Using PyQt5 GUI designer in schools

If you have used Tkinter and been un-impressed, the Qt library is easier to use and comes with a designer as well. The finished GUI is much better than Tkinter.

But there is a problem: Normal use includes use of the Cmd prompt or Powershell, something school network managers are not too keen on…

This guide will help you install and run the PyQt5 Designer, and convert the .ui files it produces to .py files you can use in your projects, using ONLY Python code.

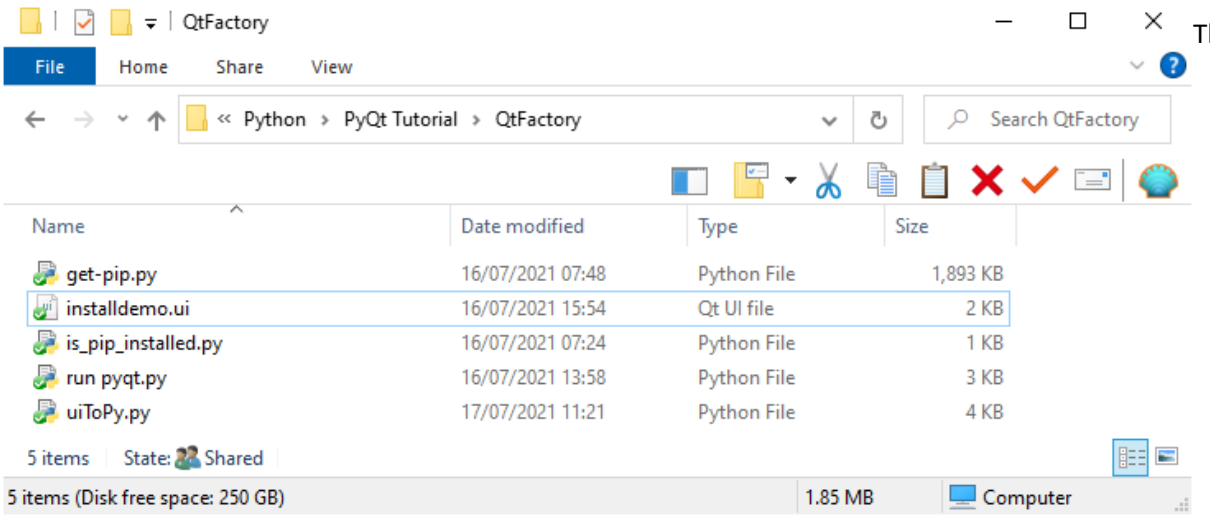
The methods shown here assume your school system has pip installed in the Python directory. If the network manager's level of paranoia is so high that pip is not installed, then you have to install pip yourself by running get-pip.py from https://bootstrap.pypa.io/ A full explanation can be found at https://github.com/pypa/get-pip .

The downside of this is the process has to be repeated for every student on every workstation they use, so that workstation's C:\Users\ folder is going to increase in size by at least 100MB for every student that uses it, by the time PyQt and PyQtDesigner is installed. It might be worth letting your network manager know this, and perhaps adding PyQt5Designer and PyQt5 to the Python install in the first place may be a better solution...

Step 1: Setup student's environment:

Using QtDesigner results in a .ui file, which has to then be converted by a command line to a.py file, so it makes sense to have a directory specifically to contain the .ui files and their .py conversions, then move them out to another directory for subsequent development. (The .py files only draw the GUI, you have to expand the code to write event handlers etc.)

Create a folder in users files with a sensible name to keep these files, eg "QtFactory" as an example.



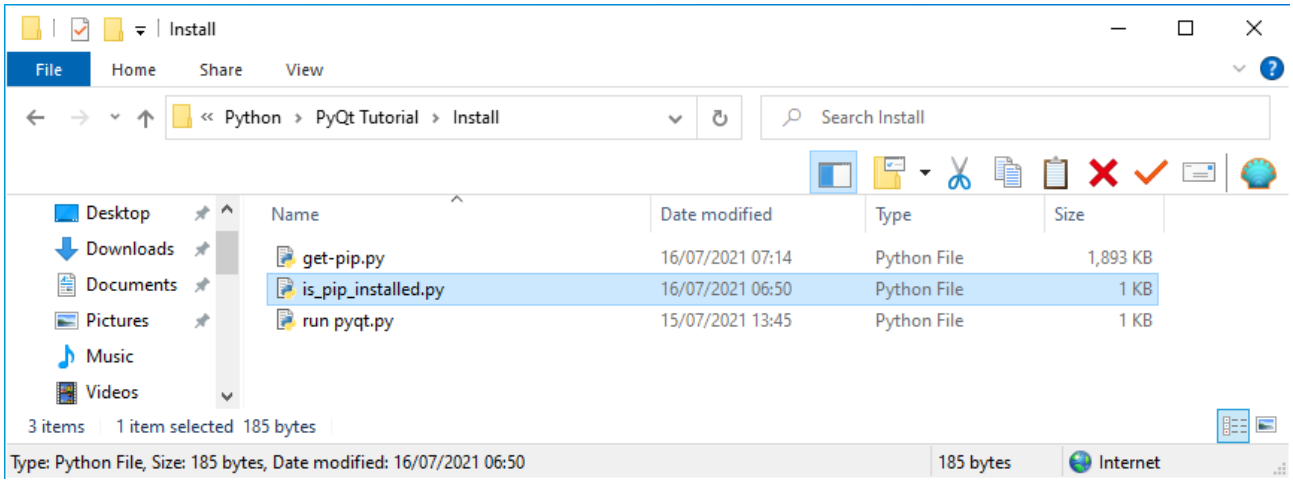
This folder will also contain a number of Python files used to setup and run PyQt. The screenshot shows these files, along with an example PyQtDesigner "installdemo.ui" file ready for conversion:

Step 2: Is pip installed?

