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

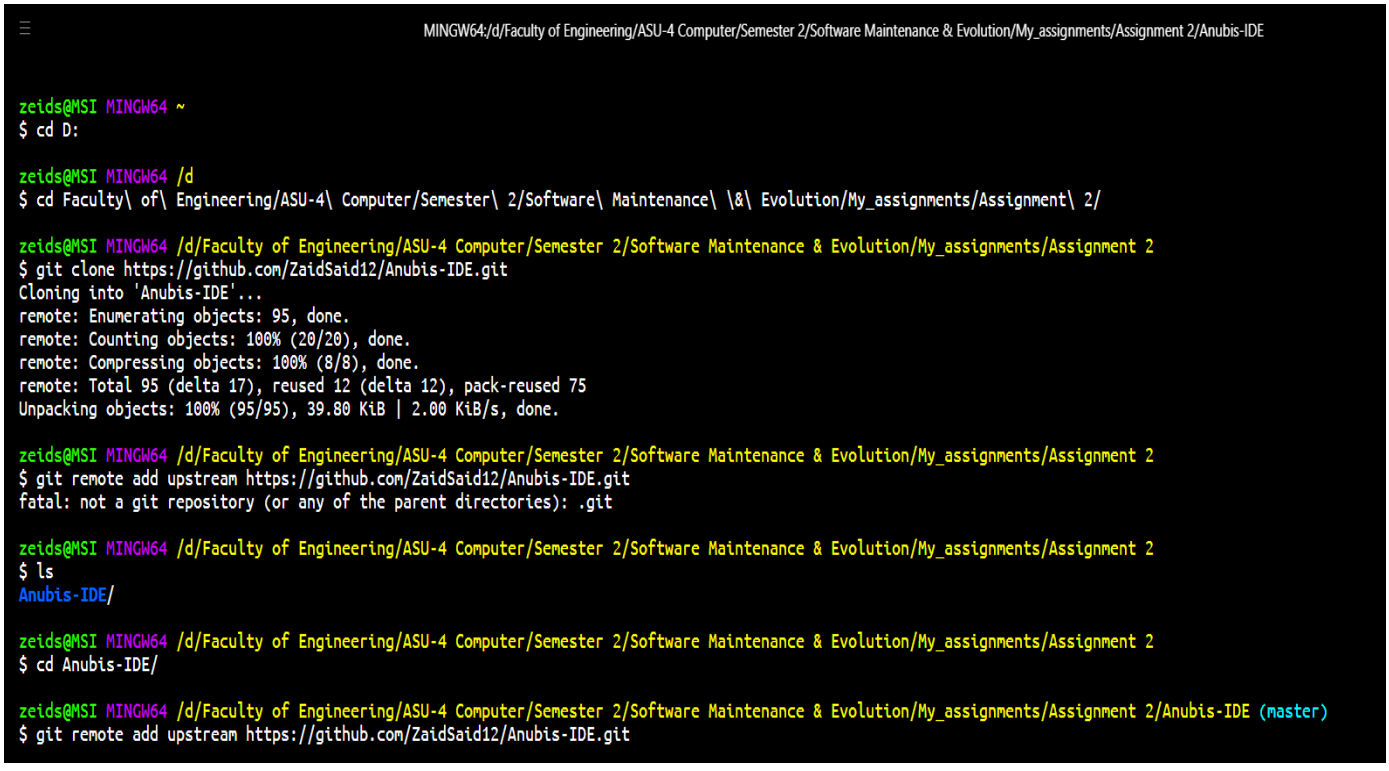
This document tends to show the detailed design of the system after being evolved, By providing the detailed class diagram, to represent the system and the methods of each class, & the sequence diagram to show the sequence of events starting from the user opening a project, editing the file and running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

