Use of Cell Block As An Indent Space In Python

Hyung Jun Yoo *hjyoo760408@hotmail.com*
Department of Electrical Engineering and Computer Science
Texas A&M University-Kingsville
Kingsville, TX 78363, U.S.A

Young Lee *young.lee@tamuk.edu*
Department of Electrical Engineering and Computer Science
Texas A&M University-Kingsville
Kingsville, TX 78363, U.S.A

Abstract

Unlike most object oriented programming languages, Python does not use braces such as "{" and "}". Therefore, mixed tabs and spaces are used for indentation. However, they are causing problems in Python. Several approaches are applied to eliminate the problem that is not only for machine-readable form but also for proof reading for human. Often, characteristics of some programming behaviors are sometimes ambiguous. In such case, it is better for human to review what machines may not handle well, but the majority of python source code editors do not provide visually attracting environment. To provide the solution to both problems, concept of using cellblock in spreadsheet as an indent space for python source code is introduced.

Keywords: Source Code, Visualization, Spreadsheet, Python, Stereopsis Algorithm.

1. INTRODUCTION

Spreadsheet contains a table of values arranged in rows and columns where each value may have predefined relationship with values in other cells [1]. Python is an interpreted, object-oriented, high-level programming language that runs in major operating systems such as, Linux/Unix, Windows, OS/2, Mac, and Amiga [2]. It can be integrated with COM, .NET, COBRA objects, or implement Python for Java Virtual Machine (JVM) using Jython [3].

There are numbers of studies concerning software visualization, where pictures, graphs or animations can acquire information about specific program. Although significant numbers of software visualization products are released, most of those products are not compatible among each other when it comes to sharing their data. One approach to solve this problem is to use a widely used application with powerful feature that is already built in it [5]. This study suggests how to use such application to write and fix errors as a first step to use its source code as a data to visualize software. Calculating numbers with visualizing it result in graphs are magnificent, but if source codes can be analyzed and counted by given conditions, it can be a commonly used program which is easy to get access and since it is widely used, sharing data between users are not difficult. Python programming language was chosen in here since it had a problem that most languages did not have and thought it could benefit more from using this method. Indent style is unnecessary for most programming languages. Rather, it is used for more clear understanding of how the source code is constructed in human readable form. For indentation, using multiple spaces and tabs were common to programmers in most programming languages. However, spaces used for indentation may vary from time to time, different programming language may use different number of spaces, or they may diverge among source code editors which programmers use preferably.

Unlike other programming languages that use braces such as "{" and "}" for block of codes, Python relies on indentation. An issue arises when several programmers get involved with editing the same source code while they use different editors that has different spaces step up for indentation. This study analyzes the Python source code and displays its indentation level on spreadsheet to resolve the confusion in mixed indent style as well as, to examine potential problem.

UNIX users or old python programmers used adding 8 spaces for each indentation, but current recommendation for one indentation is using 4 spaces [4]. This can cause problematic python source code if a programmer decides to reuse old source code when they have different size of indent block. Moreover, numbers of Microsoft Windows based Microsoft Visual Studio programmers use tab as indentation, which is default indent block, which is used by Visual Studio 2005/2007.

2. USING CELL BLOCK IN SPREADSHEET AS INDENT SPACE FOR PYTHON

2.1 Motivation

This People from different background have different way of writing source code. However, when it comes to indent style, they should all follow the same rule. Changing a way of doing things, which have been done for a while, may not be easy to fix. Eventually, there are going to be errors made. When using the cell as an indented block, this problem can be solved. A program can be written in a way to count numbers of spaces for indentation in python source code. It should count the spaces for the indentation wherever it first occurs and save its one block of indentation information on a file, so that it can be imported from spreadsheet later. If indented spaces are 8 spaces, 16 spaces, 24 spaces which is multiple of 8, then 8 spaces should be marked as one block of indentation, 16 spaces as 2 blocks and so on.