Use the following Python code, saved as '`is_pip_installed.py`' to check:

```
import sys, subprocess
try:
  import pip
  subprocess.check_call([sys.executable, '-m', 'pip', '--version'])
except ImportError:
  print("Pip not present.")

input("Enter to quit")
```

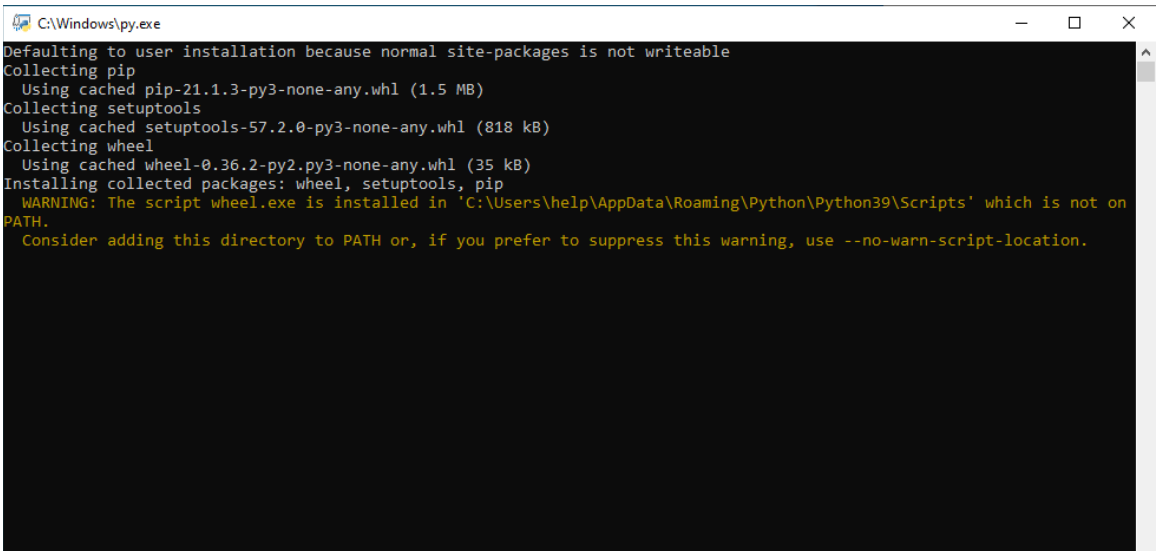
If your school system shows Python files with the Python icon, then it should be setup to run them when double-clicked:



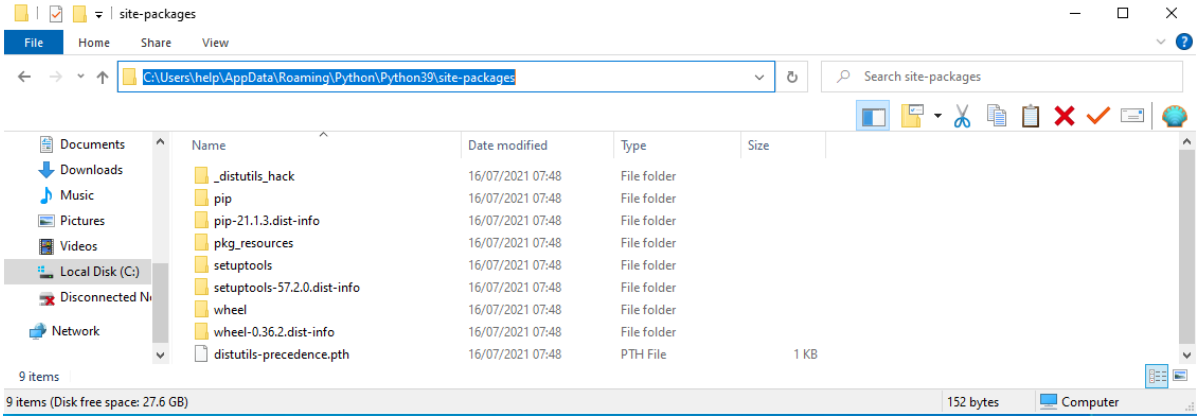
If not, then run it from whatever Python editor you have access to. Output from test VM:

Step 3: Install pip

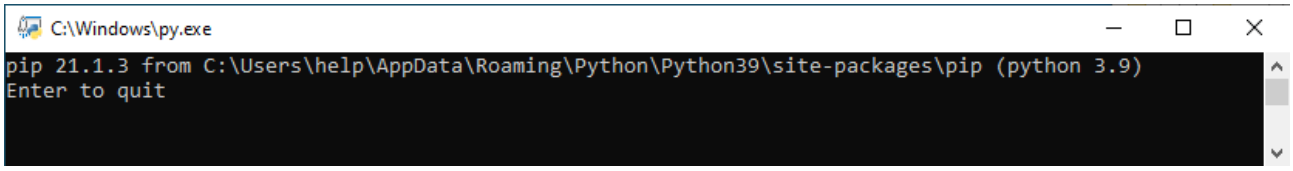
If step 2 gives a negative result, then it is time to install pip with get-pip.py. Again double-click get-pip.py, or run from an IDE.