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

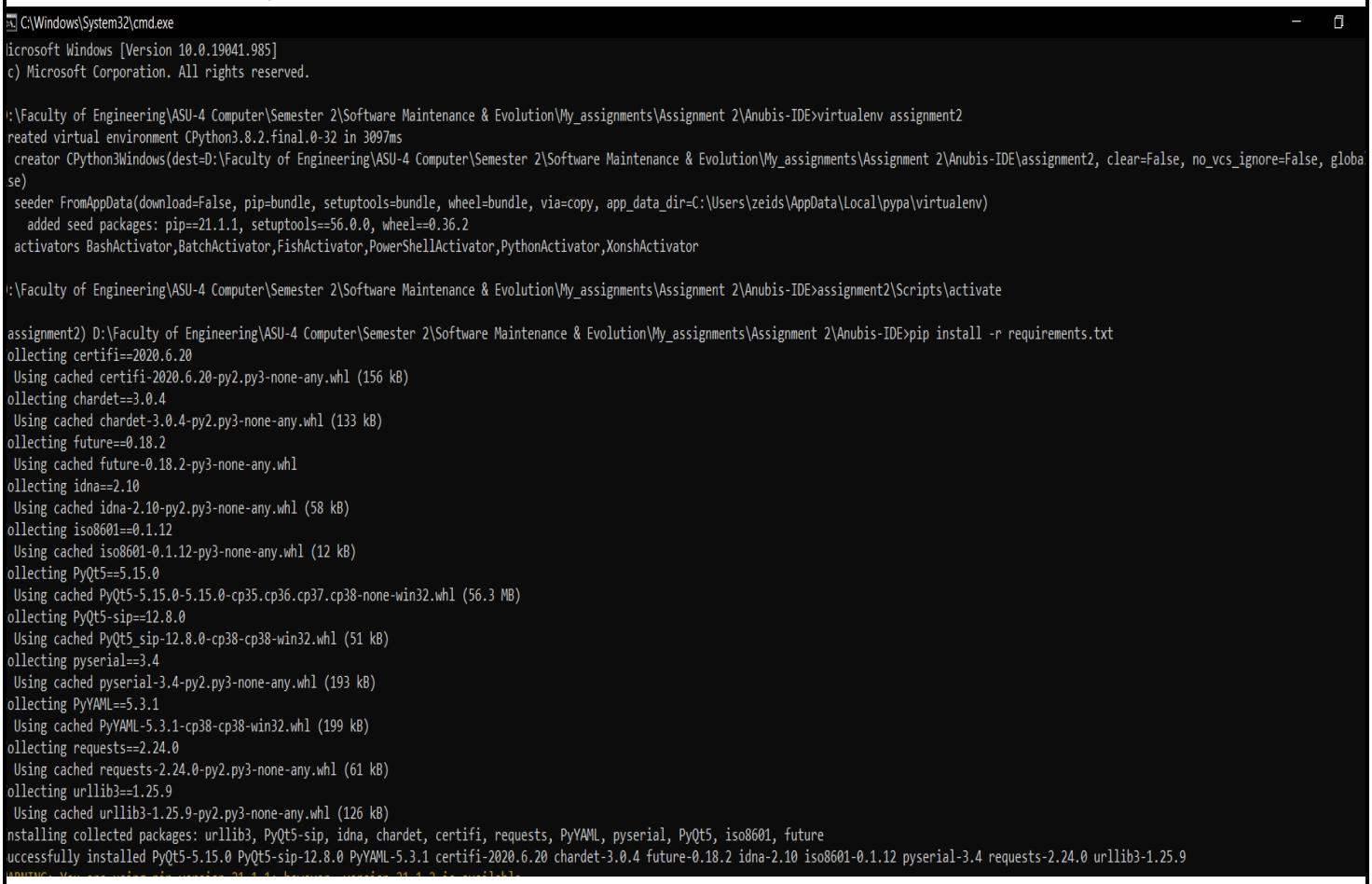
2. Added Functionality

- A fast executed for python code: in this feature the editor user will inter a code for a single function that would be automatically wrapped inside a program that has a main function that will call the function.
- The user would be asked to also to provide a list of parameters to be passed from the main to the called function.

