EVOLVING THE EDITOR

SEPTEMBER 11, 2020

AIN SHAMS UNIVERSITY – FACULTY OF ENGINEERING

AIN-SHAMS UNIVERCITY

FACULTY OF ENGINEERING

CSE426: Software Maintenance and Evolution

Evolving the Editor

Name: **Abdelrahman Amr El-Sayed Mohamed Issawi**

ID: **16P6001**

Email: **aid-issawi@hotmail.com**

Submitted to:

Prof. Dr. Ayman Bahaa.

SEPTEMBER 11, 2020

A report for Software Maintenance and Evolution Course codded CSE426 with the requirements of Ain Shams University.

Table of Contents

1. INTRODUCTION

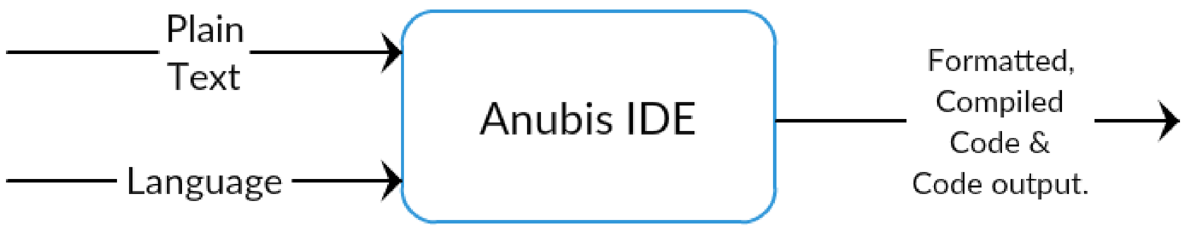
The original Anubis IDE was developed according to the requirements to write, edit, compile, and run micro python codes, the input is a normal plain text and the IDE should perform all the previous actions on this text input. Highlighting the text according to preserved python words, function names and comments. Compile the code and identify any syntax errors and highlight them, and finally run the code and provide the right output.

In the previous study we made a reverse engineering process and discussed whether the IDE reached its goal or not and what actually this IDE can do.

In this report we are going to enhance the IDE and add some functionalities. Also,

we are going to correct some of the identified issues in the original code, and the changes and corrections we are going to make will propagate in all the aspects of the project, code and design documents and requirements.

And the major enhancement in the IDE that it will support another language in addition to the original existing python support, so now the system has 2 inputs not only the plain text to be formatted, it also takes the language or automatically detect it.



When starting the IDE you will find a 3 fields and a menu bar, the first field on top left is a browser to navigate through the files, the second one is the area to write the code, and the third one at the bottom is for seeing the result.
The IDE starts by executing the UI class, this class calls the other ones to execute the overall program "will be discussed briefly in the class diagram".

2. CHANGES MADE

2.1 Enhancements in the original IDE

1. The defined unused variables are deleted; they are increasing the number of lines of code however they are not useful at all.
2. The variables and functions names are renamed to be informative not to assign any names.

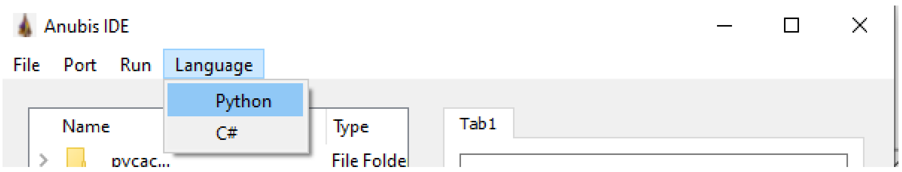
   Ex:

   reading → readingSignal,

   text2 → previewText,

   tx → textWidget,

   b → inputSignal.
3. Also, the naming format is unified as some variables were named using camel case others were named using snake case.
4. Attribute languageMode is added to control actions based on language detected or chosen. Language is chosen from menu bar.

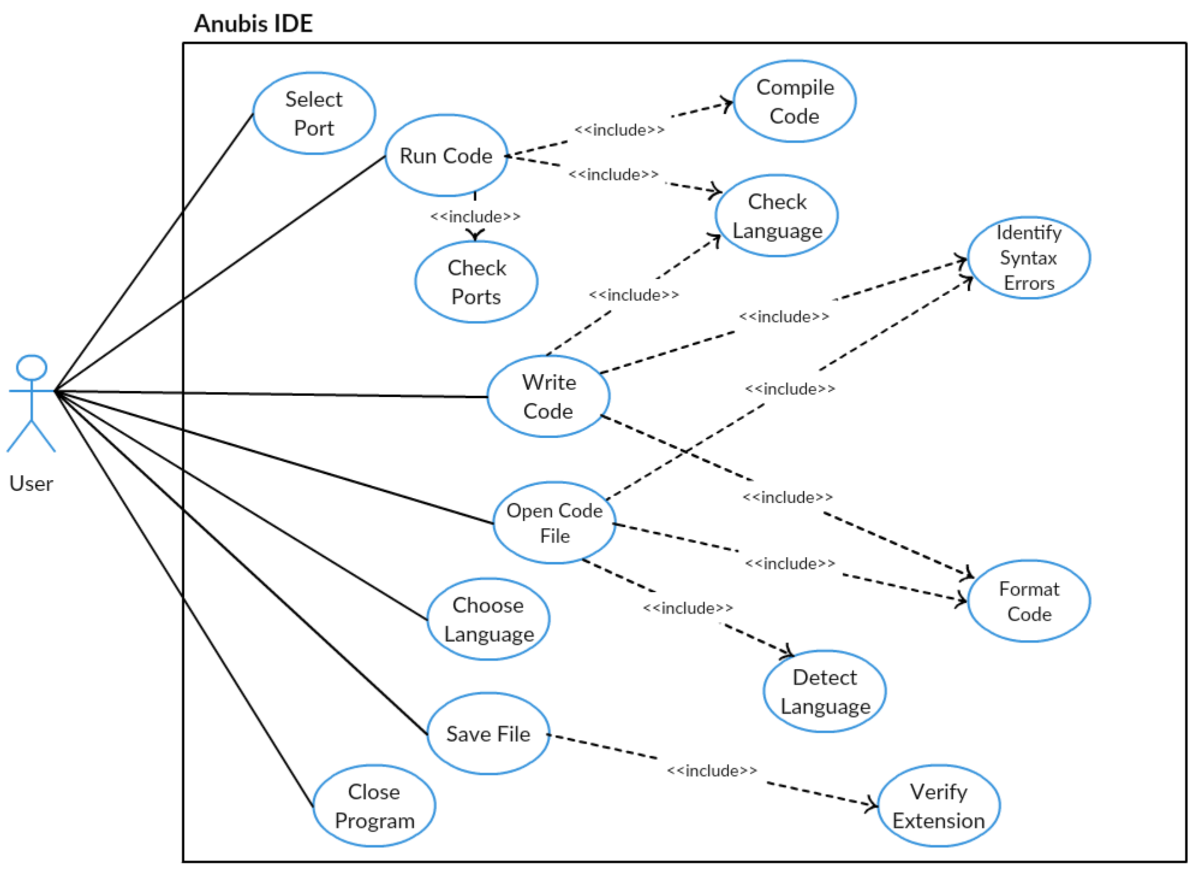


5. Supporting the coloring of built in functions ex: sorted, max, min, zip.
6. Support of the C# language and adding its vocabulary and coloring rules.
7. IDE detects language automatically and save files also with the language extension.
8. All changes are propagated to the design documents, use case diagram and requirements.

2.2 Added Functional Requirements

1. The editor must support C# in addition to python.
2. Editor must detect automatically the language based on the file's extension.
3. The user is given an option to switch between languages while editing.
4. When saving ask the user to choose the language first if it is not chosen.
5. Save the files in the same IDE directory with the chosen extension.

2.3 Updated Use Case Diagram



2.4 Updated Use Cases Description

Use Case name:	Run Code.
Goal in context:	Execute the written code and get the output.
Pre-Conditions:	A code is already written to run, language is chosen.
Successful end condition:	An output is returned, or the code function is done.
Failure end condition:	Error occurs, incorrect output, no chosen language.
Primary Actor:	User.
Secondary Actor:	None.
Trigger:	User clicked on Run item in the menu bar.
Input:	Code.
Output:	None.
Include:	Compile Code, Check ports, Check language.
Main flow:	1. User Clicks on menu bar item “Run”
Extensions:	1.1 No compiler attached. 1.2 No code written. 1.3 No language selected

Use Case name:	Compile Code.
Goal in context:	Scan and parse the written code.
Pre-Conditions:	The IDE is already opened, execution started, compiler is attached.
Successful end condition:	Code is compiled successfully.
Failure end condition:	Compilation error.
Primary Actor:	None.
Secondary Actor:	None.
Trigger:	Run button is clicked and execution started.
Input:	Code.
Output:	None.
Main flow:	1. User Clicks on menu bar item "Run". 2. Written code goes through the compiler scanner, parser and all the compilation processes. 3. The output is displayed in the text field.
Extensions:	1. Code contains errors. 2. No written code. 3. Compiler is not attached. 4. No language selected.

Use Case name:	Check Ports.
Goal in context:	Check that serial ports are selected.
Pre-Conditions:	The IDE is already opened, execution started.
Successful end condition:	Serial ports found.
Failure end condition:	No ports found.
Primary Actor:	None.
Secondary Actor:	None.
Trigger:	Run button is clicked and execution started.
Input:	Code.
Output:	None.
Main flow:	1. User Clicks on menu bar item "Run". 2. System check for serial ports selected. 3. If ports are found selected the execution starts.
Extensions:	2.1 No ports found.

Use Case name:	Select Port.
Goal in context:	To select port where microcontroller is connected.
Pre-Conditions:	The IDE is already opened, ports are available.
Successful end condition:	Port is successfully selected.
Failure end condition:	No ports selected.
Primary Actor:	User.
Secondary Actor:	None.
Trigger:	User clicked on ports item in the menu bar.
Input:	Port name.
Output:	None.
Main flow:	1. User Clicks on menu bar item "Port" 2. User selects a port from displayed ones.
Extensions:	2.1 User did not find any port available.

Use Case name:	Write Code.
Goal in context:	Enter python codes to the IDE.
Pre-Conditions:	Text filed exists.
Successful end condition:	Code is written successfully.
Failure end condition:	Code is not written.
Primary Actor:	User.
Secondary Actor:	None.
Trigger:	User put cursor on text field to write.
Include:	Identify Syntax Errors, Format Code, Check Language.
Input:	Text.
Output:	None.
Main flow:	1. User selects Language. 2. User clicks on text field to write code.
Extensions:	1.1 Language is not selected. 1.2 Text field did not accept text.

Use Case name:	Identify Syntax Errors.
Goal in context:	Highlight to the user any found syntax errors.
Pre-Conditions:	Code must exist in the text field.
Successful end condition:	All syntax errors are identified.
Failure end condition:	No or not all syntax errors are identified.
Primary Actor:	Compiler.
Secondary Actor:	None.
Trigger:	Code is written in the text field.
Input:	Code.
Output:	None.
Main flow:	1. User write code in the text field.
Extensions:	1.1 Compiler is not attached. 1.2 No language selected.

