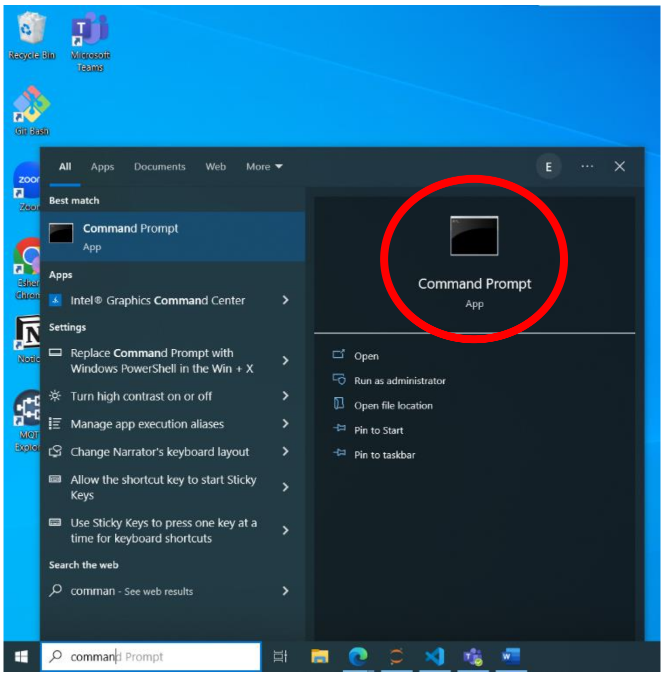
Installing PyQt and Qt Designer

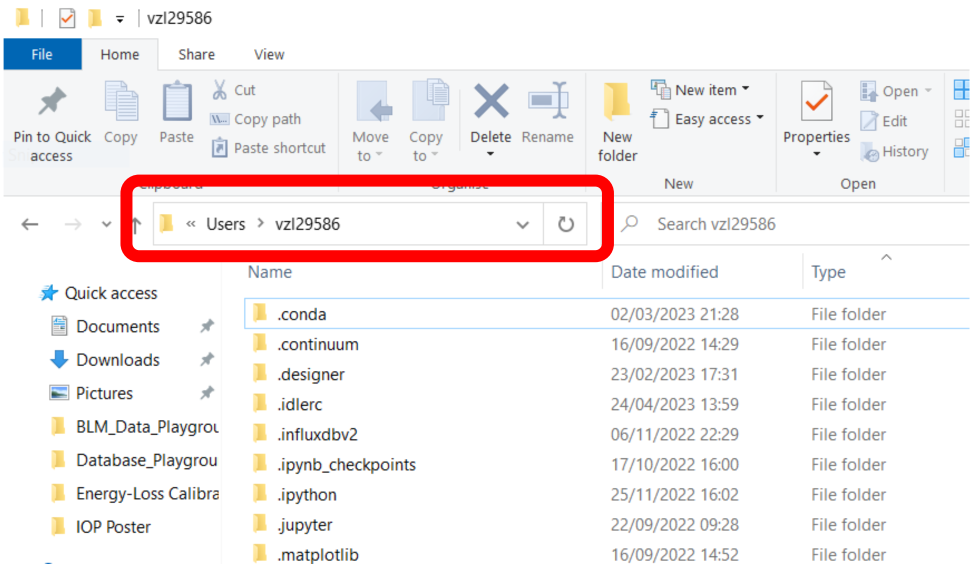
1. Open command prompt from the Windows start menu


All
Apps
Documents
Web
More
Best match
Command Prompt
App
Apps
Intel® Graphics Command Center
Settings
Replace Command Prompt with Windows PowerShell in the Win + X
Turn high contrast on or off
Manage app execution aliases
Change Narrator's keyboard layout
Allow the shortcut key to start Sticky Keys
Use Sticky Keys to press one key at a time for keyboard shortcuts
Search the web
comman - See web results
Command Prompt
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
command Prompt

2. Make sure you are in the directory C:\Users\[your username]

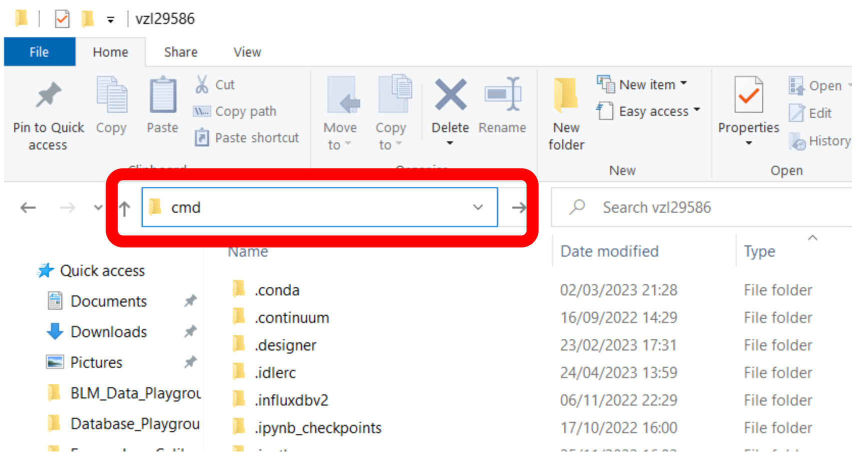
   If you know how to use bash commands, such as cd (change directory) to move around different folders, get yourself into the C:\Users\[Fed Id] folder.

   If you're not completely familiar yet, don't worry! There's an easier way to do this – open your File Explorer and find the folder: C:\Users\[Fed Id]



