



Software Maintenance and Evolution

CSE 426

Evolving the Editor

Submitted by:

Omar Saad Ahmed Hegazy

ID: 17P6008

Senior-2 CESS

Submitted to:

Dr. Ayman Bahaa

Table of Contents

- 1.Introduction..... 4
- 2. Evolution Details 5
- 3.New Design 6
- 4.Screenshots 8
- 5. Code..... 10

List of Figures

Figure 1: Anubis IDE.....	4
Figure 2: Class Diagram	6
Figure 3: Sequence Diagram.....	7
Figure 4: C# Test Code	8
Figure 5: Python Test Code	9
Figure 6: MyPath [1].....	10
Figure 7: Coloring [1]	11
Figure 8: Coloring [2]	12
Figure 9: Coloring [3]	13
Figure 10: Coloring [4]	14
Figure 11: Coloring [5]	15
Figure 12: Coloring [6]	16
Figure 13: Anubis [1]	17
Figure 14: Anubis [2].....	18
Figure 15: Anubis [3].....	19
Figure 16: Anubis [4].....	20
Figure 17: Anubis [5].....	21
Figure 18: Anubis [6].....	22
Figure 19: Anubis [7].....	23

1.Introduction



Figure 1: Anubis IDE

In this report I will explain how to evolve Anubis IDE; This is a simple IDE which will allow you to perform some basic functionalities, created by Anubis Graduation project team in faculty of engineering Ain shams university supervisor (Professor: Ayman Bahaa).

The report will contain all the evolution details, new design [class diagram and sequence diagram], testing sample code for both python and c# and finally screenshots from the evolved code.

The GitHub repository can be found here:

<https://github.com/Omar-Saad72/Anubis-IDE-Evolution>

2. Evolution Details

A new feature is added to Anubis IDE which is supporting the **C# format** as the editor can now automatically recognize which format to use based on the file extension.

To add this feature, we need to make the program more general than being specific to only programming language. So, we need to add something to detect the programming language we are using [Python or c#].

MyPath.py:

A new file is added to the program. It is used to decide whether the selected file is Python or C# file, it contains a global variable that represents the path of the opened file to work on. It is used by coloring and Anubis files.

Anubis.py:

Three changes occurred to this file:

1. Import MyPath.
2. Call [create()] function to initiate the global variable with the file extension.
3. Get path from the globally shared variable that comes from MyPath module by modifying [on_clicked()] function.

Coloring.py:

Eight changes occurred to this file:

1. Import MyPath.
2. Change the name of the file from [PythonColoring] to [Coloring] as it is responsible now for coloring of both python and C#.
3. Change the name of Class [pythonHighlighter] to [Highlighter] as it is more general now.
4. Add 2 array CsRules[] and pyRules[] instead of rules[], with each array modifications to match its language's keywords.
5. Add 2 arrays CsKeywords[] and PyKeywords[] instead of Keywords[], where each one having keywords of its programming language in [Highlighter] Class.
6. Modify highlightBlock function by checking whether the current path extension matched Python language (.py) or C# language (.cs).
7. Function matchMultiline() has been divided into two functions, one for comments in C# and the other for comments in Python.
8. The parameters of C# function have been modified, since 2 extra params are added which are ending and beginning params.

3.New Design

3.1 Static View [Class Diagram]

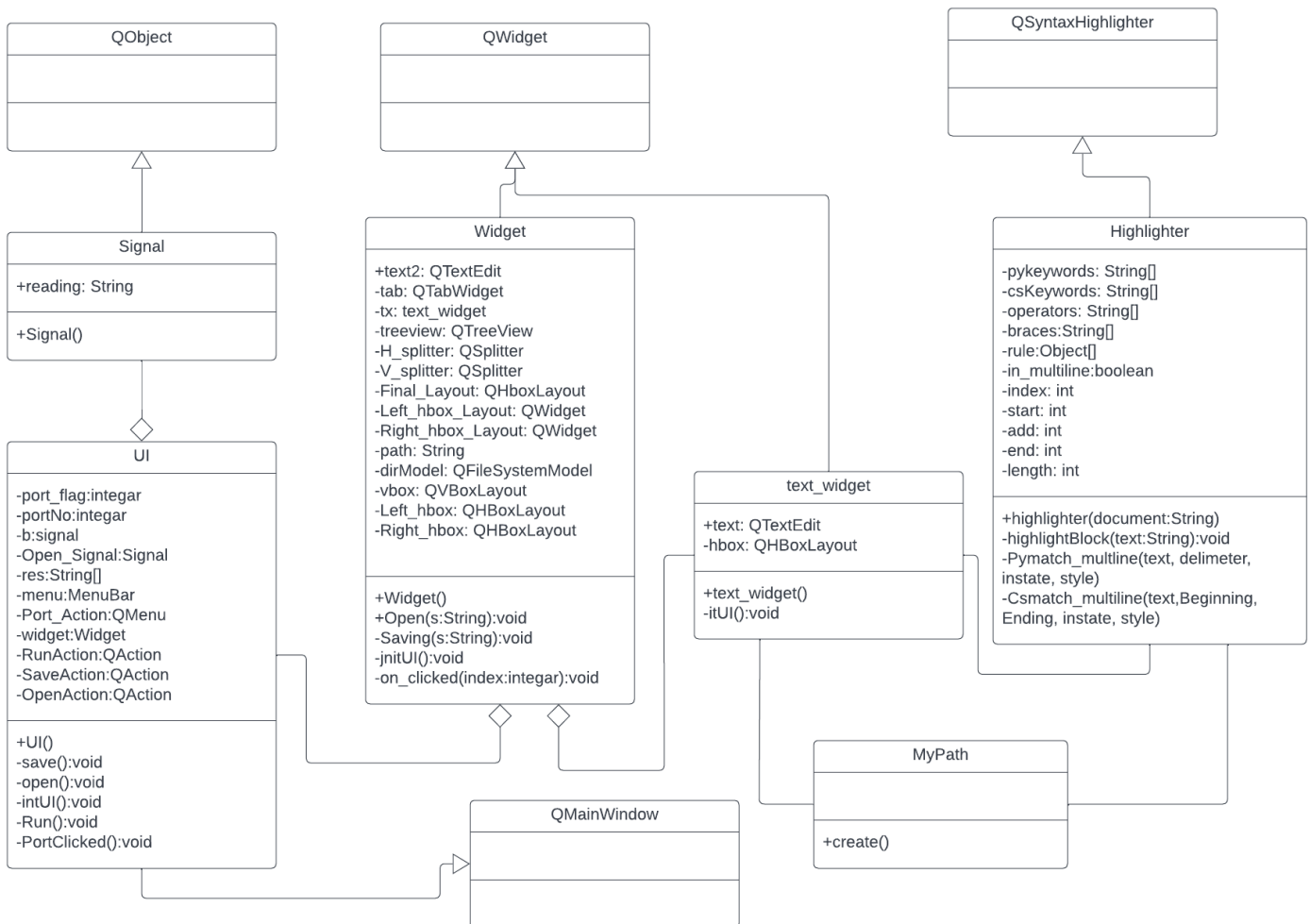


Figure 2: Class Diagram

A new class [Mypath] was added and class [Python Highlighter] name is changed to [Highlighter] to be more general to able to handle highlighting of both python and C# also its attributes and methods are modified.

3.2 Dynamic View [Sequence Diagram]

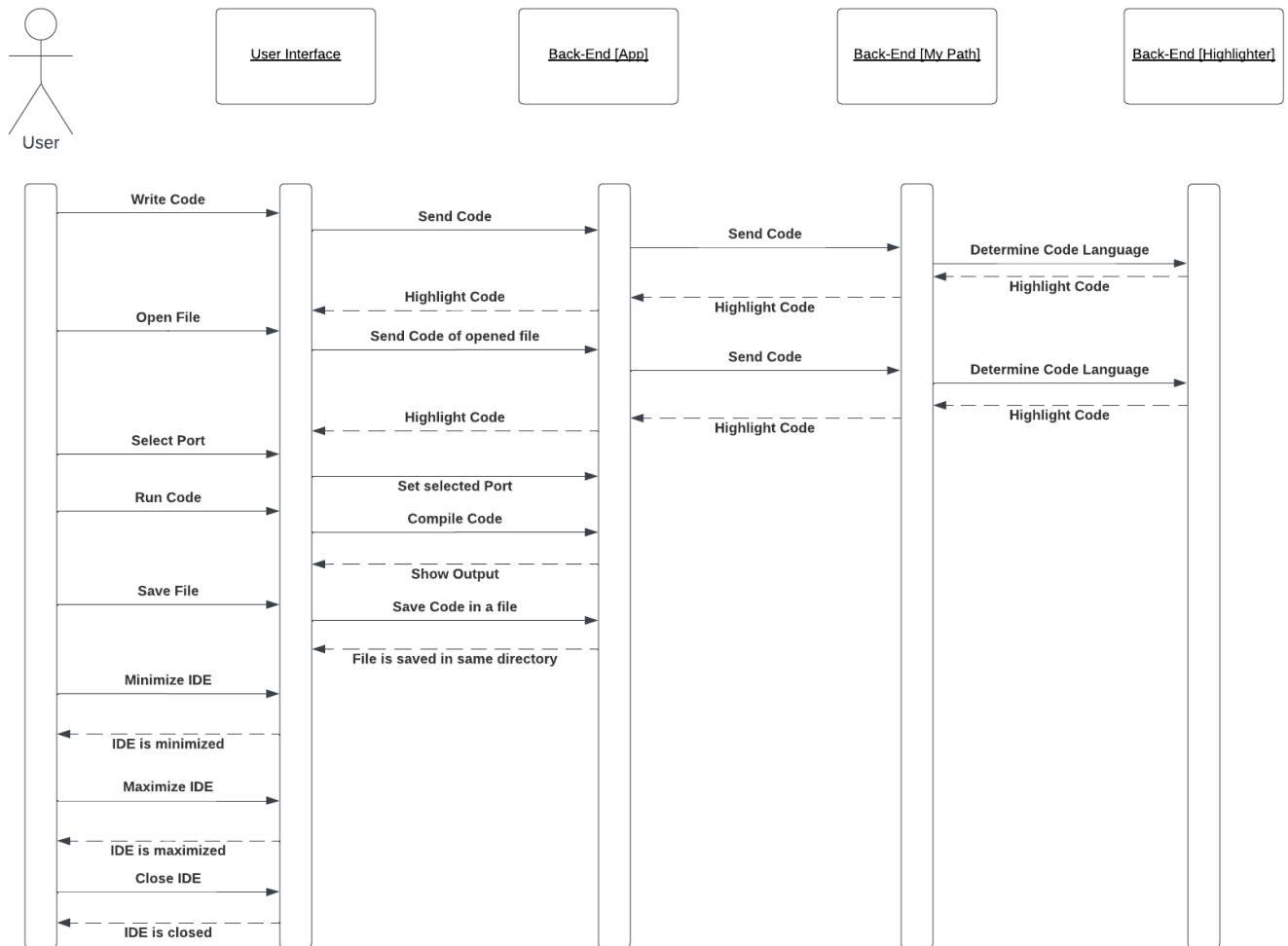


Figure 3: Sequence Diagram

A new component is added to my sequence diagram which is [My Path] which determine the language that the editor contain if it is python or C#.

