

***CSE427: Software Maintenance & Evolution***

***Evolving The Editor***

**Submitted By:**

**Zaid Said Abdelaziz Zaid 16P6066**

**Submitted To:**

**Prof. Dr. Ayman M. Bahaa Eldin**

**Eng. Mohamed Elghamry**

Cairo 2021

Contents

[1. Purpose 3](#_Toc73109803)

[2. Cloning the GitHub repo after forking the original repo 3](#_Toc73109804)

[3. Setup the environment 4](#_Toc73109805)

[4. Functionality Description 4](#_Toc73109806)

[5. Class Diagram 5](#_Toc73109807)

[6. Sequence Diagram 6](#_Toc73109808)

[7. Use Case diagram 7](#_Toc73109809)

[8. System Screenshots 8](#_Toc73109810)

[8.1. Virtual COM Ports 8](#_Toc73109811)

[8.2. Execute Normal Code “print(‘hello’)” 9](#_Toc73109812)

[8.3. Execute a user’s defined function. 10](#_Toc73109813)

[8.4. Execute a function which requires arguments 11](#_Toc73109814)

[8.5. Handle if more arguments were passed 12](#_Toc73109815)

[8.6. Handle if less arguments were passed 13](#_Toc73109816)

[8.7. Handle syntax errors 14](#_Toc73109817)

[9. Code 15](#_Toc73109818)

[10. GitHub Repo 34](#_Toc73109819)

# Purpose

This document tends to show the detailed design of the system after being evolved, By providing the detailed class diagram, to represent the system and the methods of each class, & the sequence diagram to show the sequence of events starting from the user opening a project, editing the file and running the project.

Also showing the steps of preparing the environment, issue faced & solutions, and screenshots of the program.

# Text Description automatically generatedCloning the GitHub repo after forking the original repo

# 

# Text Description automatically generatedSetup the environment

# Functionality Description

The user will write his code in the write panel “tab 1”, then select the port and click “run”.

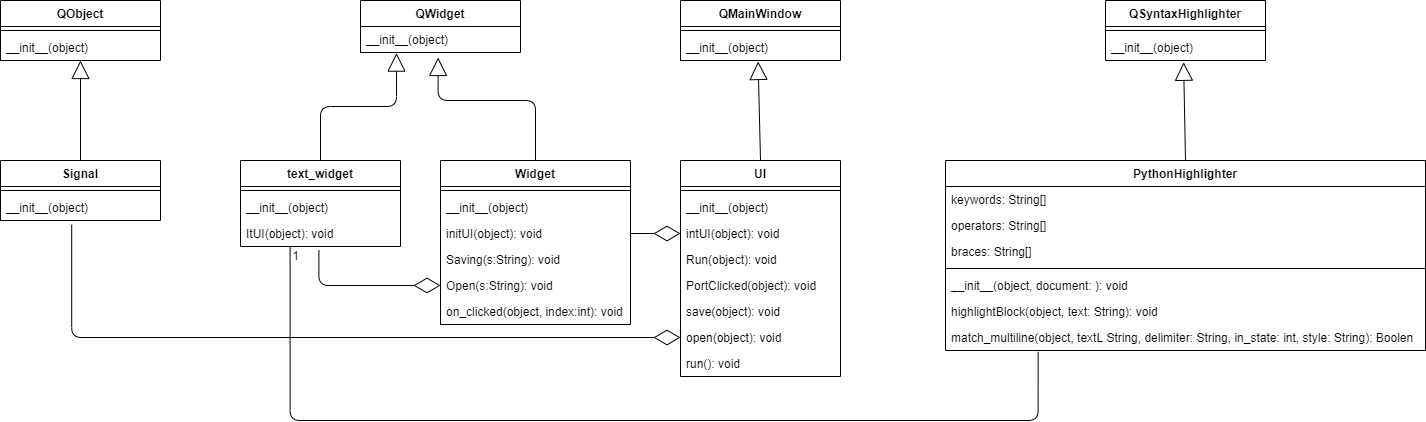
The system will be able to determine whether the user defined a new function, hence will wrap it inside the code and call the built in “exec” function to execute the user’s function.

If the user defined a function which takes arguments to run, so he\she must provide these arguments in the text field specified for it.

If the user didn’t define a new function and just used the built-in functions in python, the system will normally execute the commands if they are syntactically correct.

If the syntax was correct, then the output will be displayed in the output panel “bottom panel”, if there was any error, they are displayed in the output panel as well.

# Class Diagram



# Sequence Diagram

Diagram, schematic

Description automatically generated

# Diagram Description automatically generatedUse Case diagram

# System Screenshots

## Virtual COM Ports

Graphical user interface, application

Description automatically generated

## Execute Normal Code “print(‘hello’)”

Graphical user interface, application

Description automatically generated

## Execute a user’s defined function.

Graphical user interface, application

Description automatically generated

## Execute a function which requires arguments

Graphical user interface, application

Description automatically generated

## Handle if more arguments were passed

Graphical user interface, application

Description automatically generated

## Handle if less arguments were passed

Graphical user interface, application

Description automatically generated

## Handle syntax errors

Graphical user interface

Description automatically generated

# 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  ########

from io import StringIO

import sys

import glob

import serial

import Python\_Coloring

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

arguments = QLineEdit

#

#

#

#

############ 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):

        super().\_\_init\_\_()

        self.initUI()

    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, succeeded connections and output 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)