Use Case name:	Format Code.
Goal in context:	Format written text and color it according to the python/C# color conventions.
Pre-Conditions:	Code must exist in the text field.
Successful end condition:	All code parts are colored correctly according to the coloring conventions.
Failure end condition:	No or not all code parts are colored right.
Primary Actor:	Highlighter.
Secondary Actor:	None.
Trigger:	Code is written in the text field.
Input:	Code.
Output:	None.
Main flow:	1. User write code in the text field.
Extensions:	1.1 Highlighter class is not imported or not working. 1.2 Language is not selected.

Use Case name:	Open Code File.
Goal in context:	Open already existed code files on the device.
Pre-Conditions:	The IDE is already opened.
Successful end condition:	Files are opened successfully.
Failure end condition:	Files are not opened.
Primary Actor:	User.
Secondary Actor:	None.
Trigger:	User clicked on open in the menu bar.
Include:	Identify Syntax Errors, Format Code, Detect Language.
Input:	File path and name.
Output:	None.
Main flow:	1. User Clicks on menu bar item “File” 2. User selects open option. 3. User Navigates through files. 4. User selects the needed file.
Extensions:	4.1 User did not find any file available. 4.2 User choose unsupported file.

Use Case name:	Save File.
Goal in context:	Save formatted files into specific location.
Pre-Conditions:	The IDE is already opened, Code already exists.
Successful end condition:	Code is saved successfully.
Failure end condition:	Code is not saved.
Primary Actor:	User.
Secondary Actor:	None.
Trigger:	User clicked on save item in menu bar.
Input:	None.
Output:	None.
Include	Verify Extension.
Main flow:	1. User Clicks on menu bar item “File” 2. User selects Save option. 3. User choose the location.
Extensions:	3.1 User has no memory available.

Use Case name:	Close Program.
Goal in context:	Close the IDE.
Pre-Conditions:	The IDE is already opened.
Successful end condition:	IDE is closed.
Failure end condition:	IDE is not closed and still opened.
Primary Actor:	User.
Secondary Actor:	None.
Trigger:	User clicked on Close item in menu bar.
Input:	None.
Output:	None.
Main flow:	1. User Clicks on menu bar item “File” 2. User selects Close option.
Extensions:	None.

Use Case name:	Check Language.
Goal in context:	Identify the programming language.
Pre-Conditions:	The IDE is opened, and code is written.
Successful end condition:	Language is identified.
Failure end condition:	No language found.
Primary Actor:	compiler.
Secondary Actor:	None.
Trigger:	Program starts running.
Input:	None.
Output:	None.
Main flow:	1. Check state variable before processing code when user press run.
Extensions:	1.1 No language selected.

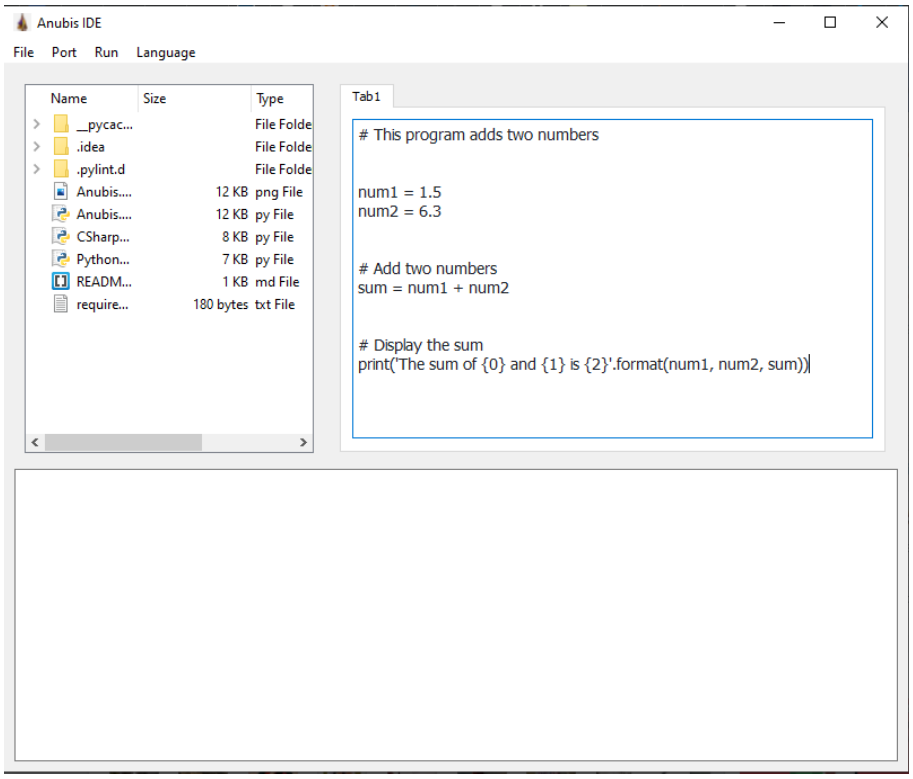
Use Case name:	Choose Language.
Goal in context:	Select programming language.
Pre-Conditions:	No language selected, IDE is opened.
Successful end condition:	Language is selected.
Failure end condition:	No language selected.
Primary Actor:	User.
Secondary Actor:	None.
Trigger:	User clicked on language item in menu bar.
Input:	None.
Output:	None.
Main flow:	1. User Clicks on menu bar item “language” 2. User chooses desired language.
Extensions:	2.1 No language selected.

Use Case name:	Detect Language.
Goal in context:	Detect language in the opened file.
Pre-Conditions:	File is chosen to be opened.
Successful end condition:	programming language is detected.
Failure end condition:	programming language is not detected.
Primary Actor:	compiler.
Secondary Actor:	None.
Trigger:	User opened a file.
Input:	None.
Output:	None.
Main flow:	1. User Clicks on menu bar item “Open” 2. User chooses file. 3. System detects language. 4. Code is reformatted based on this language
Extensions:	2.1 No File opened. 3.1 System fails to detect language. 3.2 Not supported format.

Use Case name:	Verify Extension.
Goal in context:	Check language to save file in right extension.
Pre-Conditions:	The IDE is already opened, code is written.
Successful end condition:	Extension is verified.
Failure end condition:	Unknown extension.
Primary Actor:	Compiler.
Secondary Actor:	None.
Trigger:	User clicked on Save item in menu bar.
Input:	None.
Output:	None.
Main flow:	1. User Clicks on menu bar item “save” 2. System Check state variable of language and saves the file based on it.
Extensions:	2.1 system failed to identify the extension.

3. TEST TRAILS SCREENSHOTS

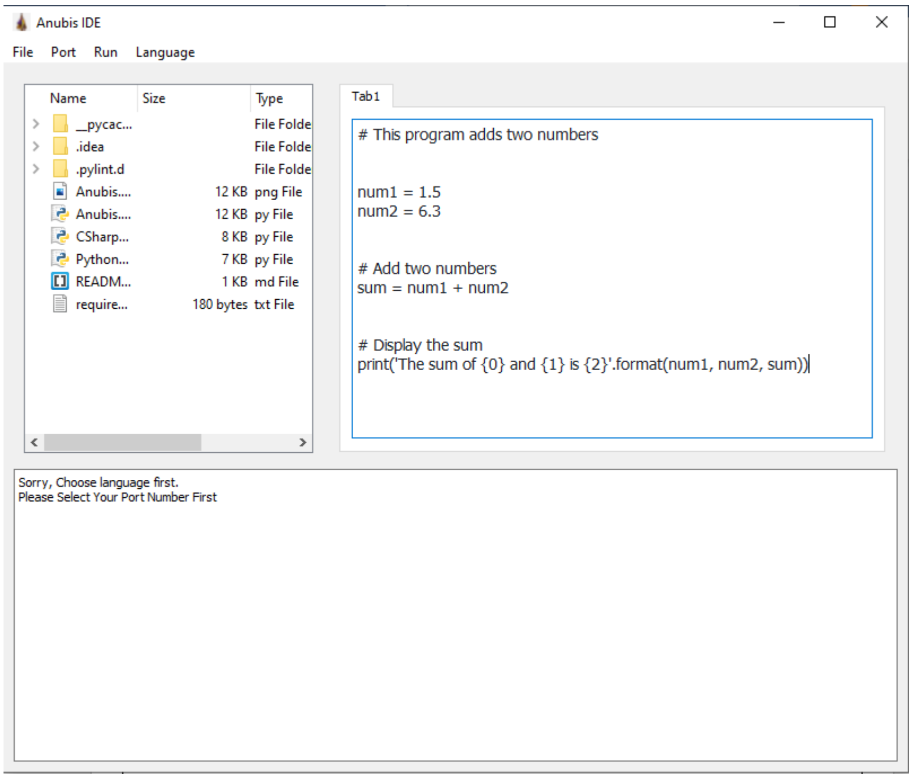
1. Write without choosing a language.



Name	Size	Type
__pycac...		File Folde
.idea		File Folde
.pylint.d		File Folde
Anubis....	12 KB	png File
Anubis....	12 KB	py File
CSharp...	8 KB	py File
Python...	7 KB	py File
READM...	1 KB	md File
require...	180 bytes	txt File

```
# This program adds two numbers


num1 = 1.5
num2 = 6.3


# Add two numbers
sum = num1 + num2


# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

2. Run without Choosing language.

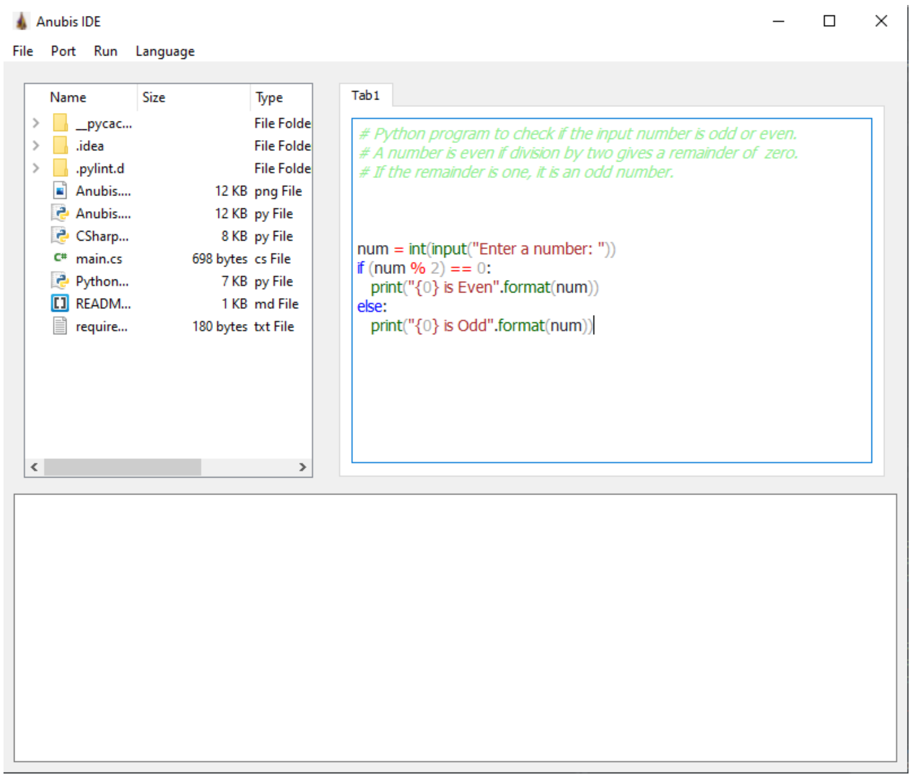


Name	Size	Type
__pycac...		File Folde
.idea		File Folde
.pylint.d		File Folde
Anubis....	12 KB	png File
Anubis....	12 KB	py File
CSharp...	8 KB	py File
Python...	7 KB	py File
READM...	1 KB	md File
require...	180 bytes	txt File