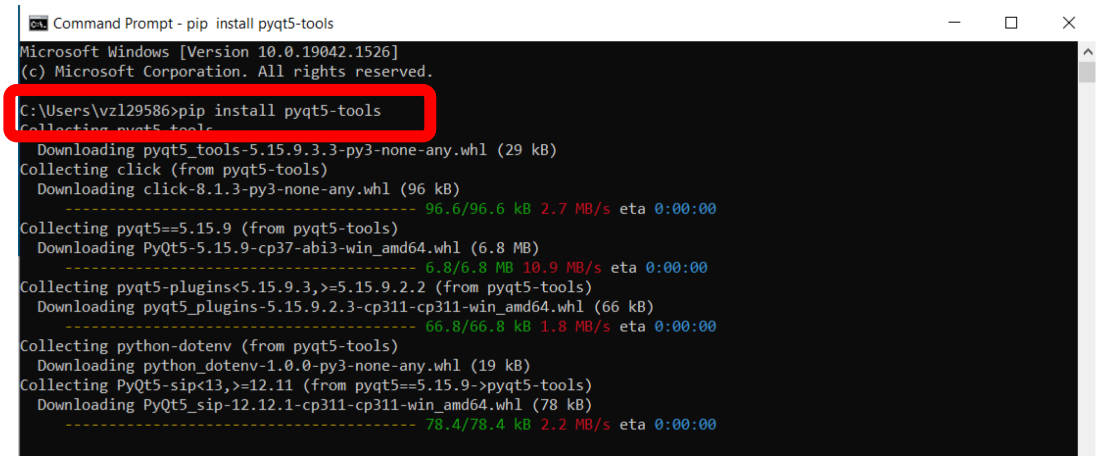
Name	Date modified	Type
.conda	02/03/2023 21:28	File folder
.continuum	16/09/2022 14:29	File folder
.designer	23/02/2023 17:31	File folder
.idlerc	24/04/2023 13:59	File folder
.influxdbv2	06/11/2022 22:29	File folder
.ipynb_checkpoints	17/10/2022 16:00	File folder
.ipython	25/11/2022 16:02	File folder
.jupyter	22/09/2022 09:28	File folder
.matplotlib	16/09/2022 14:52	File folder

In the folders selector, type out “cmd” and press enter – a command prompt should open in the correct folder.



Name	Date modified	Type
.conda	02/03/2023 21:28	File folder
.continuum	16/09/2022 14:29	File folder
.designer	23/02/2023 17:31	File folder
.idlerc	24/04/2023 13:59	File folder
.influxdbv2	06/11/2022 22:29	File folder
.ipynb_checkpoints	17/10/2022 16:00	File folder

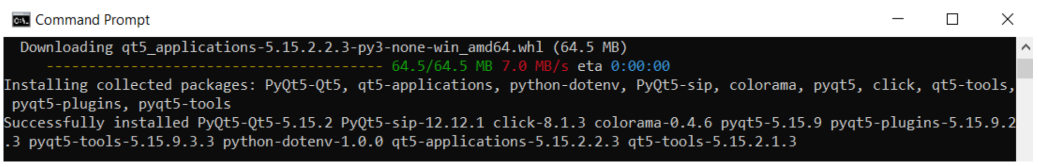
3. Type the command *pip install pyqt5-tools* exactly as is written



```
Command Prompt - pip  install pyqt5-tools
Microsoft Windows [Version 10.0.19042.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vzl29586>pip install pyqt5-tools
Collecting pyqt5-tools
  Downloading pyqt5_tools-5.15.9.3.3-py3-none-any.whl (29 kB)
Collecting click (from pyqt5-tools)
  Downloading click-8.1.3-py3-none-any.whl (96 kB)
     ---------------------------------------- 96.6/96.6 kB 2.7 MB/s eta 0:00:00
Collecting pyqt5==5.15.9 (from pyqt5-tools)
  Downloading PyQt5-5.15.9-cp37-abi3-win_amd64.whl (6.8 MB)
     ---------------------------------------- 6.8/6.8 MB 10.9 MB/s eta 0:00:00
Collecting pyqt5-plugins<5.15.9.3,>=5.15.9.2.2 (from pyqt5-tools)
  Downloading pyqt5_plugins-5.15.9.2.3-cp311-cp311-win_amd64.whl (66 kB)
     ---------------------------------------- 66.8/66.8 kB 1.8 MB/s eta 0:00:00
Collecting python-dotenv (from pyqt5-tools)
  Downloading python_dotenv-1.0.0-py3-none-any.whl (19 kB)
Collecting PyQt5-sip<13,>=12.11 (from pyqt5==5.15.9->pyqt5-tools)
  Downloading PyQt5_sip-12.12.1-cp311-cp311-win_amd64.whl (78 kB)
     ---------------------------------------- 78.4/78.4 kB 2.2 MB/s eta 0:00:00
```

If you’ve done this correctly, it should start installing pyqt5-tools – it’ll look something like this.

It’ll finish below saying: “Successfully installed PyQt5”, with a bunch of different version numbers.

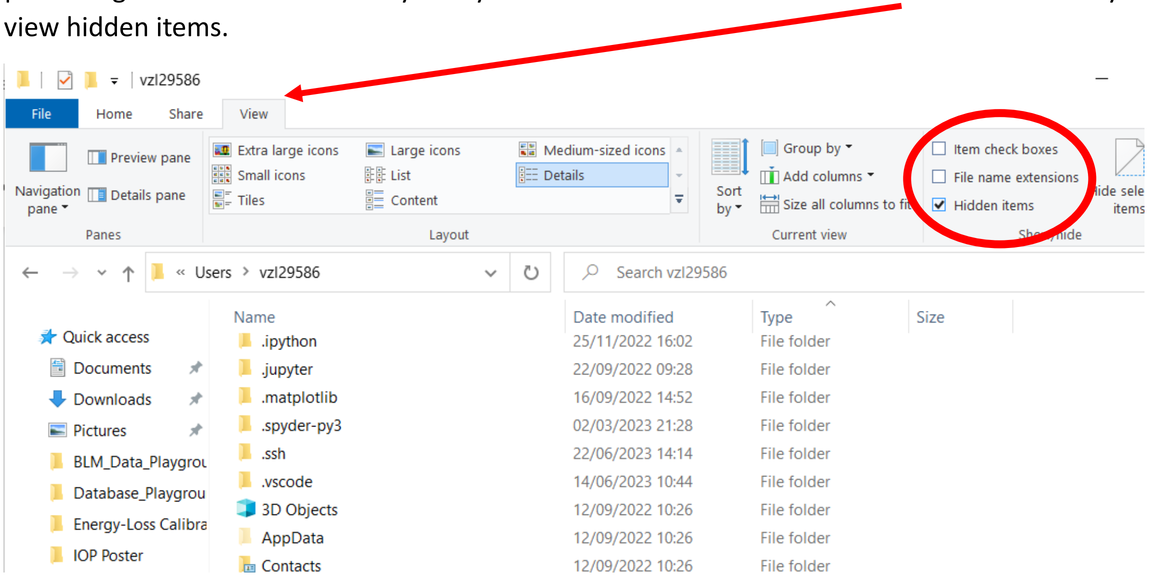


```
Command Prompt
  Downloading qt5_applications-5.15.2.2.3-py3-none-win_amd64.whl (64.5 MB)
     ---------------------------------------- 64.5/64.5 MB 7.0 MB/s eta 0:00:00
Installing collected packages: PyQt5-Qt5, qt5-applications, python-dotenv, PyQt5-sip, colorama, pyqt5, click, qt5-tools,
 pyqt5-plugins, pyqt5-tools
Successfully installed PyQt5-Qt5-5.15.2 PyQt5-sip-12.12.1 click-8.1.3 colorama-0.4.6 pyqt5-5.15.9 pyqt5-plugins-5.15.9.2
.3 pyqt5-tools-5.15.9.3.3 python-dotenv-1.0.0 qt5-applications-5.15.2.2.3 qt5-tools-5.15.2.1.3
```

4. If all went well, there should be a *designer.exe* file in the directory: *C:* → *Users* → *[username]* → *AppData* → *Local* → *Programs* → *Python* → *Python311* → *Lib* → *site-packages* → *qt5_applications* → *Qt* → *bin* → *designer.exe*

Sometimes it might a little lost and appear in *C:* → *Users* → *[username]* → *AppData* → *Local* → *Programs* → *Python* → *Python311* → *Lib* → *site-packages* → *pyqt5_tools* → *designer.exe*

AppData might be a hidden directory – so you'll have to tick the checkbox in View which allows you to view hidden items.