        # Arguments from the arguments text edit to be passed to the function call, they should be seperated by a comma

        labelForArgs = QLabel(self)

        labelForArgs.setText(

            "Arguments should be comma separated and in the correct order, ex: first\_argument, second\_argument")

        V\_splitter.addWidget(labelForArgs)

        global arguments

        arguments = QLineEdit(self)

        V\_splitter.addWidget(arguments)

        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):

        with open('main.py', '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])

        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')

        # 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)

        # Add new action in the menu for the fast execution

        MyRunAction = QAction("Run", self)

        MyRunAction.triggered.connect(self.run)

        # 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)

        # Seting the window Geometry

        self.setGeometry(200, 150, 600, 500)

        self.setWindowTitle('Anubis IDE')

        self.setWindowIcon(QtGui.QIcon('Anubis.png'))

        widget = Widget()

        self.setCentralWidget(widget)

        self.show()

    ###########################        Start OF the Functions          ##################

    '''

    This function was enhanced by 'Zaid Said Abdelaziz', this function will find whether the user entered a new function

    and wants to execute it by passing any parameters (if required)

    or the user just wants to execute a normal code using built-in functions in python

    '''

    def run(self):

        # Clear IDE console

        text2.clear()

        # text2.append("Working")

        # Get the code from the edit text

        code = text.toPlainText()

        args = arguments.text().split(',')

        function\_call\_start = code.find("def") + 4

        function\_call\_end = code.find("(")

        function\_call = code[function\_call\_start: function\_call\_end + 1]

        for arg in args:

            function\_call += arg + ','

        function\_call = function\_call[:-1] + ')'

        # print(function\_call)

        try:

            # Redirect console output to IDE console

            original\_stdout = sys.stdout

            result = StringIO()

            sys.stdout = result

            # Execute the normal code, no defined functions by the user

            if code.find('def') == -1:

                try:

                    exec(code)

                except Exception as e:

                        text.append(str(e))

            else:

                #Execute the defined function by the user

                exec(code + "\n" + function\_call, globals())

                # Show result in IDE console

            text2.append(result.getvalue())

            # Restore original stdout to print in console

            sys.stdout = original\_stdout

        except Exception as e:

            # logging.error(traceback.format\_exc())

            text2.append(str(e))

    # 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')

        if file\_name[0]:

            f = open(file\_name[0],'r')

            with f:

                data = f.read()

            self.Open\_Signal.reading.emit(data)

#

#

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

#

#

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    ex = UI()

    # ex = Widget()

    sys.exit(app.exec\_())

#############      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  ########

from io import StringIO

import sys

import glob

import serial

import Python\_Coloring

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

arguments = QLineEdit

#

#

#

#

############ 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):

        super().\_\_init\_\_()

        self.initUI()

    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, succeeded connections and output 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)

        # Arguments from the arguments text edit to be passed to the function call, they should be seperated by a comma

        labelForArgs = QLabel(self)

        labelForArgs.setText(

            "Arguments should be comma separated and in the correct order, ex: first\_argument, second\_argument")

        V\_splitter.addWidget(labelForArgs)

        global arguments

        arguments = QLineEdit(self)

        V\_splitter.addWidget(arguments)

        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):

        with open('main.py', '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])

        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')

        # 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)

        # Add new action in the menu for the fast execution

        MyRunAction = QAction("Run", self)

        MyRunAction.triggered.connect(self.run)

        # 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)

        # Seting the window Geometry

        self.setGeometry(200, 150, 600, 500)

        self.setWindowTitle('Anubis IDE')

        self.setWindowIcon(QtGui.QIcon('Anubis.png'))

        widget = Widget()

        self.setCentralWidget(widget)

        self.show()

    ###########################        Start OF the Functions          ##################

    '''

    This function was enhanced by 'Zaid Said Abdelaziz', this function will find wether the user entered a new function

    and wants to execute it by passing any parameters (if required)

    or the user just wants to execute a normal code using built-in functions in python

    '''

    def run(self):

        # Clear IDE console

        text2.clear()

        # text2.append("Working")

        # Get the code from the edit text

        code = text.toPlainText()

        args = arguments.text().split(',')

        function\_call\_start = code.find("def") + 4

        function\_call\_end = code.find("(")

        function\_call = code[function\_call\_start: function\_call\_end + 1]

        for arg in args:

            function\_call += arg + ','

        function\_call = function\_call[:-1] + ')'

        # print(function\_call)

        try:

            # Redirect console output to IDE console

            original\_stdout = sys.stdout

            result = StringIO()

            sys.stdout = result

            # Execute the normal code, no defined functions by the user

            if code.find('def') == -1:

                try:

                    exec(code)

                except Exception as e:

                        text.append(str(e))

            else:

                #Execute the defined function by the user

                exec(code + "\n" + function\_call, globals())

                # Show result in IDE console

            text2.append(result.getvalue())

            # Restore original stdout to print in console

            sys.stdout = original\_stdout

        except Exception as e:

            # logging.error(traceback.format\_exc())

            text2.append(str(e))

    # 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')

        if file\_name[0]:

            f = open(file\_name[0],'r')

            with f:

                data = f.read()

            self.Open\_Signal.reading.emit(data)

#

#

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

#

#

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    ex = UI()

    # ex = Widget()

    sys.exit(app.exec\_())

# GitHub Repo

GitHub: <https://github.com/ZaidSaid12/Anubis-IDE>

This Repo contains the images in higher quality if they weren’t clear in this report.