2.2 Proposal

If mismatch spaces of indent style is found, a program should correct it assuming a small mistake, but also inform the programmer that such error was found. When 8 spaces were used in an indentation and 7 spaces of indentation were found, it is easy to tell that one space is missing. However, some programmers use 4 spaces for indentation, which makes it when 6 spaces of indentation was found, it can be confusing to tell right away whether the indentation was meant for 4 or 8 spaces. One of the techniques to solve this problem is what we decided to call it a "Stereopsis" algorithm.

Stereopsis algorithm is similar to visualization spreadsheet where users lay out two data sets next to each other to compare to data groups [6].

One of the outputs, a "right eye view", of this program will mark indentation that if in 8 spacing indent style, a line with less than 8 spaces will be marked as no indentation, less than 16 spaces will be marked as 1 indented block, and less than 24 spaces will be marked as 2 indented blocks. The other output, a "left eye view", will show a line with spaces greater than 0 will get 1 indented block, greater than 8 spaces will get 2 indented blocks, and greater than 16 spaces will get 3 indented blocks. If source code has no error in indent style, both output will produce same result, but if there is an indent spacing that is different from others, error flag is raised which it can be check by the programmer later.

3. CASE STUDY

```
#Halstead Complexity output
#
if ((h == True and gotfile == True)or (s != True and c != True and h != True and gotfile == True)):

        print
        print '====================='
        print 'Halstead\'s Complexity'
        print '====================='
        print
        print ('Class Name').ljust(27),
        print ('Operators').ljust(12).rjust(4),
        print ('Operands').ljust(10).rjust(4),
        print ('Unique').ljust(10).rjust(4),
        print ('Unique')
        print ('').ljust(51),
        print ('Operators').ljust(10).rjust(4),
        print ('Operands')
        print '-------------------------------------------------------------------------'

        t_hal_operator_num = 0
        t_hal_operand_num = 0
        t_hal_unique_operator_num = 0
        t_hal_unique_operand_num = 0

        for key in result.keys():
                try:
                        print (key+'').ljust(29),
                except:
                        #print ('Total number of ').ljust(30),
                        dummy = 2
                if (key):
                        print (str(result[key].hal_operator_num)).rjust(5).ljust(11),
                        t_hal_operator_num += result[key].hal_operator_num
                        print (str(result[key].hal_operand_num)).rjust(4).ljust(10),
                        t_hal_operand_num += result[key].hal_operand_num
                        print (str(result[key].hal_unique_operator_num)).rjust(4).ljust(10),
                        t_hal_unique_operator_num += result[key].hal_unique_operator_num
                        print (str(result[key].hal_unique_operand_num)).rjust(4)
                        t_hal_unique_operand_num += result[key].hal_unique_operand_num
                 else
                        printf ('error occured')
        print '-------------------------------------------------------------------------'
        print ('Total Number').ljust(29),
        print str(t_hal_operator_num).rjust(5).ljust(11),
        print str(t_hal_operand_num).rjust(4).ljust(10),
        print str(t_hal_unique_operator_num).rjust(4).ljust(10),
        print str(t_hal_unique_operand_num).rjust(4)
        print
```

FIGURE 3.1: Python Source Code.

3.1 Limitation of Python Source Code Editor

Figure 3.1 shows one section of typical python source code, but there is an error, which might be hard to catch if condition statements or a loop contains several more lines of code. A programmer has to rely on the compiler to check the location of an error to find it. To be able to check errors in the source code without compiling save a lot of time, but before suggesting a solution to this, we will first go over with how an error is found and fixed just by using the source code and a compiler. After source code is ready, a programmer run compiler to see if there are any errors. If errors are found, the compiler usually returns line numbers. From compiling source code, we got an error and the line number where it points to else statement. Figure 3.2 shows where the error was in Figure 3.1. However, if this code is written or exported to a spreadsheet, it can be found fairly easy. Since spreadsheet has option to display vertical and horizontal line to clearly show the individual blocks of cells, we can use this to display the source code to catch an error.

