CSE426: Software Maintenance & Evolution

Assignment

Evolving the IDE

Prof. Dr. Ayman Bahaa

Eng. Sally Shaker

Hossam Eldin Khaled Mohamed Ali Elshaer

Sr-1 CESS 16P3025

Table of Contents

Introduction

In this assignment, we are evolving the previously studied IDE. In the previous study, we used reverse engineering concepts to come up with the design and software requirements specification documents from the source code.

To move forward, and evolve the IDE, using forward engineering. Now that we get new requirements, we need to modify our SRS, design and finally the code. Regression, unit and system testing were used to make sure the newly developed components perform their tasks isolated on their own, then to make sure that the new code did not break any older functionalities and finally to make sure that the software as a whole deliver the requirements.

Both requirements are implemented and shown in this assignment; the fast executer is available for python only through a shortcut (Ctrl+e) and also available at the top menu, and support for C# programming language format, the user can select his preferred language at the top menu, the IDE automatically sets the language when opening a file, the IDE saves the code in the correct file extensions.

In this document, the modified SRS, design, code and user guide (screenshots) are presented, new SRS or design elements are presented in a blue color to make it easier to observe the change.

The source code is available in the appendix and also available as a public repository at GitHub.

GitHub Repository:

https://github.com/hossamshh/Anubis-IDE.git

Installation Guide:

1. Open your favorite python IDE, must have integration with Git, Ex. PyCharm.
2. On your IDE, open a terminal, go to your desired installation folder.
3. Simply execute the command: git clone https://github.com/hossamshh/Anubis-IDE.git
4. Or simply download the code from the git URL, and open it with your favorite python IDE.
5. The IDE will probably offer to install the dependencies and required packages automatically, if not check readme file and install them manually.
6. Click run and that is it.

Software Requirements Specification

Functional Requirement

1. The software system is an integrated development environment (IDE).
2. The IDE is primarily needed for MicroPython.
3. The IDE supports at least one version control system.
4. The IDE provides a list of all available ports on the running computer.
5. The user needs to select a port prior to running the code.
6. The IDE offers basic functionalities: code editor to write code, save file, open file and create new file.
7. The IDE saves files in the same opened directory.
8. The IDE supports all file types necessary for software development and opens them in the correct format.
9. The IDE allows the user to open multiple files in different tabs.
10. Color coding is implemented in real time as the user is writing code.
11. The IDE has a code editor panel that uses color coding for all reserved words in the used programming language.
12. The IDE provides feedback for syntax errors.
13. The IDE has a panel that displays all files and folders in the current directory.
14. The IDE has a panel that displays current running tasks and potential errors.
15. The IDE offers space customization between the three panels.

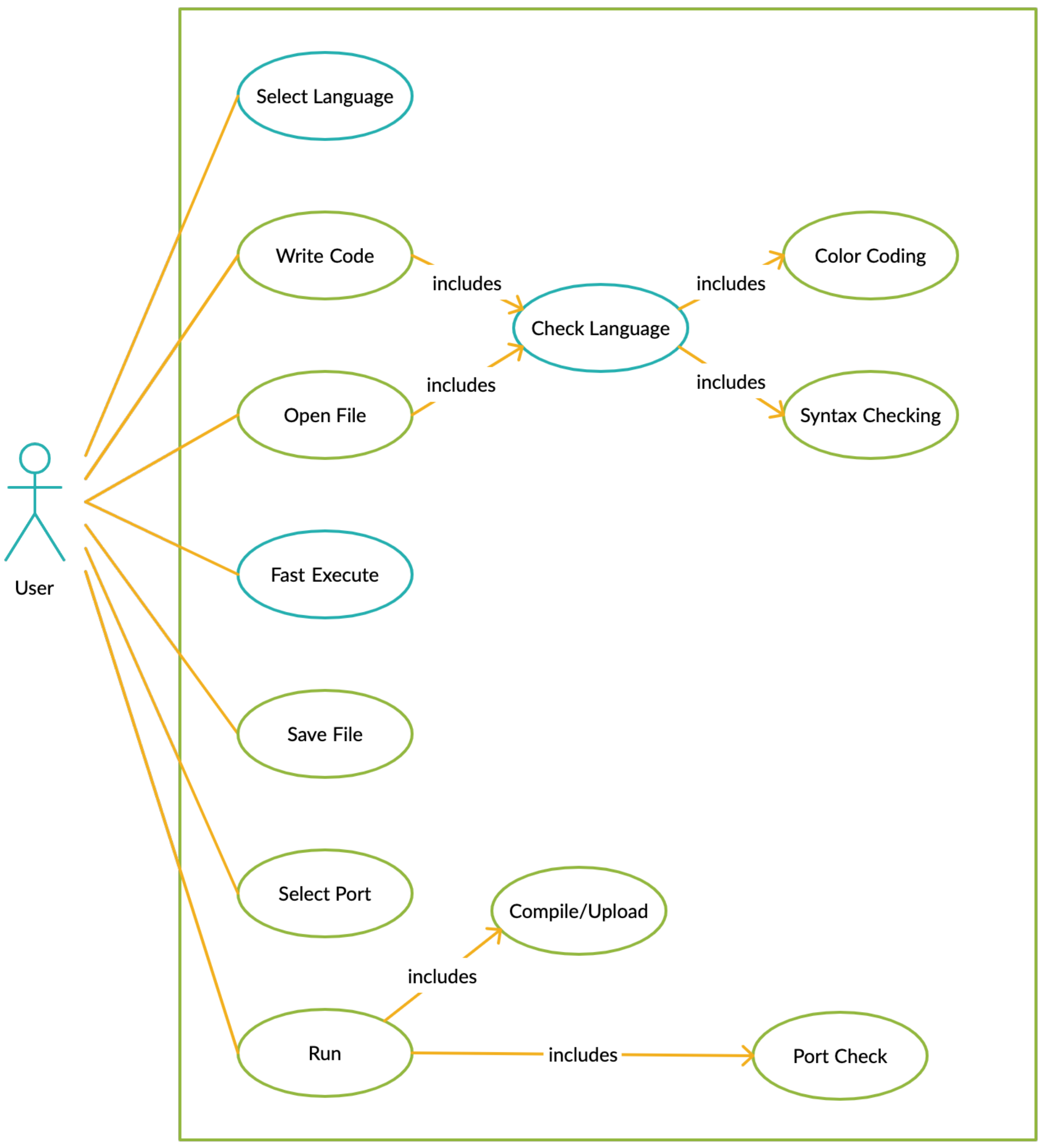
16. The IDE has fast execution option, when the user wants to test a new function in python, the IDE can generate the main function that calls the user newly created function to test it.
17. The IDE offers support for C# programming language.
18. The IDE allows the user to choose between python and C# to write new code.
19. When the user opens a file, the IDE checks the file extension and set the programming language accordingly.
20. When the user saves a file, the IDE save it in the correct extension.
21. The IDE offers different color coding for C# programming language.