```
# This program adds two numbers


num1 = 1.5
num2 = 6.3


# Add two numbers
sum = num1 + num2


# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

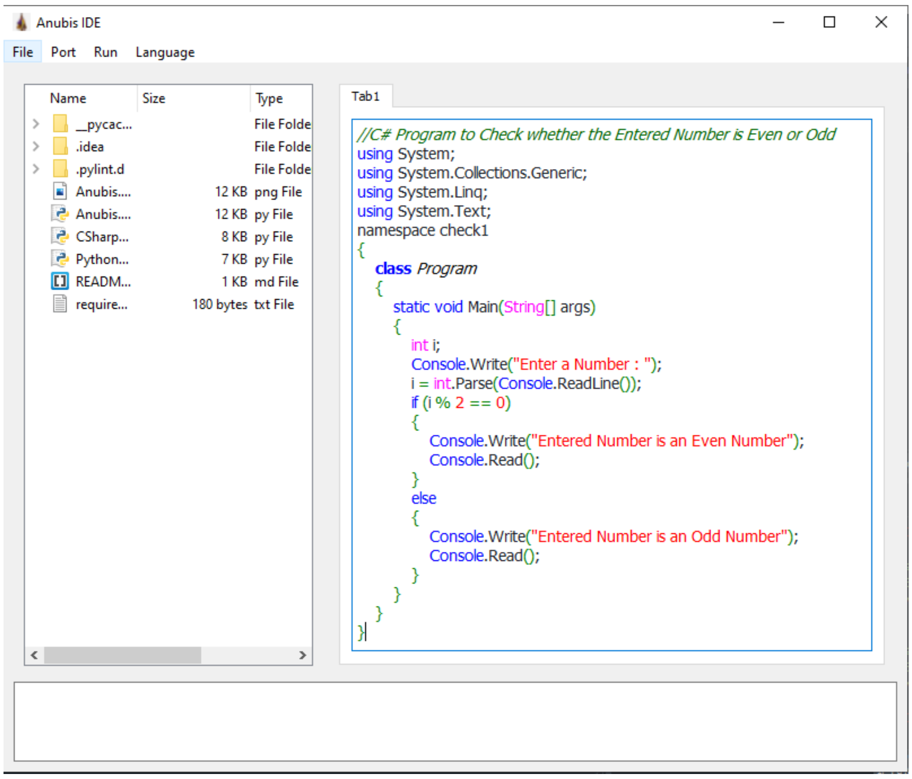
```
Sorry, Choose language first.
Please Select Your Port Number First
```

3. Write python Code to identify even and odd numbers.



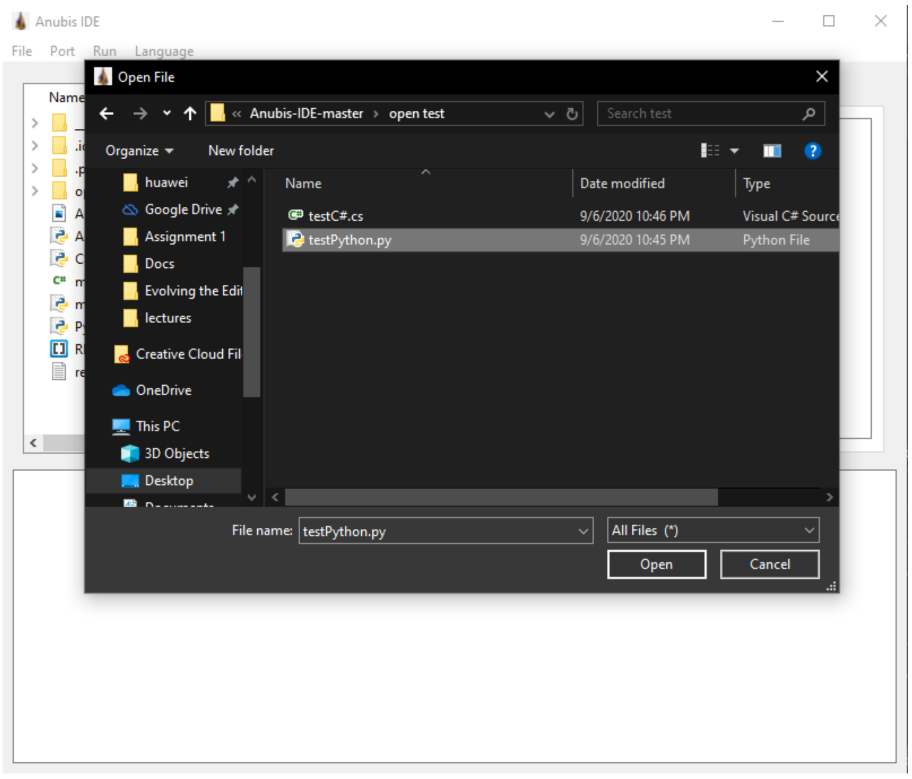
```python
# Python program to check if the input number is odd or even.
# A number is even if division by two gives a remainder of  zero.
# If the remainder is one, it is an odd number.



num = int(input("Enter a number: "))
if (num % 2) == 0:
   print("{0} is Even".format(num))
else:
   print("{0} is Odd".format(num))
```

4. Write C# Code identify even and odd number.



```csharp
//C# Program to Check whether the Entered Number is Even or Odd
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace check1
{
    class Program
    {
        static void Main(String[] args)
        {
            int i;
            Console.Write("Enter a Number : ");
            i = int.Parse(Console.ReadLine());
            if (i % 2 == 0)
            {
                Console.Write("Entered Number is an Even Number");
                Console.Read();
            }
            else
            {
                Console.Write("Entered Number is an Odd Number");
                Console.Read();
            }
        }
    }
}
```

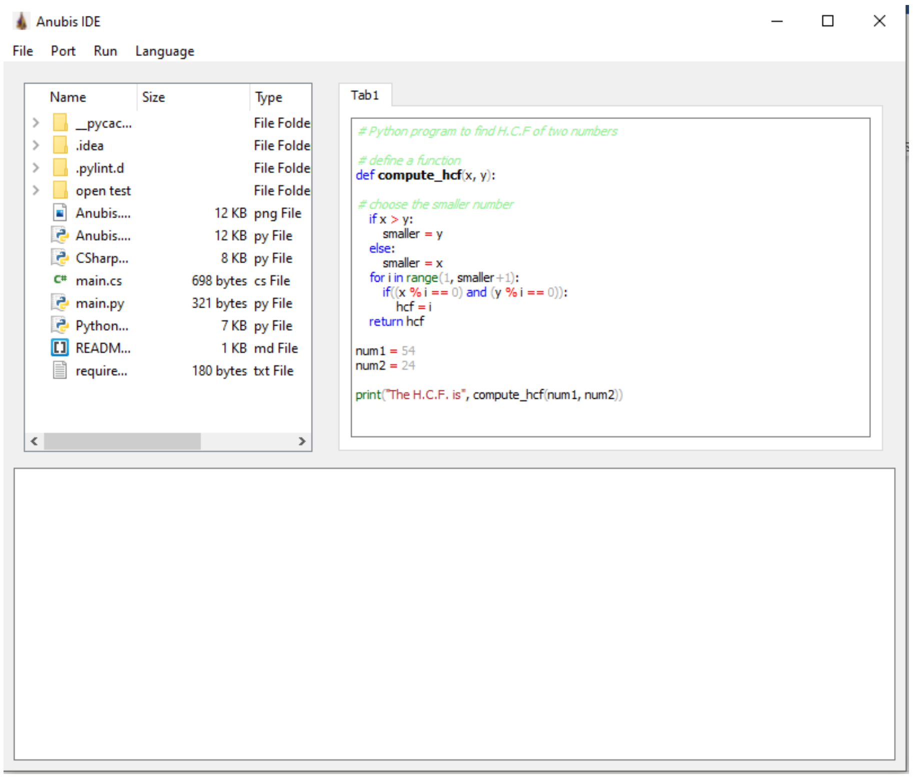
5. Open python File finds HCF between 2 numbers.



Name	Date modified	Type
testC#.cs	9/6/2020 10:46 PM	Visual C# Source
testPython.py	9/6/2020 10:45 PM	Python File

File name: testPython.py

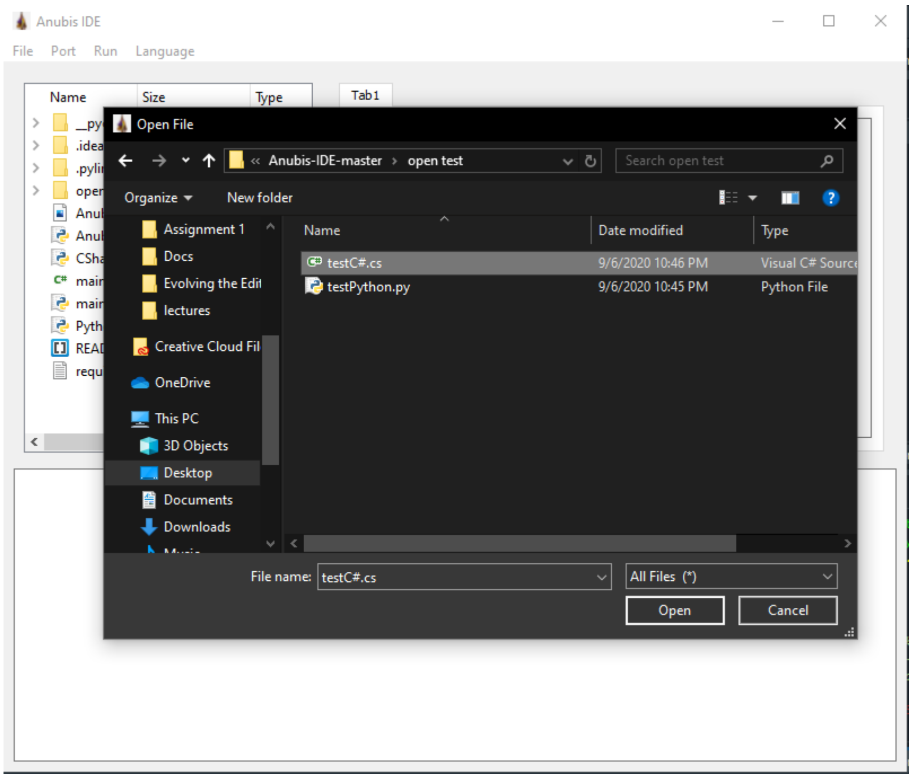
All Files (*)