This will install pip to %APPDATA%/Python...



Run 'is_pip_installed.py' again:



Success!

This means that on a school workstation, the same user on the same machine will have access to pip, but any other user will have to go through the same process. If a student moves to a different workstation, they will have to go through it again. The situation becomes worse with laptops, as the chances of using the same one again are small.

Step 4 Install PyQt5 and PyQtDesigner

```
import os, sys, subprocess, site                    # standard parts of Python install

def is_pip_installed():
    success = True
    try:
        import pip
        subprocess.check_call([sys.executable, '-m', 'pip', '--version'])
    except ImportError:
        success = False

    return success

def importPyQt5():
    ''' try to import or install PyQt5'''
    try:
        print("Checking for PyQt5")
        import PyQt5
        print("PyQt5 found")
    except ImportError:
        print("Attempting to install PyQt5")
        subprocess.check_call([sys.executable, "-m", "pip", "install", 'PyQt5'])

def importPyQt5Designer():
    ''' try to import or install PyQt5Designer'''
    try:
        print("Checking for PyQt5Designer")
        import PyQt5Designer
        print("PyQt5Designer found")
    except ImportError:
        print("Attempting to import PyQt5Designer")
        subprocess.check_call([sys.executable, "-m", "pip", "install", 'PyQt5Designer'])

def is_lib_installed(lib_name):
    ''' check if a library is already installed'''
    try:
        from pip._internal.operations import freeze        # will work as long as pip is installed
    except ImportError:  # pip < 10.0
        from pip.operations import freeze
    libs = freeze.freeze()
    found = False
    for lib in libs:
        if lib.__contains__(lib_name):
            found = True

    return found

def get_package_path(package_name):
    ''' returns the global or user path for site-packages that contains the package_name'''
    path = ""
    globalPackages = site.getsitepackages() # ['C:\\Program Files\\Python39', 'C:\\Program Files\\Python39\\lib\\site-packages']
    for package in globalPackages:
        if package.__contains__("site-packages"):
            site_path = package   # C:\\Program Files\\Python39\\lib\\site-packages

    if os.path.isdir(os.path.join(site_path, package_name)): # found this package_name
        path = site_path
    else:
        site_path = site.getusersitepackages() #C:\Users\help\AppData\Roaming\Python\Python39\site-packages
        if os.path.isdir(os.path.join(site_path, package_name)): # found
            path = site_path

    return os.path.join(path, package_name) # return full path including package

def main():
    if not is_pip_installed:
        print("Pip is not installed. Run get-pip.py first, then try again")
        input("Enter to quit")
    else:
        if not is_lib_installed("PyQt5"):               # PyQt5 not installed
            importPyQt5()                                   # so install it!
        if is_lib_installed("PyQt5"):                   # PyQt5 now installed, just checking
            if not is_lib_installed("Designer"):        # PyQt5Designer not installed
                importPyQt5Designer()                   # so install it!

        if is_lib_installed("Designer"):                # designer installed?
            path = get_package_path("QtDesigner")       # C:\Users\<user>\AppData\Roaming\Python\Python39\site-packages\QtDesigner
            file = os.path.join(path, "designer.exe")   # C:\Users\<user>\AppData\Roaming\Python\Python39\site-packages\QtDesigner\designer.exe
            subprocess.Popen(file)                          # start QtDesigner
        else:
            print("Unable to locate Qt Designer")
            input("Enter to continue")

main()
```

This python file, saved as run pyqt.py is an all-in-one that will:

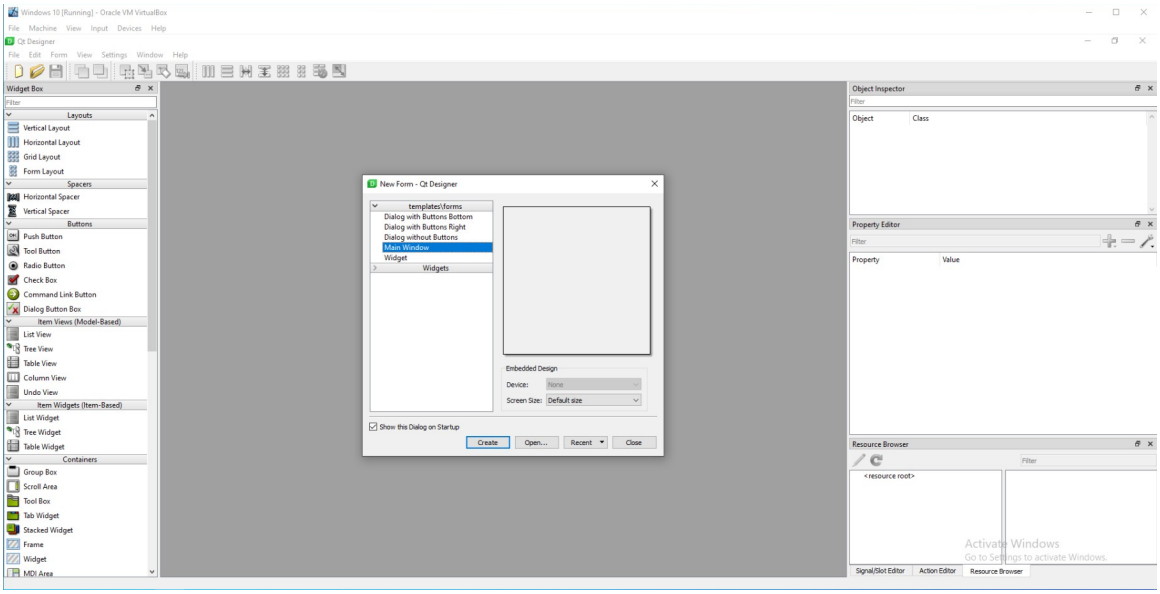
1. check if PyQt is present else install

```
C:\Program Files\Python39\python.exe
Attempting to import PyQt5
Defaulting to user installation because normal site-packages is not writeable
Collecting PyQt5
  Downloading PyQt5-5.15.4-cp36.cp37.cp38.cp39-none-win_amd64.whl (6.8 MB)
     |□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□| 6.8 MB 111 kB/s
Collecting PyQt5-sip<13,>=12.8
  Downloading PyQt5_sip-12.9.0-cp39-cp39-win_amd64.whl (63 kB)
     |□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□| 63 kB 58 kB/s
Collecting PyQt5-Qt5>=5.15
  Downloading PyQt5_Qt5-5.15.2-py3-none-win_amd64.whl (50.1 MB)
     |□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□| 50.1 MB 25 kB/s
Installing collected packages: PyQt5-sip, PyQt5-Qt5, PyQt5
  WARNING: The scripts pylupdate5.exe, pyrcc5.exe and pyuic5.exe are installed in 'C:\Users\help\AppData\Roaming\Python\
Python39\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed PyQt5-5.15.4 PyQt5-Qt5-5.15.2 PyQt5-sip-12.9.0
```

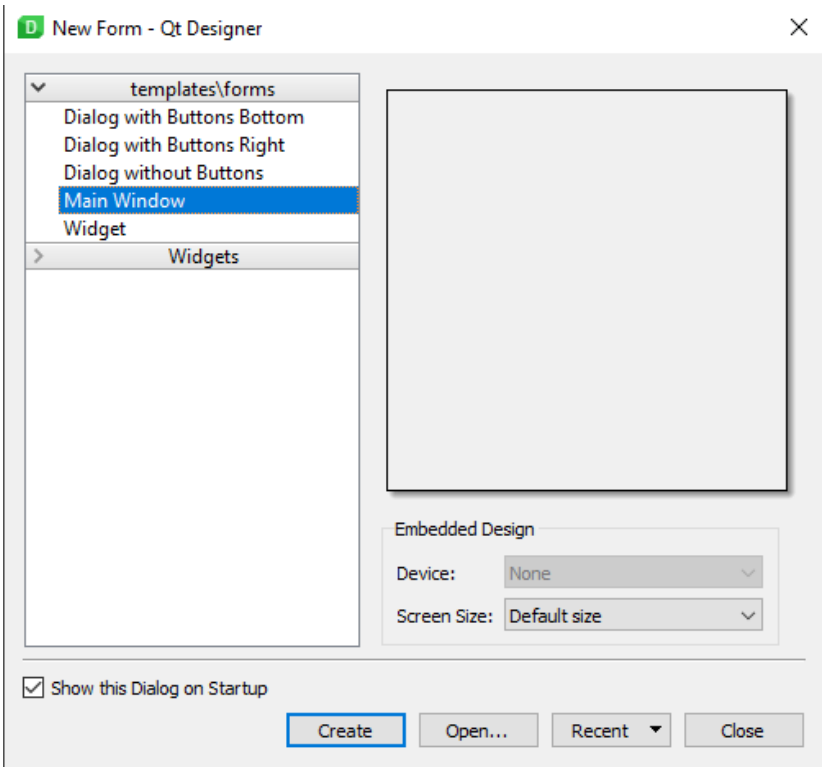
2. Install PyQt5Designer

```
C:\Program Files\Python39\python.exe
Attempting to import PyQt5Designer
Defaulting to user installation because normal site-packages is not writeable
Collecting PyQt5Designer
  Downloading PyQt5Designer-5.14.1-py3-none-win_amd64.whl (40.8 MB)
     |□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□| 40.8 MB 3.3 MB/s
Installing collected packages: PyQt5Designer
  WARNING: The script qmlview.exe is installed in 'C:\Users\help\AppData\Roaming\Python\Python39\Scripts' which is not o
n PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed PyQt5Designer-5.14.1
```