Non-Functional Requirement

1. The software must be written in python.
2. Must use agile process model and deliver increments in 2 months cycle.
3. The software must include detailed installation guide.
4. The software must declare dependencies and how to solve them.
5. The software must be compatible with major operating systems: Windows, Linux and macOS
6. The software installation requirements must not exceed 2GB of RAM.
7. The software must be delivered in 6 months (3 cycles)
8. Syntax errors detection and feedback must occur within one second.
9. The source code must follow one naming convention.

10. Must update user guide when new requirements are added.

Use Case Diagram



Use Case Description

1. Select language:

Name	Select language
Main actor	User.
Description	User chooses a programming language between python and C#.
Goal	The user selects his preferred programming language.
Trigger	The user clicks on the language menu.
Include	None.
Pre-condition	The IDE is installed correctly and running, the language menu appears at the top.
Input	Programming language from the menu.
Post-condition	The selected language is set as the current language.
Output	The selected language is displayed in the top menu.
Normal path	1. The user opens the languages menu. 2. The user selects a programming language. 3. The language is set and displayed in the top menu.
Alternative path	3.1 a. The language is not set. b. The user can report the error.

2. Write code:

Name	Write code
Main actor	User
Description	User is writing new code or editing existing code using the IDE.
Goal	User writes code that gets color coding and highlights for syntax errors.
Trigger	User starts writing in the opened tab.
Include	Check Language
Pre-condition	The IDE is installed correctly and running, a new tab is opened automatically to write code in a new file.
Input	Code
Post-condition	The written code is colored according to color codes and syntax errors are underlined.
Output	Color coded and underlined for syntax errors code.
Normal path	1. Start writing code in a new tab or open a file. 2. The code is colored, and syntax errors are underlined.
Alternative path	1.1 a. The code is not colored correctly. b. The user can report the error. 2.2 a. The code has syntax errors but not underlined. b. The user can report the error.

3. Check language:

Name	Check language
Main actor	None.
Description	The IDE checks the current language or the selected file extension and perform color coding and syntax analysis accordingly.
Goal	The system assigns the correct color codes and syntax analyzer.
Trigger	User writes new code (check based on selected language) or opens an existing file (check based on file extension)
Include	Color Coding, Syntax Checking
Pre-condition	Code exists in the opened tab.
Input	Plain code
Post-condition	The written code is colored according to color codes and syntax errors are underlined.
Output	Code is color coded and highlighted for syntax error code.
Normal path	1. Start writing code in a new tab or open a file. 2. The code is colored, and syntax errors are underlined according to the chosen language.
Alternative path	2.1 a. Code is not colored, or syntax error are not underlined. b. The user can report the error. 2.2 a. The language is not set correctly. b. The user can report the error.

4. Color coding:

Name	Color coding
Main actor	None.
Description	Color code the keywords in the code.
Goal	The code is color coded for enhance readability.
Trigger	User writes some code or opens an existing file.
Include	None.
Pre-condition	Code exists in the opened tab.
Input	Plain code.
Post-condition	Code is color coded.
Output	Colored code.
Normal path	1. User writes some code or opens a file. 2. The code is color coded.
Alternative path	2.1 a. The code is not color coded. b. The user can report the error.

5. Syntax checking:

Name	Syntax checking
Main actor	None
Description	Syntax errors are underlined.
Goal	Underlined syntax errors to help the user correcting them.
Trigger	User writes some code or opens a file.
Include	None.
Pre-condition	Code exists in the opened tab.
Input	Plain code.
Post-condition	The code is underlined where syntax errors exist.
Output	Underlined code.
Normal path	1. User writes some code or opens a file. 2. Syntax errors are underlined.
Alternative path	2.1 a. The code has syntax errors but not underlined. b. The user can report the error.

6. Open file:

Name	Open file
Main actor	User
Description	The user opens a file into the IDE.
Goal	The user opens a file into the IDE to get color coding and syntax errors check.
Trigger	The clicks on a file from the left panel or click on the open file menu.
Include	Check language
Pre-condition	The user has a file to open.
Input	File.
Post-condition	The selected file is opened in the IDE, code is colored, and syntax errors are underlined.
Output	Color coded and underlined for syntax errors code.
Normal path	1. The user clicks on the open menu. 2. The user chooses the desired file. 3. The file is opened in the IDE. 4. Code is colored and syntax errors are underlined.
Alternative path	3 a. The user chooses an unsupported file. b. The program displays a message “wrong file type”.

7. Fast execute:

Name	Fast execute
Main actor	User
Description	The user wants to test a new function in python, the IDE helps by writing the main and calls the new function inside.
Goal	The IDE helps the user by writing the main and calls the new function inside.
Trigger	The clicks on the fast execute menu.
Include	None.
Pre-condition	Python is selected as programming language
Input	Code.
Post-condition	The main function is written, and the new function is called inside the main.
Output	Code is appended with the main function.
Normal path	1. The user selected python as programming language. 2. The user clicks on the fast execute menu. 3. The main function is appended to the code and the new function is called inside it.
Alternative path	3.1 a. The user did not choose python b. The program displays a message "fast execute is only available for python". 3.2 a. The user did write a function to test b. The program appends an empty main function.

8. Select port:

Name	Select port
Main actor	User
Description	User selects the port of the pyboard to upload the code.
Goal	The port is selected to upload the code to the pyboard.
Trigger	The user clicks on the select port menu.
Include	None.
Pre-condition	A pyboard is connected to the computer.
Input	A port from the available ports list.
Post-condition	A port is selected.
Output	The selected port is marked as selected.
Normal path	1. The user connects a pyboard to the computer. 2. The user opens the ports menu. 3. The user selects a port.
Alternative path	3.1 a. The user cannot find his port in the list. b. Display a message try reconnecting the board.