Name	Size	Type
__pycac...		File Folde
.idea		File Folde
.pylint.d		File Folde
open test		File Folde
Anubis....	12 KB	png File
Anubis....	12 KB	py File
CSharp...	8 KB	py File
main.cs	698 bytes	cs File
main.py	321 bytes	py File
Python...	7 KB	py File
READM...	1 KB	md File
require...	180 bytes	txt File

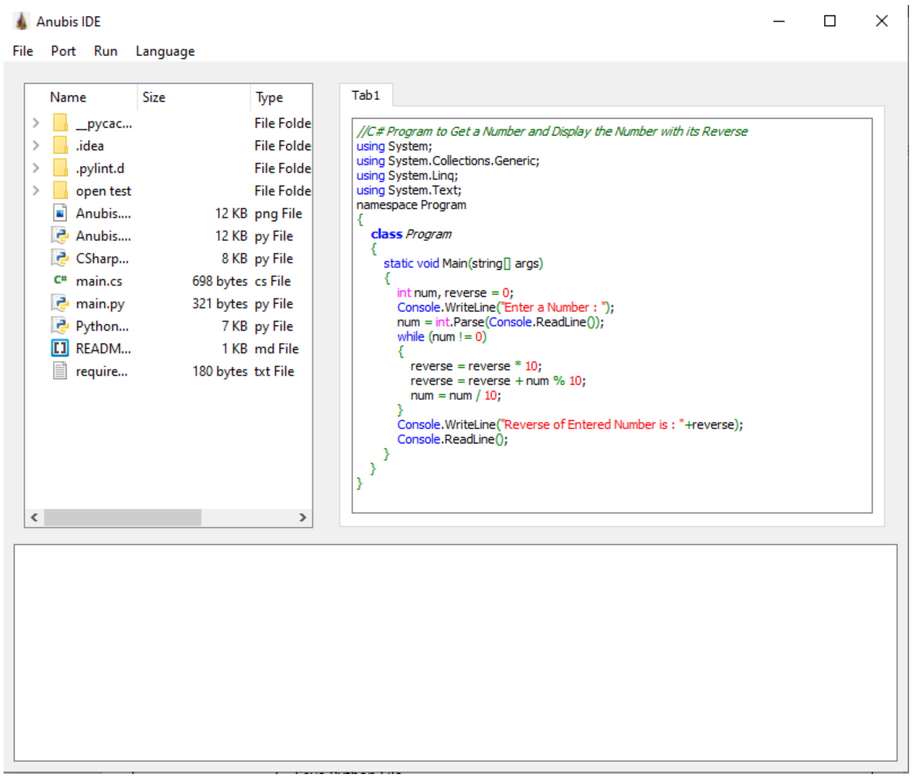
```
# Python program to find H.C.F of two numbers

# define a function
def compute_hcf(x, y):

# choose the smaller number
   if x > y:
      smaller = y
   else:
      smaller = x
   for i in range(1, smaller+1):
      if((x % i == 0) and (y % i == 0)):
         hcf = i
   return hcf

num1 = 54
num2 = 24

print("The H.C.F. is", compute_hcf(num1, num2))
```

6. Open C# File displays number in reverse.



Name	Date modified	Type
testC#.cs	9/6/2020 10:46 PM	Visual C# Source
testPython.py	9/6/2020 10:45 PM	Python File

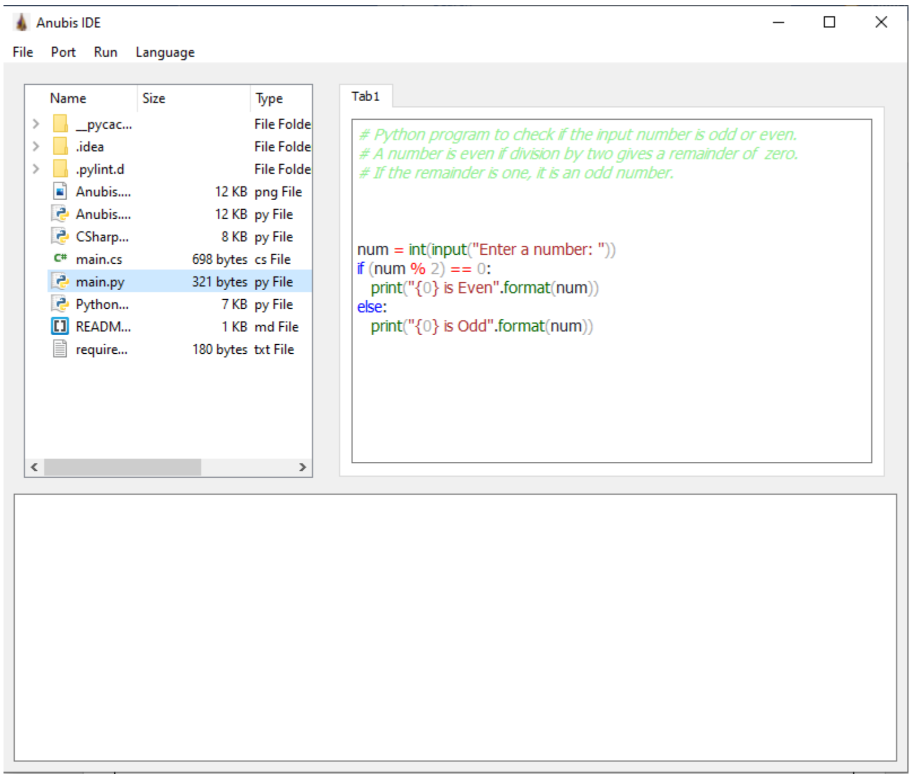
File name: testC#.cs   All Files (*)



Name	Size	Type
__pycac...		File Folde
.idea		File Folde
.pylint.d		File Folde
open test		File Folde
Anubis....	12 KB	png File
Anubis....	12 KB	py File
CSharp...	8 KB	py File
main.cs	698 bytes	cs File
main.py	321 bytes	py File
Python...	7 KB	py File
READM...	1 KB	md File
require...	180 bytes	txt File

```
//C# Program to Get a Number and Display the Number with its Reverse
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Program
{
   class Program
   {
      static void Main(string[] args)
      {
         int num, reverse = 0;
         Console.WriteLine("Enter a Number : ");
         num = int.Parse(Console.ReadLine());
         while (num != 0)
         {
            reverse = reverse * 10;
            reverse = reverse + num % 10;
            num = num / 10;
         }
         Console.WriteLine("Reverse of Entered Number is : "+reverse);
         Console.ReadLine();
      }
   }
}
```

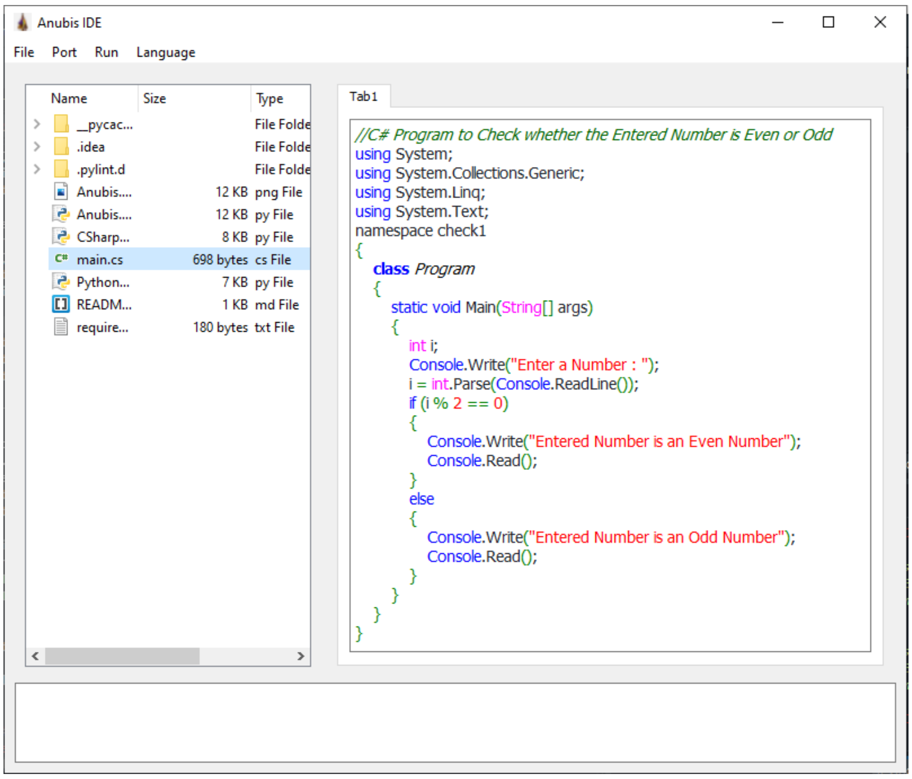
7. Save Python File.



Name	Size	Type
__pycac...		File Folde
.idea		File Folde
.pylint.d		File Folde
Anubis....	12 KB	png File
Anubis....	12 KB	py File
CSharp...	8 KB	py File
main.cs	698 bytes	cs File
main.py	321 bytes	py File
Python...	7 KB	py File
READM...	1 KB	md File
require...	180 bytes	txt File

```
# Python program to check if the input number is odd or even.
# A number is even if division by two gives a remainder of  zero.
# If the remainder is one, it is an odd number.


num = int(input("Enter a number: "))
if (num % 2) == 0:
   print("{0} is Even".format(num))
else:
   print("{0} is Odd".format(num))
```

8. Save C# Code.



Name	Size	Type
__pycac...		File Folde
.idea		File Folde
.pylint.d		File Folde
Anubis....	12 KB	png File
Anubis....	12 KB	py File
CSharp...	8 KB	py File
main.cs	698 bytes	cs File
Python...	7 KB	py File
READM...	1 KB	md File
require...	180 bytes	txt File