3. Cloning the GitHub repo after forking the original repo



4. Setup the environment



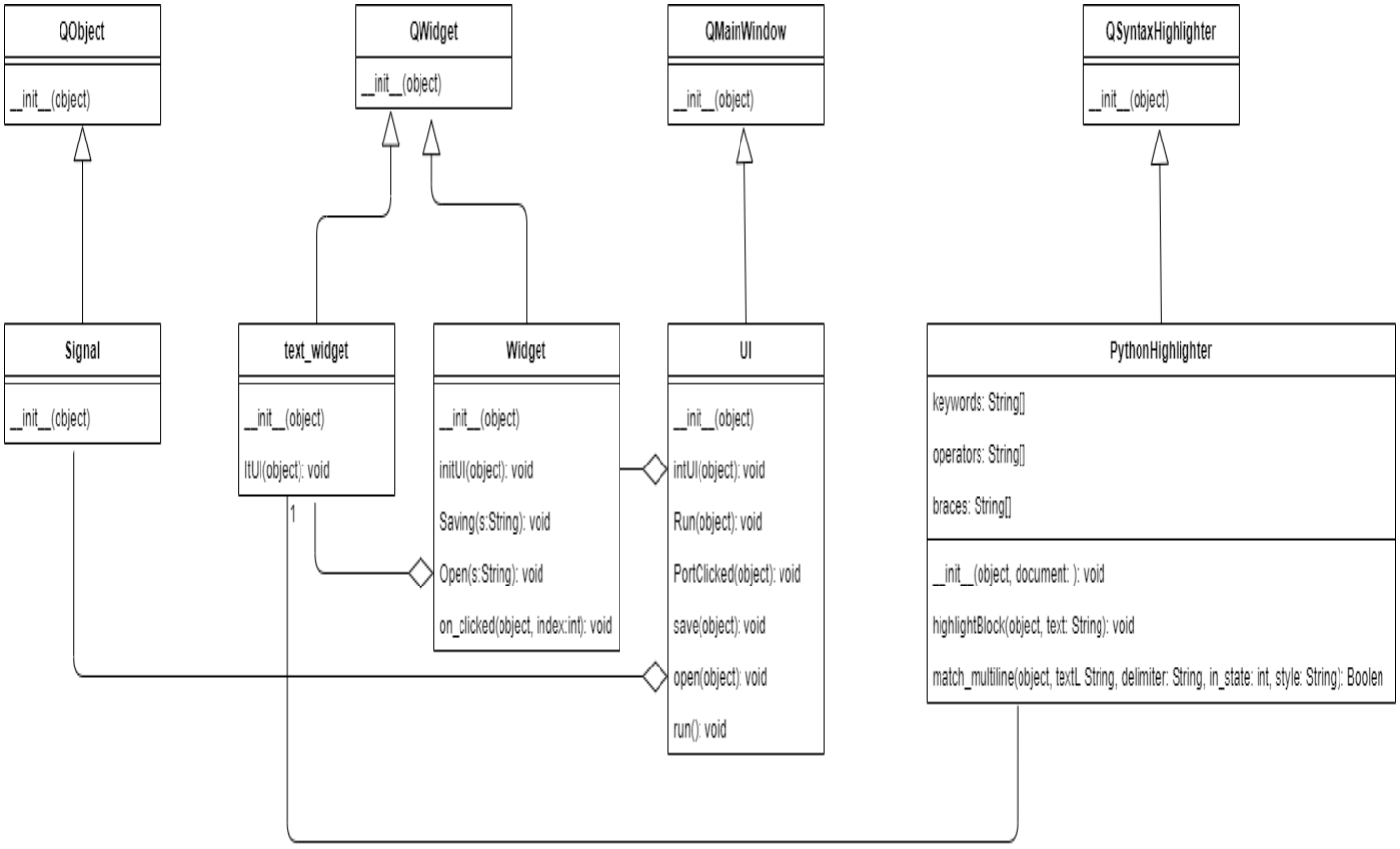
5. Functionality Description

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.

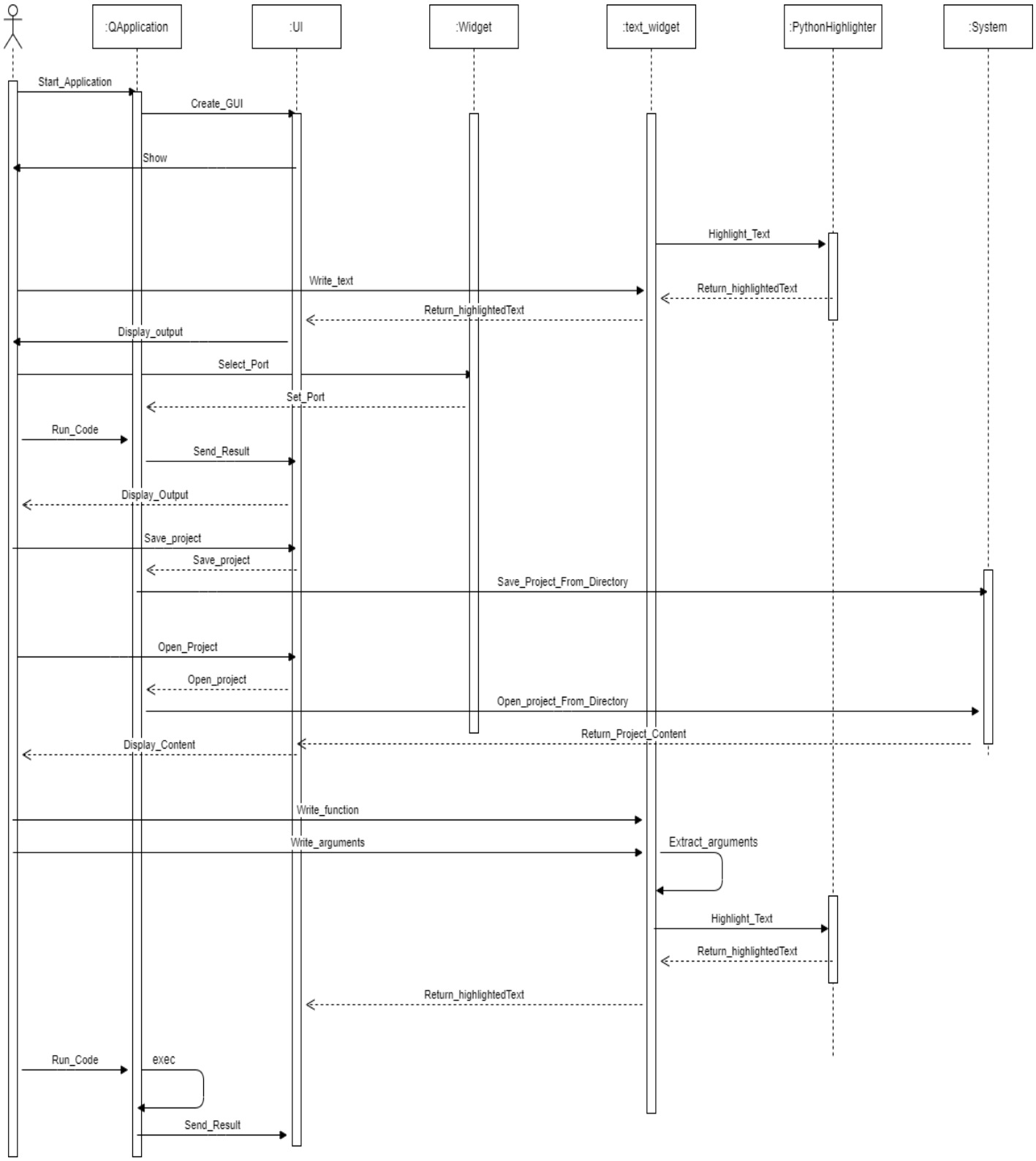
If the user defined a function which takes arguments to run, so he\she must provide these arguments in the text field specified for it.

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.