4.Screenshots

4.1 Test Sample C# code

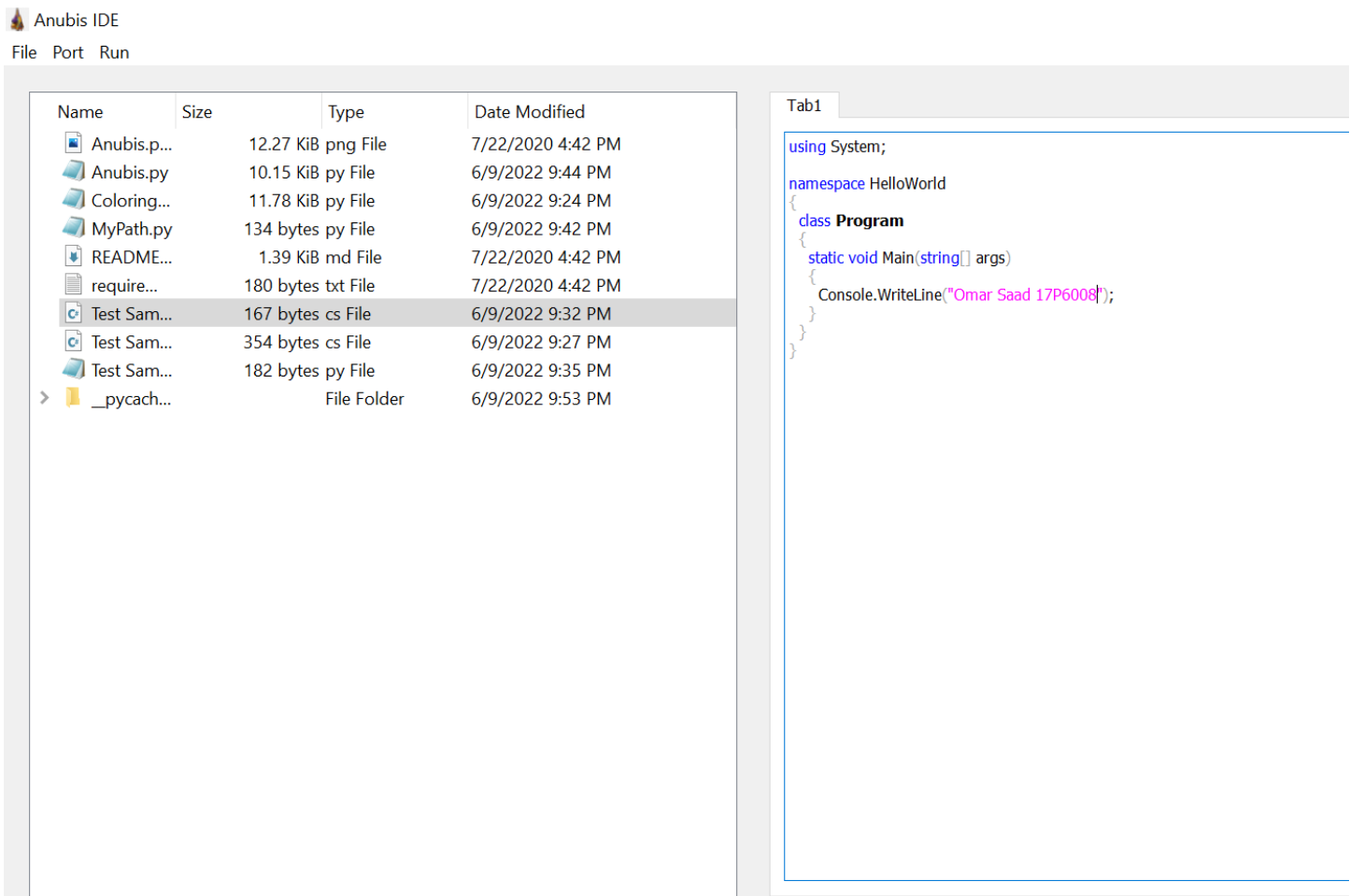


Figure 4: C# Test Code

Here we test a sample C# code:

1. We can see that syntax coloring is working correctly.
2. Class.
3. Function.
4. Console statement.

4.2 Test Sample Python code

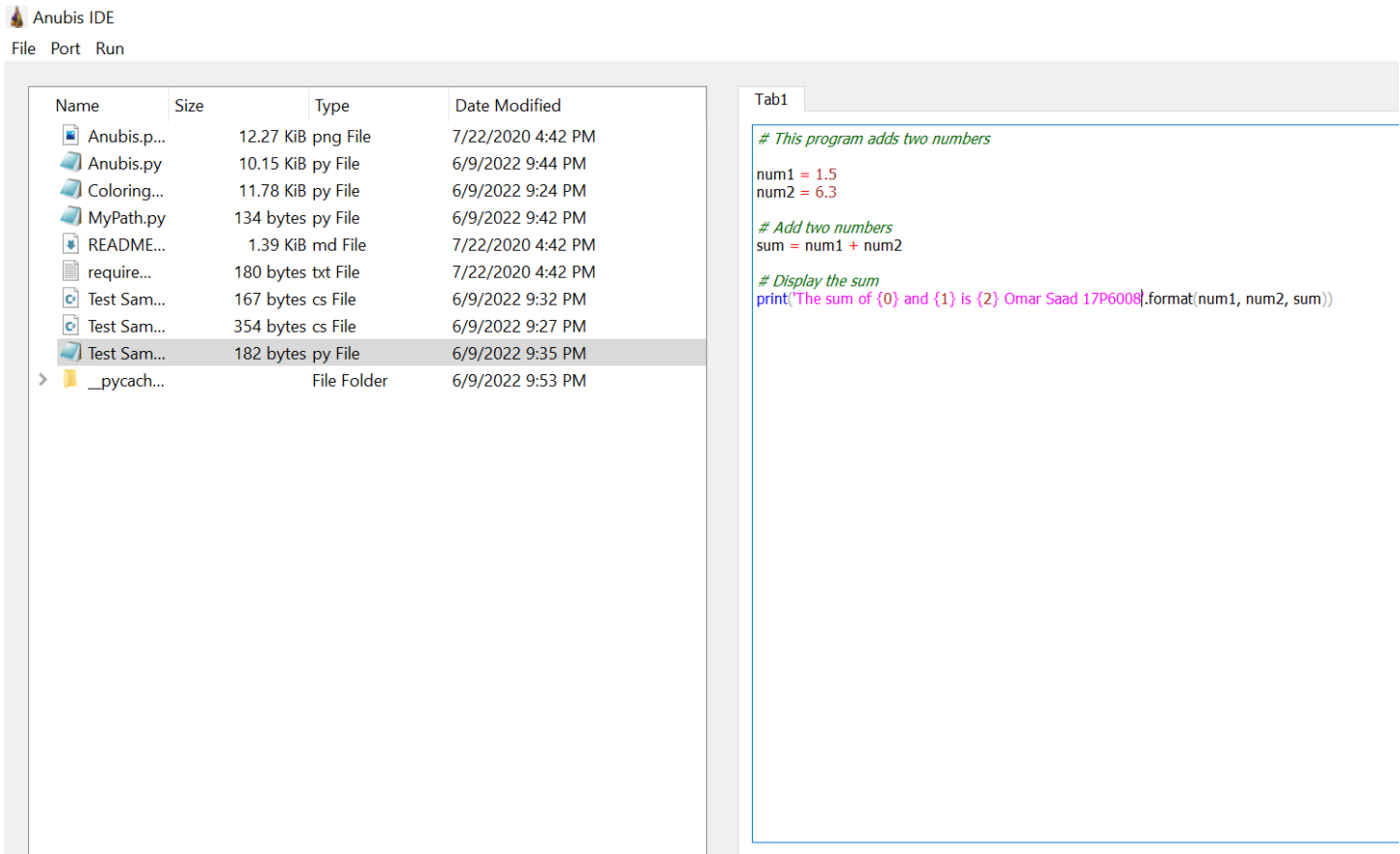


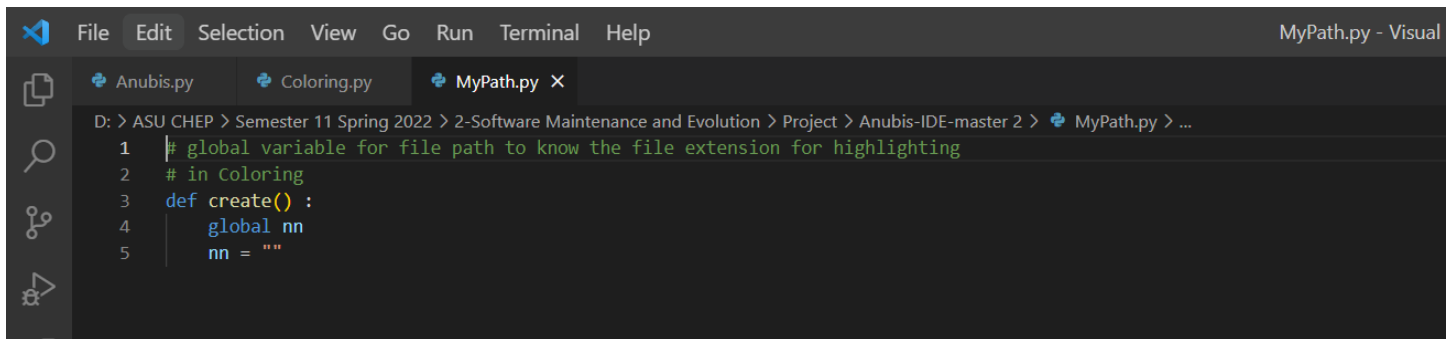
Figure 5: Python Test Code

Here we test a sample Python code:

1. We can see that syntax coloring is working correctly.
2. Addition of two variables.
3. Print Statement.

5. Code

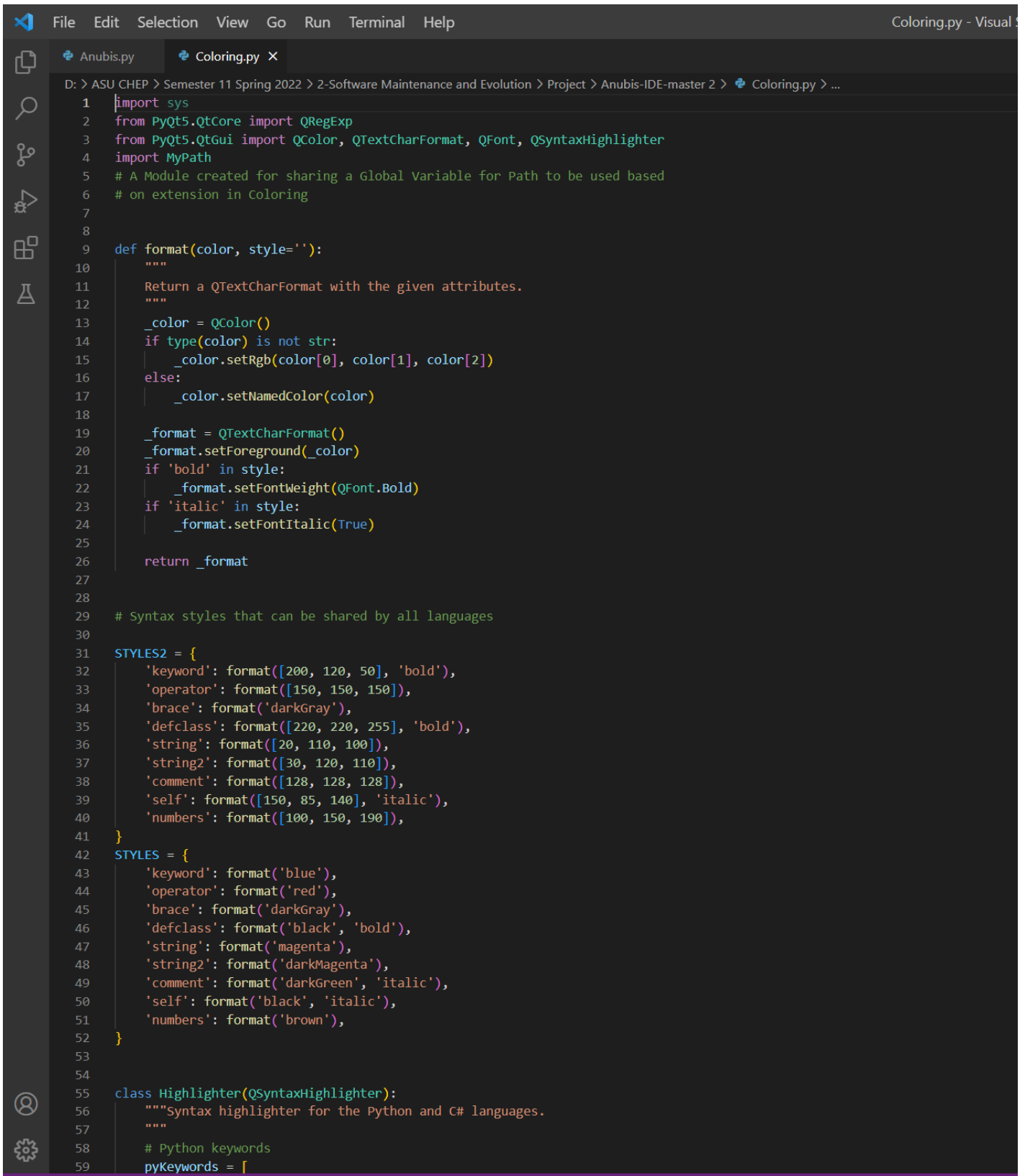
5.1 MyPath.py



```
File Edit Selection View Go Run Terminal Help MyPath.py - Visual
Anubis.py Coloring.py MyPath.py X
D: > ASU CHEP > Semester 11 Spring 2022 > 2-Software Maintenance and Evolution > Project > Anubis-IDE-master 2 > MyPath.py > ...
1 # global variable for file path to know the file extension for highlighting
2 # in Coloring
3 def create() :
4     global nn
5     nn = ''
```

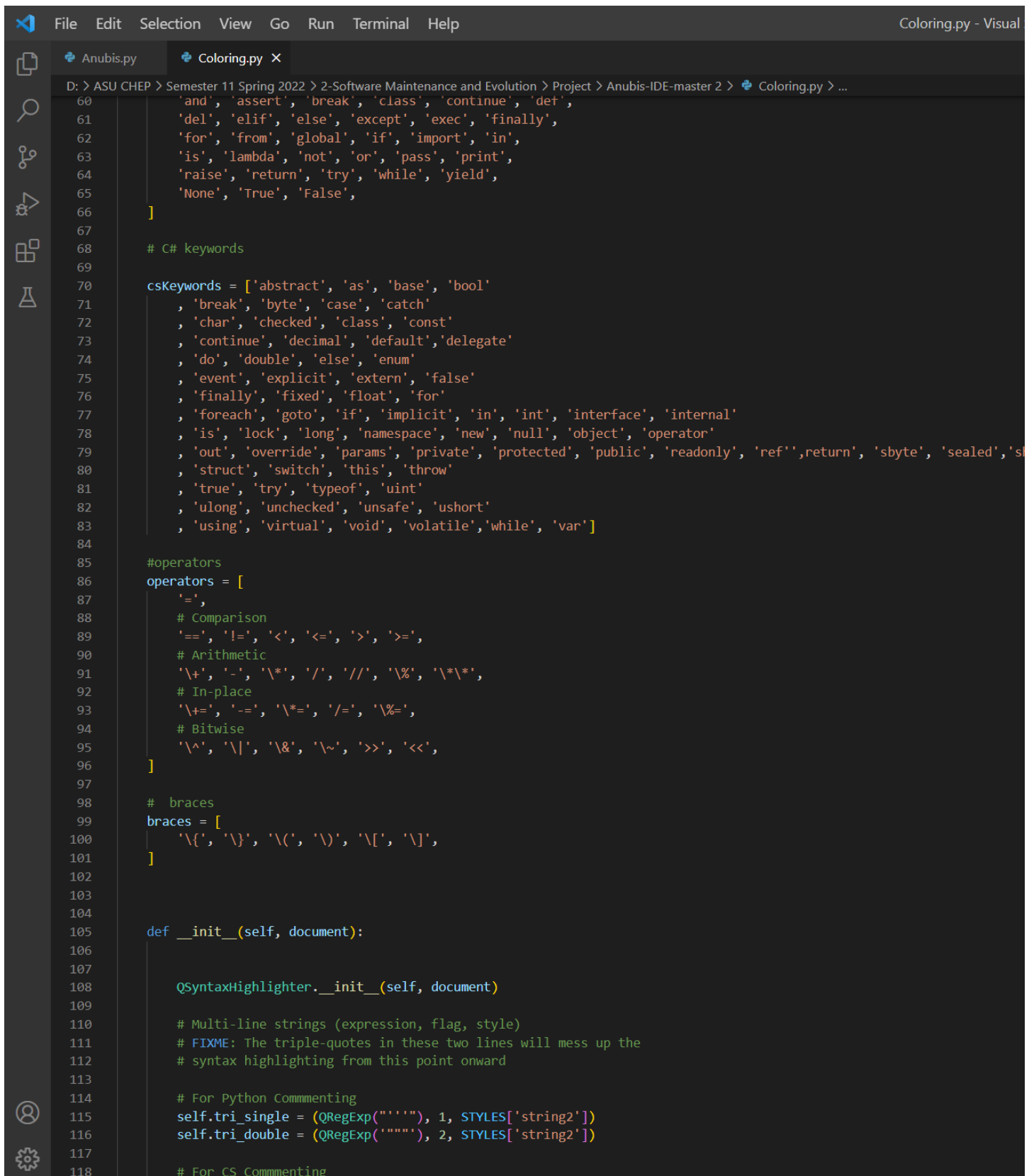
Figure 6: MyPath [1]

5.2 Coloring.py



```
1 import sys
2 from PyQt5.QtCore import QRegExp
3 from PyQt5.QtGui import QColor, QTextCharFormat, QFont, QSyntaxHighlighter
4 import MyPath
5 # A Module created for sharing a Global Variable for Path to be used based
6 # on extension in Coloring
7
8
9 def format(color, style=''):
10     """
11     Return a QTextCharFormat with the given attributes.
12     """
13     _color = QColor()
14     if type(color) is not str:
15         _color.setRgb(color[0], color[1], color[2])
16     else:
17         _color.setNamedColor(color)
18
19     _format = QTextCharFormat()
20     _format.setForeground(_color)
21     if 'bold' in style:
22         _format.setFontWeight(QFont.Bold)
23     if 'italic' in style:
24         _format.setFontItalic(True)
25
26     return _format
27
28
29 # Syntax styles that can be shared by all languages
30
31 STYLES2 = {
32     'keyword': format([200, 120, 50], 'bold'),
33     'operator': format([150, 150, 150]),
34     'brace': format('darkGray'),
35     'defclass': format([220, 220, 255], 'bold'),
36     'string': format([20, 110, 100]),
37     'string2': format([30, 120, 110]),
38     'comment': format([128, 128, 128]),
39     'self': format([150, 85, 140], 'italic'),
40     'numbers': format([100, 150, 190]),
41 }
42
43 STYLES = {
44     'keyword': format('blue'),
45     'operator': format('red'),
46     'brace': format('darkGray'),
47     'defclass': format('black', 'bold'),
48     'string': format('magenta'),
49     'string2': format('darkMagenta'),
50     'comment': format('darkGreen', 'italic'),
51     'self': format('black', 'italic'),
52     'numbers': format('brown'),
53 }
54
55 class Highlighter(QSyntaxHighlighter):
56     """Syntax highlighter for the Python and C# languages.
57     """
58     # Python keywords
59     pyKeywords = [
```