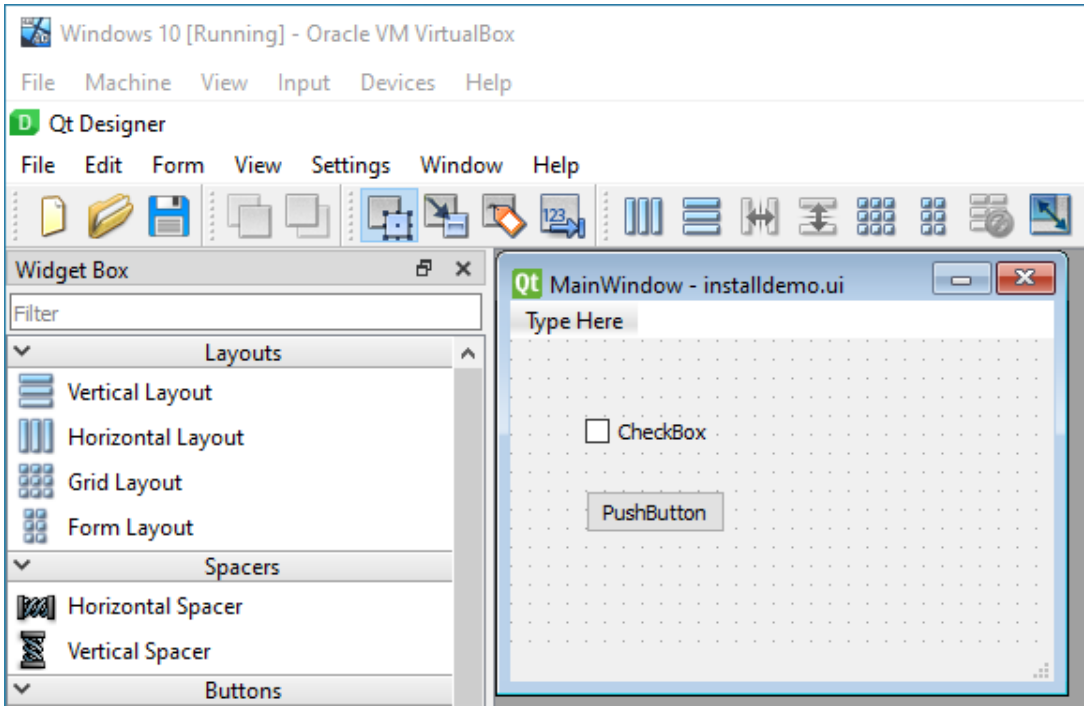
3. Run PyQt5Designer:



Select Main Window and click “Create”



Create your application by designing the GUI, then save it. This one was saved as installdemo.ui



Next you have to convert the .ui to .py file with the cmd:

`pyuic5 -x installdemo.ui -o installdemo.py.` Oh wait. No cmd on school computers!

And pyuic5.exe will not have been entered into the Path Environment Variable!

The following code, saved as uiToPy.py and kept in the “QtFactory” folder you created earlier will do the job:

```python
import os, sys, subprocess, site, glob                  # standard parts of Python install

def get_package_path(package_name):
    ''' returns the global or user path for site-packages that contains the package_name'''
    path = ""
    source = "global"
    globalPackages = site.getsitepackages()         # ['C:\\Program Files\\Python39', 'C:\\Program Files\\Python39\\lib\\
site-packages']
    for package in globalPackages:
        if package.__contains__("site-packages"):
            site_path = package                         # C:\\Program Files\\Python39\\lib\\site-packages

    if os.path.isdir(os.path.join(site_path, package_name)): # found this package_name
        path = site_path
    else:
        site_path = site.getusersitepackages()          #C:\Users\help\AppData\Roaming\Python\Python39\site-packages
        if os.path.isdir(os.path.join(site_path, package_name)): # found
            path = site_path
            source = "user"

    return  path, source                            #return path and whether the package is 'global' or 'user'

def get_scripts_path(path, source):
    ''' locate correct path for Scripts '''
    if source == "user": # C:\Users\<user>\AppData\Roaming\Python\Python39\site-packages; C:\Users\<user>\AppData\Roaming\
Python\Python39\Scripts
        parts = os.path.split(path)
        scripts_path = os.path.join(parts[0],"Scripts")
    else: # C:\Program Files\Python39\Lib\site-packages; C:\Program Files\Python39\Scripts
        parts = os.path.split(path)
        parts = os.path.split(parts[0])
        scripts_path = os.path.join(parts[0],"Scripts")
    return scripts_path # loction of Scripts varies depending on global or user

def convert(converter, ui_file, py_file):
    ''' run the pyuic5.exe: usually done in a cmd propmt '''
    if os.path.exists(converter):
        print(f"Converting {os.path.split(ui_file)[1]}...")
        subprocess.check_call([converter, "-x", ui_file , "-o" , py_file])
    else:
        print(f"pyuic5.exe not found at calculated path: {converter}")

def main():
    ''' everything runs from here '''
    cwd = os.getcwd()                                   # current working directory: convert all .ui files found here
    print(f"Checking for .ui files in {cwd}:")
    ui_files = glob.glob("*.ui")                        # list of .ui files (name only, not full path)
    for ui_file in ui_files:
        print(f"\tFound {ui_file}")
    py_files = glob.glob("*.py")                        # list of .py files (name only, not full path)
    print("\nChecking for converted .py files:")
    for py_file in py_files:
        if os.path.splitext(py_file)[0] + ".ui" in ui_files:# print any pre-converted .ui to .py files
            print(f"\tFound previously converted {py_file}")
    path, source = get_package_path("QtDesigner")          # C:\Users\<user>\AppData\Roaming\Python\Python39\site-packages or
Python install folder
    print(f"\nPyQt5 package found in {path} ({source})")
    scripts_path = get_scripts_path(path, source)          # C:\Program Files\Python39\Scripts or C:\Users\<user>\AppData\
Roaming\Python\Python39\Scripts
    print(f"Scripts path found at: {scripts_path}\n")
    converter = os.path.join(scripts_path, "pyuic5.exe")   # C:\Users\<user>\AppData\Roaming\Python\Python39\Scripts\
pyuic5.exe or C:\Program Files\Python39\Scripts\pyuic5.exe
    list_converted = []                                    # empty list to store names of converted files
    for ui_file in ui_files:                            # iterate all .ui files
        py_file = os.path.splitext(ui_file)[0] + ".py"     # py_file = .py file with same name as .ui -> test.ui found,
py_file = test.py
        if not py_file in py_files:                        # Look for equivalent, previously converted .py file with same
name
            list_converted.append(ui_file)                 # add to list of converted files
            ui_file = os.path.join(cwd, ui_file)        # ui_file given full pathname
            py_file = os.path.join(cwd, py_file)        # py_file given full pathname
            convert(converter, ui_file, py_file)        # create .py from .ui
    # Display a list of converted files
    if len(list_converted) > 0:
        print("The following files were converted from .ui to .py:")
        for file in list_converted:
            print(f"\t{file}")
        print(f"Move them from {cwd} to a suitable location for their associated projects")

    else:
        print("\tNo new .ui files found for conversion")

    input("\nEnter to quit")

main()
```