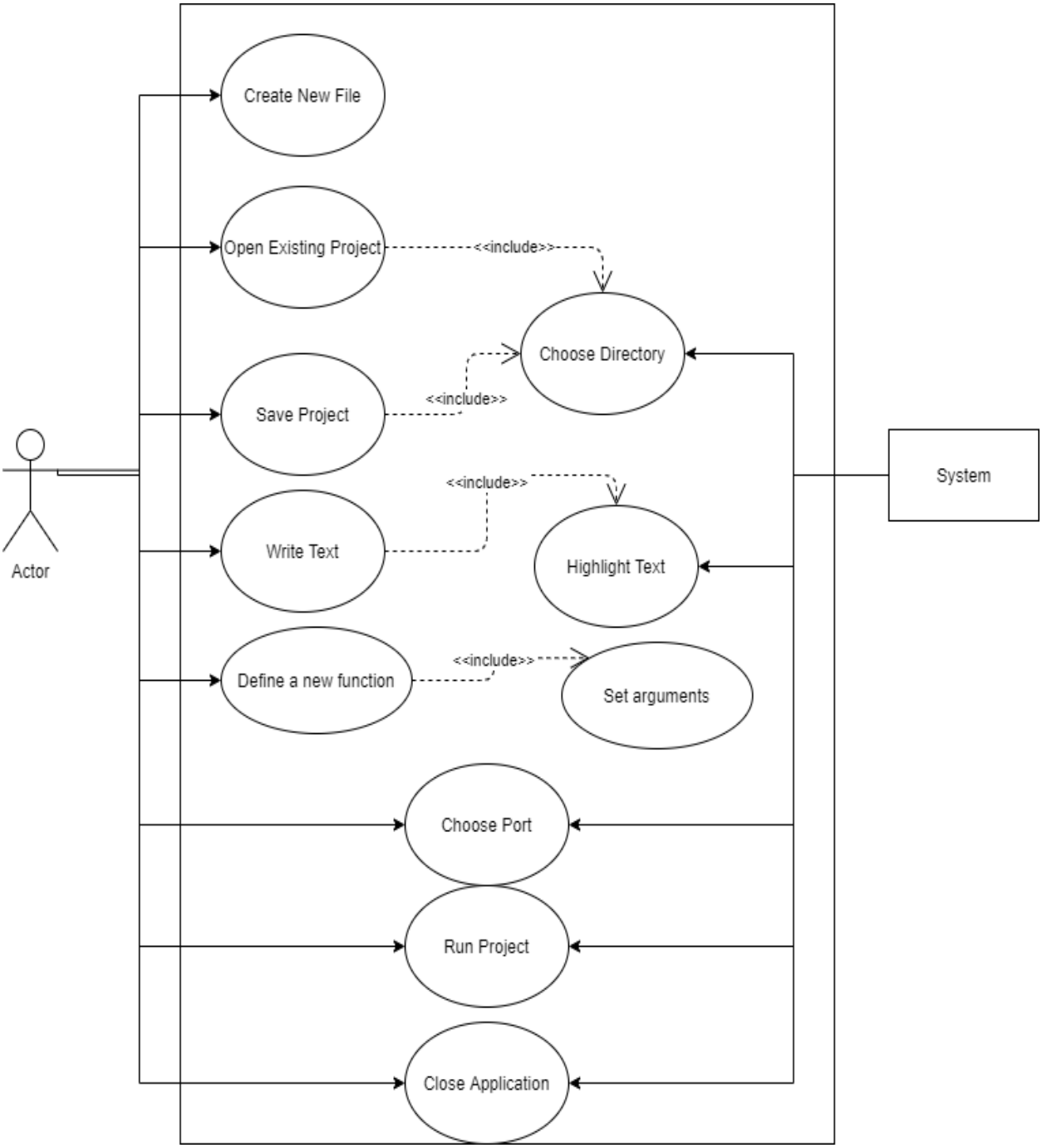
6. Class Diagram



7. Sequence Diagram

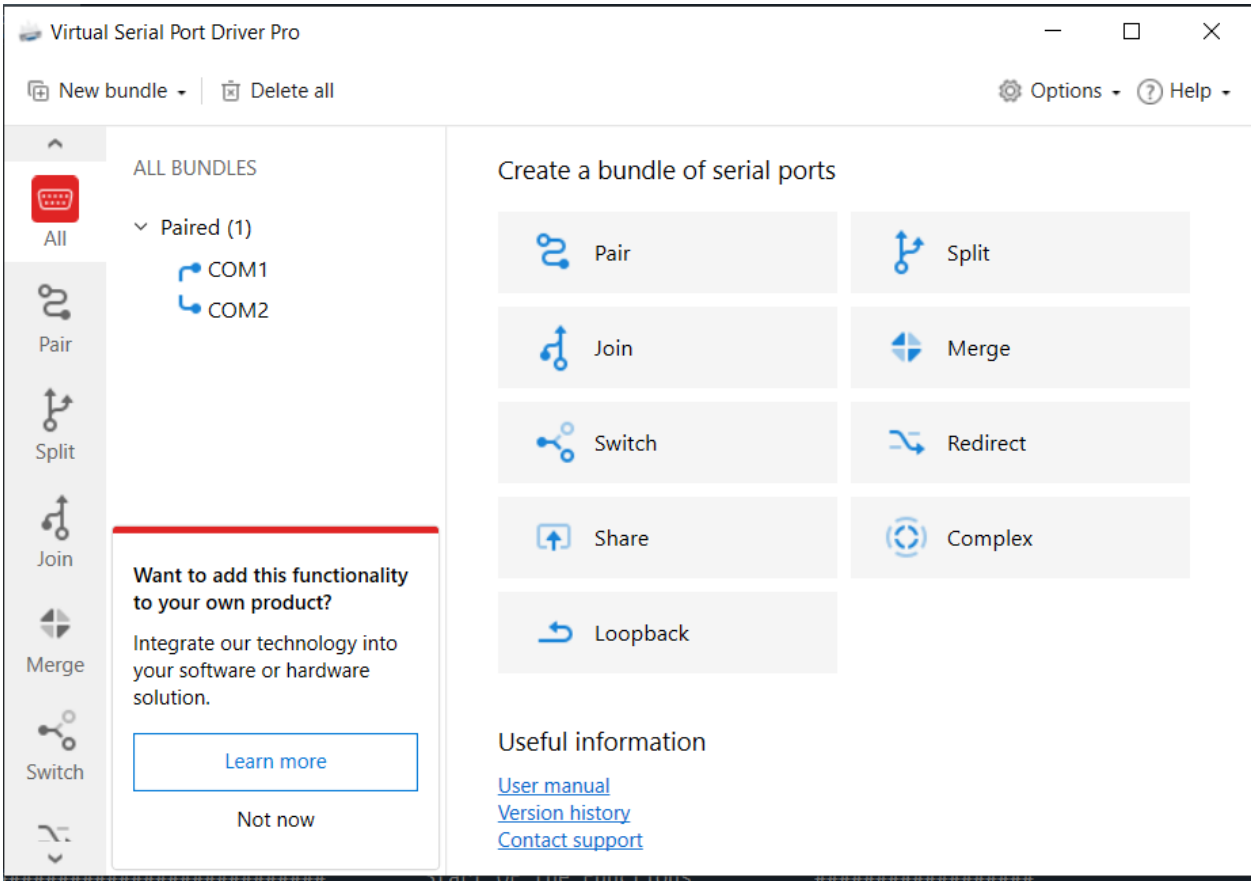