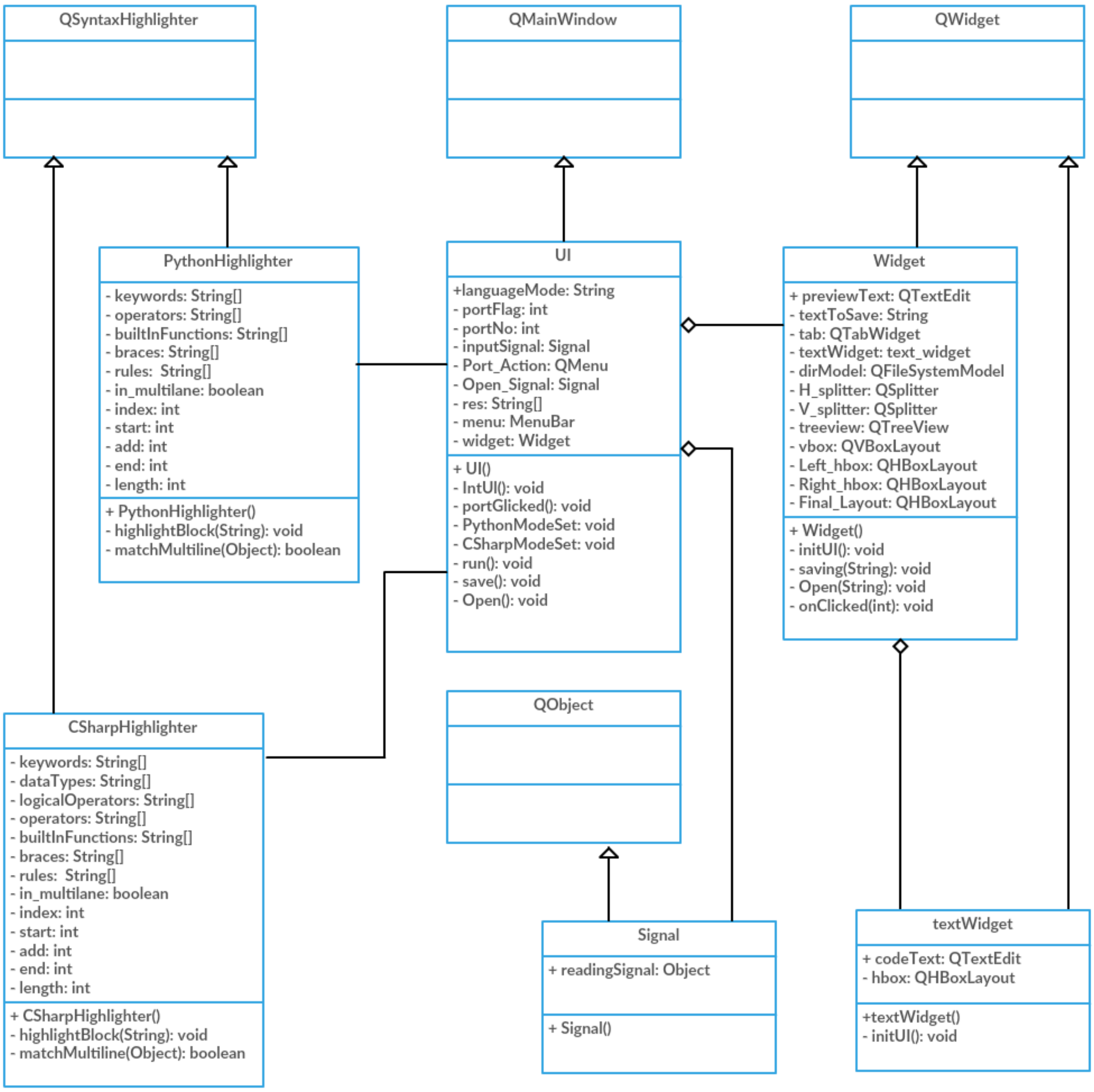
```
//C# Program to Check whether the Entered Number is Even or Odd
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace check1
{
    class Program
    {
        static void Main(String[] args)
        {
            int i;
            Console.Write("Enter a Number : ");
            i = int.Parse(Console.ReadLine());
            if (i % 2 == 0)
            {
                Console.Write("Entered Number is an Even Number");
                Console.Read();
            }
            else
            {
                Console.Write("Entered Number is an Odd Number");
                Console.Read();
            }
        }
    }
}
```

4. UPDATED DESIGN DOCUMENTS

4.1 Updated Class Diagram

Updates:

1. The hierarchy of classes or the sequence of program is changed in the original one the UI was calling the widget and the signal classes, widget calls textWidget, and textWidget calls Python highlighter class.
2. Now, UI calls signal and widget classes and as well is call the python and C# highlighter classes and finally the widget class calls the textWidget one.
3. Attributes, methods, classes names are adjusted and reformatted, new functions and attributes are added.
4. "CSharpHighlighter" class is added which is responsible for formatting the C# code.

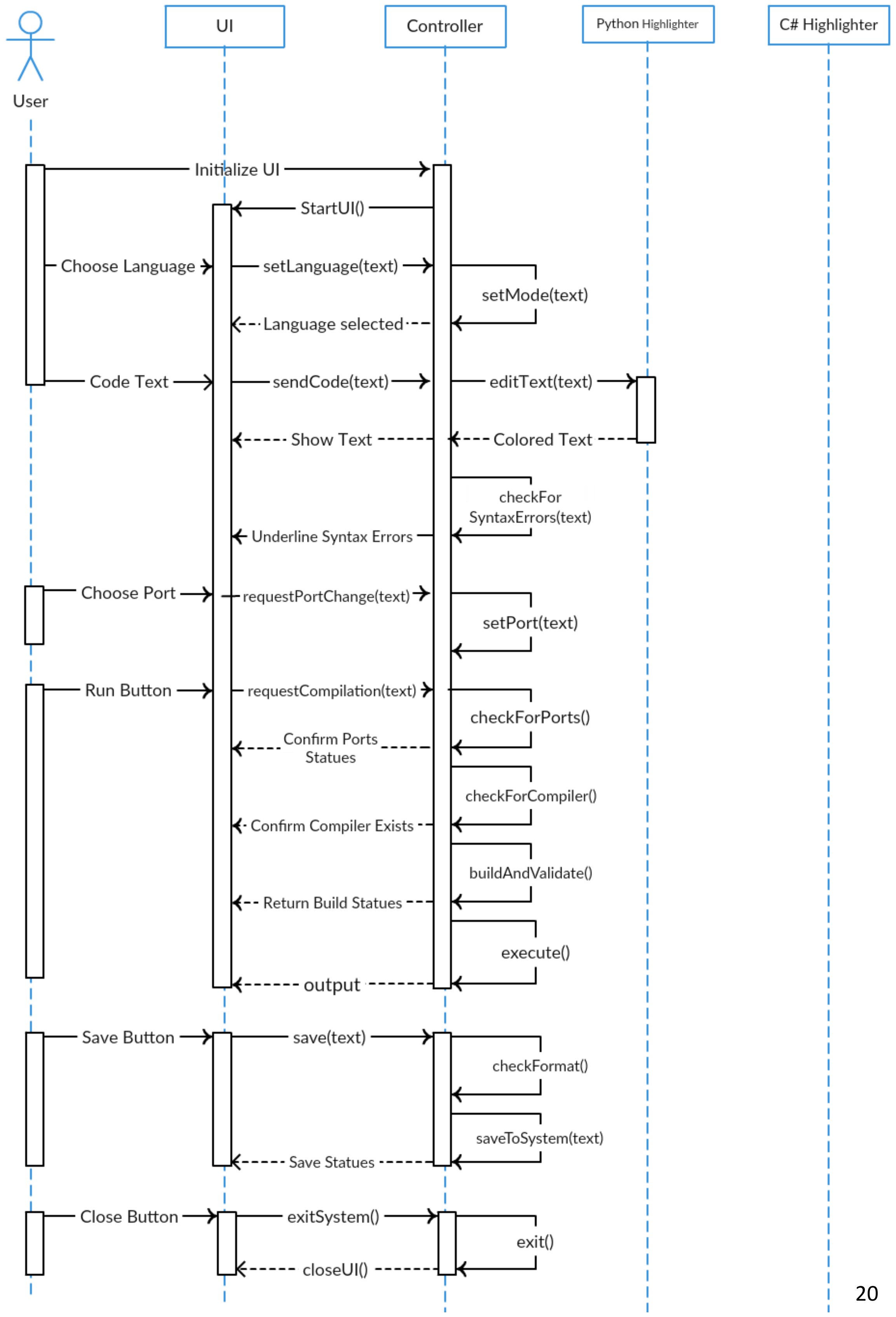


4.2 Updated Sequence Diagram

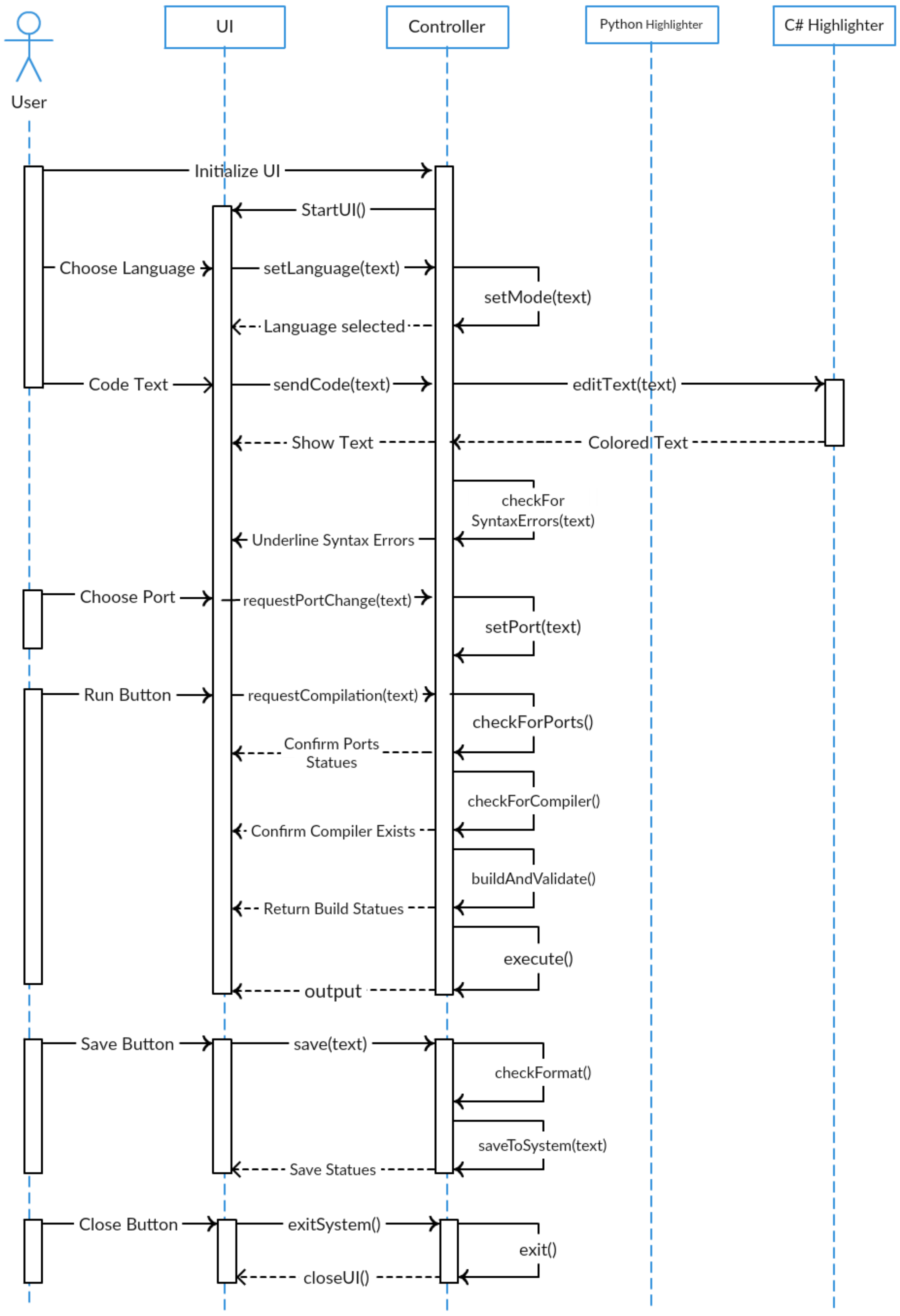
Updates:

The user now may write codes using python or C#, or the use can also open python or C# files.