Figure 7: Coloring [1]



```
File Edit Selection View Go Run Terminal Help
Coloring.py - Visual

Anubis.py Coloring.py X
D: > ASU CHEP > Semester 11 Spring 2022 > 2-Software Maintenance and Evolution > Project > Anubis-IDE-master 2 > Coloring.py > ...
60 'and', 'assert', 'break', 'class', 'continue', 'def',
61 'del', 'elif', 'else', 'except', 'exec', 'finally',
62 'for', 'from', 'global', 'if', 'import', 'in',
63 'is', 'lambda', 'not', 'or', 'pass', 'print',
64 'raise', 'return', 'try', 'while', 'yield',
65 'None', 'True', 'False',
66 ]
67
68 # C# keywords
69
70 csKeywords = ['abstract', 'as', 'base', 'bool'
71 , 'break', 'byte', 'case', 'catch'
72 , 'char', 'checked', 'class', 'const'
73 , 'continue', 'decimal', 'default', 'delegate'
74 , 'do', 'double', 'else', 'enum'
75 , 'event', 'explicit', 'extern', 'false'
76 , 'finally', 'fixed', 'float', 'for'
77 , 'foreach', 'goto', 'if', 'implicit', 'in', 'int', 'interface', 'internal'
78 , 'is', 'lock', 'long', 'namespace', 'new', 'null', 'object', 'operator'
79 , 'out', 'override', 'params', 'private', 'protected', 'public', 'readonly', 'ref', 'return', 'sbyte', 'sealed', 's
80 , 'struct', 'switch', 'this', 'throw'
81 , 'true', 'try', 'typeof', 'uint'
82 , 'ulong', 'unchecked', 'unsafe', 'ushort'
83 , 'using', 'virtual', 'void', 'volatile', 'while', 'var']
84
85 #operators
86 operators = [
87     '=',
88     # Comparison
89     '==', '!=', '<', '<=', '>', '>=',
90     # Arithmetic
91     '+', '-', '*', '/', '//', '%', '*%',
92     # In-place
93     '+=', '-=', '*=', '/=', '%=',
94     # Bitwise
95     '^', '|', '&', '~', '>>', '<<',
96 ]
97
98 # braces
99 braces = [
100     '{', '}', '(', ')', '[', ']',
101 ]
102
103
104
105 def __init__(self, document):
106
107     QSyntaxHighlighter.__init__(self, document)
108
109     # Multi-line strings (expression, flag, style)
110     # FIXME: The triple-quotes in these two lines will mess up the
111     # syntax highlighting from this point onward
112
113     # For Python Commenting
114     self.tri_single = (QRegExp("'''"), 1, STYLES['string2'])
115     self.tri_double = (QRegExp('"""'), 2, STYLES['string2'])
116
117     # For CS Commenting
118
```

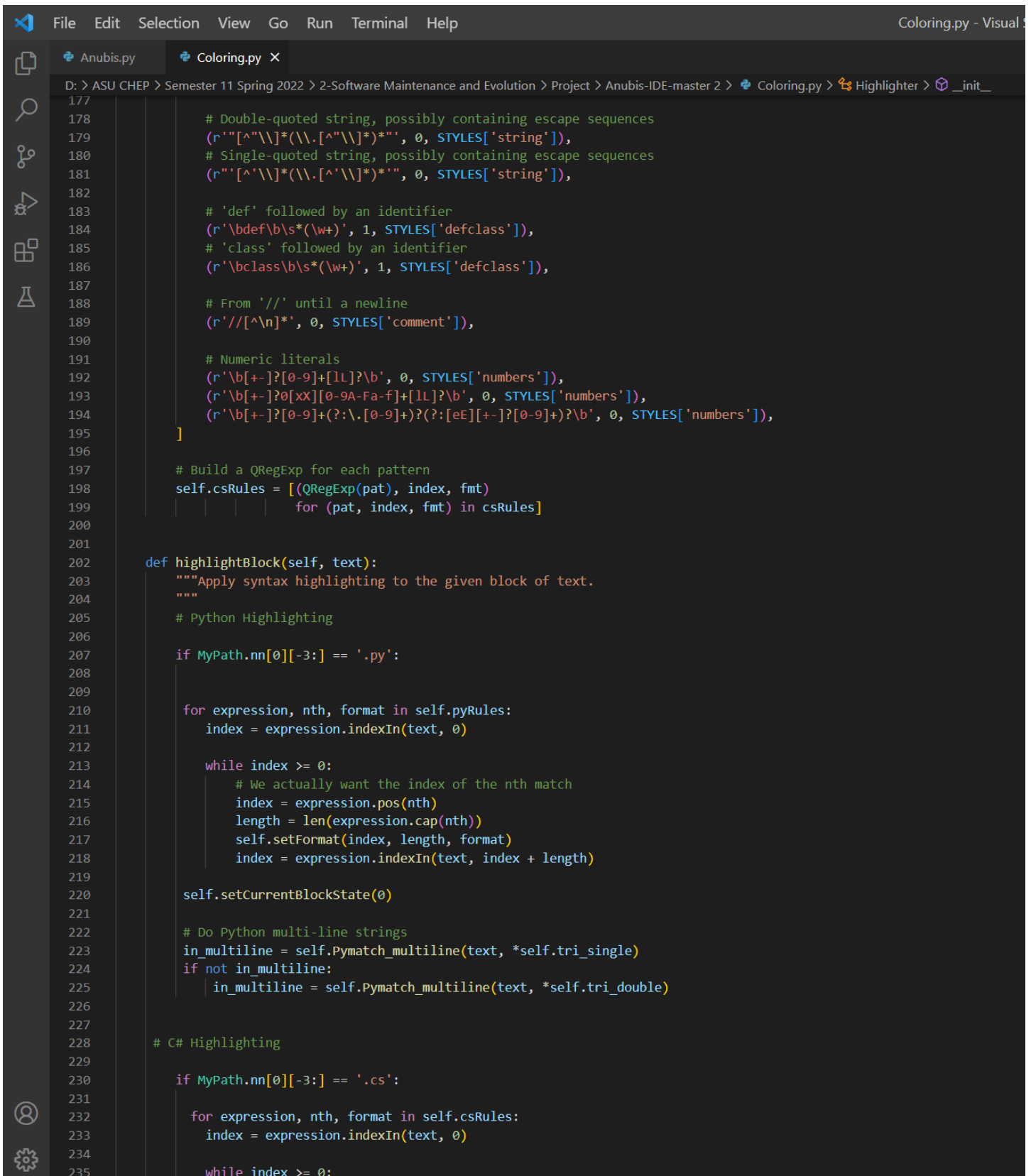
Figure 8: Coloring [2]

```

118 # For CS Commenting
119 self.CS_Comment = (QRegExp('/\/*'), QRegExp('\*/'), 3, STYLES['string2'])
120
121
122 # Python regular Expression Rules
123 pyRules = []
124
125 # Keyword, operator, and brace pyRules
126 pyRules += [(r'\b%s\b' % w, 0, STYLES['keyword'])
127             for w in Highlighter.pyKeywords]
128 pyRules += [(r'%s' % o, 0, STYLES['operator'])
129             for o in Highlighter.operators]
130 pyRules += [(r'%s' % b, 0, STYLES['brace'])
131             for b in Highlighter.braces]
132
133 # All other pyRules
134 pyRules += [
135     # 'self'
136     (r'\bself\b', 0, STYLES['self']),
137
138     # Double-quoted string, possibly containing escape sequences
139     (r'"[^\\]*(\\.[^\\"]*)"', 0, STYLES['string']),
140     # Single-quoted string, possibly containing escape sequences
141     (r"'[^\\]*(\\.[^\\'])*'", 0, STYLES['string']),
142
143     # 'def' followed by an identifier
144     (r'\bdef\b\s*(\w+)', 1, STYLES['defclass']),
145     # 'class' followed by an identifier
146     (r'\bclass\b\s*(\w+)', 1, STYLES['defclass']),
147
148     # From '#' until a newline
149     (r'#[^\n]*', 0, STYLES['comment']),
150
151     # Numeric literals
152     (r'\b[+-]?[0-9]+[lL]?[bB]', 0, STYLES['numbers']),
153     (r'\b[+-]?0[xX][0-9A-Fa-f]+[lL]?[bB]', 0, STYLES['numbers']),
154     (r'\b[+-]?[0-9]+(?:\.[0-9]+)?(?:[eE][+-]?[0-9]+)?[bB]', 0, STYLES['numbers']),
155 ]
156
157 # Build a QRegExp for each pattern
158 self.pyRules = [(QRegExp(pat), index, fmt)
159                 for (pat, index, fmt) in pyRules]
160
161
162 # C# regular Expression Rules
163 csRules = []
164
165 # Keyword, operator, and brace C# Rules
166 csRules += [(r'\b%s\b' % w, 0, STYLES['keyword'])
167             for w in Highlighter.csKeywords]
168 csRules += [(r'%s' % o, 0, STYLES['operator'])
169             for o in Highlighter.operators]
170 csRules += [(r'%s' % b, 0, STYLES['brace'])
171             for b in Highlighter.braces]
172
173 # All other C# Rules
174 csRules += [
175     # 'self'
176     (r'\bself\b', 0, STYLES['self']),

```

Figure 9: Coloring [3]



The screenshot shows the Anubis-IDE interface with the 'Coloring.py' file open. The file contains a list of regular expressions and their corresponding styles for syntax highlighting. The rules are organized into two main sections: Python highlighting and C# highlighting. The Python section includes rules for double-quoted strings, single-quoted strings, 'def' and 'class' statements, comments, and numeric literals. The C# section includes rules for multi-line strings and comments. The code is written in Python and uses the 'STYLE' module for defining styles.