3.2 Stereopsis Algorithm

3.2.1 Left Eye View

When this source code is passed through Stereopsis algorithm, since there is an error in the source code, two different results will occur. Following two figures will show how they appear and what can be done to fix it. These two figures will display mostly where the error occurred.

```
#Halstead Complexity output
#
if ((h == True and gotfile == True)or (s != True and c != True and h != True and gotfile == True)):

        print
        print '====================='
        print 'Halstead\'s Complexity'
        print '====================='
        print
        print ('Class Name').ljust(27),
        print ('Operators').ljust(12).rjust(4),
        print ('Operands').ljust(10).rjust(4),
        print ('Unique').ljust(10).rjust(4),
        print ('Unique')
        print ('').ljust(51),
        print ('Operators').ljust(10).rjust(4),
        print ('Operands')
        print '-------------------------------------------------------------------------'

        t_hal_operator_num = 0
        t_hal_operand_num = 0
        t_hal_unique_operator_num = 0
        t_hal_unique_operand_num = 0

        for key in result.keys():
                try:
                        print (key+'').ljust(29),
                except:
                        #print ('Total number of ').ljust(30),
                        dummy = 2
                if (key):
                        print (str(result[key].hal_operator_num)).rjust(5).ljust(11),
                        t_hal_operator_num += result[key].hal_operator_num
                        print (str(result[key].hal_operand_num)).rjust(4).ljust(10),
                        t_hal_operand_num += result[key].hal_operand_num
                        print (str(result[key].hal_unique_operator_num)).rjust(4).ljust(10),
                        t_hal_unique_operator_num += result[key].hal_unique_operator_num
                        print (str(result[key].hal_unique_operand_num)).rjust(4)
                        t_hal_unique_operand_num += result[key].hal_unique_operand_num
                else
                        printf ('error occured')
        print '-------------------------------------------------------------------------'
        print ('Total Number').ljust(29),
        print str(t_hal_operator_num).rjust(5).ljust(11),
        print str(t_hal_operand_num).rjust(4).ljust(10),
        print str(t_hal_unique_operator_num).rjust(4).ljust(10),
        print str(t_hal_unique_operand_num).rjust(4)
        print
```

Extra Space Causing Error

FIGURE 3.2: Line Drawn for Finding an Error.

```
if (key):
        print (str(result[key].hal_operator_num)).rjust(5).ljust(11),
        t_hal_operator_num += result[key].hal_operator_num
        print (str(result[key].hal_operand_num)).rjust(4).ljust(10),
        t_hal_operand_num += result[key].hal_operand_num
        print (str(result[key].hal_unique_operator_num)).rjust(4).ljust(10),
        t_hal_unique_operator_num += result[key].hal_unique_operator_num
        print (str(result[key].hal_unique_operand_num)).rjust(4)
        t_hal_unique_operand_num += result[key].hal_unique_operand_num
        else
        printf ('error occured')
```

FIGURE 3.3: Source Code from the Left Eye Method.

The left eye method inserts a block of indentation when default number of spaces in indent style is counted. Any spaces from 0 to 7 will get no block of indentation. Spaces from 8 to 15 will get one block of indentation. Spaces from 16 to 23 will get two block of indentation. This python code used 8 spaces for one block of indentation. The left eye method inserted 2 blocks of indentation when if statement had 16 spaces in front. However, 17 spaces where found before else statement started, thus it added 3 blocks of indentation. In the Left Eye method, one example of formula may be written as: Number of blocks of indentation equals spaces in front of current line of code divided by default indented spaces and add one if the remainder is greater than zero.

3.2.2 Right Eye View

Next figure is an output of the Right Eye method.

```
if (key):
        print (str(result[key].hal_operator_num)).rjust(5).ljust(11),
        t_hal_operator_num += result[key].hal_operator_num
        print (str(result[key].hal_operand_num)).rjust(4).ljust(10),
        t_hal_operand_num += result[key].hal_operand_num
        print (str(result[key].hal_unique_operator_num)).rjust(4).ljust(10),
        t_hal_unique_operator_num += result[key].hal_unique_operator_num
        print (str(result[key].hal_unique_operand_num)).rjust(4)
        t_hal_unique_operand_num += result[key].hal_unique_operand_num
else
        printf ('error occured')
```

FIGURE 3.4: Source Code from the Right Eye Method.