9. Save file:

Name	Save file
Main actor	User
Description	The user saves a file from the IDE on his disk space.
Goal	The file is saved on the disk for future use.
Trigger	The user clicks on the save menu.
Include	None.
Pre-condition	A file is opened in the IDE.
Input	A file.
Post-condition	The opened file is saved on the disk space.
Output	A confirmation message that the file has been saved.
Normal path	1. The user clicks on the save menu. 2. If it’s not an existing file, the user chooses file name and location. 3. The file is saved on the disk space.
Alternative path	3.1 a. The IDE could not save the file due to ungranted permissions. b. Display a message to grant write permission to the IDE. 3.2 a. The IDE could not save the file due to insufficient space. b. Display a message “insufficient space”.

10. Run:

Name	Run
Main actor	User
Description	The IDE checks the port, complies and uploads the code
Goal	The code is compiled and uploaded to the board to run
Trigger	The user clicks on the run menu.
Include	Compile/Upload, Port Check.
Pre-condition	A file is opened in the IDE.
Input	Code.
Post-condition	The code is uploaded to the board
Output	A successful message that the code is uploaded.
Normal path	1. The user clicks on the run menu. 2. The program checks the port 3. The program checks for compilation errors. 4. The program uploads the code to the board
Alternative path	2 a. The program found no port or port error. b. The program displays a message “Please select a port”. c. The program does not upload the code. 3 a. The program found compilation errors. b. The program underlines the code causing the errors. c. The program does not upload the code.

11.Port check:

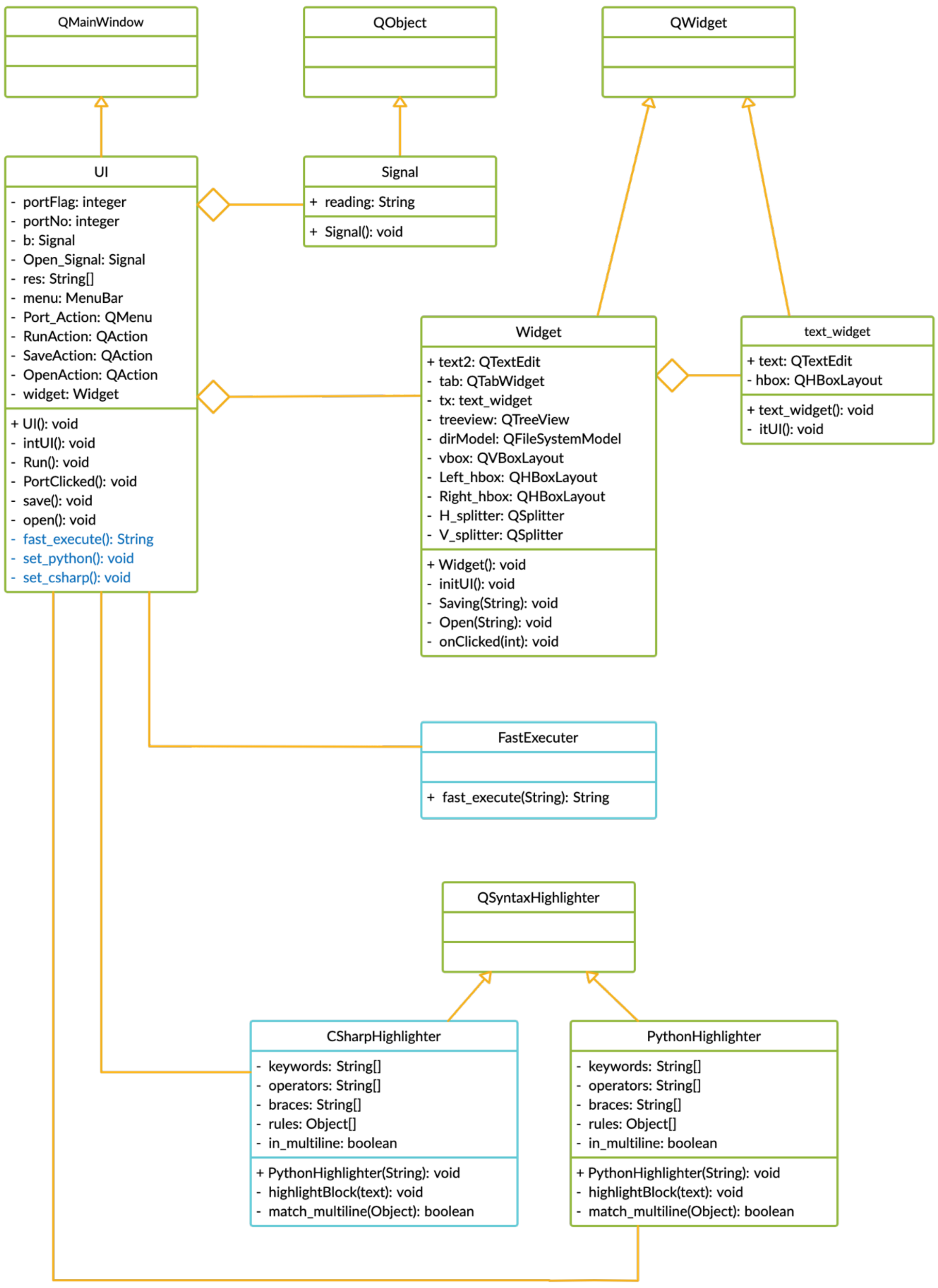
Name	Port check
Main actor	None.
Description	The IDE checks that a port is selected and that a board is connected to that port.
Goal	Check if a valid port exists to upload the code.
Trigger	The user clicks on the run menu.
Include	None.
Pre-condition	None.
Input	Selected port.
Post-condition	Port is validated.
Output	Boolean, if the port is valid or not.
Normal path	1. The IDE checks if a port is valid.
Alternative path	1.1 a. The program found no port or port error. b. The program displays a message "Please select a port". c. The program does not upload the code.

12.Compile/Upload:

Name	Compile/Upload
Main actor	None
Description	The IDE complies the code and upload it to the board to run.
Goal	The code is compiled and uploaded to the board to run
Trigger	The user clicks on the run menu.
Include	None.
Pre-condition	The IDE found a valid port.
Input	Code.
Post-condition	The code is uploaded to the board
Output	Compiled code.
Normal path	1. The program checks for compilation errors. 2. The program uploads the code to the board 3. Displays a success message
Alternative path	3 a. The program found compilation errors. b. The program underlines the code causing the errors. c. The program does not upload the code. d. The program displays unsuccessful message.