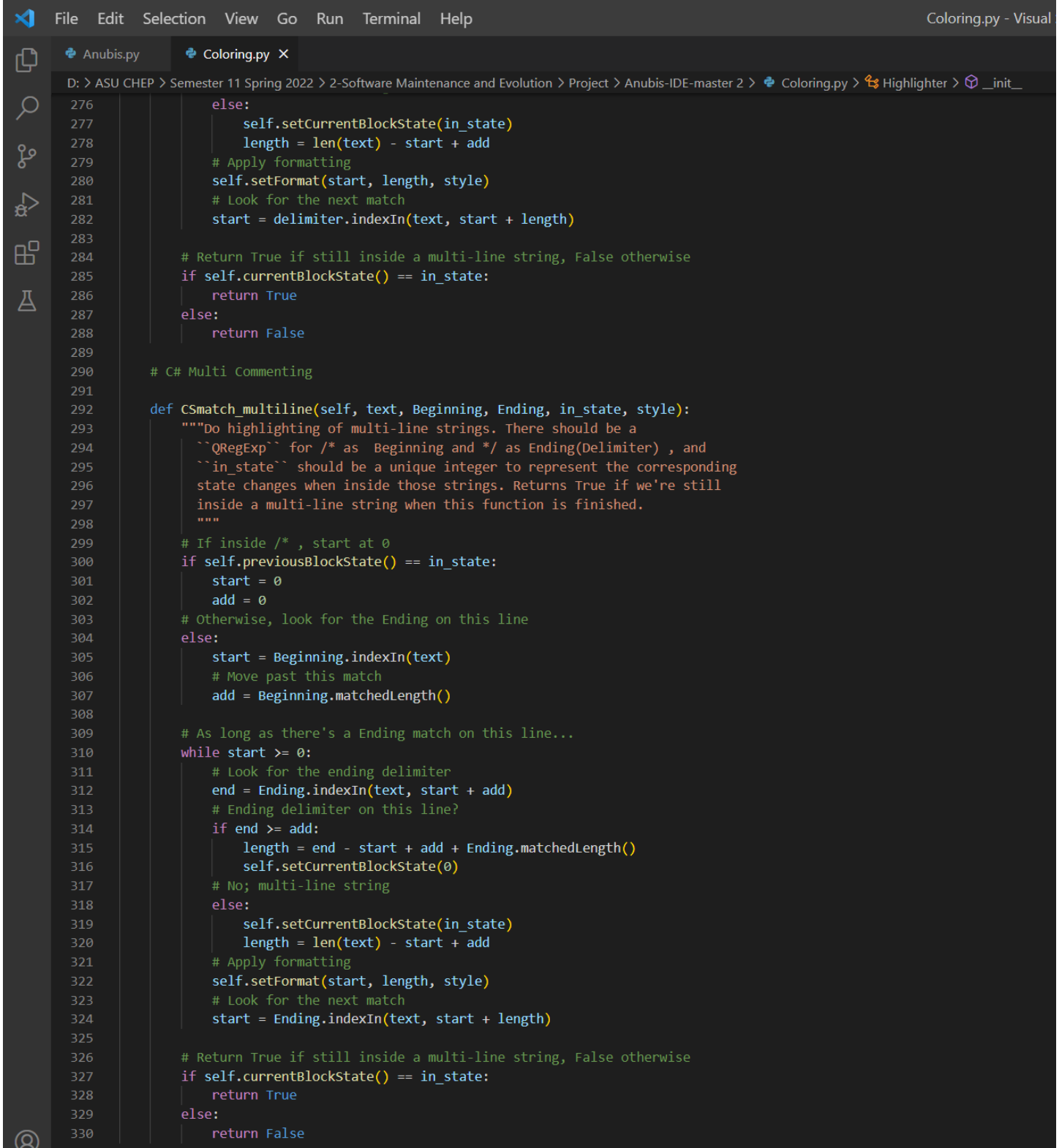
```
177
178     # Double-quoted string, possibly containing escape sequences
179     (r'"([^\\"]*(\\.["\\"]*)*)"', 0, STYLES['string']),
180     # Single-quoted string, possibly containing escape sequences
181     (r'"'([^\\"]*(\\.["\\"]*)*)"', 0, STYLES['string']),
182
183     # 'def' followed by an identifier
184     (r'\bdef\b\s*(\w+)', 1, STYLES['defclass']),
185     # 'class' followed by an identifier
186     (r'\bclass\b\s*(\w+)', 1, STYLES['defclass']),
187
188     # From '/' until a newline
189     (r'//[^\n]*', 0, STYLES['comment']),
190
191     # Numeric literals
192     (r'\b[+-]?[0-9]+[lL]?[bB]', 0, STYLES['numbers']),
193     (r'\b[+-]?[0-9]+[lL]?[bB]', 0, STYLES['numbers']),
194     (r'\b[+-]?[0-9]+(?:\.[0-9]+)?(?:[eE][+-]?[0-9]+)?[bB]', 0, STYLES['numbers']),
195 ]
196
197 # Build a QRegExp for each pattern
198 self.csRules = [(QRegExp(pat), index, fmt)
199                 for (pat, index, fmt) in csRules]
200
201
202 def highlightBlock(self, text):
203     """Apply syntax highlighting to the given block of text.
204     """
205     # Python Highlighting
206
207     if MyPath.nn[0][-3:] == '.py':
208
209         for expression, nth, format in self.pyRules:
210             index = expression.indexIn(text, 0)
211
212             while index >= 0:
213                 # We actually want the index of the nth match
214                 index = expression.pos(nth)
215                 length = len(expression.cap(nth))
216                 self.setFormat(index, length, format)
217                 index = expression.indexIn(text, index + length)
218
219         self.setCurrentBlockState(0)
220
221     # Do Python multi-line strings
222     in_multiline = self.Pymatch_multiline(text, *self.tri_single)
223     if not in_multiline:
224         in_multiline = self.Pymatch_multiline(text, *self.tri_double)
225
226
227
228     # C# Highlighting
229
230     if MyPath.nn[0][-3:] == '.cs':
231
232         for expression, nth, format in self.csRules:
233             index = expression.indexIn(text, 0)
234
235             while index >= 0:
```

Figure 10: Coloring [4]



```
219
220     self.setCurrentBlockState(0)
221
222     # Do Python multi-line strings
223     in_multiline = self.Pymatch_multiline(text, *self.tri_single)
224     if not in_multiline:
225         in_multiline = self.Pymatch_multiline(text, *self.tri_double)
226
227
228     # C# Highlighting
229
230     if MyPath.nn[0][-3:] == '.cs':
231
232         for expression, nth, format in self.csRules:
233             index = expression.indexIn(text, 0)
234
235             while index >= 0:
236                 # We actually want the index of the nth match
237                 index = expression.pos(nth)
238                 length = len(expression.cap(nth))
239                 self.setFormat(index, length, format)
240                 index = expression.indexIn(text, index + length)
241
242     self.setCurrentBlockState(0)
243
244     # Do C# Multi-Line Strings
245     self.CSmatch_multiline(text, *self.CS_Comment)
246
247
248     #Python Multi Commenting Function
249
250     def Pymatch_multiline(self, text, delimiter, in_state, style):
251         """Do highlighting of multi-line strings. ``delimiter`` should be a
252         ``QRegExp`` for triple-single-quotes or triple-double-quotes, and
253         ``in_state`` should be a unique integer to represent the corresponding
254         state changes when inside those strings. Returns True if we're still
255         inside a multi-line string when this function is finished.
256         """
257         # If inside triple-single quotes, start at 0
258         if self.previousBlockState() == in_state:
259             start = 0
260             add = 0
261         # Otherwise, look for the delimiter on this line
262         else:
263             start = delimiter.indexIn(text)
264             # Move past this match
265             add = delimiter.matchedLength()
266
267         # As long as there's a delimiter match on this line...
268         while start >= 0:
269             # Look for the ending delimiter
270             end = delimiter.indexIn(text, start + add)
271             # Ending delimiter on this line?
272             if end >= add:
273                 length = end - start + add + delimiter.matchedLength()
274                 self.setCurrentBlockState(0)
275                 # No; multi-line string
276             else:
277                 self.setCurrentBlockState(in_state)
```

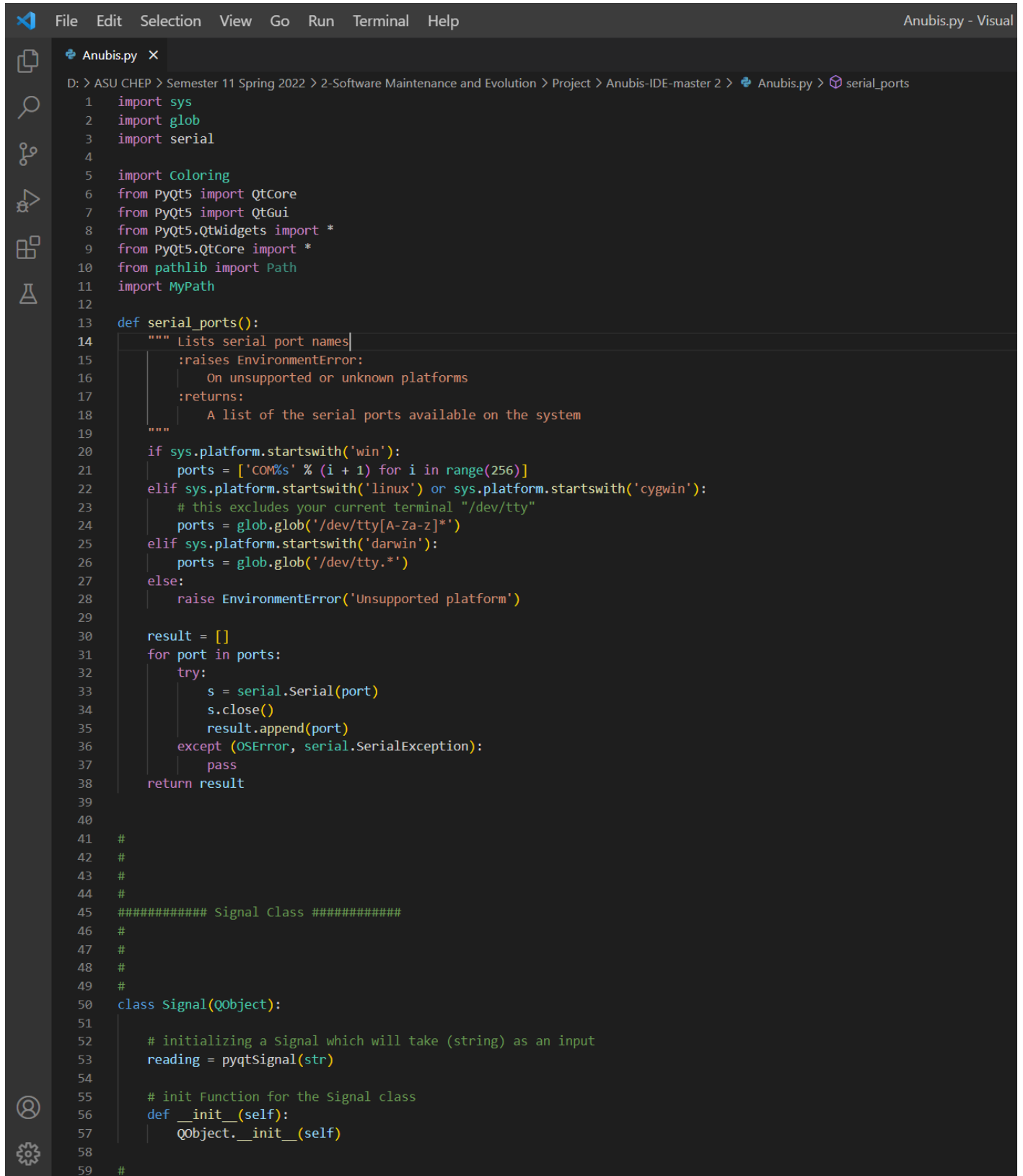
Figure 11: Coloring [5]



```
276         else:
277             self.setCurrentBlockState(in_state)
278             length = len(text) - start + add
279             # Apply formatting
280             self.setFormat(start, length, style)
281             # Look for the next match
282             start = delimiter.indexIn(text, start + length)
283
284             # Return True if still inside a multi-line string, False otherwise
285             if self.currentBlockState() == in_state:
286                 return True
287             else:
288                 return False
289
290     # C# Multi Commenting
291
292     def CSmatch_multiline(self, text, Beginning, Ending, in_state, style):
293         """Do highlighting of multi-line strings. There should be a
294         ``QRegExp`` for /* as Beginning and */ as Ending(Delimiter) , and
295         ``in_state`` should be a unique integer to represent the corresponding
296         state changes when inside those strings. Returns True if we're still
297         inside a multi-line string when this function is finished.
298         """
299         # If inside /* , start at 0
300         if self.previousBlockState() == in_state:
301             start = 0
302             add = 0
303         # Otherwise, look for the Ending on this line
304         else:
305             start = Beginning.indexIn(text)
306             # Move past this match
307             add = Beginning.matchedLength()
308
309         # As long as there's a Ending match on this line...
310         while start >= 0:
311             # Look for the ending delimiter
312             end = Ending.indexIn(text, start + add)
313             # Ending delimiter on this line?
314             if end >= add:
315                 length = end - start + add + Ending.matchedLength()
316                 self.setCurrentBlockState(0)
317             # No; multi-line string
318             else:
319                 self.setCurrentBlockState(in_state)
320                 length = len(text) - start + add
321             # Apply formatting
322             self.setFormat(start, length, style)
323             # Look for the next match
324             start = Ending.indexIn(text, start + length)
325
326         # Return True if still inside a multi-line string, False otherwise
327         if self.currentBlockState() == in_state:
328             return True
329         else:
330             return False
```