It is possible to find an error from examining from Figure 3.4 only but, when two different source codes are present, finding an error can be more convenient, because a programmer does not have to go through whole source code but find the difference in those source codes and make a correction. In this case, a programmer should select a result from the Right Eye method and choose it as code to be use. In the Right Eye method, formula can be written as: Number of blocks of indentation equals spaces in front of current line of code divided by default indented spaces.

3.2.3 Simple Error Correction

For simple error correction and handling, an error correction algorithm can be implemented while migrated to spreadsheet to remove problem in Figure 3.1. If the default indentation has 4 spaces, put no indentation to a code that has 0~1 spaces in front, level one indentation to a code with 3~5 spaces, and level two indentation to a code with 7~9 spaces in front. This will get rid of problems when a space bar is pressed a little bit more (or less) than intended. In Figure 3.1, default indentation is 8 spaces, which makes code with 0~2 spaces in front will have no indentation, code with 6~10 spaces will have level one indentation, code with 14~18 spaces will have level two indentation, and so on. This case, the result of simple error correction has same output as the right eye view. Indent spaces that is close to middle of the first indentation and next indentation should be alerted to a programmer, because it is risky to depend wholly on the computer to correct it automatically. These errors are sometimes ambiguous even to the programmer hence, the decision should be made manually by the programmer with clear information by presenting both result from Stereopsis algorithm.

3.2.4 Stereopsis Result

A python source code passed through Stereopsis algorithm will have 3 output files in Comma Separated Value (CSV) files and a message displayed on monitor screen. “_l” is added to file name which contains output through the left eye view whereas, “_r” and “_sec” will be added to output files passed through the right eye view and simple error correction, respectively. Messages on the monitor should contain location of indent mismatch from default indent spacing, suggested

indent space, and which indent style should be used. With this information, a python programmer should have sufficient enough knowledge to correct the source code. To demonstrate how the spaces were either indented or not, a sample file name “test.txt” was created. This file contains a sentence with tabs and whitespaces inserted in front.

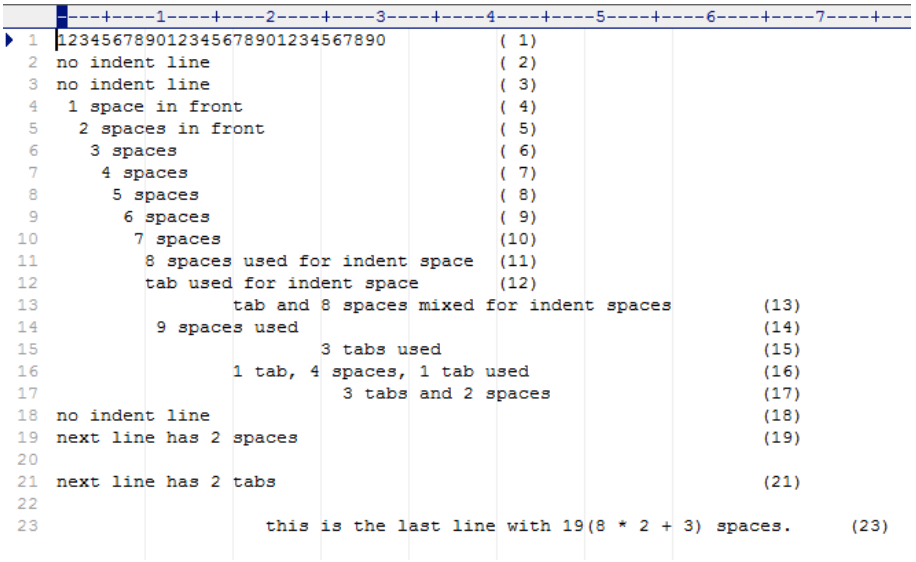


FIGURE 3.5: test.txt.