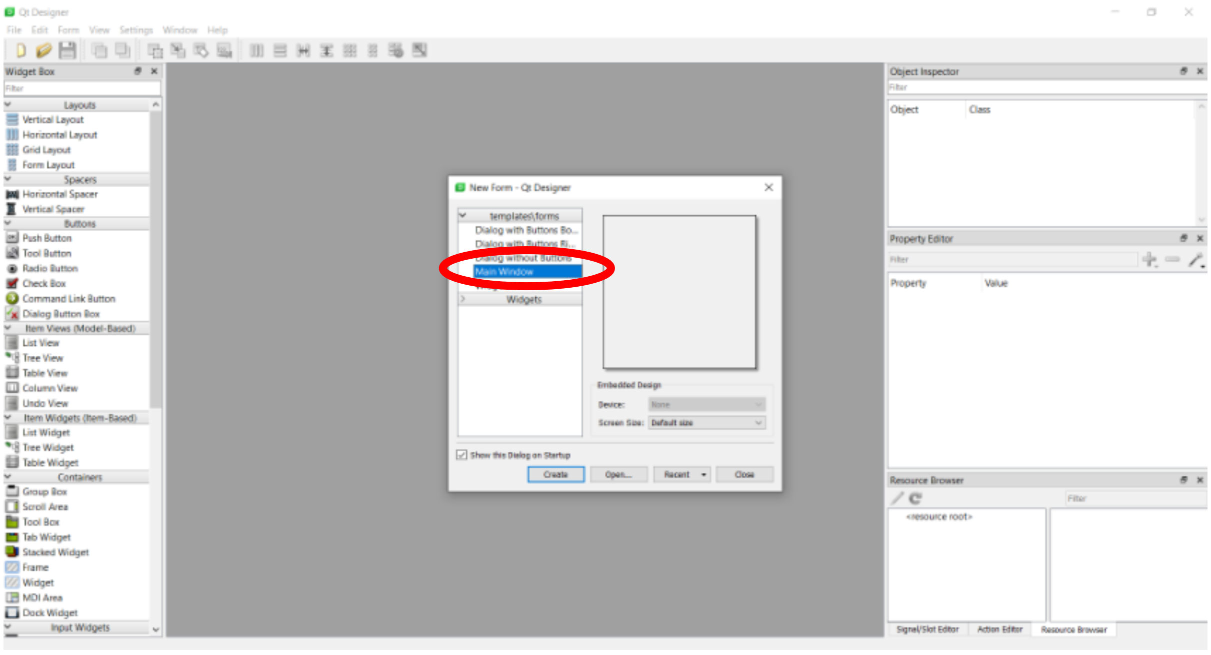
- [ ] Item check boxes
- [ ] File name extensions
- [✓] Hidden items

Name	Date modified	Type	Size
.ipython	25/11/2022 16:02	File folder	
.jupyter	22/09/2022 09:28	File folder	
.matplotlib	16/09/2022 14:52	File folder	
.spyder-py3	02/03/2023 21:28	File folder	
.ssh	22/06/2023 14:14	File folder	
.vscode	14/06/2023 10:44	File folder	
3D Objects	12/09/2022 10:26	File folder	
AppData	12/09/2022 10:26	File folder	
Contacts	12/09/2022 10:26	File folder	

Once you can see the designer.exe file, double click to open it.

5. Qt Designer

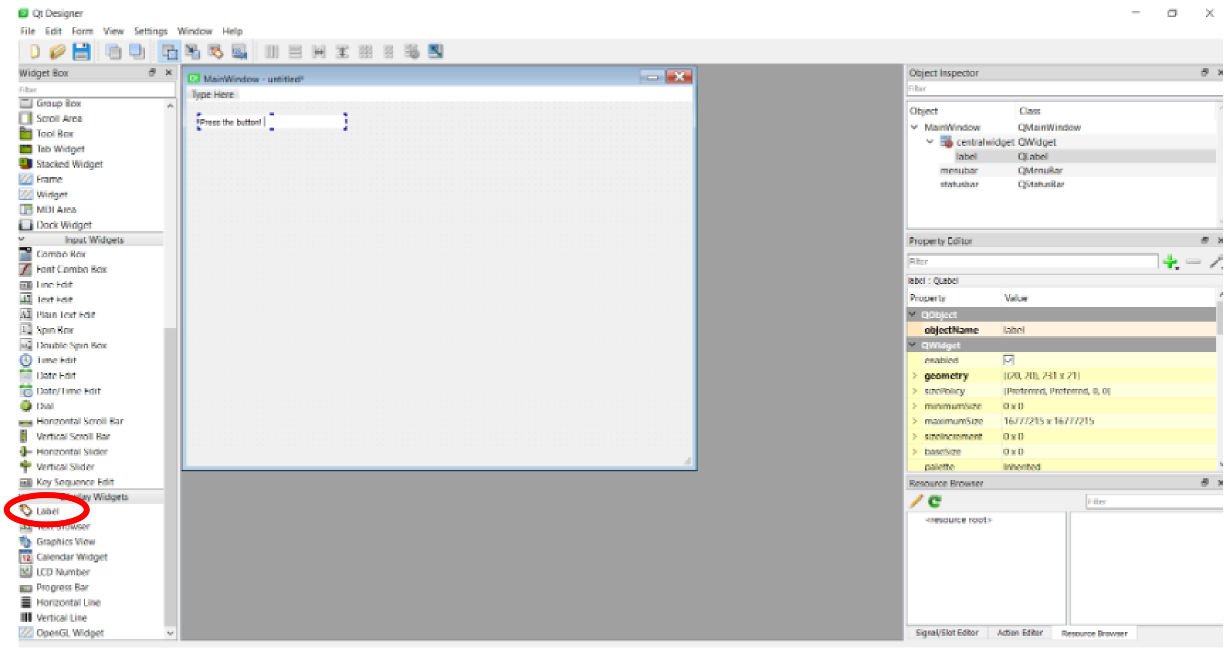
Qt Designer should look something like this. For our first exercise, we'll need to create a Main Window.