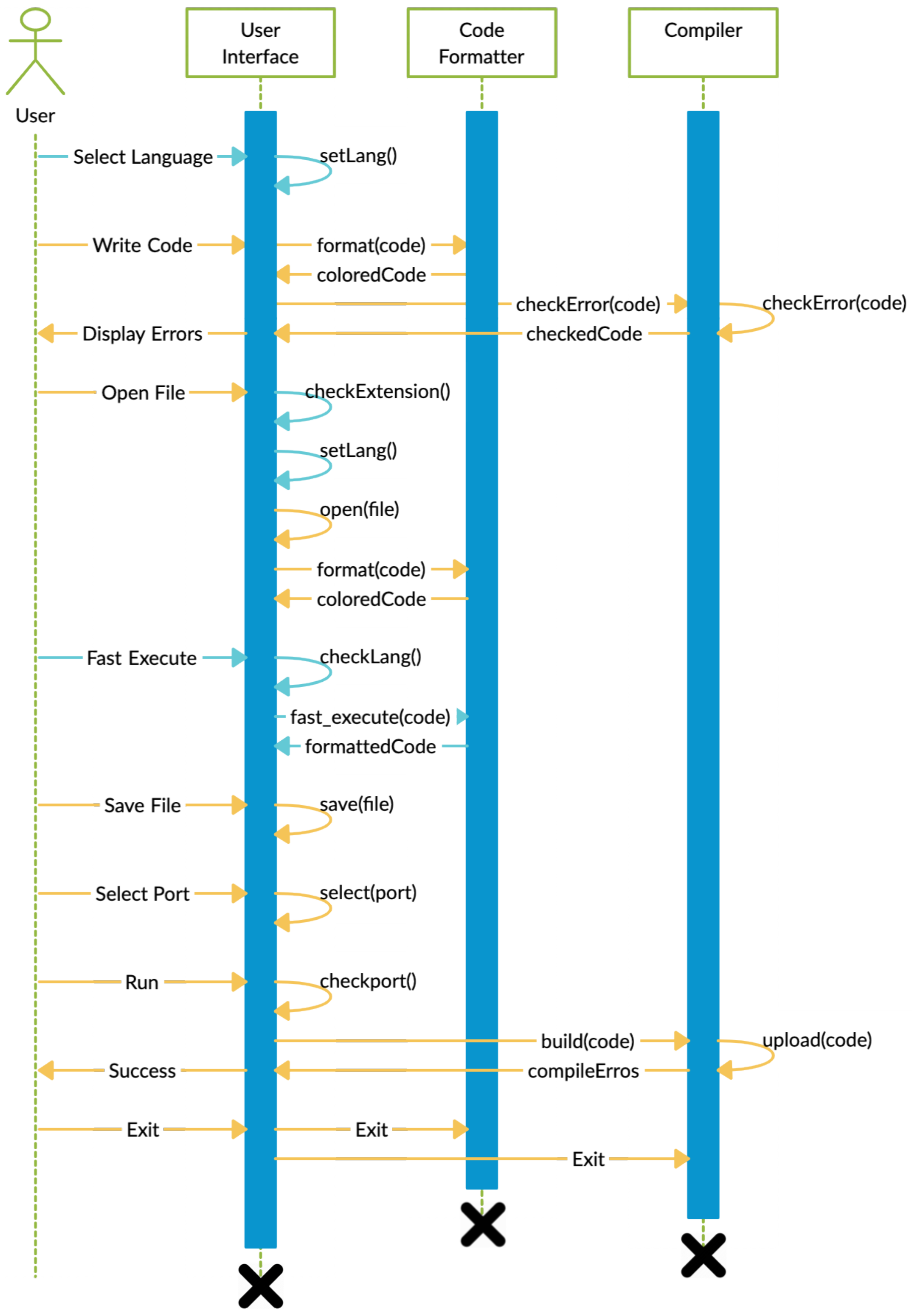
Design Diagrams

Class Diagram

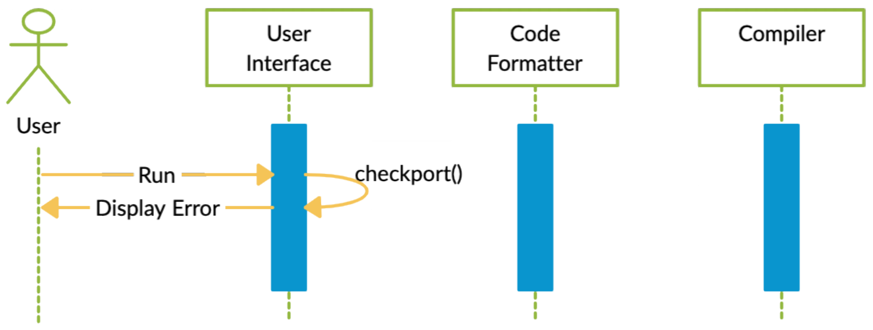


Sequence Diagrams

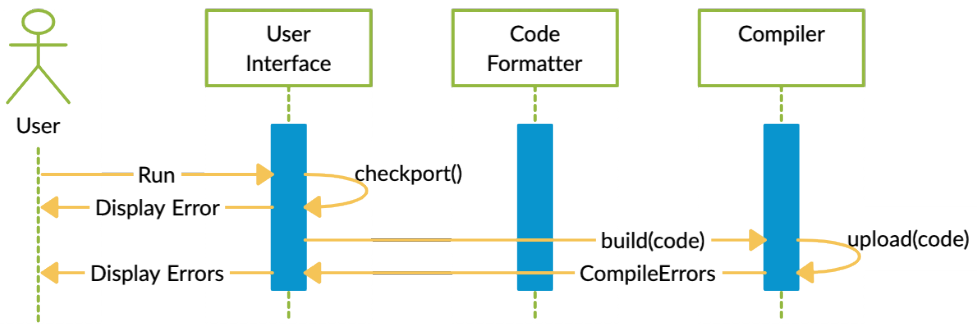
1. Normal Scenarios:



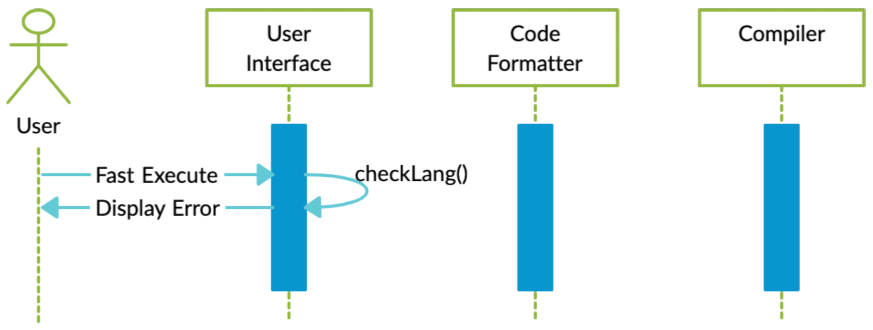
2. Port check error:



3. Compilation error:

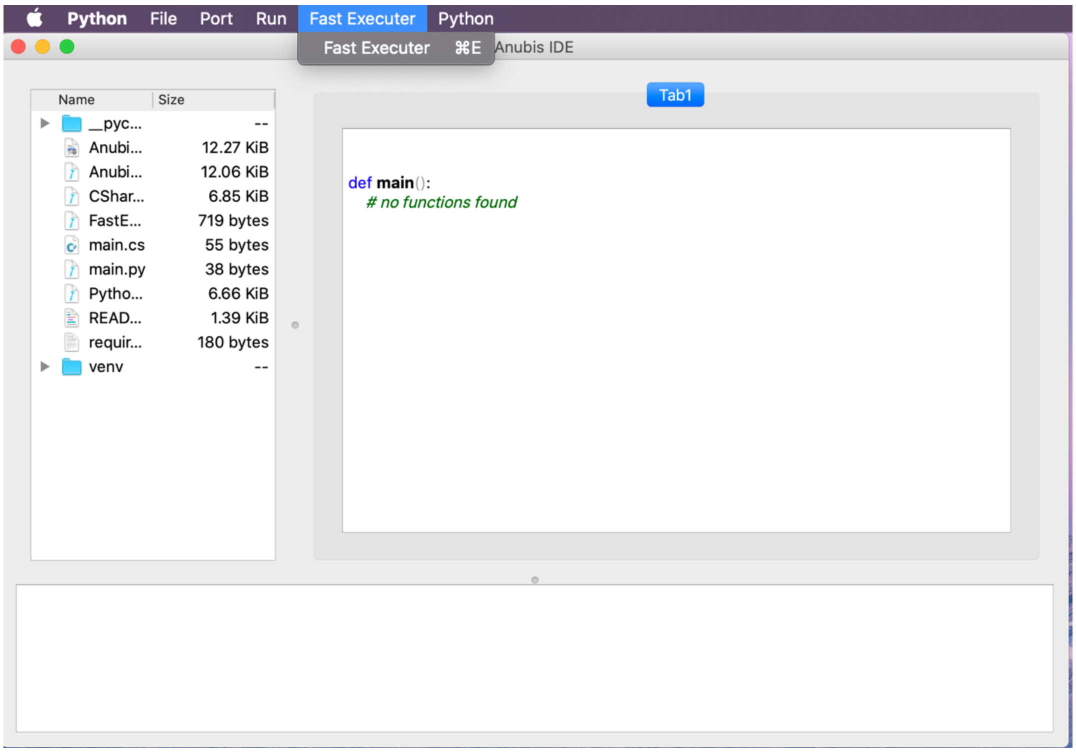


4. Fast execute language error:



Test Runs Screenshots

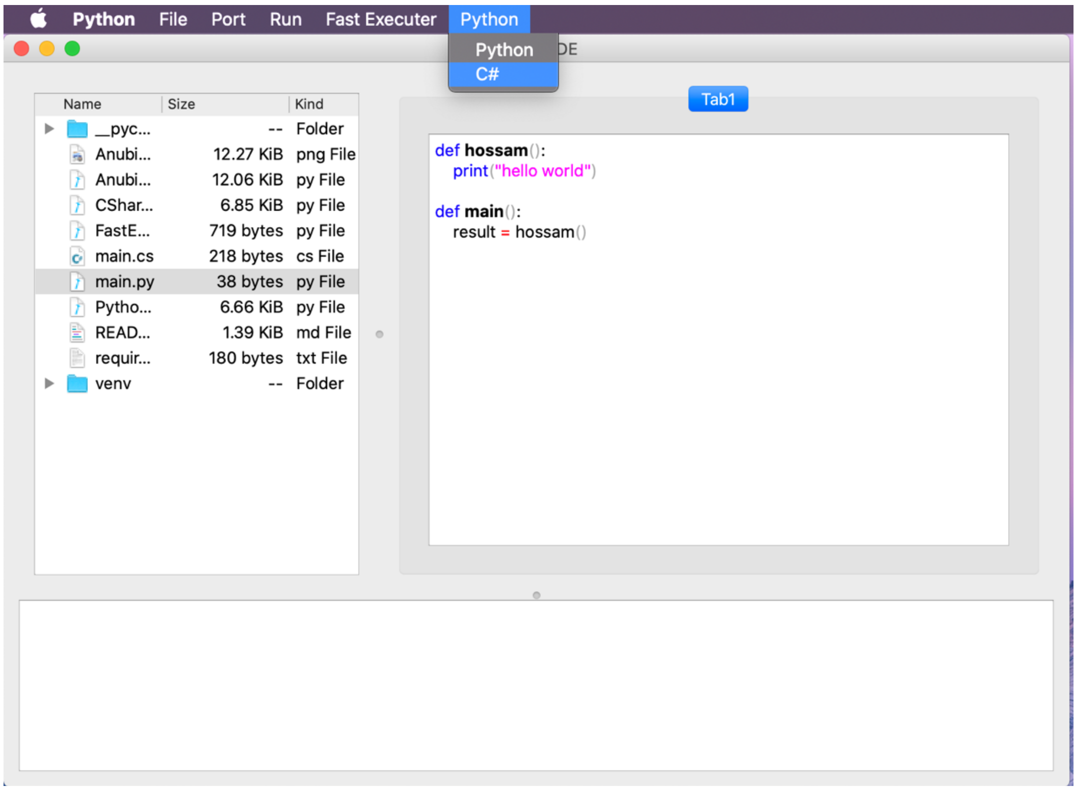
1. Fast executer on empty file python



Name	Size
__pyc...	--
Anubi...	12.27 KiB
Anubi...	12.06 KiB
CShar...	6.85 KiB
FastE...	719 bytes
main.cs	55 bytes
main.py	38 bytes
Pytho...	6.66 KiB
READ...	1.39 KiB
requir...	180 bytes
venv	--

```
def main():
  # no functions found
```