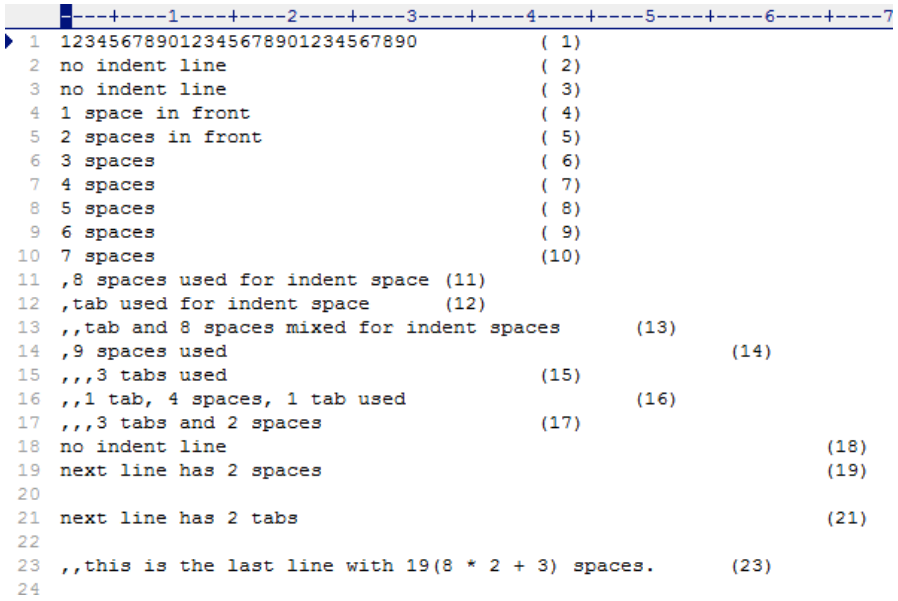
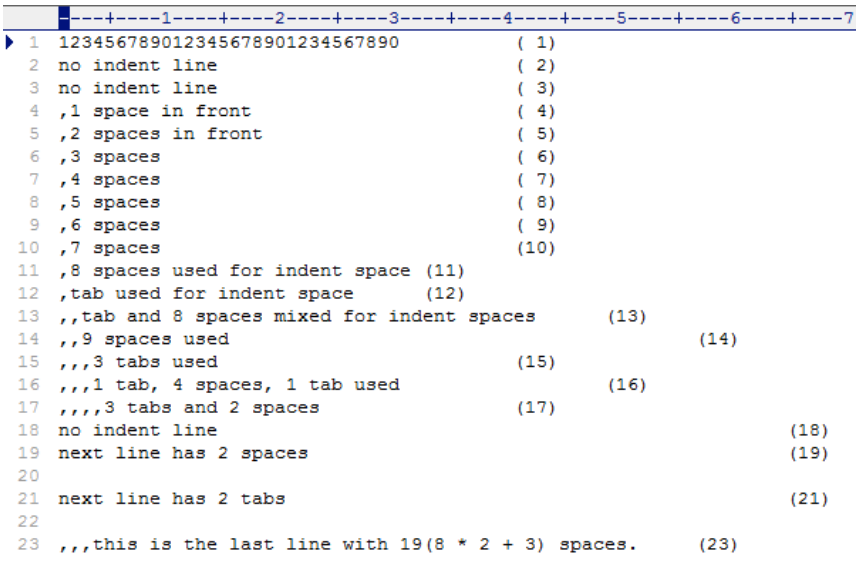


FIGURE 3.6: test_r.csv.

When passed through Stereopsis algorithm, 3 outputs are generated along with result message. In Figure 3.6, file “test_r.csv” will mark cell block when there are 8 or more spaces. Cell blocks are added if whitespaces are 8 to 15 spaces. Two cell blocks are added if whitespaces are 16 to 23. Tabs are treated as 8 spaces.