Currently the QtFactory folder looks like the screenshot on page 1

Double-click (or run from an IDE) uiToPy.py:

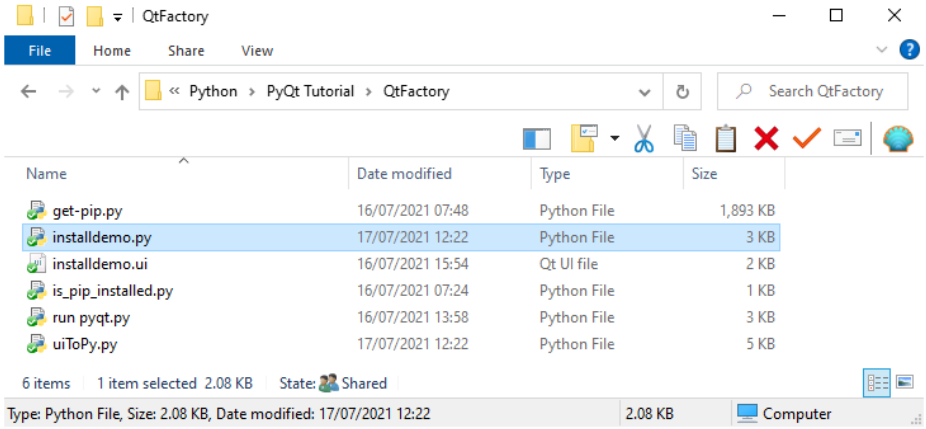
```
C:\Program Files\Python38\python.exe
Checking for .ui files in c:\Users\graha\Dropbox\Computer Club\Python\PyQt Tutorial\QtFactory:

        Found installdemo.ui

Checking for converted .py files:

PyQt5 package found in C:\Program Files\Python38\lib\site-packages (global)
Scripts path found at: C:\Program Files\Python38\Scripts

Converting installdemo.ui...
The following files were converted from .ui to .py:
        installdemo.ui
Move them from c:\Users\graha\Dropbox\Computer Club\Python\PyQt Tutorial\QtFactory to a suitab
le location for their associated projects

Enter to quit
```

The output gives a summary of events, and the file installdemo.ui has been converted to installdemo.py

Run it again to see what happens:

```
C:\Program Files\Python38\python.exe
Checking for .ui files in c:\Users\graha\Dropbox\Computer Club\Python\PyQt Tutorial\QtFactory:

        Found installdemo.ui

Checking for converted .py files:
        Found previously converted installdemo.py

PyQt5 package found in C:\Program Files\Python38\lib\site-packages (global)
Scripts path found at: C:\Program Files\Python38\Scripts

        No new .ui files found for conversion

Enter to quit
```

As you can see nothing was done as no new .ui files were found. The folder now looks like this:

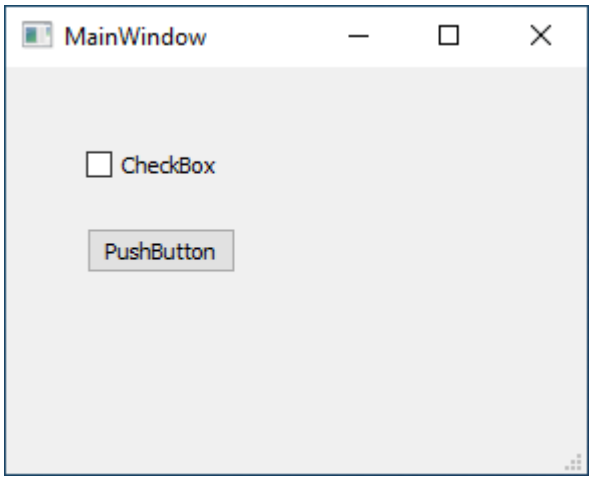


The converted file code is:

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'c:\Users\graha\Dropbox\Computer Club\
Python\PyQt Tutorial\QtFactory\installdemo.ui'
#
# Created by: PyQt5 UI code generator 5.15.3
#
# 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(290, 203)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(40, 80, 75, 23))
        self.pushButton.setObjectName("pushButton")
        self.checkBox = QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox.setGeometry(QtCore.QRect(40, 40, 70, 17))
        self.checkBox.setObjectName("checkBox")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 290, 21))
        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"))
        self.pushButton.setText(_translate("MainWindow", "PushButton"))
        self.checkBox.setText(_translate("MainWindow", "CheckBox"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