8. Use Case diagram

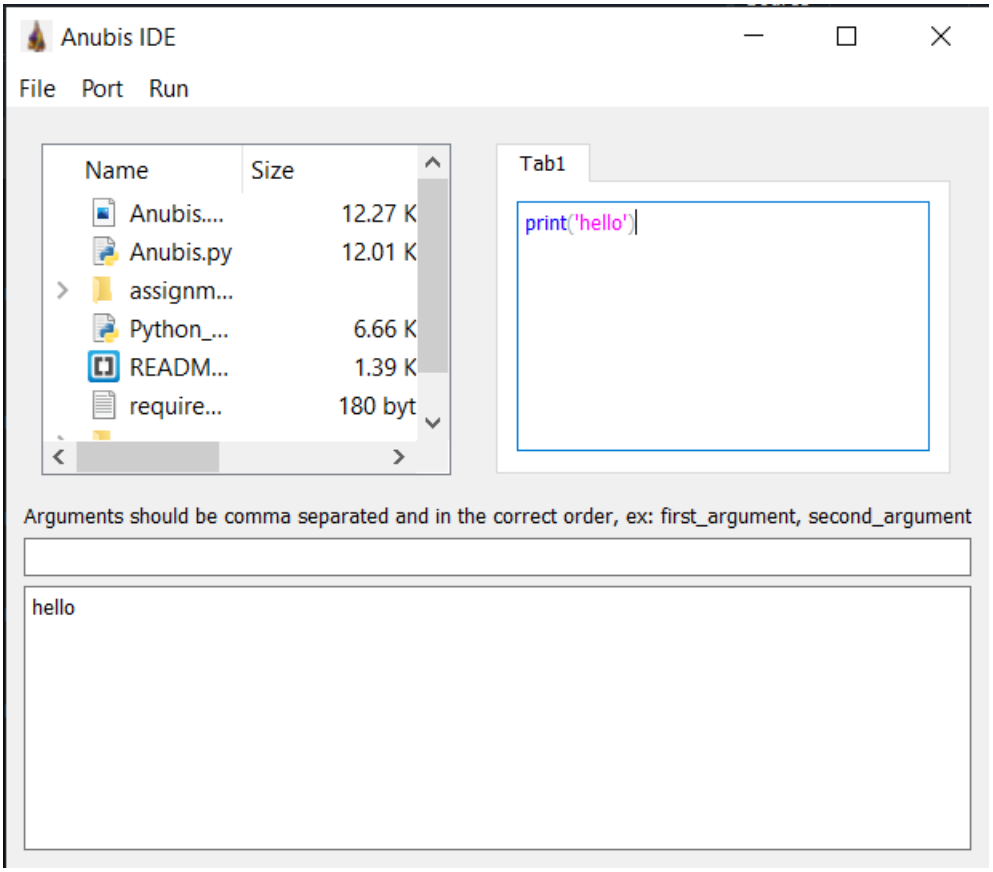


9. System Screenshots

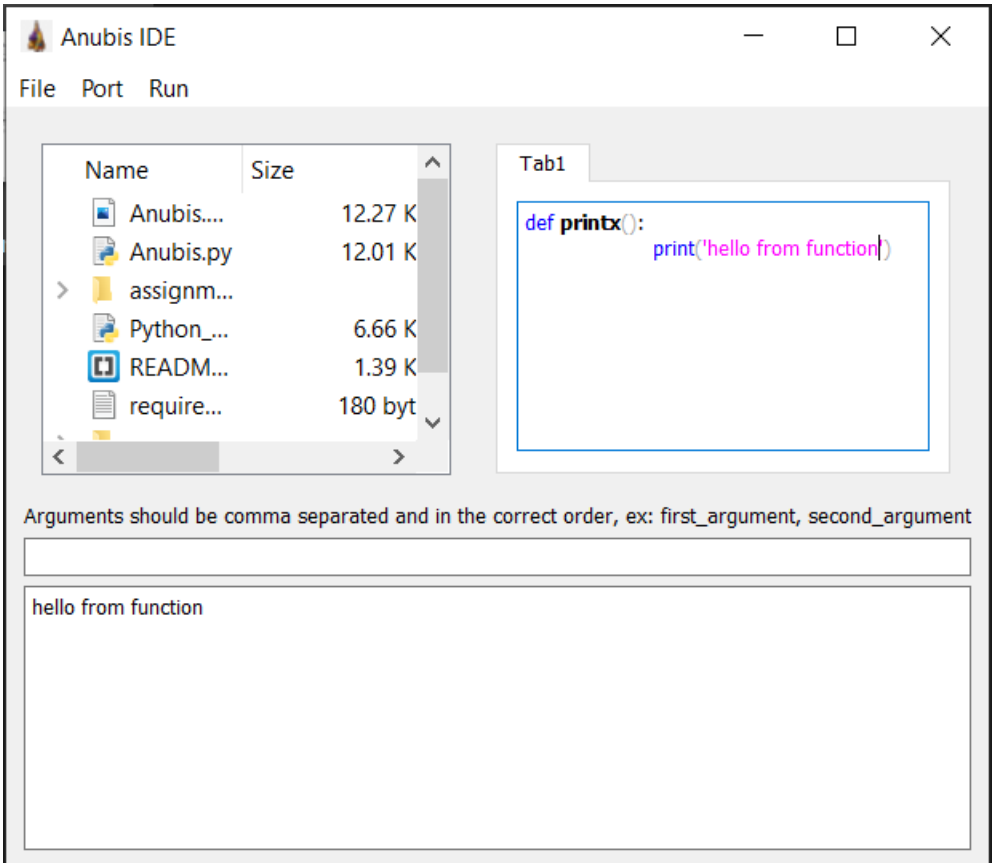