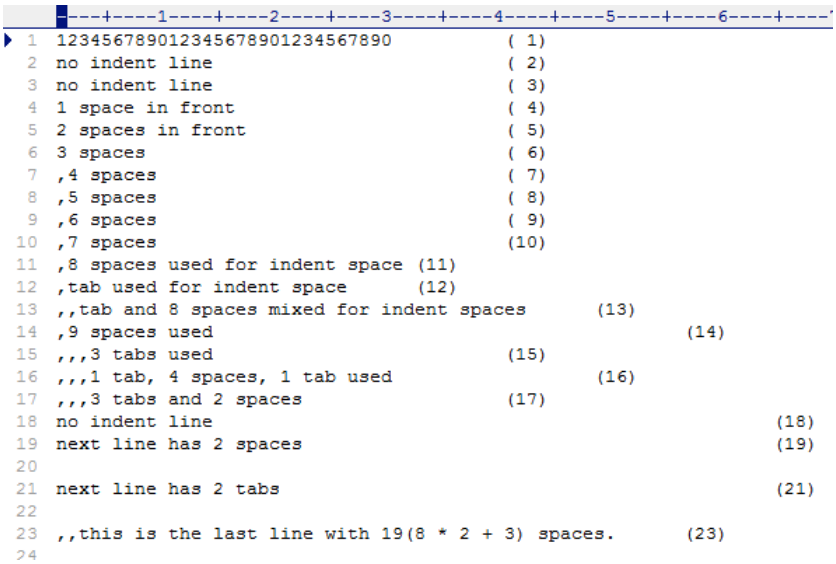
File “test_l.csv” will mark a cell block in front, if there is any space greater than one. Cell blocks are added if whitespaces are 1 to 8 spaces. Two cell blocks are added if whitespaces are 9 to 16. Tabs are treated as 8 spaces just like in the right eye view. Figure 3.7 shows the result of “test.txt” in “test_l.csv” with appropriate cell block inserted.

File “test_sec.csv” is the one that most programmers may like, since it corrects any little mistakes in intent spacing. This is not true if mistakes were made outside the correction range. One of the examples that simple error correction cannot correct is when indentation error exceeds more than 8 spaces. However, Stereopsis algorithm will still catch the mismatching indent style and inform the programmer where the error was made.



```
    ----+----1----+----2----+----3----+----4----+----5----+----6----+----7
 1  12345678901234567890123456789O          ( 1)
 2  no indent line                          ( 2)
 3  no indent line                          ( 3)
 4  ,1 space in front                       ( 4)
 5  ,2 spaces in front                      ( 5)
 6  ,3 spaces                               ( 6)
 7  ,4 spaces                               ( 7)
 8  ,5 spaces                               ( 8)
 9  ,6 spaces                               ( 9)
10  ,7 spaces                               (10)
11  ,8 spaces used for indent space (11)
12  ,tab used for indent space      (12)
13  ,,tab and 8 spaces mixed for indent spaces      (13)
14  ,,9 spaces used                                      (14)
15  ,,,3 tabs used                          (15)
16  ,,,1 tab, 4 spaces, 1 tab used                  (16)
17  ,,,,3 tabs and 2 spaces                 (17)
18  no indent line                                                (18)
19  next line has 2 spaces                                        (19)
20
21  next line has 2 tabs                                          (21)
22
23  ,,,this is the last line with 19(8 * 2 + 3) spaces.     (23)
```

FIGURE 3.7: test_l.csv.



```
    ----+----1----+----2----+----3----+----4----+----5----+----6----+----
 1  12345678901234567890123456789O          ( 1)
 2  no indent line                          ( 2)
 3  no indent line                          ( 3)
 4  1 space in front                        ( 4)
 5  2 spaces in front                       ( 5)
 6  3 spaces                                ( 6)
 7  ,4 spaces                               ( 7)
 8  ,5 spaces                               ( 8)
 9  ,6 spaces                               ( 9)
10  ,7 spaces                               (10)
11  ,8 spaces used for indent space (11)
12  ,tab used for indent space      (12)
13  ,,tab and 8 spaces mixed for indent spaces      (13)
14  ,9 spaces used                                       (14)
15  ,,,3 tabs used                          (15)
16  ,,,1 tab, 4 spaces, 1 tab used                  (16)
17  ,,,3 tabs and 2 spaces                  (17)
18  no indent line                                                (18)
19  next line has 2 spaces                                        (19)
20
21  next line has 2 tabs                                          (21)
22
23  ,,this is the last line with 19(8 * 2 + 3) spaces.      (23)
24
```

FIGURE 3.8: test_sec.csv.