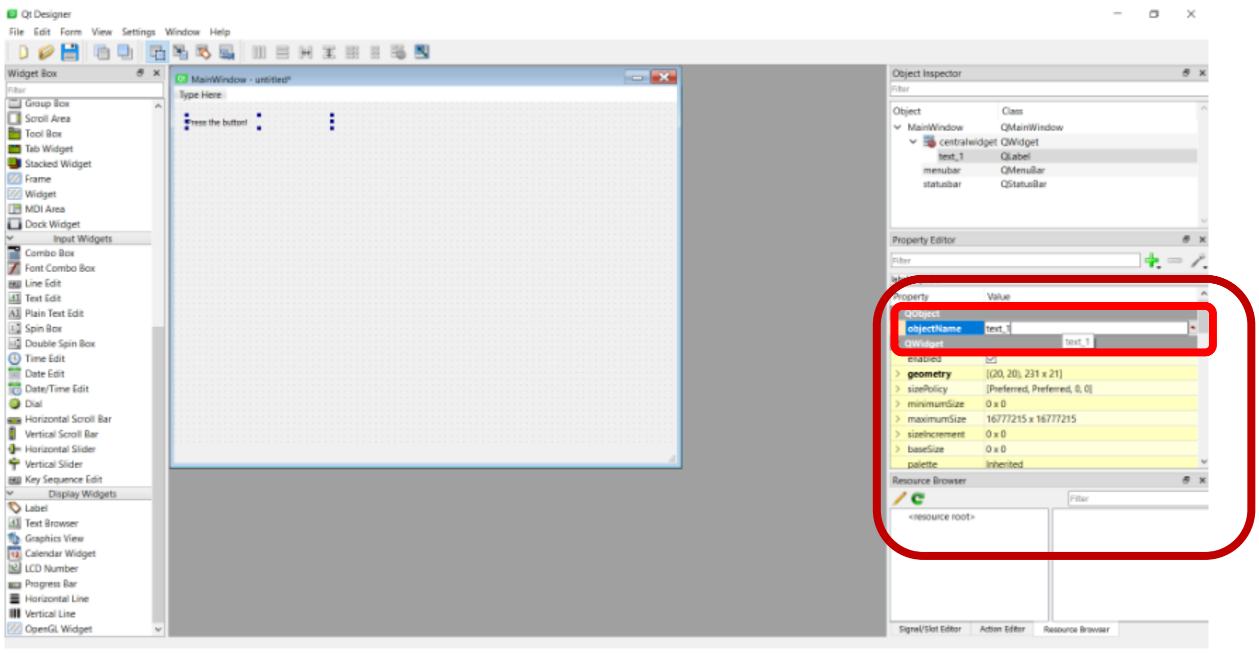
6. Making things: Simple counter

Scroll down the left hand side bar and have a look at all of the widgets available.

Let's make a simple application together. Click and drag the Label onto the Main Window, play around with the size and shape of the object. Double click on the Label to edit the text.



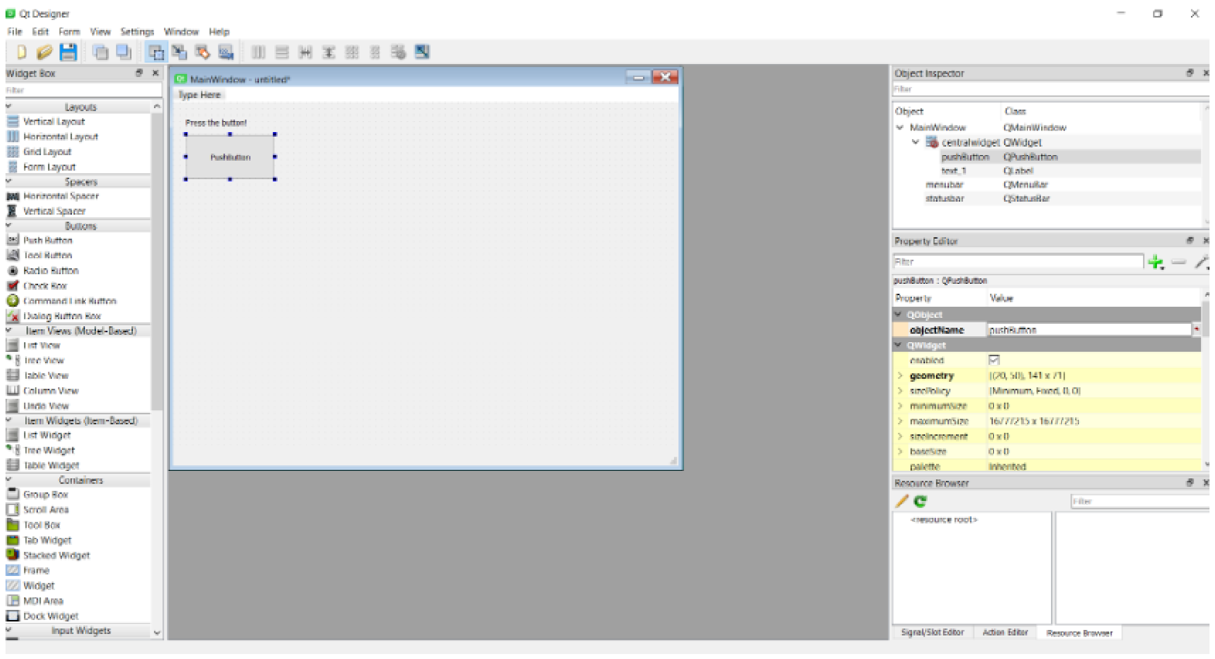
Whenever we place any widget on the Main Window, look at the right hand side bar and find the objectName field. This is how this object will be referred to in our Python code, so it'll be important to change this to a more useful name.