9.1. Virtual COM Ports



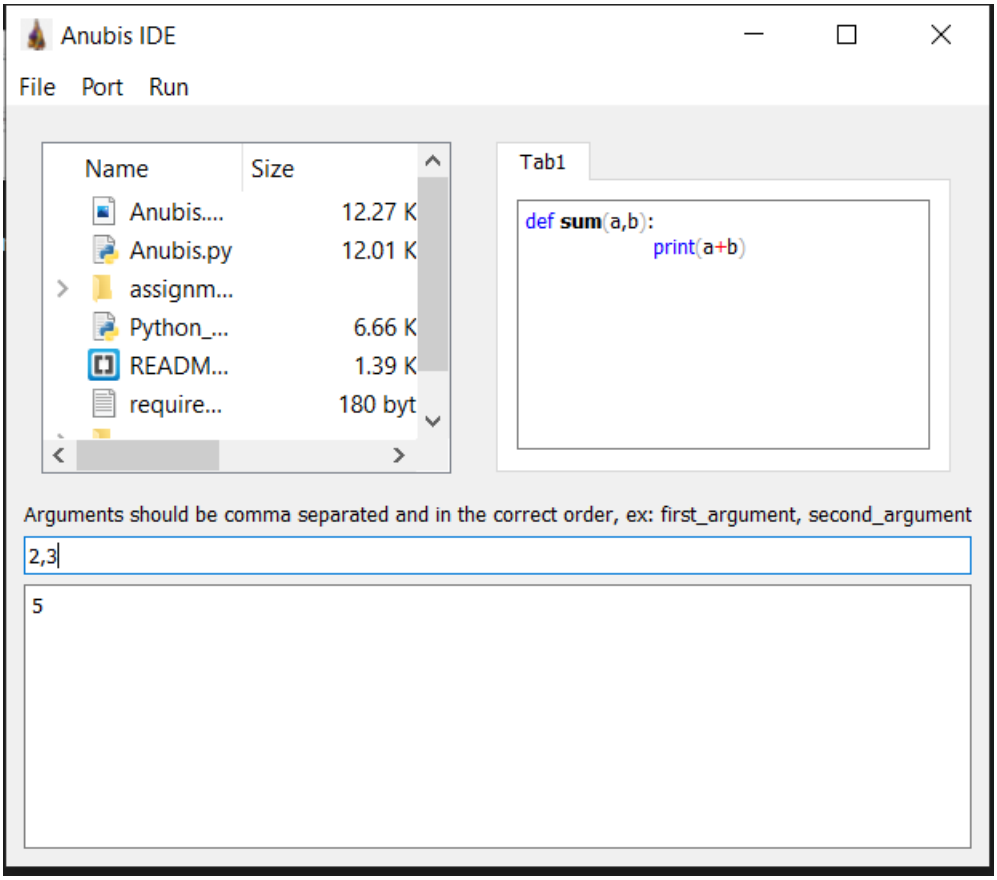
9.2. Execute Normal Code “print(‘hello’)”