Any mistakes equal to or smaller than 8 spaces or one tab will be corrected, but if indent error is greater than 8 spaces, the programmer may use output from the left or right eye view to make an adjustment.

Since spreadsheet eliminates and mismatch in indent style, this is a good way to write a python source code. However, direct python compiler is still needed to get the python to work, which is written, in spreadsheet format. This should be included in the future work, until then, python source code in spreadsheet can be exported to text file to be run.

Following figures 3.9, 3.10 and 3.11 shows each python source code in spreadsheet.

	A	B	C	D	E	F	G	H
1	123456789012345678901234567890(1)							
2	no indent line(2)							
3	no indent line(3)							
4		1 space in front(4)						
5		2 spaces in front(5)						
6		3 spaces(6)						
7		4 spaces(7)						
8		5 spaces(8)						
9		6 spaces(9)						
10		7 spaces(10)						
11		8 spaces used for indent space(11)						
12		tab used for indent space(12)						
13			tab and 8 spaces mixed for indent spaces(13)					
14			9 spaces used(14)					
15				3 tabs used(15)				
16				1 tab	4 spaces	1 tab used(16)		
17					3 tabs and 2 spaces(17)			
18	no indent line(18)							
19	next line has 2 spaces(19)							
20								
21	next line has 2 tabs(21)							
22								
23				this is the last line with 19(8 * 2 + 3) spaces.(23)				
24								

FIGURE 3.9: test_l.csv in Spreadsheet.

	A	B	C	D	E	F	G
1	123456789012345678901234567890(1)						
2	no indent line(2)						
3	no indent line(3)						
4	1 space in front(4)						
5	2 spaces in front(5)						
6	3 spaces(6)						
7	4 spaces(7)						
8	5 spaces(8)						
9	6 spaces(9)						
10	7 spaces(10)						
11		8 spaces used for indent space(11)					
12		tab used for indent space(12)					
13			tab and 8 spaces mixed for indent spaces(13)				
14		9 spaces used(14)					
15				3 tabs used(15)			
16			1 tab	4 spaces	1 tab used(16)		
17				3 tabs and 2 spaces(17)			
18	no indent line(18)						
19	next line has 2 spaces(19)						
20							
21	next line has 2 tabs(21)						
22							
23			this is the last line with 19(8 * 2 + 3) spaces.(23)				
24							

FIGURE 3.10: test_r.csv in Spreadsheet.

	A	B	C	D	E	F	G
1	123456789012345678901234567890(1)						
2	no indent line(2)						
3	no indent line(3)						
4	1 space in front(4)						
5	2 spaces in front(5)						
6	3 spaces(6)						
7		4 spaces(7)					
8		5 spaces(8)					
9		6 spaces(9)					
10		7 spaces(10)					
11		8 spaces used for indent space(11)					
12		tab used for indent space(12)					
13			tab and 8 spaces mixed for indent spaces(13)				
14		9 spaces used(14)					
15				3 tabs used(15)			
16				1 tab	4 spaces	1 tab used(16)	
17				3 tabs and 2 spaces(17)			
18	no indent line(18)						
19	next line has 2 spaces(19)						
20							
21	next line has 2 tabs(21)						
22							
23			this is the last line with 19(8 * 2 + 3) spaces.(23)				
24							