2. Fast executer result on python file, and language menu

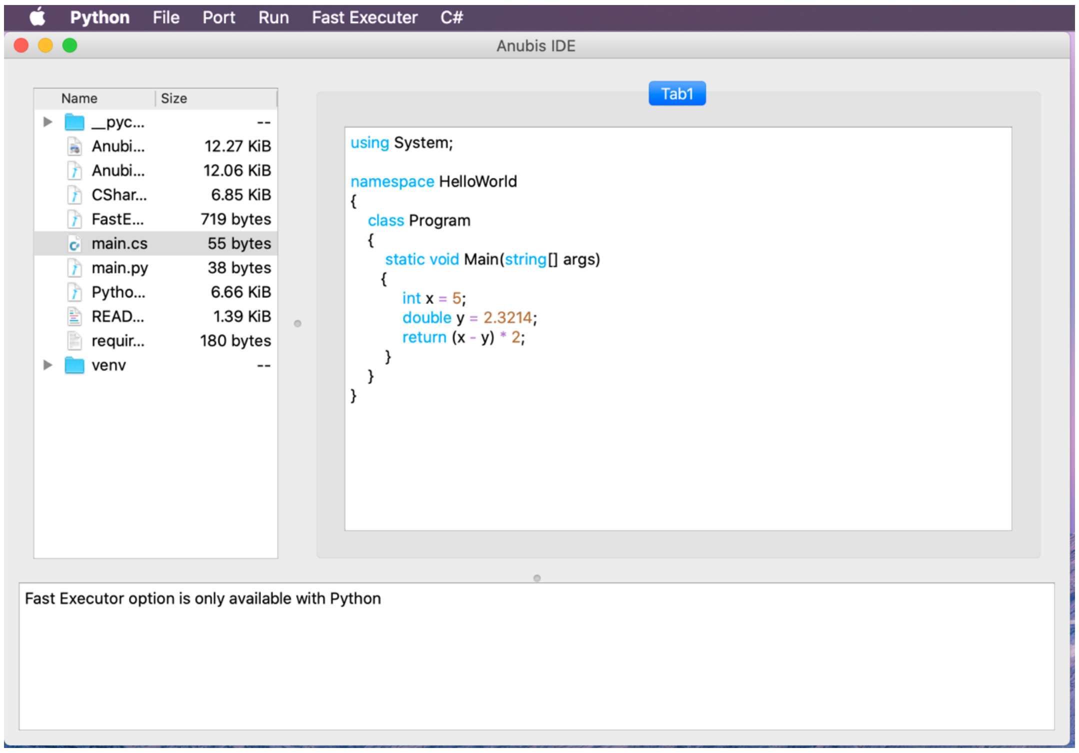


Name	Size	Kind
__pyc...	--	Folder
Anubi...	12.27 KiB	png File
Anubi...	12.06 KiB	py File
CShar...	6.85 KiB	py File
FastE...	719 bytes	py File
main.cs	218 bytes	cs File
main.py	38 bytes	py File
Pytho...	6.66 KiB	py File
READ...	1.39 KiB	md File
requir...	180 bytes	txt File
venv	--	Folder

```
def hossam():
    print("hello world")

def main():
    result = hossam()
```

3. Color coding on fast executer error message on C# sample program



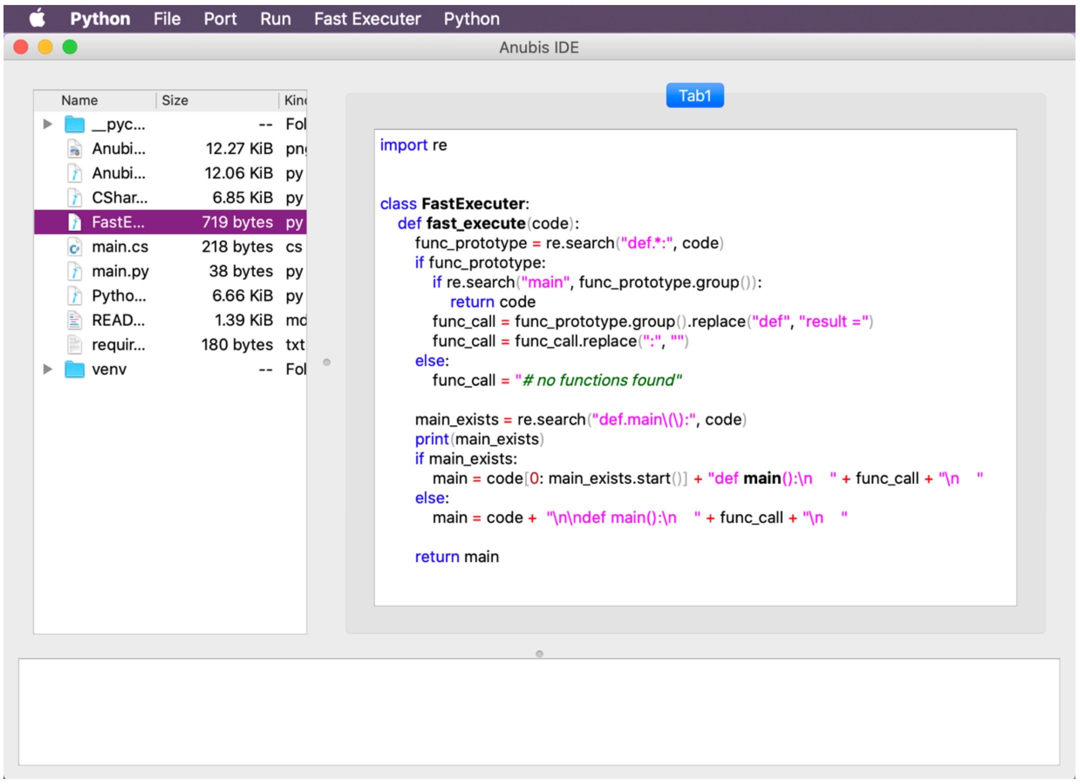
Name	Size
__pyc...	--
Anubi...	12.27 KiB
Anubi...	12.06 KiB
CShar...	6.85 KiB
FastE...	719 bytes
main.cs	55 bytes
main.py	38 bytes
Pytho...	6.66 KiB
READ...	1.39 KiB
requir...	180 bytes
venv	--

```
using System;

namespace HelloWorld
{
   class Program
   {
      static void Main(string[] args)
      {
         int x = 5;
         double y = 2.3214;
         return (x - y) * 2;
      }
   }
}
```

```
Fast Executor option is only available with Python
```