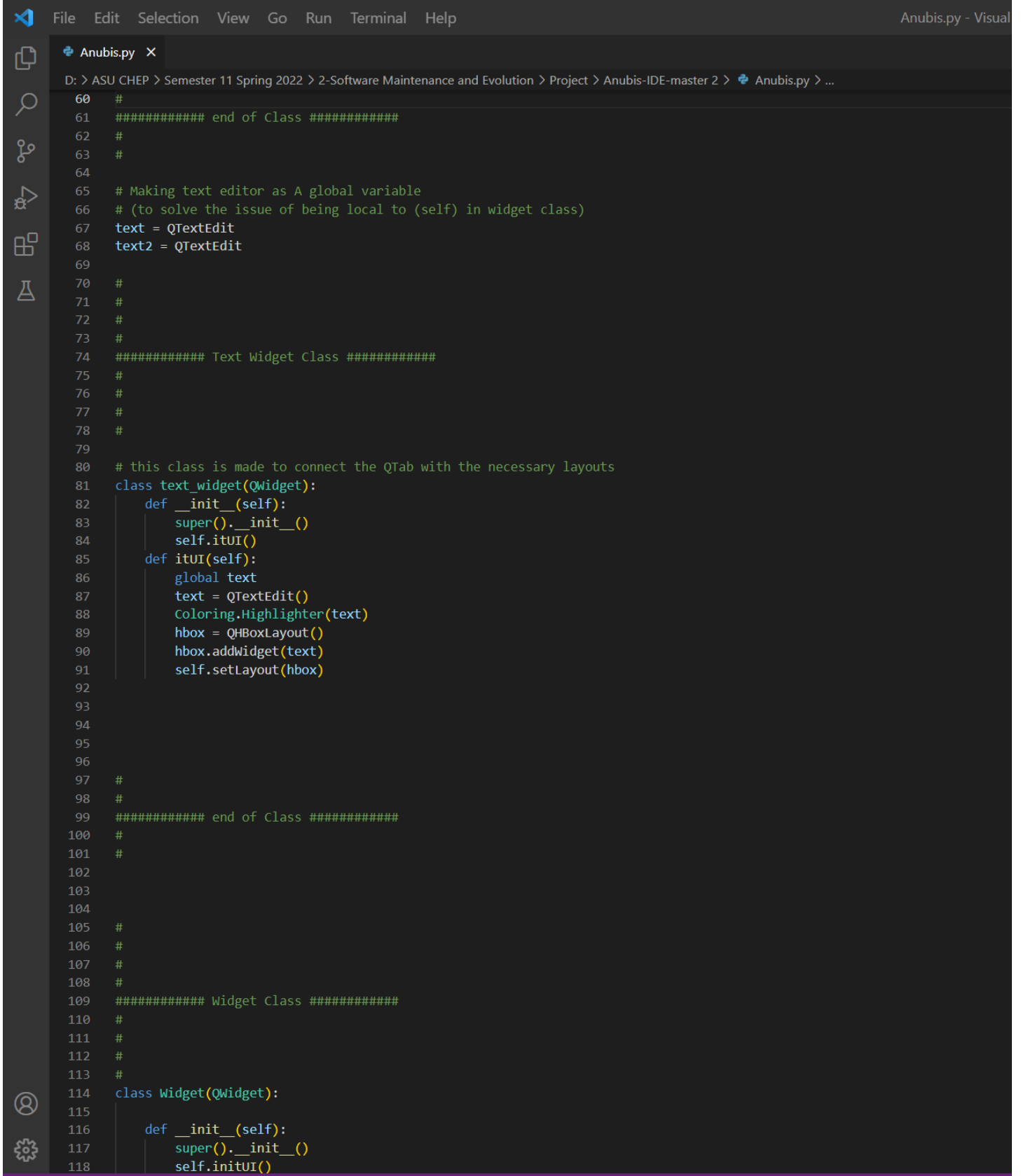
Figure 12: Coloring [6]

5.3 Anubis.py



```
File Edit Selection View Go Run Terminal Help Anubis.py - Visual
Anubis.py X
D: > ASU CHEP > Semester 11 Spring 2022 > 2-Software Maintenance and Evolution > Project > Anubis-IDE-master 2 > Anubis.py > serial_ports
1 import sys
2 import glob
3 import serial
4
5 import Coloring
6 from PyQt5 import QtCore
7 from PyQt5 import QtGui
8 from PyQt5.QtWidgets import *
9 from PyQt5.QtCore import *
10 from pathlib import Path
11 import MyPath
12
13 def serial_ports():
14     """ Lists serial port names|
15         :raises EnvironmentError:
16             On unsupported or unknown platforms
17         :returns:
18             A list of the serial ports available on the system
19     """
20     if sys.platform.startswith('win'):
21         ports = ['COM%s' % (i + 1) for i in range(256)]
22     elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
23         # this excludes your current terminal "/dev/tty"
24         ports = glob.glob('/dev/tty[A-Za-z]*')
25     elif sys.platform.startswith('darwin'):
26         ports = glob.glob('/dev/tty.*')
27     else:
28         raise EnvironmentError('Unsupported platform')
29
30     result = []
31     for port in ports:
32         try:
33             s = serial.Serial(port)
34             s.close()
35             result.append(port)
36         except (OSError, serial.SerialException):
37             pass
38     return result
39
40
41 #
42 #
43 #
44 #
45 ##### Signal Class #####
46 #
47 #
48 #
49 #
50 class Signal(QObject):
51
52     # initializing a Signal which will take (string) as an input
53     reading = pyqtSignal(str)
54
55     # init Function for the Signal class
56     def __init__(self):
57         QObject.__init__(self)
58
59 #
```

Figure 13: Anubis [1]



The image shows a screenshot of the Anubis.py IDE. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar on the right says "Anubis.py - Visual". The left sidebar contains icons for file explorer, search, source control, and other IDE features. The main editor area displays a Python script for a text widget class. The script includes comments, class definitions, and method implementations. The code is as follows:

```
60 #
61 ##### end of Class #####
62 #
63 #
64 #
65 # Making text editor as A global variable
66 # (to solve the issue of being local to (self) in widget class)
67 text = QTextEdit
68 text2 = QTextEdit
69
70 #
71 #
72 #
73 #
74 ##### Text Widget Class #####
75 #
76 #
77 #
78 #
79
80 # this class is made to connect the QTab with the necessary layouts
81 class text_widget(QWidget):
82     def __init__(self):
83         super().__init__()
84         self.itUI()
85     def itUI(self):
86         global text
87         text = QTextEdit()
88         Coloring.Highlighter(text)
89         hbox = QHBoxLayout()
90         hbox.addWidget(text)
91         self.setLayout(hbox)
92
93
94
95
96
97 #
98 #
99 ##### end of Class #####
100 #
101 #
102
103
104
105 #
106 #
107 #
108 #
109 ##### Widget Class #####
110 #
111 #
112 #
113 #
114 class Widget(QWidget):
115
116     def __init__(self):
117         super().__init__()
118         self.initUI()
```