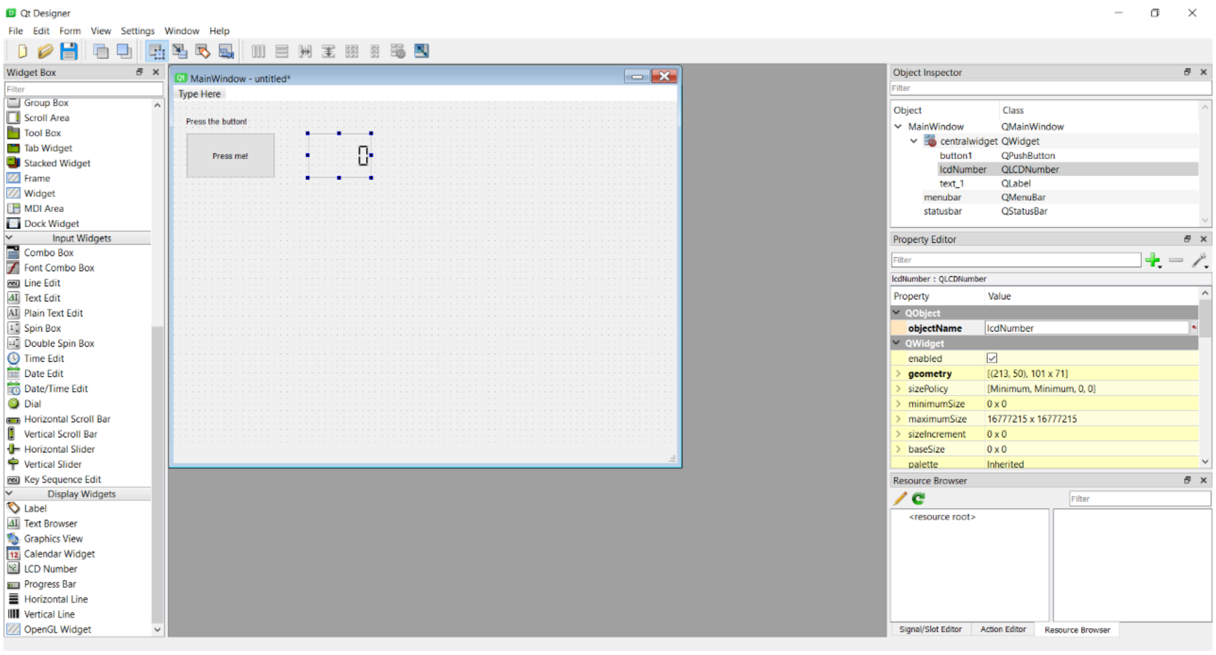
Though it may seem tempting to play around with all of the fonts, font sizes, colours and other customisable options in this bar, let's not play around too much right now – it'll make our Python code more difficult to read.

The purpose of this exercise is to create a simple interactive Python application – having simple code right now will help us understand exactly what is going on. Further along in your project, you are welcome to customise the appearance of your GUIs as you see fit.

We'll now add an interactive button so that our user can give us some input. We can choose a simple Push Button. Again we might find it useful to change the object name as above.

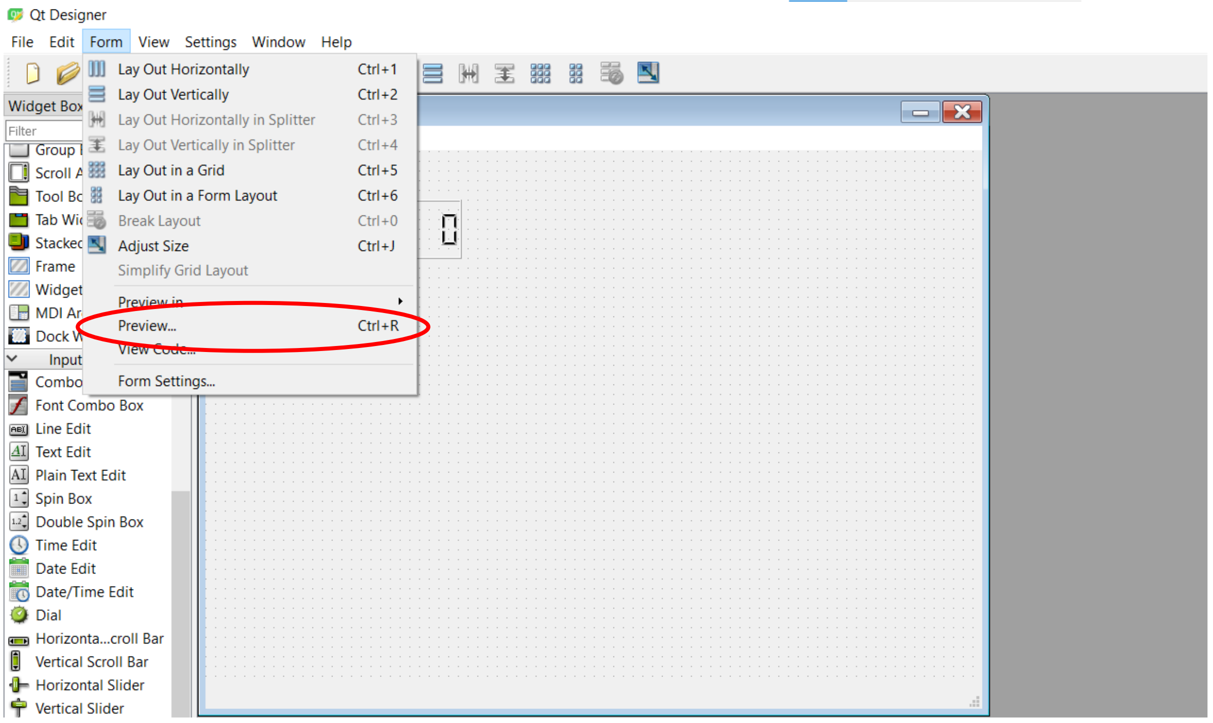


Let’s add a simple LCD Number. This will display a number to the user.