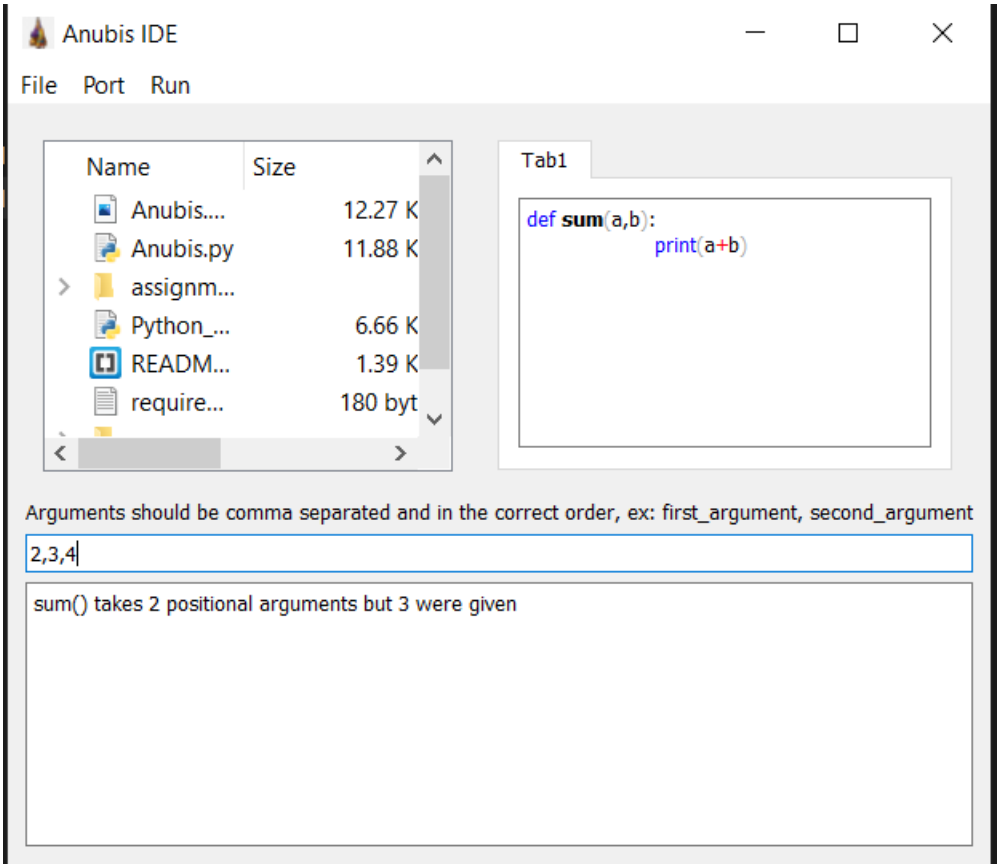
9.3. Execute a user's defined function.



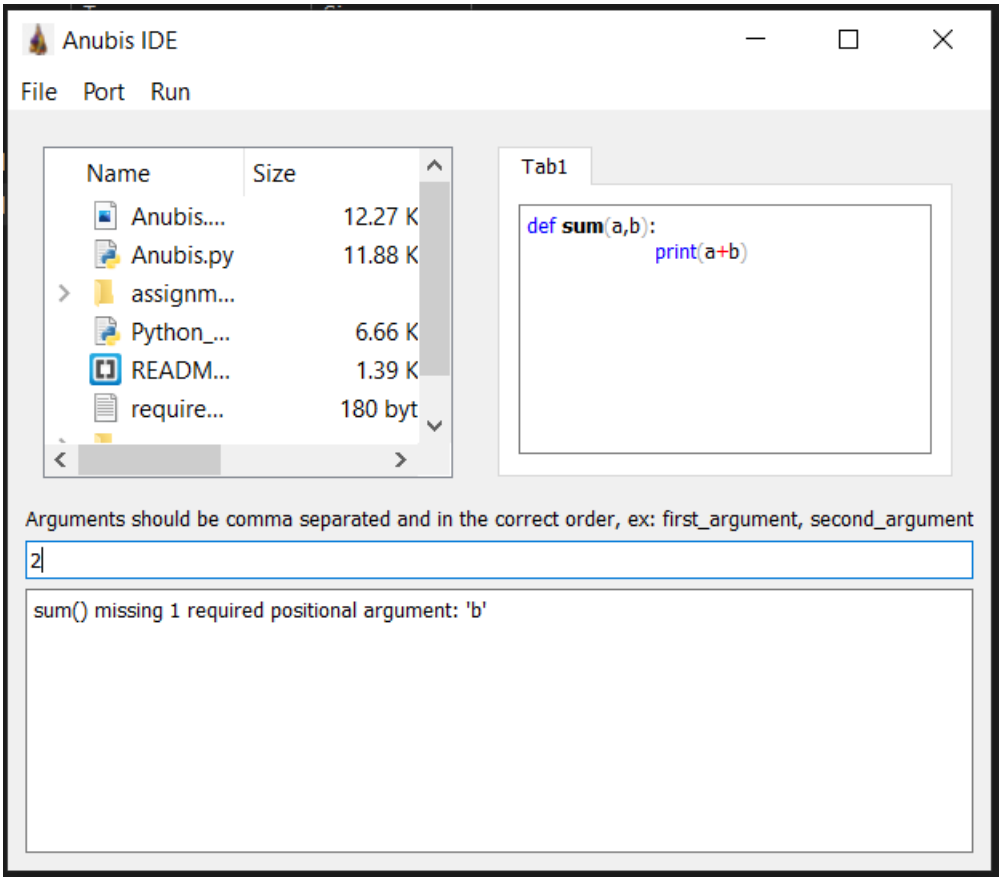
9.4. Execute a function which requires arguments