FIGURE 3.11: test_sec.csv in Spreadsheet.

```
#
if ((h == True and gotfile == True)or (s != True and c != True and h != True and gotfile == True)):
class ContainerType(object):
    '''ContainerType is a type to describe abstracted
    tokens that contain other tokens'''

    # a list of valid operators
    OPERATORS = reserved.keys() + ['+','-','*','/','++','--',';',':','[',']','(',')','{','}','&&','||','<=','>=','<','>','==','=','!']

    # a list of tokens that are neither operators nor operands
    IGNORE = [',']

    def __init__(self, name=None):
        '''ContainerType([str]) -> ContainerType'''
        self.name = name
        self.tokens = []
        self.counter = 0
        self.bodyless = False

    def work(self, tokens, index=0):
        '''self.work(list, [int]) -> int'''
        while True:
            if index >= len(tokens):
                break
            current = tokens[index]

            # track open braces and close braces
            if current.value == '{':
                self.counter += 1
                self.tokens.append(current)
            elif current.value == '}':
                self.counter -= 1
                self.tokens.append(current)
                if self.counter == 0:
                    break
```

Figure 3.12: Colored Indented Blocks

3.3 Coloring Cell Block

In information visualization spreadsheets, cells may have abstract data sets, selection criteria, viewing specifications and other information required to customize specific views, have been developed to allow end users access to rich visualizations of data [7]. An idea of containing information about surrounding cells can boost the visualization if coordinated carefully. With each blocks of cell represent an indentation, a programmer still need to count the number of blocks to check when two or more functions or statements are apart from each other but expected to be in same indentation. Using a built-in feature in Microsoft Excel 2005/2007, cell blocks can be colored when it meets given conditions. When block of cell is colored with pre-defined color, it will save time counting blocks of cells in front of the code.

In Figure 3.12, each *n*-th level of indented blocks is colored to show the location of the indented block in a source code. In this case, the yellow represents the first indent block, green the second, orange the third, purple the fourth, red the fifth and blue which represents it is on the sixth indented block. If empty line was inserted to a source code, none of the blocks were colored to avoid confusion. Coloring cell by condition statements are used to on Excel feature which allows user to fill the cell with data additionally, uses its data to give information for visualization [8].

4. CONCLUSION

Further study of this subject may be to add options work with spreadsheets with different types that are widely used, such as OpenOffice.org Calc, Apple Numbers, etc. Using spreadsheet features to analyze the source code, for example, count the number of classes, functions, variables, and lines of code to compute the complexity of the program. In Microsoft Excel VBA(Visual Basic Application) is provided when OpenOffice has StarOffice Basic in CALC(spreadsheets) to allow complex calculation using programming language based on the data from the spreadsheet [9]. Grouping and Outling in Excel as well as hiding cells features will provide grouping classes or structure when editing python source code which is in most object oriented language source code editor. This will allow programmers to look at the source code with abstract information where it only shows classes, functions and structures name.

When there are many programmers using numbers of different types of editor to work on one program, it is hard to maintain a single style of way of writing code. Thus, an application is needed to recognize different styles and synthesize them for ease of collaboration in work among programmers with different background. Such application should display the problem with more clear and in meaningful matter.

5. REFERENCES

[1] Byron S. Gottfried, **"**Spreadsheet Tools for Engineers Using Excel", McGraw-Hill, 2009

[2] Mark Lutz, "programming Python", O'Reilly Media, 2011

[3] "BeginnersGuide Overview", Python Official Site. Python Software Foundation, n.d. Web. 10

[4] Rossum, Warsaw, "Style Guide for Python Code", www.python.org/psf/, Python Software Foundation

[5] Martin Erwig, Robin Abraham, Steve Kollmansberger, Irene Cooperstein. “Gencel: a program generator for correct spreadsheets”, Journal of Functional Programming, Cambridge University Press, Vol. 16, Issue 3, (2006): 293-325.

[6] Ed Huai-hsin Chi, John Riedl, Phillip Barry, Joseph Konstan, Principles for Information Visualization Spreadsheets, Vol. 18, no.4 IEEE. (1998): 30-38.

[7] Robin Abraham, Margaret Burnett, Martin Erwig. “Spreadsheet Programming”, Encyclopedia of Computer Science and Engineering, (2009): 2804-2810.

[8] “Microsoft Conditional Formatting: Adding Customized Rules to Excel 2007”, Microsoft Developer Network (MSDN), Microsoft Corporation

[9] Solveig Hauglan, “StarOffice 6.0 Office Suite Companion”, Prentice Hall, 2002