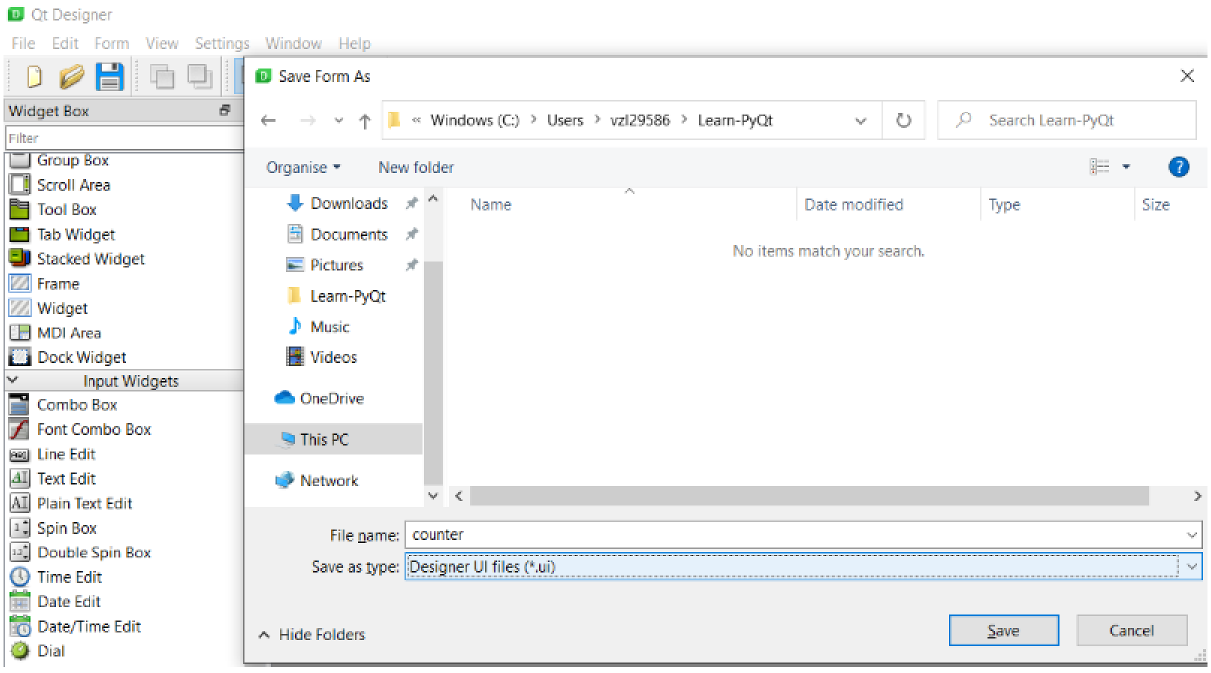
We can preview our GUI – have a look at what it will look like – by clicking Preview in the Form Menu, or “Ctrl” + R.

It is not recommended to view code or preview in Python in this menu, as the Python code uses deprecated packages (it’s better for us to convert to Python another way – detailed below!)


Qt Designer
File Edit Form View Settings Window Help
Lay Out Horizontally Ctrl+1
Lay Out Vertically Ctrl+2
Lay Out Horizontally in Splitter Ctrl+3
Lay Out Vertically in Splitter Ctrl+4
Lay Out in a Grid Ctrl+5
Lay Out in a Form Layout Ctrl+6
Break Layout Ctrl+0
Adjust Size Ctrl+J
Simplify Grid Layout
Preview in
Preview... Ctrl+R
View Code...
Form Settings...

7. Converting our .ui file into Python code.

Let’s save our GUI as a .ui file in a good file location.

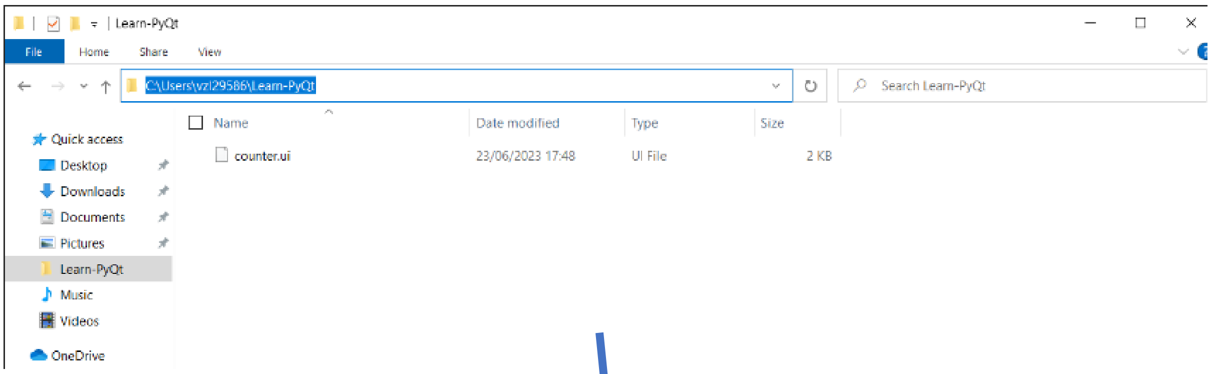


No items match your search.

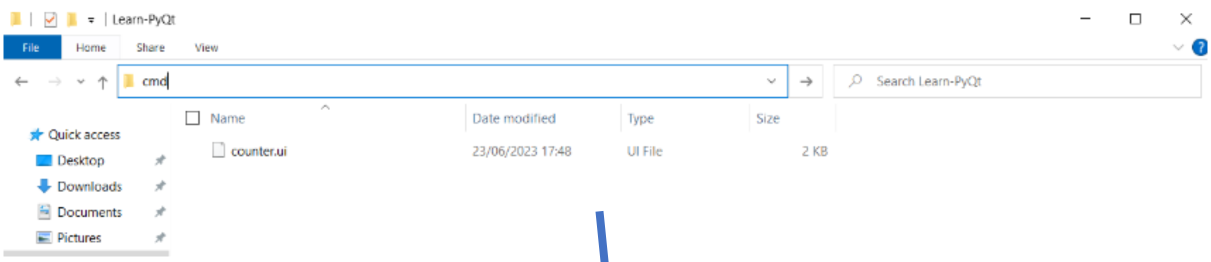
File name: counter

Save as type: Designer UI files (*.ui)

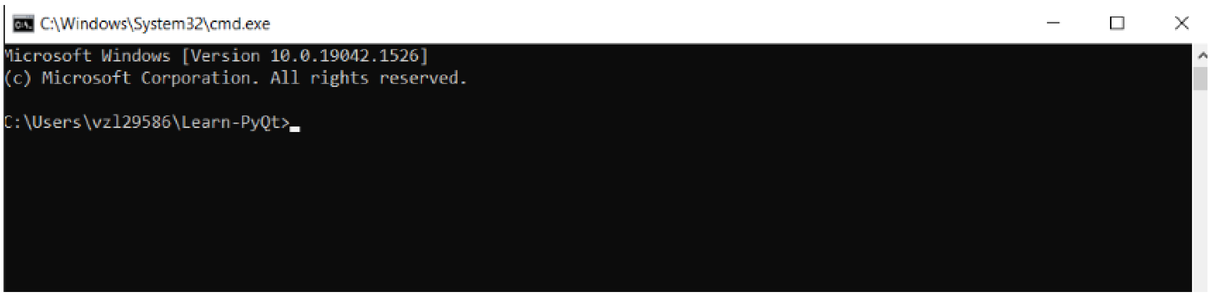
Let’s open this file location using our file explorer again and type “cmd” in the file field to open a command prompt in this file location.