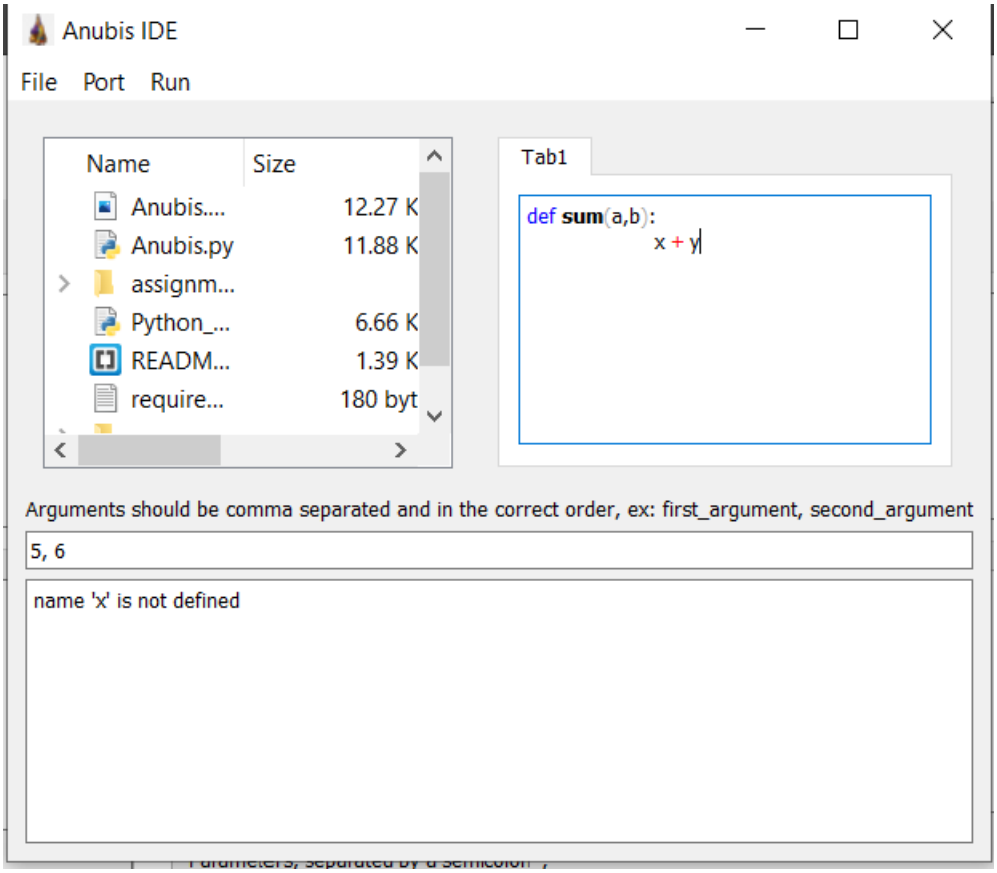
9.5. Handle if more arguments were passed



9.6. Handle if less arguments were passed



9.7. Handle syntax errors



10. Code

```
#############      author => Anubis Graduation Team        ############
#############      this project is part of my graduation project and it intends t
o 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 f
unction, 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 ed
itor looks nice
        tab = QTabWidget()
        tx = text_widget()
        tab.addTab(tx, "Tab"+"1")
```

```
        # second editor in which the error messeges, succeeded connections and ou
tput will be shown
        global text2
        text2 = QTextEdit()
        text2.setReadOnly(True)
        # defining a Treeview variable to use it in showing the directory include
d 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 somefi
lters (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 directo
ry 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 edit
or 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 QHBox
Layout
        # 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 ma
ke 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 cal
l, they should be seperated by a comma
        labelForArgs = QLabel(self)
        labelForArgs.setText(
            "Arguments should be comma separated and in the correct order, ex: fi
rst_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 e
ditor
    @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 th
e 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()) func
tion
        # copyrights of serial_ports() function goes back to a guy from stackover
flow(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 e
ditor
    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 t
o 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 f
unction, 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 ed
itor looks nice
        tab = QTabWidget()
        tx = text_widget()
        tab.addTab(tx, "Tab"+"1")

        # second editor in which the error messeges, succeeded connections and ou
tput will be shown
        global text2
        text2 = QTextEdit()
        text2.setReadOnly(True)
        # defining a Treeview variable to use it in showing the directory include
d 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 somefi
lters (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 directo
ry 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 edit
or 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 QHBox
Layout
        # 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 ma
ke 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 cal
l, they should be seperated by a comma
```

```
        labelForArgs = QLabel(self)
        labelForArgs.setText(
            "Arguments should be comma separated and in the correct order, ex: fi
rst_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 e
ditor
    @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 th
e 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()) func
tion
        # copyrights of serial_ports() function goes back to a guy from stackover
flow(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 e
ditor
    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_())
```

11. 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.