You will note the class has no constructor, as it is intended to be used as a static class

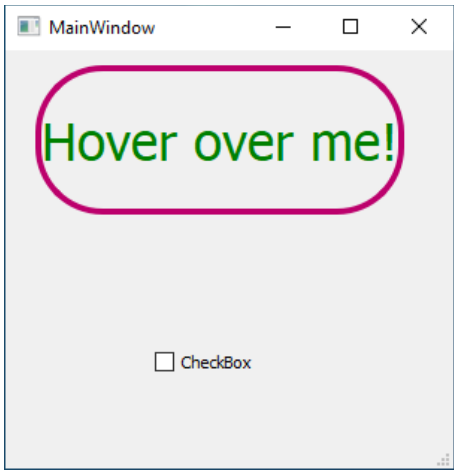
And this is what it looks like:

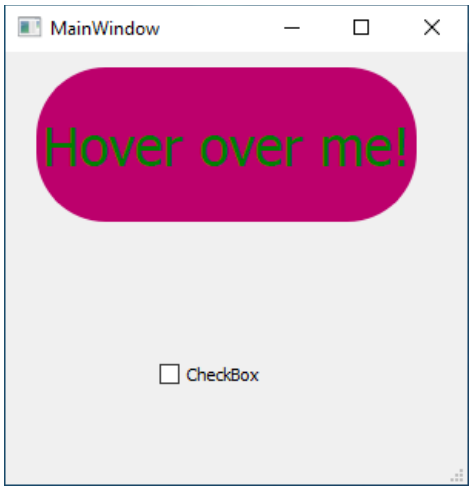


So why is this better than Tkinter??

Make some changes to the code using CSS which alters the properties of any element in the GUI, run it and move the mouse over the button:

```
        MainWindow.resize(300, 280)
    self.centralwidget = QtWidgets.QWidget(MainWindow)
    self.centralwidget.setObjectName("centralwidget")
    self.pushButton = QtWidgets.QPushButton(self.centralwidget)
    #self.pushButton.setGeometry(QtCore.QRect(40, 80, 75, 23))
    self.pushButton.setObjectName("PushButton")
    self.pushButton.setCursor(QtCore.Qt.PointingHandCursor)
    self.pushButton.setStyleSheet(
        "*{border: 4px solid '#BC006C';" +
        "border-radius: 45px;" +
        "font-size: 35px;" +
        "color: 'green';" +
        "padding: 25px 0;" +
        "margin: 10px 20px;}" +
        "*:hover{background: '#BC006C';}"
        )
    self.checkBox = QtWidgets.QCheckBox(self.centralwidget)
    self.checkBox.setGeometry(QtCore.QRect(100, 200, 70, 17))
```

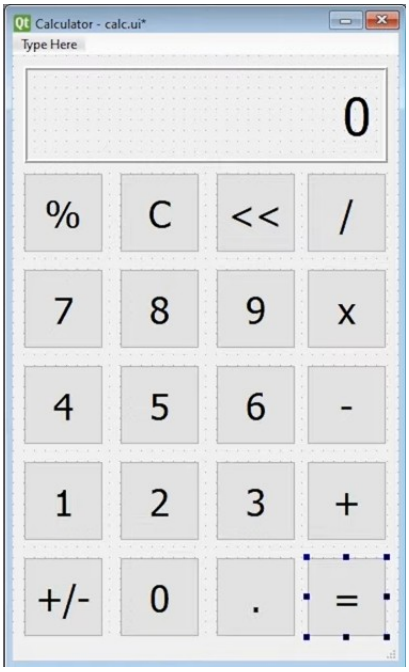




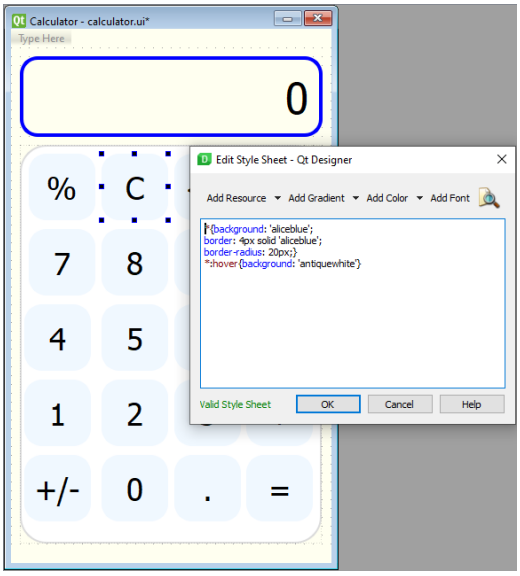
The CSS changes made manually can also be handled in PyQt5Designer