Name	Date modified	Type	Size
counter.ui	23/06/2023 17:48	UI File	2 KB



Name	Date modified	Type	Size
counter.ui	23/06/2023 17:48	UI File	2 KB



```
Microsoft Windows [Version 10.0.19042.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vzl29586\Learn-PyQt>_
```

In the command prompt, we want to type the following carefully:

pyuic5 -x <filename>.ui -o <filename>.py

In my case, I have saved my .ui file as counter.ui, so I'll need to type the following:

pyuic5 -x counter.ui -o counter.py

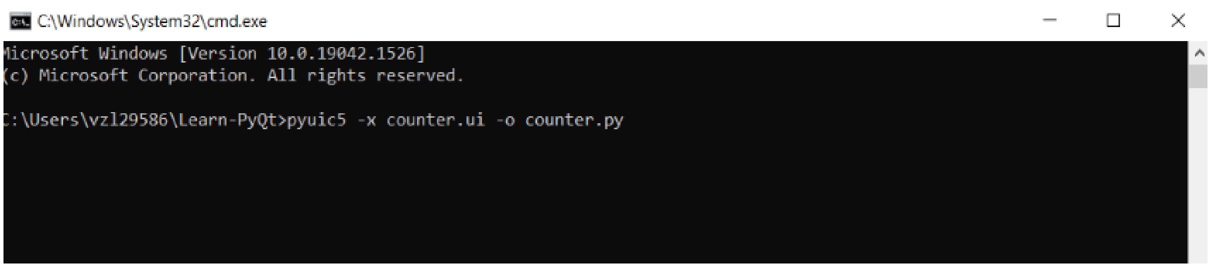
We can actually name the python file anything we like – it doesn't have to be the same filename, so if I wanted to, I could write:

pyuic5 -x counter.ui -o test.py

and the python file would be saved as *test.py* instead.

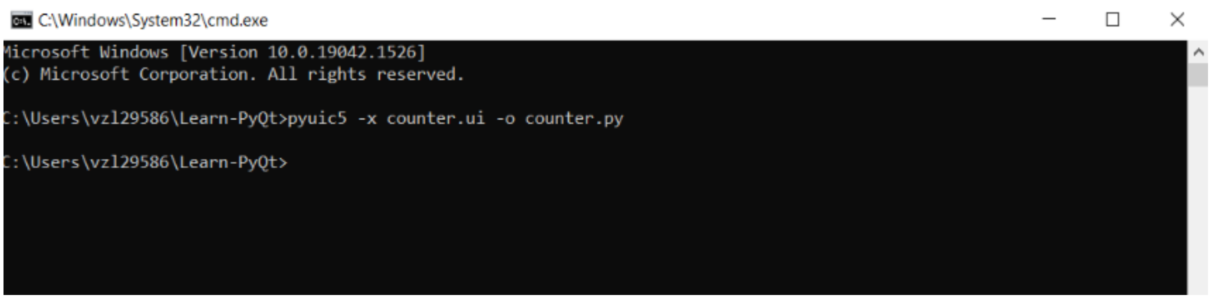
The *-x* before the .ui file tells the computer to *execute* this file.

The *-o* before the python file stands for *output file*.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vzl29586\Learn-PyQt>pyuic5 -x counter.ui -o counter.py
```

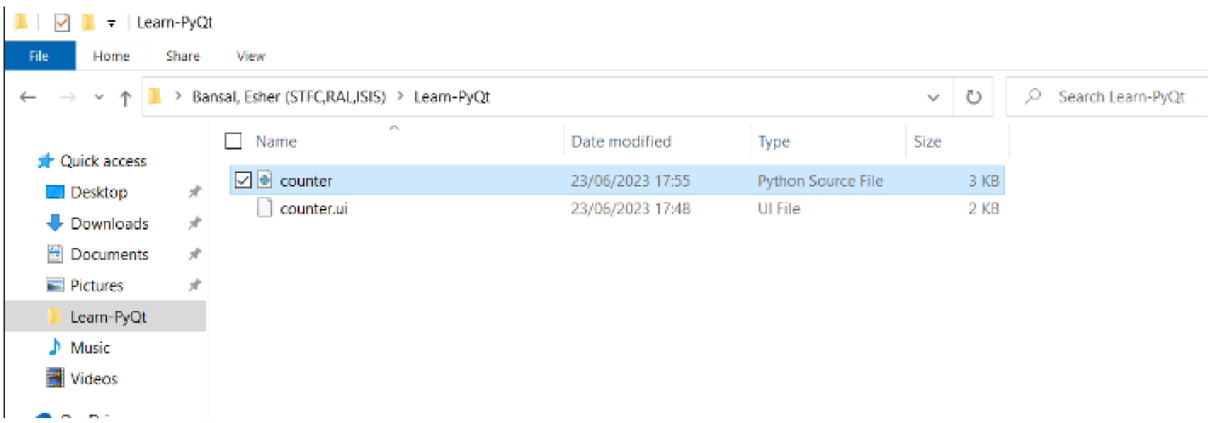
Once you press enter after typing this command, you should get the screen below (with no errors). This means that this has been run successfully.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vzl29586\Learn-PyQt>pyuic5 -x counter.ui -o counter.py

C:\Users\vzl29586\Learn-PyQt>
```

If we open up our file explorer again, we will find a new python file has been created!

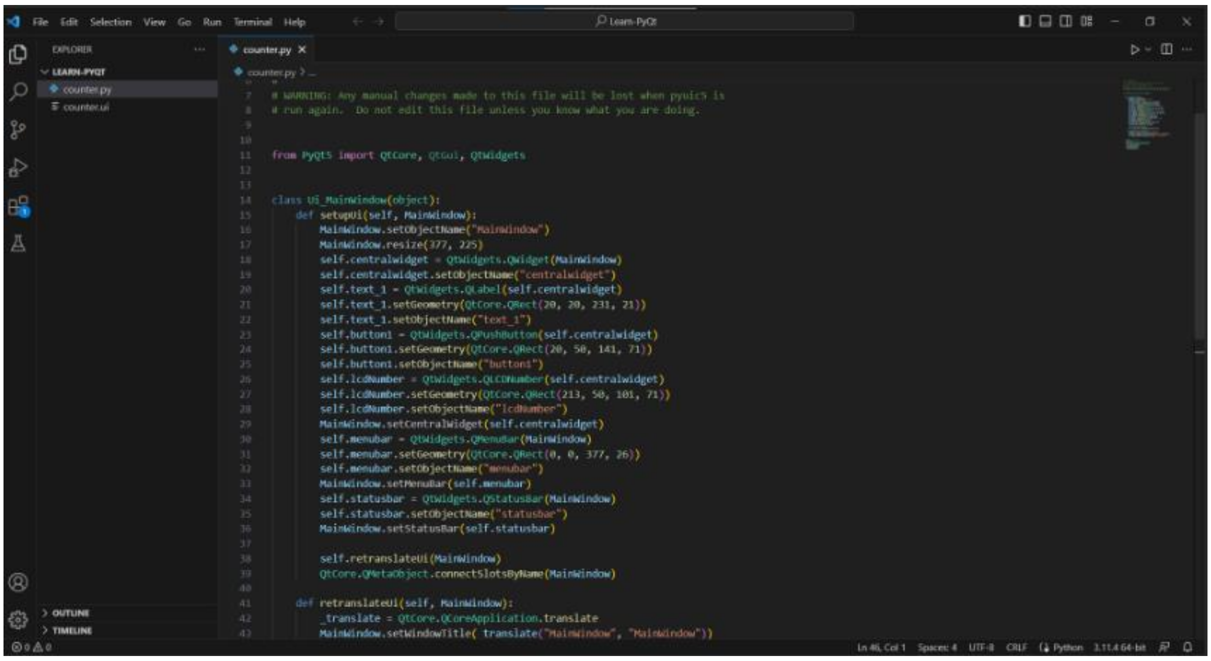
Let's open it up and see what we've got.