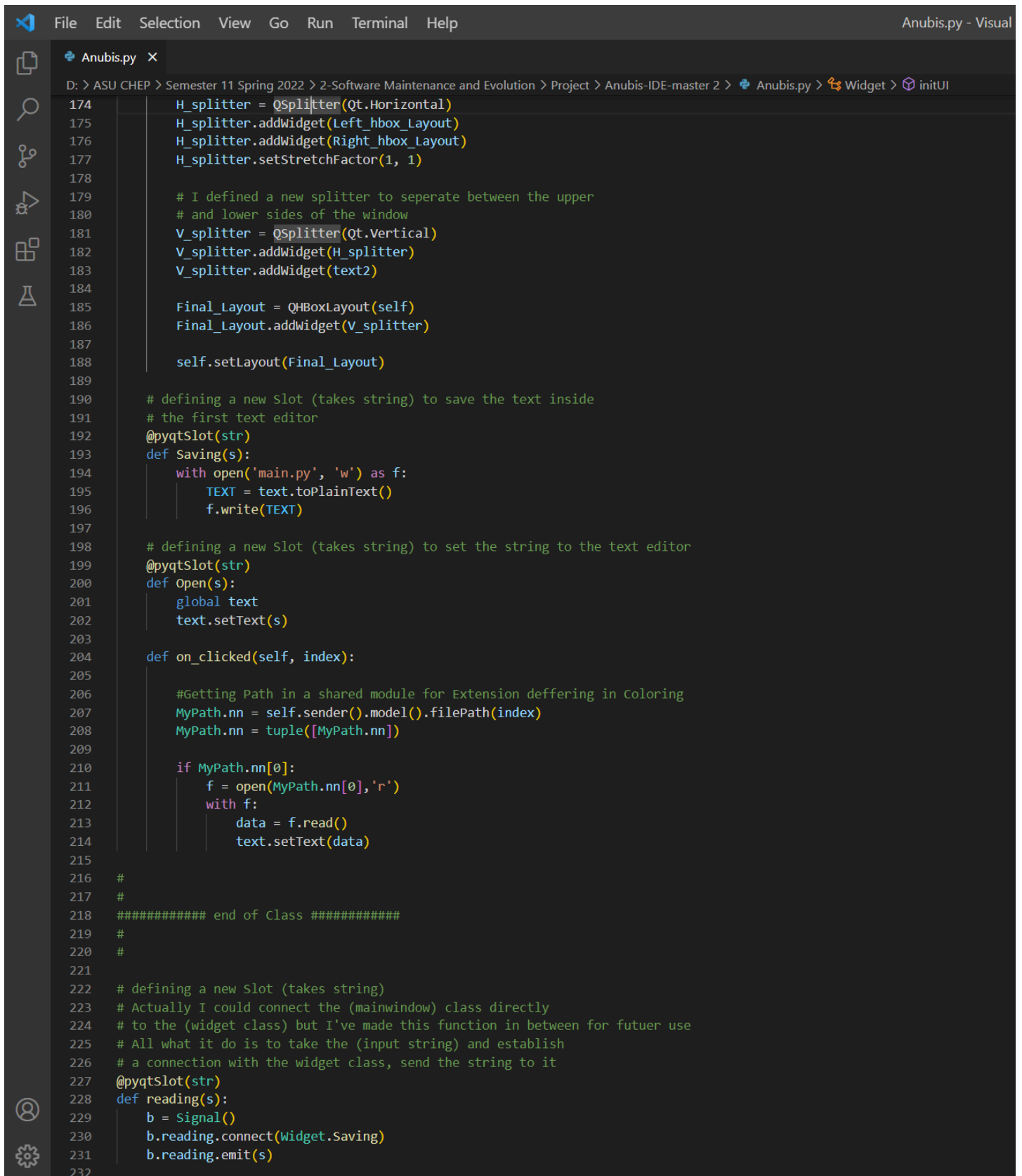
Figure 14: Anubis [2]



```
File Edit Selection View Go Run Terminal Help
Anubis.py X
D: > ASU CHEP > Semester 11 Spring 2022 > 2-Software Maintenance and Evolution > Project > Anubis-IDE-master 2 > Anubis.py > Widget

115
116     def __init__(self):
117         super().__init__()
118         self.initUI()
119
120     def initUI(self):
121
122         # This widget is responsible of making Tab in IDE which makes the
123         # Text editor looks nice
124         tab = QTabWidget()
125         tx = text_widget()
126         tab.addTab(tx, "Tab"+"1")
127
128         # second editor in which the error messages and succeeded connections will be shown
129         global text2
130         text2 = QTextEdit()
131         text2.setReadOnly(True)
132         # defining a Treeview variable to use it in showing the directory included files
133         self.treeview = QTreeView()
134
135         # making a variable (path) and setting it to the root path
136         # (surely I can set it to whatever the root I want, not the default)
137         #path = QDir.rootPath()
138
139         path = QDir.currentPath()
140
141         # making a FileSystem variable, setting its root path and applying somefilters
142         # (which I need) on it
143         self.dirModel = QFileSystemModel()
144         self.dirModel.setRootPath(QDir.rootPath())
145
146         # NoDotAndDotDot => Do not list the special entries "." and "..".
147         # AllDirs =>List all directories; i.e. don't apply the filters to directory names.
148         # Files => List files.
149         self.dirModel.setFilter(QDir.NoDotAndDotDot | QDir.AllDirs | QDir.Files)
150         self.treeview.setModel(self.dirModel)
151         self.treeview.setRootIndex(self.dirModel.index(path))
152         self.treeview.clicked.connect(self.on_clicked)
153
154         vbox = QVBoxLayout()
155         Left_hbox = QHBoxLayout()
156         Right_hbox = QHBoxLayout()
157
158         # after defining variables of type QVBoxLayout and QHBoxLayout
159         # I will Assign treeviews variable to the left one and the first text editor
160         # in which the code will be written to the right one
161         Left_hbox.addWidget(self.treeview)
162         Right_hbox.addWidget(tab)
163
164         # defining another variable of type QWidget to set its layout as an QHBoxLayout
165         # I will do the same with the right one
166         Left_hbox_Layout = QWidget()
167         Left_hbox_Layout.setLayout(Left_hbox)
168
169         Right_hbox_Layout = QWidget()
170         Right_hbox_Layout.setLayout(Right_hbox)
171
172         # I defined a splitter to separate the two variables (left, right) and
173         # make it more easily to change the space between them
```

Figure 15: Anubis [3]



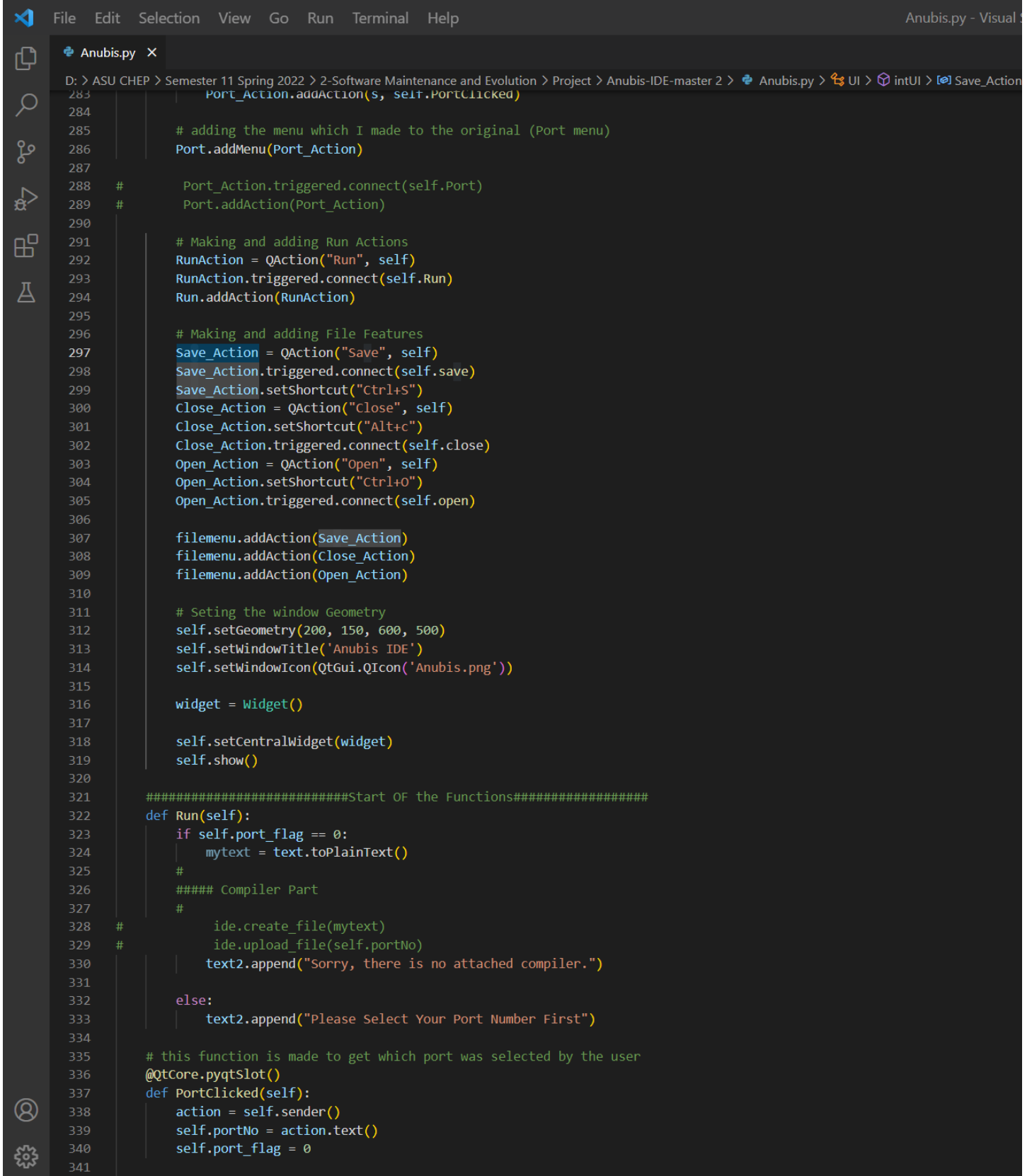
```
174 H_splitter = QSplitter(Qt.Horizontal)
175 H_splitter.addWidget(Left_hbox_Layout)
176 H_splitter.addWidget(Right_hbox_Layout)
177 H_splitter.setStretchFactor(1, 1)
178
179 # I defined a new splitter to separate between the upper
180 # and lower sides of the window
181 V_splitter = QSplitter(Qt.Vertical)
182 V_splitter.addWidget(H_splitter)
183 V_splitter.addWidget(text2)
184
185 Final_Layout = QHBoxLayout(self)
186 Final_Layout.addWidget(V_splitter)
187
188 self.setLayout(Final_Layout)
189
190 # defining a new Slot (takes string) to save the text inside
191 # the first text editor
192 @pyqtSlot(str)
193 def Saving(s):
194     with open('main.py', 'w') as f:
195         TEXT = text.toPlainText()
196         f.write(TEXT)
197
198 # defining a new Slot (takes string) to set the string to the text editor
199 @pyqtSlot(str)
200 def Open(s):
201     global text
202     text.setText(s)
203
204 def on_clicked(self, index):
205
206     #Getting Path in a shared module for Extension deffering in Coloring
207     MyPath.nn = self.sender().model().filePath(index)
208     MyPath.nn = tuple([MyPath.nn])
209
210     if MyPath.nn[0]:
211         f = open(MyPath.nn[0], 'r')
212         with f:
213             data = f.read()
214             text.setText(data)
215
216 #
217 #
218 ##### end of Class #####
219 #
220 #
221
222 # defining a new Slot (takes string)
223 # Actually I could connect the (mainwindow) class directly
224 # to the (widget class) but I've made this function in between for futuer use
225 # All what it do is to take the (input string) and establish
226 # a connection with the widget class, send the string to it
227 @pyqtSlot(str)
228 def reading(s):
229     b = Signal()
230     b.reading.connect(Widget.Saving)
231     b.reading.emit(s)
232
```