4. Fast executer source code and normal python color coding



Name	Size	Kin
__pyc...	--	Fol
Anubi...	12.27 KiB	pn
Anubi...	12.06 KiB	py
CShar...	6.85 KiB	py
FastE...	719 bytes	py
main.cs	218 bytes	cs
main.py	38 bytes	py
Pytho...	6.66 KiB	py
READ...	1.39 KiB	md
requir...	180 bytes	txt
venv	--	Fol

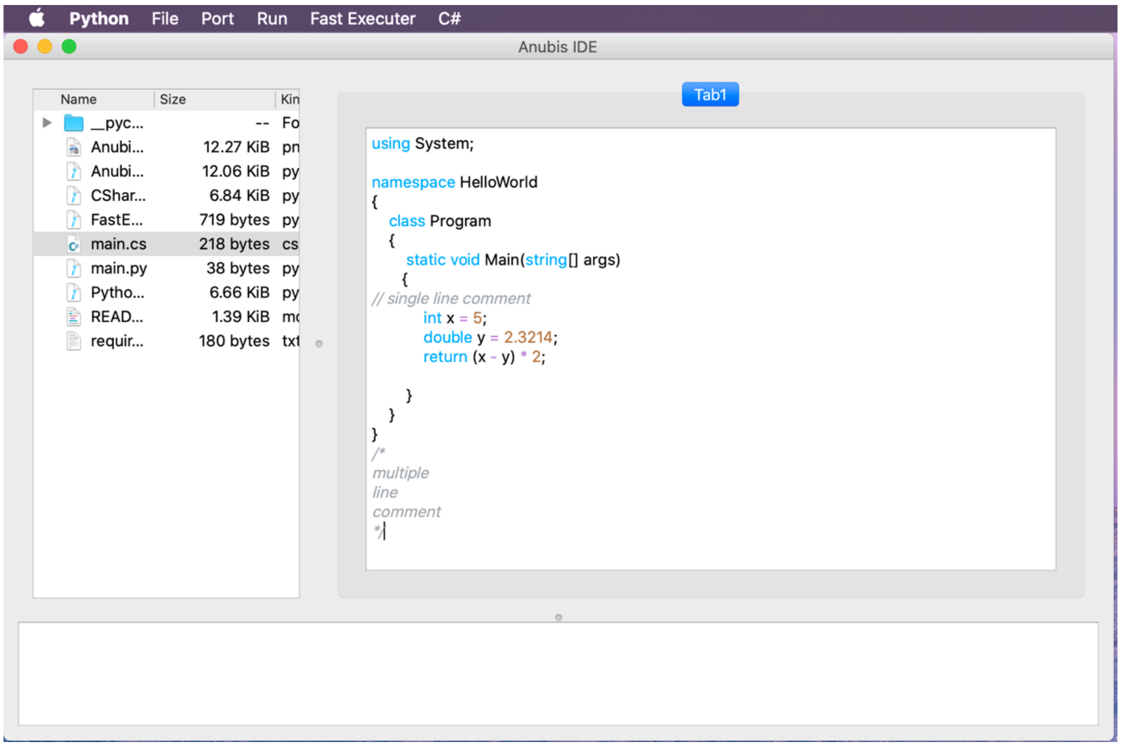
```python
import re


class FastExecuter:
   def fast_execute(code):
      func_prototype = re.search("def.*:", code)
      if func_prototype:
         if re.search("main", func_prototype.group()):
            return code
         func_call = func_prototype.group().replace("def", "result =")
         func_call = func_call.replace(":", "")
      else:
         func_call = "# no functions found"

      main_exists = re.search("def.main\(\):", code)
      print(main_exists)
      if main_exists:
         main = code[0: main_exists.start()] + "def main():\n    " + func_call + "\n    "
      else:
         main = code +  "\n\ndef main():\n    " + func_call + "\n    "

      return main
```

5. C# color coding example with comments and saved file “main.cs”



```
Name        Size        Kin
_pyc...     --          Fo
Anubi...    12.27 KiB   pn
Anubi...    12.06 KiB   py
CShar...    6.84 KiB    py
FastE...    719 bytes   py
main.cs     218 bytes   cs
main.py     38 bytes    py
Pytho...    6.66 KiB    py
READ...     1.39 KiB    mc
requir...   180 bytes   txt
```

```
using System;

namespace HelloWorld
{
   class Program
   {
      static void Main(string[] args)
      {
// single line comment
         int x = 5;
         double y = 2.3214;
         return (x - y) * 2;

      }
   }
}
/*
multiple
line
comment
*/
```