Name	Date modified	Type	Size
counter	23/06/2023 17:55	Python Source File	3 KB
counter.ui	23/06/2023 17:48	UI File	2 KB

8. Just like Magic

It’s written all of our python code for us! Let’s see what happens when we run this file.

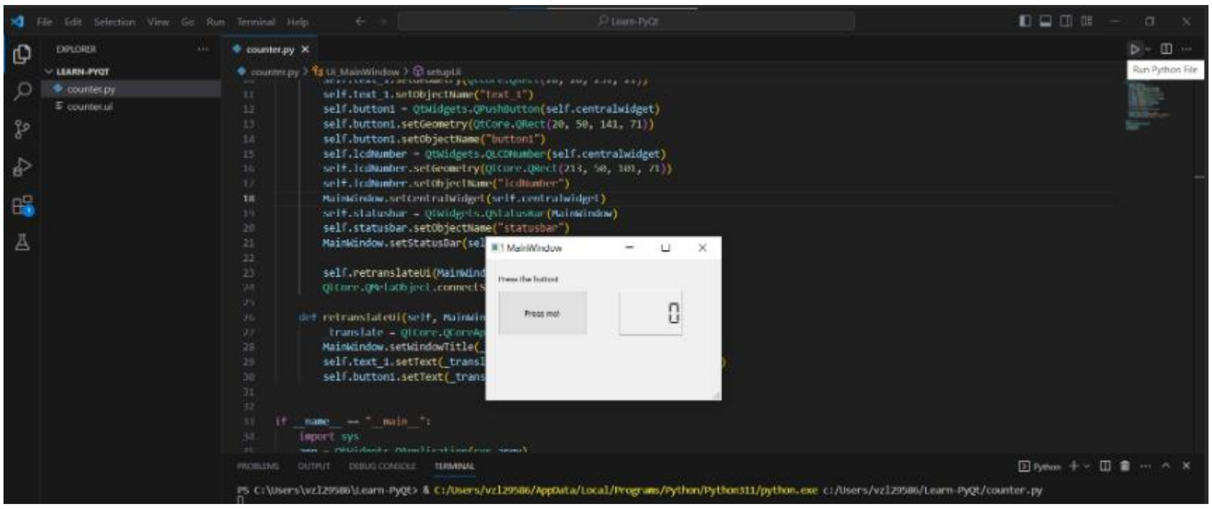


```python
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(377, 225)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.text_1 = QtWidgets.QLabel(self.centralwidget)
        self.text_1.setGeometry(QtCore.QRect(20, 20, 231, 21))
        self.text_1.setObjectName("text_1")
        self.button1 = QtWidgets.QPushButton(self.centralwidget)
        self.button1.setGeometry(QtCore.QRect(20, 50, 141, 71))
        self.button1.setObjectName("button1")
        self.lcdNumber = QtWidgets.QLCDNumber(self.centralwidget)
        self.lcdNumber.setGeometry(QtCore.QRect(213, 50, 101, 71))
        self.lcdNumber.setObjectName("lcdNumber")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 377, 26))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
```

Our application pops up and runs!

Right now it isn’t doing anything with the user input, but it looks exactly the way that we designed it.

This is where we come in. We’ll need to write the functions which tell our GUI exactly what to do with different user inputs (e.g: which graphs to plot, which numbers to output, etc.).



```python
        self.text_1.setObjectName("text_1")
        self.button1 = QtWidgets.QPushButton(self.centralwidget)
        self.button1.setGeometry(QtCore.QRect(20, 50, 141, 71))
        self.button1.setObjectName("button1")
        self.lcdNumber = QtWidgets.QLCDNumber(self.centralwidget)
        self.lcdNumber.setGeometry(QtCore.QRect(213, 50, 101, 71))
        self.lcdNumber.setObjectName("lcdNumber")
        MainWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(sel

        self.retranslateUi(MainWind
        QtCore.QMetaObject.connectS

    def retranslateUi(self, MainWin
        _translate = QtCore.QCoreAp
        MainWindow.setWindowTitle(_
        self.text_1.setText(_transl
        self.button1.setText(_trans


if __name__ == "__main__":
    import sys
```

```
PS C:\Users\vzl29586\Learn-PyQt> & C:/Users/vzl29586/AppData/Local/Programs/Python/Python311/python.exe c:/Users/vzl29586/Learn-PyQt/counter.py
```

9. Video walkthrough

If you missed the workshop or just want to see a video walkthrough, have a look at this link:

[(4) PyQt5 Tutorial - How to Use Qt Designer - YouTube]

10. Over to you

Play around with Qt Designer and PyQt and see what you can make!

Tasks:

- **Functional counter** – make the counter increase the value of the LCD screen each time the button is pressed
- **Counter with "+1" and "-1" buttons** – make changes to the simple counter to see if you can allow the user to add *and* subtract.
- **Simple multiplier** – Use radio buttons or checkboxes to allow the user to select two numbers, from a total of 6 different number displayed to the user and output the product of those numbers.

Extensions:

- **Sine wave plotter** – Have a go at plotting a sine wave with user input. Maybe have a slider for the user to apply a multiple or offset the sine wave, or allow the user to interact with the sine wave in some other way.
  Hint: Have a look at *np.sin, np.linspace* and have a look into using a *vbox (vertical box layout)* in PyQt to place your matplotlib figure inside. For more hints, ask Esher!