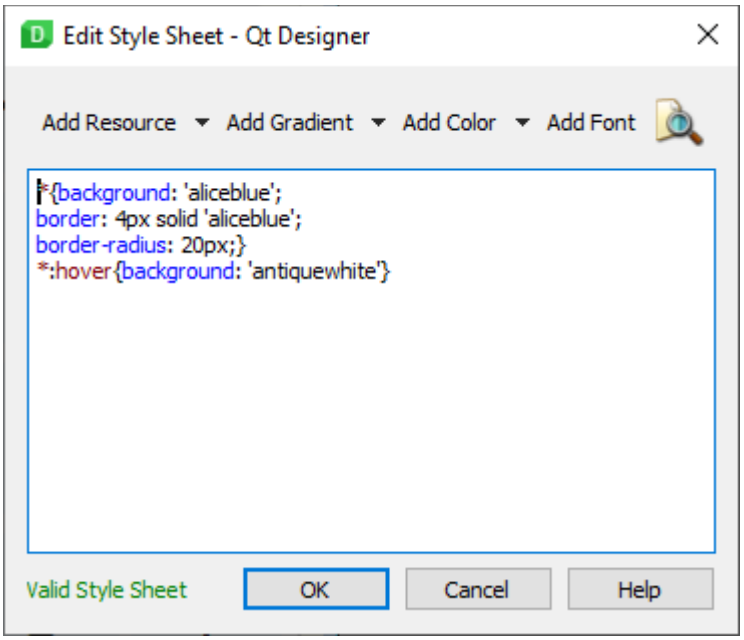
There is an excellent PyQt5 tutorial on Youtube at https://www.youtube.com/watch?v=H1FpwbavWIk&list=PLCC34OHNcOtpmCA8s_dpPMvQLyHbvxocY&index=8

Here is his design for a calculator made with PyQt5Designer. He keeps it simple and continues later to add event handlers in the converted code. It is functional, much easier to design than Tkinter, but the appearance is not spectacular:

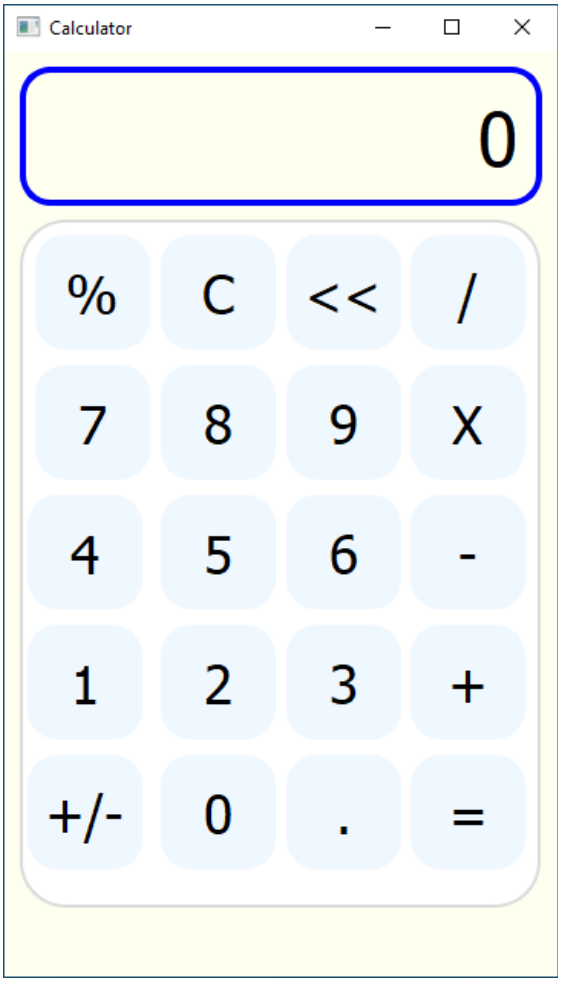


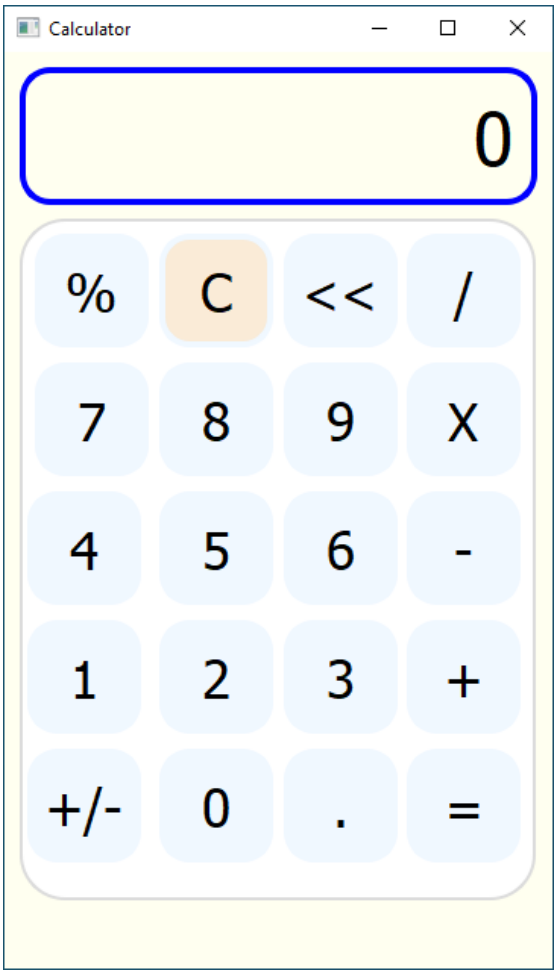
The designer has a built-in CSS tool, used here to style the 'C' button:





The file is saved, converted and moved out into a separate folder to continue with the code.
Here it is running, showing the action of hover over the ‘C’ button:





~~So why is this better than Tkinter??~~

Your students will be much more motivated to use a GUI when they can produce modern apps using Python.

Enjoy