Appendix

Anubis source code

```
#############      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 CSharpColoring
import FastExecuter
from PyQt5 import QtCore
from PyQt5 import QtGui
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from pathlib import Path

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
    reading = 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)
text = QTextEdit
text2 = QTextEdit
language = "Python"

#
#
#
#
############ 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 text
        text = QTextEdit()
        Python_Coloring.PythonHighlighter(text)
        hbox = QHBoxLayout()
        hbox.addWidget(text)
        self.setLayout(hbox)



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




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

    def __init__(self, ui):
        super().__init__()
        self.initUI()
        self.ui = ui

    def initUI(self):

        # This widget is responsible of making Tab in IDE which makes the Text
editor looks nice
        tab = QTabWidget()
        tx = text_widget()
        tab.addTab(tx, "Tab"+"1")

        # second editor in which the error messeges and succeeded connections will
be shown
        global text2
        text2 = QTextEdit()
        text2.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.on_clicked)

        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(text2)

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

        self.setLayout(Final_Layout)

    # defining a new Slot (takes string) to save the text inside the first text
editor
    @pyqtSlot(str)
    def Saving(s):
        if language == "Python":
            with open('main.py', 'w') as f:
                TEXT = text.toPlainText()
                f.write(TEXT)
        else:
            with open('main.cs', 'w') as f:
                TEXT = text.toPlainText()
                f.write(TEXT)

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

    def on_clicked(self, index):

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

        file_ext = nn[0].split(".")[1]
        if file_ext == "py":
            UI.set_python(self.ui)
        else:
            UI.set_csharp(self.ui)

        if nn[0]:
            f = open(nn[0],'r')
            with f:
                data = f.read()
                text.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):
    b = Signal()
    b.reading.connect(Widget.Saving)
    b.reading.emit(s)

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

    def intUI(self):
        self.port_flag = 1
        self.b = Signal()

        self.Open_Signal = Signal()

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

        # connecting (self.b) with reading function
        self.b.reading.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')
        fast_menu = menu.addMenu('Fast Executer')
        self.language_menu = menu.addMenu('Python')

        # 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)

        fast_action = QAction("Fast Executer", self)
        fast_action.triggered.connect(self.fast_execute)
        fast_action.setShortcut("Ctrl+e")
        fast_menu.addAction(fast_action)

        python_action = QAction('Python', self)
        python_action.triggered.connect(self.set_python)
        csharp_action = QAction('C#', self)
        csharp_action.triggered.connect(self.set_csharp)

        self.language_menu.addAction(python_action)
        self.language_menu.addAction(csharp_action)

        # Seting the window Geometry
        self.setGeometry(200, 150, 600, 500)
        self.setWindowTitle('Anubis IDE')
        self.setWindowIcon(QtGui.QIcon('Anubis.png'))


        widget = Widget(self)

        self.setCentralWidget(widget)
        self.show()

    ###########################        Start OF the Functions
##################
    def Run(self):
        if self.port_flag == 0:
            mytext = text.toPlainText()
        #
        ##### Compiler Part
        #
#             ide.create_file(mytext)
#             ide.upload_file(self.portNo)
            text2.append("Sorry, there is no attached compiler.")

        else:
            text2.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.b.reading.emit("name")

    # I made this function to open a file and exhibits it to the user in a text
editor
```

```
    def open(self):
        file_name = QFileDialog.getOpenFileName(self,'Open File','/home')
        file_ext = file_name[0].split(".")[1]
        if file_ext == "py":
            self.set_python()
        else:
            self.set_csharp()
        if file_name[0]:
            f = open(file_name[0],'r')
            with f:
                data = f.read()
            self.Open_Signal.reading.emit(data)

    def fast_execute(self):
        if language == "Python":
            main_func = FastExecuter.FastExecuter.fast_execute(text.toPlainText())
            text.setText(main_func)
        else:
            text2.append("Fast Executor option is only available with Python")

    def set_python(self):
        self.language_menu.setTitle("Python")
        global language
        language = "Python"
        Python_Coloring.PythonHighlighter(text)

    def set_csharp(self):
        self.language_menu.setTitle("C#")
        global language
        language = "C#"
        CSharpColoring.CSharpHighlighter(text)

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

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

Fast executer source code

```python
import re


class FastExecuter:
    def fast_execute(code):
        func_prototype = re.search("def.*:", code)
        if func_prototype:
            if re.search("main", func_prototype.group()):
                return code
            func_call = func_prototype.group().replace("def", "result =")
            func_call = func_call.replace(":", "")
        else:
            func_call = "# no functions found"

        main_exists = re.search("def.main\(\):", code)
        print(main_exists)
        if main_exists:
            main = code[0: main_exists.start()] + "def main():\n    " + func_call +
"\n    "
        else:
            main = code +  "\n\ndef main():\n    " + func_call + "\n    "

        return main
```

CSharpColoring source code

```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
STYLES = {
    'keyword': format([3, 169, 252]), #blue
    'operator': format([176, 124, 207]), #purple
    'brace': format('black'),
    'string': format([85, 171, 79]), #green
    'string2': format([104, 214, 96]), #light green
    'comment': format([163, 168, 173], 'italic'), #grey
    'numbers': format([171, 114, 65]), #brown
}

class CSharpHighlighter(QSyntaxHighlighter):
    """Syntax highlighter for the Python language.
    """
    # C# keywords
    keywords = [
        'abstract', 'bool',        'continue',    'decimal', 'default',
        'event',    'explicit',    'extern',  'char',        'checked',
        'class',   'const',   'break',   'as',      'base',
        'delegate',    'is,'      'lock',        'long',        'num',
        'byte',     'case',        'catch',   'false',   'finally',
        'fixed',   'float',   'for',     'foreach',  'static',
        'goto',        'if',      'implicit',    'in',      'int',
        'interface','internal',     'do',      'double',  'else',
        'namespace','new',      'null',        'object',  'operator',
        'out',     'override',    'params',  'private', 'protected',
        'public',  'readonly',    'sealed',  'short',   'sizeof',
        'ref',     'return',  'sbyte',   'stackalloc','static',
        'string',  'struct',  'void',        'volatile',    'while',
        'true',        'try',     'switch',  'this',        'throw',
        'unchecked' 'unsafe',  'ushort',  'using',   'using',
        'virtual', 'typeof',  'uint',        'ulong',   'out',
        'add',     'alias',   'async',   'await',       'dynamic',
        'from',    'get',     'orderby', 'ascending','decending',
        'group',   'into',        'join',        'let',     'nameof',
        'global',  'partial', 'set',     'remove',   'select',
        'value',   'var',     'when',        'Where',   'yield'
    ]
    # C# operators
    operators = [
        '=',
        # logical
        '!', '?', ':',
        # Comparison
        '==', '!=', '<', '<=', '>', '>=',
        # Arithmetic
        '\+', '-', '\*', '/', '\%', '\+\+', '--',
        # self assignment
        '\+=', '-=', '\*=', '/=', '\%=', '<<=', '>>=', '\&=', '\^=', '\|=',
        # Bitwise
        '\^', '\|', '\&', '\~', '>>', '<<',
    ]

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

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

            # comments // & /**/
            (r'//[^\n]*', 0, STYLES['comment']),
            (r'/\*[\s\S]*\*/$', 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.match_multiline(text, *self.tri_single)
        if not in_multiline:
            in_multiline = self.match_multiline(text, *self.tri_double)

    def match_multiline(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
```