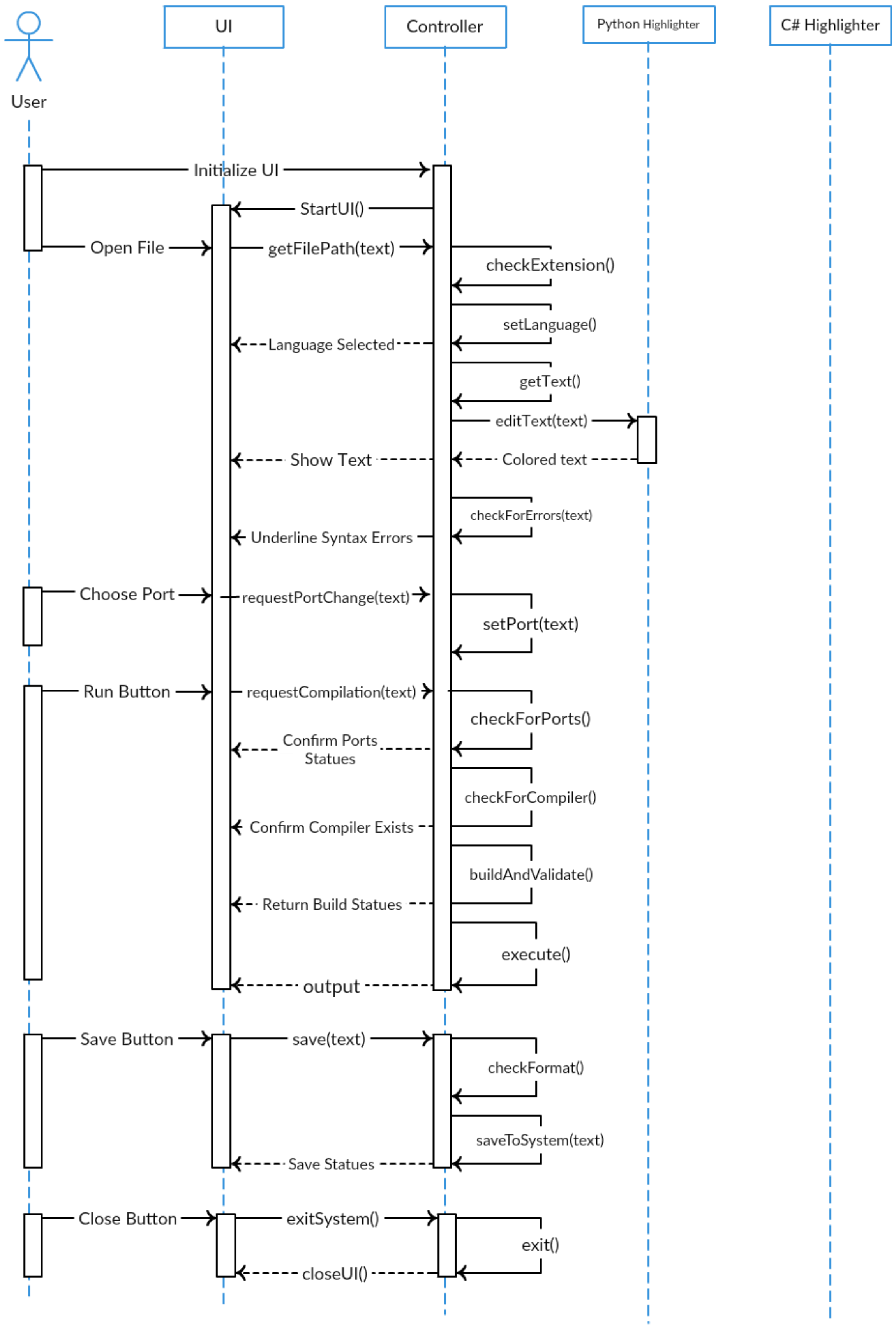
4.2.1 Writing Python Code



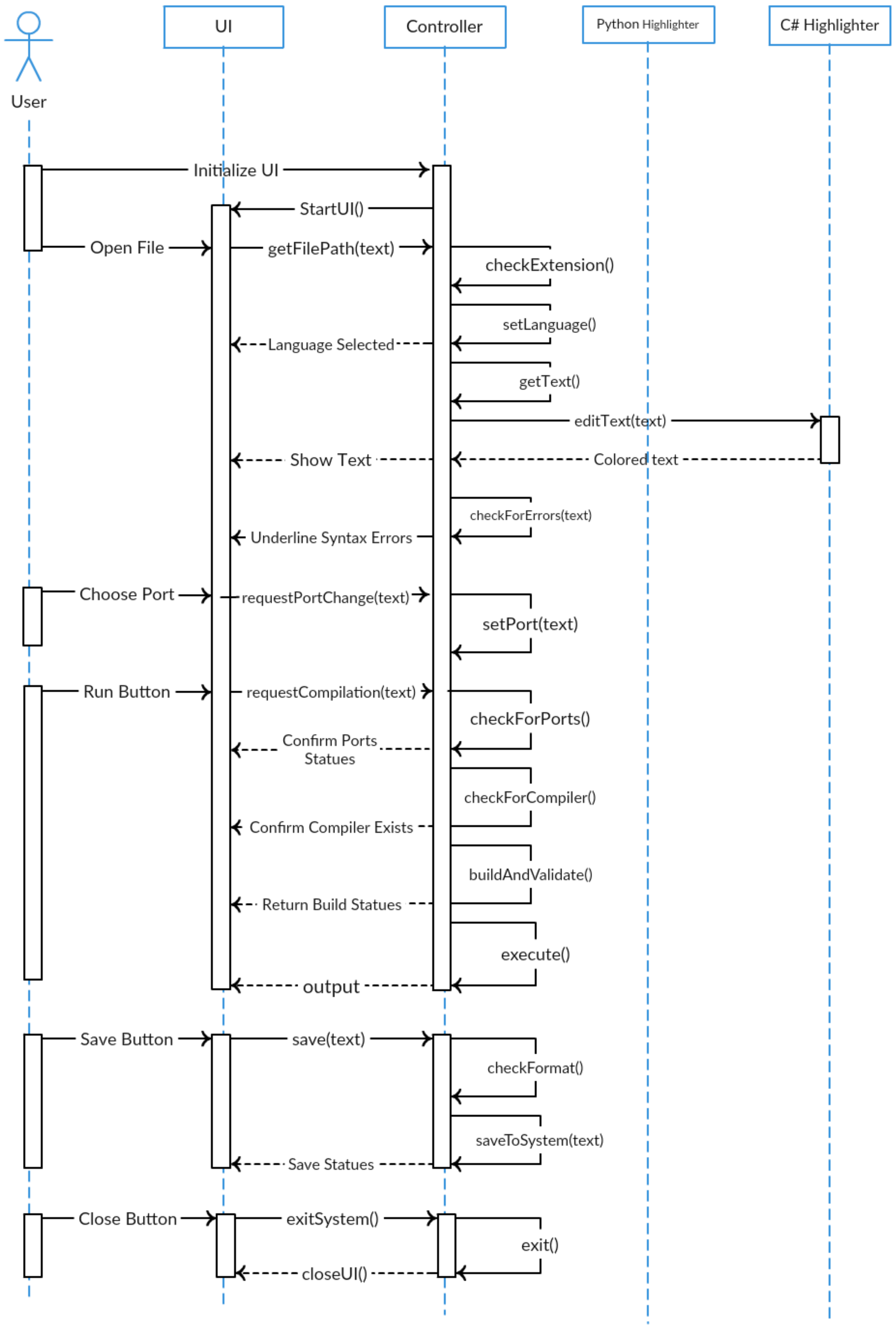
4.2.2 Writing C# Code



4.2.3 Opening python File



4.2.4 Opening C# File



5. CODE

5.1 CSharp_Coloring.py

```python
"""
Created on Sat Sep  5 21:00:41 2020
@author: Abdelrahman Issawi 16P6001
@reason: To support C# Formating.
"""
import sys
from PyQt5.QtCore import QRegExp
from PyQt5.QtGui import QColor, QTextCharFormat, QFont, QSyntaxHighlighter

# Coloring, formating function
def format(color, style=''):
    """
    Return a QTextCharFormat with the given attributes.
    """
    _color = QColor()
    if type(color) is not str:
        _color.setRgb(color[0], color[1], color[2])
    else:
        _color.setNamedColor(color)

    _format = QTextCharFormat()
    _format.setForeground(_color)
    if 'bold' in style:
        _format.setFontWeight(QFont.Bold)
    if 'italic' in style:
        _format.setFontItalic(True)

    return _format


# Syntax styles that can be shared by all languages
STYLES = {
       'keyword': format('blue'),
       'operator': format('green'),
       'brace': format('green'),
       'class': format('blue', 'bold'),
       'string': format('red'),
       'string2': format('darkMagenta'),
       'comment': format('darkGreen', 'italic'),
       'classID': format('black', 'italic'),
       'numbers': format('red'),
       'builtInFunctions': format('green'),
       'dataTypes': format('magenta'),
       'logicalOperators': format('green')
   }

class CSharpHighlighter(QSyntaxHighlighter):
    """Syntax highlighter for the C# language.
    """
    # C# keywords
    keywords = [
       "public","private","protected","internal","protected","internal","using",
       "abstract","async","const","event","extern","new","override","partial",
       "readonly","sealed","static","unsafe","virtual","volatile","if","else","true",
       "class","false","switch","case","default","break","continue",
       "for","foreach","while","do","try","catch","finally","in","void","delegate","Console"
    ]

    #built in functions
    builtInFunctions = [
        "ToBoolean","ToByte","ToChar","ToDateTime","ToDecimal","ToDouble","ToInt32","ToInt64",
        "ToSbyte","ToSingle","ToString","ToType","ToUInt32","ToUInt64",
         "sizeof","typeof","is","as","AsInt","IsInt","AsFloat",
         "IsFloat","AsDecimal","IsDecimal","AsDateTime","IsDateTime","AsBool","IsBool","Clone","CompareTo",
         "Contains","EndsWith","Equals","GetHashCode","GetType","IndexOf","ToLower","ToUpper",
         "Insert","IsNormalized","LastIndexOf","Length","Remove","Split","Replace","StartsWith","ToCharArray","Trim",
         "IsFixedSize","LongLength","TrueForAll"
         "BinarySearch","Clear","Clone","Copy","CopyTo","Empty","CreateInstance","Exists","find","FindIndex",
         "FindAll","GetEnumerator","GetType","Initialize","Resize","Reverse","Sort","ToString"
    ]

    #data types
    dataTypes = [
        "int","double","float","String","char","byte","long","decimal","bool","enums","struct"
    ]

    # C# operators
    operators = [
        '=',
        # Comparison
        '==', '!=', '<', '<=', '>', '>=',
        # Arithmetic
        '\+', '-', '\*', '/', '\%',
        # In-place
        '\+=', '-=', '\*=', '/=', '\%=','&=',
        # Bitwise
        '\^', '\|', '\&', '\~',
        #shift
        '<<=','<<='

    ]

    # Logical and Bitwise Operators
    logicalOperators = [
        # logicalOperators
        '&&','\|\|', '!','<<','>>'
    ]

    # braces
    braces = [
        '\{', '\}', '\(', '\)', '\[', '\]',
    ]

    def __init__(self, document):
        QSyntaxHighlighter.__init__(self, document)
        # Multi-line strings (expression, flag, style)
        # FIXME: The triple-quotes in these two lines will mess up the
        # syntax highlighting from this point onward
        self.tri_single = (QRegExp("'''"), 1, STYLES['string2'])
        self.tri_double = (QRegExp('"""'), 2, STYLES['string2'])

        rules = []

        # Keyword, operator, and brace rules
        rules += [(r'\b%s\b' % w, 0, STYLES['keyword'])
                  for w in CSharpHighlighter.keywords]

        rules += [(r'\b%s\b' % w, 0, STYLES['dataTypes'])
                  for w in CSharpHighlighter.dataTypes]

        rules += [(r'\b%s\b' % w, 0, STYLES['builtInFunctions'])
                  for w in CSharpHighlighter.builtInFunctions]

        rules += [(r'%s' % o, 0, STYLES['operator'])
                  for o in CSharpHighlighter.operators]

        rules += [(r'%s' % b, 0, STYLES['brace'])
                  for b in CSharpHighlighter.braces]

        rules += [(r'%s' % b, 0, STYLES['logicalOperators'])
                  for b in CSharpHighlighter.logicalOperators]


        # All other rules
        rules += [
            # 'class'
            (r'\bclass\b', 0, STYLES['class']),

            # Double-quoted string, possibly containing escape sequences
            (r'"[^"\\]*(\\.[^"\\]*)*"', 0, STYLES['string']),
            # Single-quoted string, possibly containing escape sequences
            (r"'[^'\\]*(\\.[^'\\]*)*'", 0, STYLES['string']),

            # 'class' followed by an identifier
            (r'\bclass\b\s*(\w+)', 1, STYLES['classID']),

            # From '//' until a newline
            (r'//[^\n]*', 0, STYLES['comment']),

            # From '//' until a newline
            #(r'/*[^\n]*/*', 0, STYLES['comment']),


            # Numeric literals
            (r'\b[+-]?[0-9]+[lL]?\b', 0, STYLES['numbers']),
            (r'\b[+-]?0[xX][0-9A-Fa-f]+[lL]?\b', 0, STYLES['numbers']),
            (r'\b[+-]?[0-9]+(?:\.[0-9]+)?(?:[eE][+-]?[0-
9]+)?\b', 0, STYLES['numbers']),
        ]

        # Build a QRegExp for each pattern
        self.rules = [(QRegExp(pat), index, fmt)
                      for (pat, index, fmt) in rules]

    def highlightBlock(self, text):
        """Apply syntax highlighting to the given block of text.
        """
        # Do other syntax formatting
        for expression, nth, format in self.rules:
            index = expression.indexIn(text, 0)

            while index >= 0:
                # We actually want the index of the nth match
                index = expression.pos(nth)
                length = len(expression.cap(nth))
                self.setFormat(index, length, format)
                index = expression.indexIn(text, index + length)

        self.setCurrentBlockState(0)

        # Do multi-line strings
        in_multiline = self.matchMultiline(text, *self.tri_single)
        if not in_multiline:
            in_multiline = self.matchMultiline(text, *self.tri_double)

    def matchMultiline(self, text, delimiter, in_state, style):
        """Do highlighting of multi-line strings. ``delimiter`` should be a
        ``QRegExp`` for triple-single-quotes or triple-double-quotes, and
        ``in_state`` should be a unique integer to represent the corresponding
        state changes when inside those strings. Returns True if we're still
        inside a multi-line string when this function is finished.
        """
        # If inside triple-single quotes, start at 0
        if self.previousBlockState() == in_state:
            start = 0
            add = 0
        # Otherwise, look for the delimiter on this line
        else:
            start = delimiter.indexIn(text)
            # Move past this match
            add = delimiter.matchedLength()

        # As long as there's a delimiter match on this line...
        while start >= 0:
            # Look for the ending delimiter
            end = delimiter.indexIn(text, start + add)
            # Ending delimiter on this line?
            if end >= add:
                length = end - start + add + delimiter.matchedLength()
                self.setCurrentBlockState(0)
            # No; multi-line string
            else:
                self.setCurrentBlockState(in_state)
                length = len(text) - start + add
            # Apply formatting
            self.setFormat(start, length, style)