Figure 16: Anubis [4]



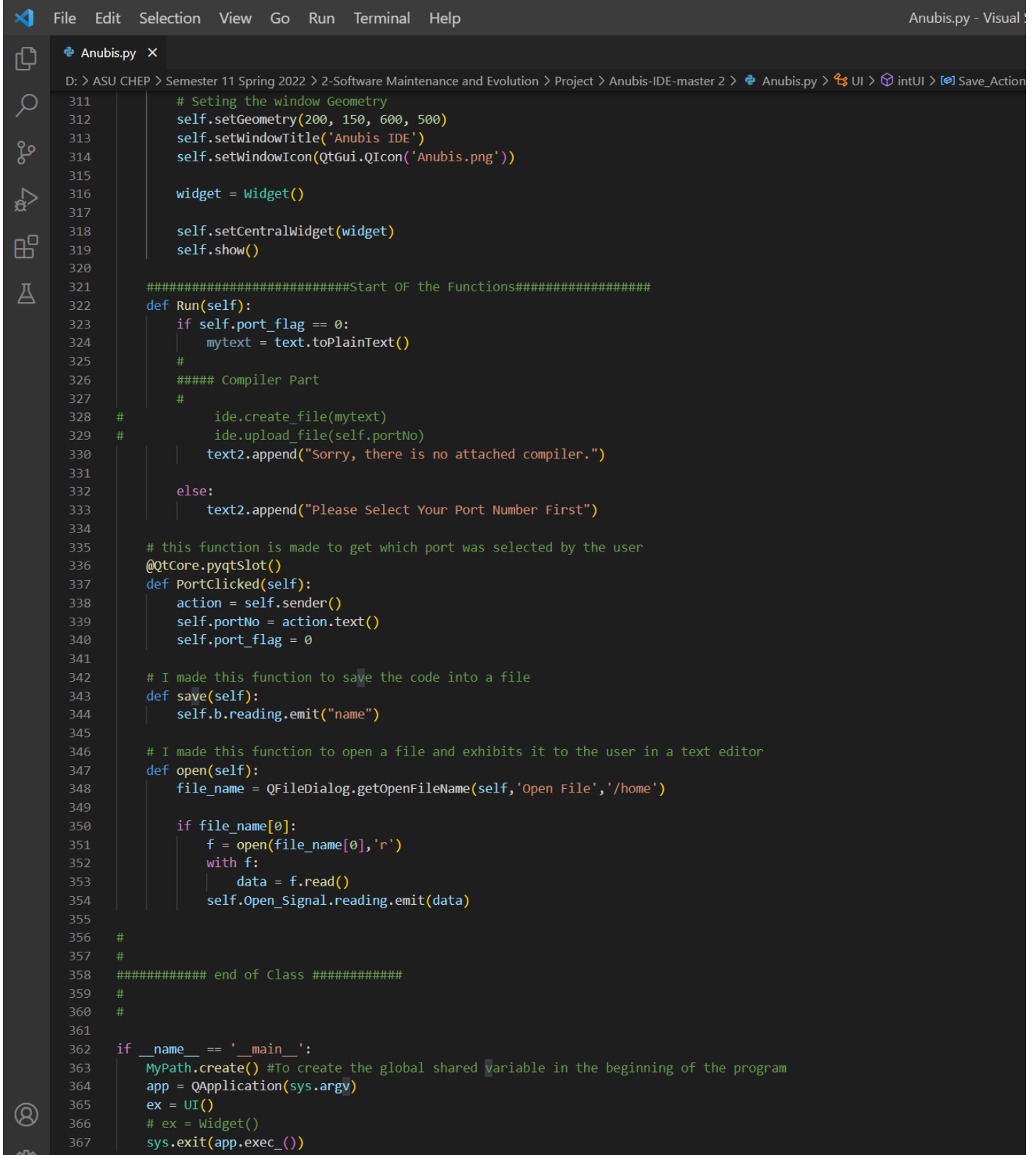
```
232 |
233 | # same as reading Function
234 | @pyqtSlot(str)
235 | def Openning(s):
236 |     b = Signal()
237 |     b.reading.connect(Widget.Open)
238 |     b.reading.emit(s)
239 | #
240 | #
241 | #
242 | #
243 | ##### MainWindow Class #####
244 | #
245 | #
246 | #
247 | #
248 | class UI(QMainWindow):
249 |     def __init__(self):
250 |         super().__init__()
251 |         self.intUI()
252 |
253 |     def intUI(self):
254 |         self.port_flag = 1
255 |         self.b = Signal()
256 |
257 |         self.Open_Signal = Signal()
258 |
259 |         # connecting (self.Open_Signal) with Openning function
260 |         self.Open_Signal.reading.connect(Openning)
261 |
262 |         # connecting (self.b) with reading function
263 |         self.b.reading.connect(reading)
264 |
265 |         # creating menu items
266 |         menu = self.menuBar()
267 |
268 |         # I have three menu items
269 |         filemenu = menu.addMenu('File')
270 |         Port = menu.addMenu('Port')
271 |         Run = menu.addMenu('Run')
272 |
273 |         # As any PC or laptop have many ports, so I need to list them to the User
274 |         # so I made (Port_Action) to add the Ports got from (serial_ports()) function
275 |         # copyrights of serial_ports() function goes back to a guy from stackoverflow
276 |         # (whome I can't remember his name), so thank you (unknown)
277 |         Port_Action = QMenu('port', self)
278 |
279 |         res = serial_ports()
280 |
281 |         for i in range(len(res)):
282 |             s = res[i]
283 |             Port_Action.addAction(s, self.PortClicked)
284 |
285 |         # adding the menu which I made to the original (Port menu)
286 |         Port.addAction(Port_Action)
287 |
288 |         # Port_Action.triggered.connect(self.Port)
289 |         # Port.addAction(Port_Action)
290 |
```

Figure 17: Anubis [5]



```
File Edit Selection View Go Run Terminal Help
Anubis.py - Visual Studio Code
Anubis.py X
D: > ASU CHEP > Semester 11 Spring 2022 > 2-Software Maintenance and Evolution > Project > Anubis-IDE-master 2 > Anubis.py > UI > intUI > Save_Action
283 Port_Action.addAction(S, self.PortClicked)
284
285 # adding the menu which I made to the original (Port menu)
286 Port.addAction(Port_Action)
287
288 # Port_Action.triggered.connect(self.Port)
289 # Port.addAction(Port_Action)
290
291 # Making and adding Run Actions
292 RunAction = QAction("Run", self)
293 RunAction.triggered.connect(self.Run)
294 Run.addAction(RunAction)
295
296 # Making and adding File Features
297 Save_Action = QAction("Save", self)
298 Save_Action.triggered.connect(self.save)
299 Save_Action.setShortcut("Ctrl+S")
300 Close_Action = QAction("Close", self)
301 Close_Action.setShortcut("Alt+c")
302 Close_Action.triggered.connect(self.close)
303 Open_Action = QAction("Open", self)
304 Open_Action.setShortcut("Ctrl+O")
305 Open_Action.triggered.connect(self.open)
306
307 filemenu.addAction(Save_Action)
308 filemenu.addAction(Close_Action)
309 filemenu.addAction(Open_Action)
310
311 # Seting the window Geometry
312 self.setGeometry(200, 150, 600, 500)
313 self.setWindowTitle('Anubis IDE')
314 self.setWindowIcon(QtGui.QIcon('Anubis.png'))
315
316 widget = Widget()
317
318 self.setCentralWidget(widget)
319 self.show()
320
321 #####Start OF the Functions#####
322 def Run(self):
323     if self.port_flag == 0:
324         mytext = text.toPlainText()
325         #
326         ##### Compiler Part
327         #
328         # ide.create_file(mytext)
329         # ide.upload_file(self.portNo)
330         text2.append("Sorry, there is no attached compiler.")
331
332     else:
333         text2.append("Please Select Your Port Number First")
334
335 # this function is made to get which port was selected by the user
336 @QtCore.pyqtSlot()
337 def PortClicked(self):
338     action = self.sender()
339     self.portNo = action.text()
340     self.port_flag = 0
341
```

Figure 18: Anubis [6]



```
File Edit Selection View Go Run Terminal Help Anubis.py - Visual S
Anubis.py X
D: > ASU CHEP > Semester 11 Spring 2022 > 2-Software Maintenance and Evolution > Project > Anubis-IDE-master 2 > Anubis.py > UI > intUI > Save_Action
311     # Seting the window Geometry
312     self.setGeometry(200, 150, 600, 500)
313     self.setWindowTitle('Anubis IDE')
314     self.setWindowIcon(QtGui.QIcon('Anubis.png'))
315
316     widget = Widget()
317
318     self.setCentralWidget(widget)
319     self.show()
320
321     #####Start OF the Functions#####
322     def Run(self):
323         if self.port_flag == 0:
324             mytext = text.toPlainText()
325             #
326             ##### Compiler Part
327             #
328             #         ide.create_file(mytext)
329             #         ide.upload_file(self.portNo)
330             text2.append("Sorry, there is no attached compiler.")
331
332         else:
333             text2.append("Please Select Your Port Number First")
334
335     # this function is made to get which port was selected by the user
336     @QtCore.pyqtSlot()
337     def PortClicked(self):
338         action = self.sender()
339         self.portNo = action.text()
340         self.port_flag = 0
341
342     # I made this function to save the code into a file
343     def save(self):
344         self.b.reading.emit("name")
345
346     # I made this function to open a file and exhibits it to the user in a text editor
347     def open(self):
348         file_name = QFileDialog.getOpenFileName(self, 'Open File', '/home')
349
350         if file_name[0]:
351             f = open(file_name[0], 'r')
352             with f:
353                 data = f.read()
354                 self.Open_Signal.reading.emit(data)
355
356     #
357     #
358     ##### end of Class #####
359     #
360     #
361
362     if __name__ == '__main__':
363         MyPath.create() #To create the global shared Variable in the beginning of the program
364         app = QApplication(sys.argv)
365         ex = UI()
366         # ex = Widget()
367         sys.exit(app.exec_())
```

Figure 19: Anubis [7]