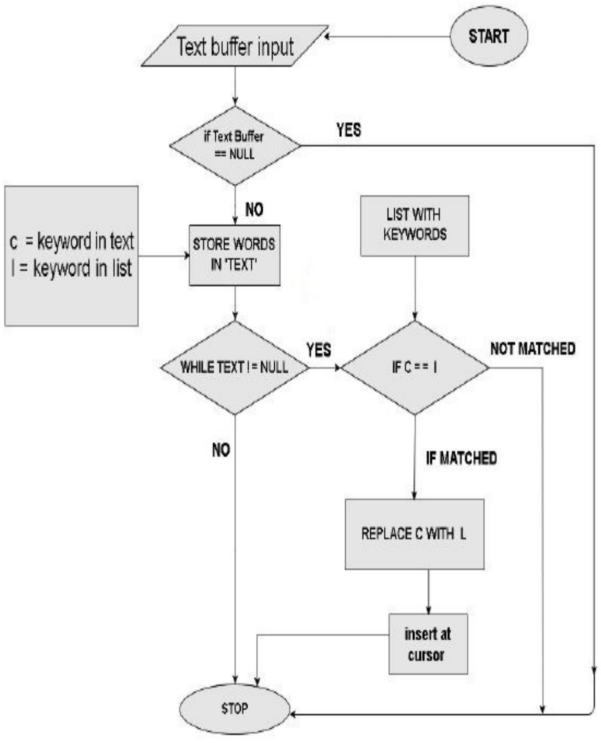


Fig. 10. Auto-Spacing algorithm flow

V. Experimental Results

The proposed system helps to make coding more interactive and easy to use, so that the user can focus more on the logic of the code rather than the syntax or formatting. Currently only two languages are supported and used for testing purposes viz. Java and Python.

The proposed system provides the following functionalities: auto-completion, and auto-formatting that includes auto-indentation, auto-spacing and auto-comments. The system will automatically format the source code as the user types it into the editor. The system also possesses syntax highlighting, side tree-view, bracket-matching functionality and language chooser at the editor startup.

Figure 11 shows language selection window of the editor whereas Fig. 12 shows a sample program in the editor.

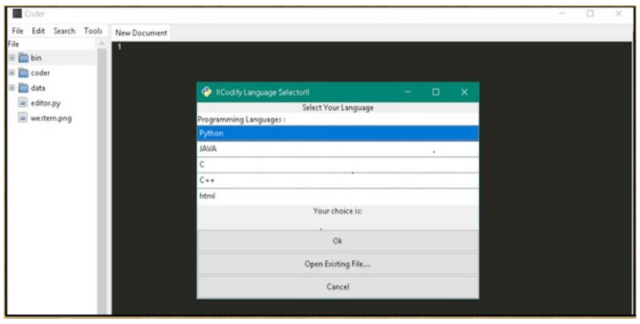
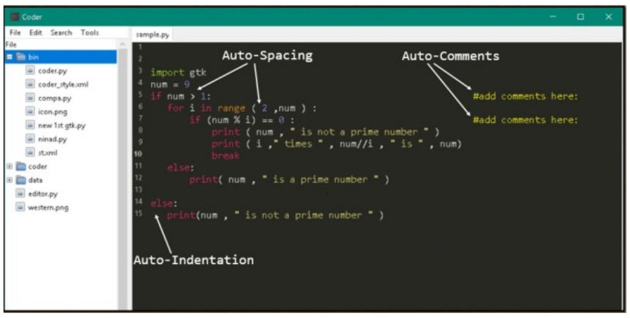


Fig. 11. Language Selection Window



```
import gtk
num = 9
if num > 1:
   for i in range ( 2 ,num ) :
       if (num % i) == 0 :
           print ( num , " is not a prime number " )
           print ( i ," times " , num//i , " is " , num)
           break
   else:
       print( num , " is a prime number " )

else:
   print(num , " is not a prime number " )
```
```
#add comments here:
#add comments here:
```

Fig. 12. Sample program in the editor (Final Result)

VI. Conclusions

With the current systems, not up to the mark when it comes to developer requirements such as formatting, the proposed system would help programmers to follow a valid and well-defined structure when it comes to writing the code.

Thus, the proposed system has been developed into a Dynamic Code editor with many different features and functionalities. The system would help programmers to stick to their logic of program and help them worry less about the formatting part of it. This would not only make a better code with good logic but also would make the code look even better than it should be.

Future Scope

The proposed system can be developed in many different directions which have vast scope for improvements in the system. These include:

1. Providing support for various programming languages other than Java and Python.
2. Improvising the algorithms to add more efficiency of the system and enhance its working.
3. Adding more functionalities other than the current ones to make it flexible for the developers to use it.
4. To make it a complete IDE, compiler support can be integrated into the system along with a debugger, version control systems and build automation tools.

References

[1] A. Hindle, M. Godfrey, and R,Holt. "From Indentation Shapes to Code Structures". IEEE Conference 28 September, 2008.

[2] R.Miara,J.Musselman,J. Navarro and B. Shneiderman,"Program Indentation And Comprehensibility". November 1983.

[3] Anuj Vaijapurkar, Rishon Patani and Vaibhav Kashyap Jha "An Auto-completion Algorithm Using Conditional Probability" 2 April. 2016, pp. 61-64.

[4] Thomas E. Kesler, Randy B. Uram, Ferial Magareh-Abed, Ann Fritzsche, Carl Amport, H.E. Dunsmore "The effect of indentation on program comprehension" 3 November 1983.

[5] http://www.hongkiat.com/blog/free-code-editors-reviewed, last accessed on: 05/01/2017.

[6] https://en.wikipedia.org/wiki/Comment_(computer_programming), last accessed on: 10/01/2017.

[7] http://www.cprogramming.com/tutorial/style.html, last accessed on: 10/01/2017.

[8] https://en.wikipedia.org/wiki/Whitespace_character, last accessed on: 07/01/2017.

[9] https://en.wikipedia.org/wiki/Indent_style, last accessed on: 06/01/2017.

[10] https://en.wikipedia.org/wiki/Programming_style, last accessed on: 05/01/2017.

[11] https://www.tutorialspoint.com/Java/Java_documentation.htm,last accessed on: 07/01/2017.

[12] http://www.pygtk.org/pygtk2reference,last accessed on: 12/01/2017

[13] https://google.github.io/styleguide/Javaguide.html#s4-formatting, last accessed on: 11/01/2017

[14] https://developer.gnome.org/pygtk/stable/, last accessed on: 10/01/2017.

[15] https://www.python.org/dev/peps/pep-0008/, last accessed on: 10/01/2017.

[16] http://javarevisited.blogspot.in/2013/11/java-vs-python-which-programming-laungage-to-learn-first.html, last accessed on: 12/01/2017