            # Look for the next match
            start = delimiter.indexIn(text, start + length)

        # Return True if still inside a multi-line string, False otherwise
        if self.currentBlockState() == in_state:
            return True
        else:
            return False
```

5.2 Python_Coloring.py

```python
import sys
from PyQt5.QtCore import QRegExp
from PyQt5.QtGui import QColor, QTextCharFormat, QFont, QSyntaxHighlighter


def format(color, style=''):
    """
    Return a QTextCharFormat with the given attributes.
    """
    _color = QColor()
    if type(color) is not str:
        _color.setRgb(color[0], color[1], color[2])
    else:
        _color.setNamedColor(color)

    _format = QTextCharFormat()
    _format.setForeground(_color)
    if 'bold' in style:
        _format.setFontWeight(QFont.Bold)
    if 'italic' in style:
        _format.setFontItalic(True)

    return _format


# Syntax styles that can be shared by all languages

STYLES2 = {
    'keyword': format([200, 120, 50], 'bold'),
    'operator': format([150, 150, 150]),
    'brace': format('darkGray'),
    'defclass': format([220, 220, 255], 'bold'),
    'string': format([20, 110, 100]),
    'string2': format([30, 120, 110]),
    'comment': format([128, 128, 128]),
    'self': format([150, 85, 140], 'italic'),
    'numbers': format([100, 150, 190]),
}
STYLES = {
        'keyword': format('blue'),
        'operator': format('red'),
        'brace': format('darkGray'),
        'defclass': format('black', 'bold'),
        'string': format('brown'),
        'string2': format('darkMagenta'),
        'comment': format('light Green', 'italic'),
        'self': format('black', 'italic'),
        'numbers': format('dark grey'),
        'builtInFunctions': format('dark green'),
        'escapeCharacters': format('darkBlue')
    }

class PythonHighlighter(QSyntaxHighlighter):
    """Syntax highlighter for the Python language.
    """
    # Python keywords


    keywords = [
        'and', 'assert', 'break', 'class', 'continue', 'def',
        'del', 'elif', 'else', 'except', 'exec', 'finally',
        'for', 'from', 'global', 'if', 'import', 'in',
        'is', 'lambda', 'not', 'or', 'pass',
        'raise', 'return', 'try', 'while', 'yield',
        'None', 'True', 'False','as'
    ]
    #built in functions
    builtInFunctions = [
        "abs","delattr","hash","memoryview","set","all","dict","help","min","setattr","any","dir","hex",
         "next","slice","ascii","divmod","id","object","sorted","bin",
         "enumerate","input","oct","staticmethod","bool","eval","int","open","str","breakpoint",
         "exec","isinstance","ord","sum","bytearray","filter","issubclass","pow","super","bytes",
         "float","iter","print","tuple","callable","format","len","property","type","chr",
         "frozenset","list","range","vars","classmethod","getattr","locals","repr","zip","compile",
         "globals","map","reversed","__import__","complex","hasattr","max","round"
    ]

    # Python operators
    operators = [
        '=',
        # Comparison
        '==', '!=', '<', '<=', '>', '>=',
        # Arithmetic
        '\+', '-', '\*', '/', '//', '\%', '\*\*',
        # In-place
        '\+=', '-=', '\*=', '/=', '\%=',
        # Bitwise
        '\^', '\|', '\&', '\~', '>>', '<<',
    ]

    # Python braces
    braces = [
        '\{', '\}', '\(', '\)', '\[', '\]',
    ]


    def __init__(self, document):
        QSyntaxHighlighter.__init__(self, document)
        # Multi-line strings (expression, flag, style)
            # FIXME: The triple-quotes in these two lines will mess up the
            # syntax highlighting from this point onward
            self.tri_single = (QRegExp("'''"), 1, STYLES['string2'])
            self.tri_double = (QRegExp('"""'), 2, STYLES['string2'])

            rules = []

            # Keyword,built in functions ,operator, and brace rules
            rules += [(r'\b%s\b' % w, 0, STYLES['keyword'])
                      for w in PythonHighlighter.keywords]
            rules += [(r'\b%s\b' % bi, 0, STYLES['builtInFunctions'])
                      for bi in PythonHighlighter.builtInFunctions]
            rules += [(r'%s' % o, 0, STYLES['operator'])
                      for o in PythonHighlighter.operators]
            rules += [(r'%s' % b, 0, STYLES['brace'])
                      for b in PythonHighlighter.braces]

            # All other rules
            rules += [
                # 'self'
                (r'\bself\b', 0, STYLES['self']),

                # Double-quoted string, possibly containing escape sequences
                (r'"[^"\\]*(\\.[^"\\]*)*"', 0, STYLES['string']),
                # Single-quoted string, possibly containing escape sequences
                (r"'[^'\\]*(\\.[^'\\]*)*'", 0, STYLES['string']),

                # 'def' followed by an identifier
                (r'\bdef\b\s*(\w+)', 1, STYLES['defclass']),
```

```python
            # 'class' followed by an identifier
            (r'\bclass\b\s*(\w+)', 1, STYLES['defclass']),

            # From '#' until a newline
            (r'#[^\n]*', 0, STYLES['comment']),

            # Numeric literals
            (r'\b[+-]?[0-9]+[lL]?\b', 0, STYLES['numbers']),
            (r'\b[+-]?0[xX][0-9A-Fa-f]+[lL]?\b', 0, STYLES['numbers']),
            (r'\b[+-]?[0-9]+(?:\.[0-9]+)?(?:[eE][+-]?[0-9]+)?\b', 0, STYLES['numbers']),
        ]

        # Build a QRegExp for each pattern
        self.rules = [(QRegExp(pat), index, fmt)
                      for (pat, index, fmt) in rules]

    def highlightBlock(self, text):
        """Apply syntax highlighting to the given block of text.
        """
        # Do other syntax formatting
        for expression, nth, format in self.rules:
            index = expression.indexIn(text, 0)

            while index >= 0:
                # We actually want the index of the nth match
                index = expression.pos(nth)
                length = len(expression.cap(nth))
                self.setFormat(index, length, format)
                index = expression.indexIn(text, index + length)

        self.setCurrentBlockState(0)

        # Do multi-line strings
        in_multiline = self.matchMultiline(text, *self.tri_single)
        if not in_multiline:
            in_multiline = self.matchMultiline(text, *self.tri_double)

    def matchMultiline(self, text, delimiter, in_state, style):
        """Do highlighting of multi-line strings. ``delimiter`` should be a
        ``QRegExp`` for triple-single-quotes or triple-double-quotes, and
        ``in_state`` should be a unique integer to represent the corresponding
        state changes when inside those strings. Returns True if we're still
        inside a multi-line string when this function is finished.
        """
        # If inside triple-single quotes, start at 0
        if self.previousBlockState() == in_state:
            start = 0
            add = 0
        # Otherwise, look for the delimiter on this line
        else:
            start = delimiter.indexIn(text)
            # Move past this match
            add = delimiter.matchedLength()

        # As long as there's a delimiter match on this line...
        while start >= 0:
            # Look for the ending delimiter
            end = delimiter.indexIn(text, start + add)
            # Ending delimiter on this line?
            if end >= add:
                length = end - start + add + delimiter.matchedLength()
                self.setCurrentBlockState(0)
            # No; multi-line string
            else:
                self.setCurrentBlockState(in_state)
                length = len(text) - start + add
            # Apply formatting
```

```
                self.setFormat(start, length, style)
                # Look for the next match
                start = delimiter.indexIn(text, start + length)

            # Return True if still inside a multi-line string, False otherwise
            if self.currentBlockState() == in_state:
                return True
            else:
                return False
```

5.3 Anubis.py

```python
#############      author => Anubis Graduation Team        ############
#############      this project is part of my graduation project and it intends to make a fully functioned IDE from scratch    ########
#############      I've borrowed a function (serial_ports()) from a guy in stack overflow whome I can't remember his name, so I gave hime the copyrights of this function, thank you  ########


import sys
import glob
import serial

import Python_Coloring
import CSharp_Coloring
from PyQt5 import QtCore
from PyQt5 import QtGui
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from pathlib import Path
global languageMode
languageMode = "" #by default no language

def serial_ports():
    """ Lists serial port names
        :raises EnvironmentError:
            On unsupported or unknown platforms
        :returns:
            A list of the serial ports available on the system
    """
    if sys.platform.startswith('win'):
        ports = ['COM%s' % (i + 1) for i in range(256)]
    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        # this excludes your current terminal "/dev/tty"
        ports = glob.glob('/dev/tty[A-Za-z]*')
    elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.*')
    else:
        raise EnvironmentError('Unsupported platform')

    result = []
    for port in ports:
        try:
            s = serial.Serial(port)
            s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass


    return result


#
#
#
#
############ Signal Class ############
#
#
#
#
class Signal(QObject):

    # initializing a Signal which will take (string) as an input
    readingSignal = pyqtSignal(str)

    # init Function for the Signal class
    def __init__(self):
        QObject.__init__(self)

#
#
############ end of Class ############
#
#

# Making text editor as A global variable (to solve the issue of being local to (self) in widget class)
codeText = QTextEdit
previewText = QTextEdit

#
#
#
#
############ Text Widget Class ############
#
#
#
#

# this class is made to connect the QTab with the necessary layouts
class text_widget(QWidget):
    def __init__(self):
        super().__init__()
        self.itUI()
    def itUI(self):
        global codeText
        codeText = QTextEdit()
        hbox = QHBoxLayout()
        hbox.addWidget(codeText)
        self.setLayout(hbox)



    #
    #
    ############ end of Class ############
    #
    #



    #
    #
    #
    #
    ############ Widget Class ############
    #
    #
    #
    #
    class Widget(QWidget):

        def __init__(self):
            super().__init__()
            self.initUI()

        def initUI(self):

            # This widget is responsible of making Tab in IDE which makes the Text edit or looks nice
            tab = QTabWidget()
```

```python
        textWidget = text_widget()
        tab.addTab(textWidget, "Tab"+"1")

        # second editor in which the error messeges and succeeded connections will be shown
        global previewText
        previewText = QTextEdit()
        previewText.setReadOnly(True)
        # defining a Treeview variable to use it in showing the directory included files
        self.treeview = QTreeView()

        # making a variable (path) and setting it to the root path (surely I can set it to whatever the root I want, not the default)
        #path = QDir.rootPath()

        path = QDir.currentPath()

        # making a Filesystem variable, setting its root path and applying somefilters (which I need) on it
        self.dirModel = QFileSystemModel()
        self.dirModel.setRootPath(QDir.rootPath())

        # NoDotAndDotDot => Do not list the special entries "." and "..".
        # AllDirs =>List all directories; i.e. don't apply the filters to directory names.
        # Files => List files.
        self.dirModel.setFilter(QDir.NoDotAndDotDot | QDir.AllDirs | QDir.Files)
        self.treeview.setModel(self.dirModel)
        self.treeview.setRootIndex(self.dirModel.index(path))
        self.treeview.clicked.connect(self.onClicked)

        vbox = QVBoxLayout()
        Left_hbox = QHBoxLayout()
        Right_hbox = QHBoxLayout()

        # after defining variables of type QVBox and QHBox
        # I will Assign treevies variable to the left one and the first text editor in which the code will be written to the right one
        Left_hbox.addWidget(self.treeview)
        Right_hbox.addWidget(tab)

        # defining another variable of type Qwidget to set its layout as an QHBoxLayout
        # I will do the same with the right one
        Left_hbox_Layout = QWidget()
        Left_hbox_Layout.setLayout(Left_hbox)

        Right_hbox_Layout = QWidget()
        Right_hbox_Layout.setLayout(Right_hbox)

        # I defined a splitter to seperate the two variables (left, right) and make it more easily to change the space between them
        H_splitter = QSplitter(Qt.Horizontal)
        H_splitter.addWidget(Left_hbox_Layout)
        H_splitter.addWidget(Right_hbox_Layout)
        H_splitter.setStretchFactor(1, 1)

        # I defined a new splitter to seperate between the upper and lower sides of the window
        V_splitter = QSplitter(Qt.Vertical)
        V_splitter.addWidget(H_splitter)
        V_splitter.addWidget(previewText)

        Final_Layout = QHBoxLayout(self)
        Final_Layout.addWidget(V_splitter)

        self.setLayout(Final_Layout)
```

```python

    # defining a new Slot (takes string) to save the text inside the first text editor
    @pyqtSlot(str)
    def saving(s):
        if languageMode == "python":
           with open('main.py', 'w') as f:
               textToSave = codeText.toPlainText()
               f.write(textToSave)
        elif languageMode == "C#":
           with open('main.cs', 'w') as f:
               textToSave = codeText.toPlainText()
               f.write(textToSave)
        else:
           previewText.append("Please, Choose language first.")


    # defining a new Slot (takes string) to set the string to the text editor
    @pyqtSlot(str)
    def Open(s):
        global codeText
        codeText.setText(s)

    def onClicked(self, index):

        readData = self.sender().model().filePath(index)
        readData = tuple([readData])

        if readData[0]:
            f = open(readData[0],'r')
            with f:
                data = f.read()
                codeText.setText(data)

#
#
############ end of Class ############
#
#

# defining a new Slot (takes string)
# Actually I could connect the (mainwindow) class directly to the (widget class) but I've made this function in between for futuer use
# All what it do is to take the (input string) and establish a connection with the widget class, send the string to it
@pyqtSlot(str)
def reading(s):
    inputSignal = Signal()
    inputSignal.readingSignal.connect(Widget.saving)
    inputSignal.readingSignal.emit(s)

# same as reading Function
@pyqtSlot(str)
def Openning(s):
    inputSignal = Signal()
    inputSignal.readingSignal.connect(Widget.Open)
    inputSignal.readingSignal.emit(s)
#
#
#
#
############ MainWindow Class ############
#
#
#
#
class UI(QMainWindow):
    def __init__(self):
        super().__init__()
        self.intUI()

    def intUI(self):
        global languageMode
        self.port_flag = 1
        self.inputSignal = Signal()

        self.Open_Signal = Signal()

        # connecting (self.Open_Signal) with Openning function
        self.Open_Signal.readingSignal.connect(Openning)

        # connecting (self.b) with reading function
        self.inputSignal.readingSignal.connect(reading)

        # creating menu items
        menu = self.menuBar()

        # I have three menu items
        filemenu = menu.addMenu('File')
        Port = menu.addMenu('Port')
        Run = menu.addMenu('Run')

        language = menu.addMenu('Language')

        # As any PC or laptop have many ports, so I need to list them to the User
        # so I made (Port_Action) to add the Ports got from (serial_ports()) function
        # copyrights of serial_ports() function goes back to a guy from stackoverflow(whome I can't remember his name), so thank you (unknown)
        Port_Action = QMenu('port', self)

        res = serial_ports()

        for i in range(len(res)):
            s = res[i]
            Port_Action.addAction(s, self.PortClicked)

        # adding the menu which I made to the original (Port menu)
        Port.addMenu(Port_Action)

#        Port_Action.triggered.connect(self.Port)
#        Port.addAction(Port_Action)

        # Making and adding Run Actions
        RunAction = QAction("Run", self)
        RunAction.triggered.connect(self.Run)
        Run.addAction(RunAction)

        # Making and adding File Features
        Save_Action = QAction("Save", self)
        Save_Action.triggered.connect(self.save)
        Save_Action.setShortcut("Ctrl+S")
        Close_Action = QAction("Close", self)
        Close_Action.setShortcut("Alt+c")
        Close_Action.triggered.connect(self.close)
        Open_Action = QAction("Open", self)
        Open_Action.setShortcut("Ctrl+O")
        Open_Action.triggered.connect(self.Open)


        filemenu.addAction(Save_Action)
        filemenu.addAction(Close_Action)
        filemenu.addAction(Open_Action)

        # Languages handeling
        pythonLanguage = QAction("Python", self)
        cSharp = QAction("C#", self)
        pythonLanguage.triggered.connect(self.PythonModeSet)
        cSharp.triggered.connect(self.CSharpModeSet)
        language.addAction(pythonLanguage)
        language.addAction(cSharp)

        # Seting the window Geometry
        self.setGeometry(200, 150, 800, 650)
        self.setWindowTitle('Anubis IDE')
        self.setWindowIcon(QtGui.QIcon('Anubis.png'))


        widget = Widget()

        self.setCentralWidget(widget)
        self.show()

    ###########################        Start OF the Functions          ##################
    def PythonModeSet(self):
       global languageMode
       languageMode = "python"
       Python_Coloring.PythonHighlighter(codeText)

    def CSharpModeSet(self):
       global languageMode
       languageMode = "C#"
       CSharp_Coloring.CSharpHighlighter(codeText)


    def Run(self):
        #print(languageMode)

        if languageMode == "":
            previewText.append("Sorry, Choose language first.")

        if self.port_flag == 0:
            mytext = codeText.toPlainText()
        #
        ##### Compiler Part
        #
#            ide.create_file(mytext)
#            ide.upload_file(self.portNo)
            previewText.append("Sorry, there is no attached compiler.")

        else:
            previewText.append("Please Select Your Port Number First")


    # this function is made to get which port was selected by the user
    @QtCore.pyqtSlot()
    def PortClicked(self):
        action = self.sender()
        self.portNo = action.text()
        self.port_flag = 0



    # I made this function to save the code into a file
    def save(self):
        self.inputSignal.readingSignal.emit("name")


    # I made this function to open a file and exhibits it to the user in a text editor
    def Open(self):
        global languageMode
        file_name = QFileDialog.getOpenFileName(self,'Open File','/home')
        extension = file_name[0].split(".")[1]
        if file_name[0]:
            f = open(file_name[0],'r')
            with f:
               if extension == "py":
                  data = f.read()
                  self.Open_Signal.readingSignal.emit(data)
                  languageMode = "python"
                  Python_Coloring.PythonHighlighter(codeText)

               if extension == "cs":
                  data = f.read()
                  self.Open_Signal.readingSignal.emit(data)
                  languageMode = "C#"
                  CSharp_Coloring.CSharpHighlighter(codeText)


#
#
############ end of Class ############
#
#

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = UI()
    # ex = Widget()
    sys.exit(app.exec_())
```