Python GUI for Data Acquisition

Prof. David R. Loker, Pennsylvania State University, Behrend College

David R. Loker received the M.S.E.E. degree from Syracuse University in 1986. In 1984, he joined General Electric (GE) Company, AESD, as a design engineer. In 1988, he joined the faculty at Penn State Erie, The Behrend College. In 2007, he became the Chair of the Electrical and Computer Engineering Technology Program. His research interests include wireless sensor networks, data acquisition systems, and communications systems.

Prof. Teck Meng (Jonathan) Liaw, Pennsylvania State University, Behrend College

Jonathan Liaw earned his B.Sc. in Applied Physics from the University of Portsmouth, UK, in 1983, followed by an M.Sc. in Computer Science from the University of Wyoming in 1985. He later obtained a second M.Sc. in Embedded Software from Gannon University in 2009. After serving as an adjunct professor at Gannon University, he joined Penn State Erie, The Behrend College, as a full-time faculty member in 2019. His research interests include machine learning, embedded systems design, and quantum computing.

Dr. Steven Nozaki, Pennsylvania State University, Behrend College

Ph.D. Engineering Education - The Ohio State University

Python GUI for Data Acquisition

Abstract

Data acquisition (DAQ) involves sampling signals utilizing sensors that measure various electrical parameters, processing these signals, and displaying real-world results. This system is often implemented with a USB DAQ device (e.g., myDAQ, etc.) connected to a PC, or a controller module (e.g., Raspberry Pi Pico, Arduino, etc.), and software employed with graphical or text-based programming (e.g., LabVIEW, Matlab, etc.). With the increasing availability of free and open-source software package availability in Python and the increasing demand for Python skills among graduates, some courses are now incorporating Python for software development in practical applications. Python comes with a module called *tkinter* (short for "Tk interface") that allows users to create simple and quick GUI programs. The *tkinter* module provides various graphical widgets (e.g., Button, Entry, Label, Radiobutton, etc.) with which the user can interact or view. A more robust method for creating advanced Python GUI program uses PySide6 and Qt Designer.

The goal of this paper is to utilize Python with the myDAQ to develop GUI programs suitable for lab projects in electrical and computer engineering and engineering technology, software engineering, and computer science programs. First, a lab project is shown to introduce students to Python GUI development using the *tkinter* module for simulating various waveforms (sinusoidal, square, and triangular), determining the spectrum of these signals, and displaying both the time-domain and frequency-domain representation of these waveforms. The GUI allows the user to select the waveform type, amplitude characteristics (peak amplitude, DC offset), frequency, signal duration, and sample rate. The second lab project introduces students to Python GUI development, using *tkinter*, for data collection with the myDAQ and plotting the time-domain and frequency-domain representation of the data collected. Then, these programs are developed using PySide6 and Qt Designer. Each project includes detailed engineering requirements, software code, and outcomes.

Introduction to Data Acquisition

Data acquisition (DAQ) involves sampling signals utilizing sensors that provide electrical outputs based on physical inputs, processing these signals into real-world values, and displaying the results. This system can be implemented with a USB DAQ device (e.g., myDAQ, etc.) connected to a PC, and software employed with graphical or text-based programming (e.g., LabVIEW, Matlab, etc.).

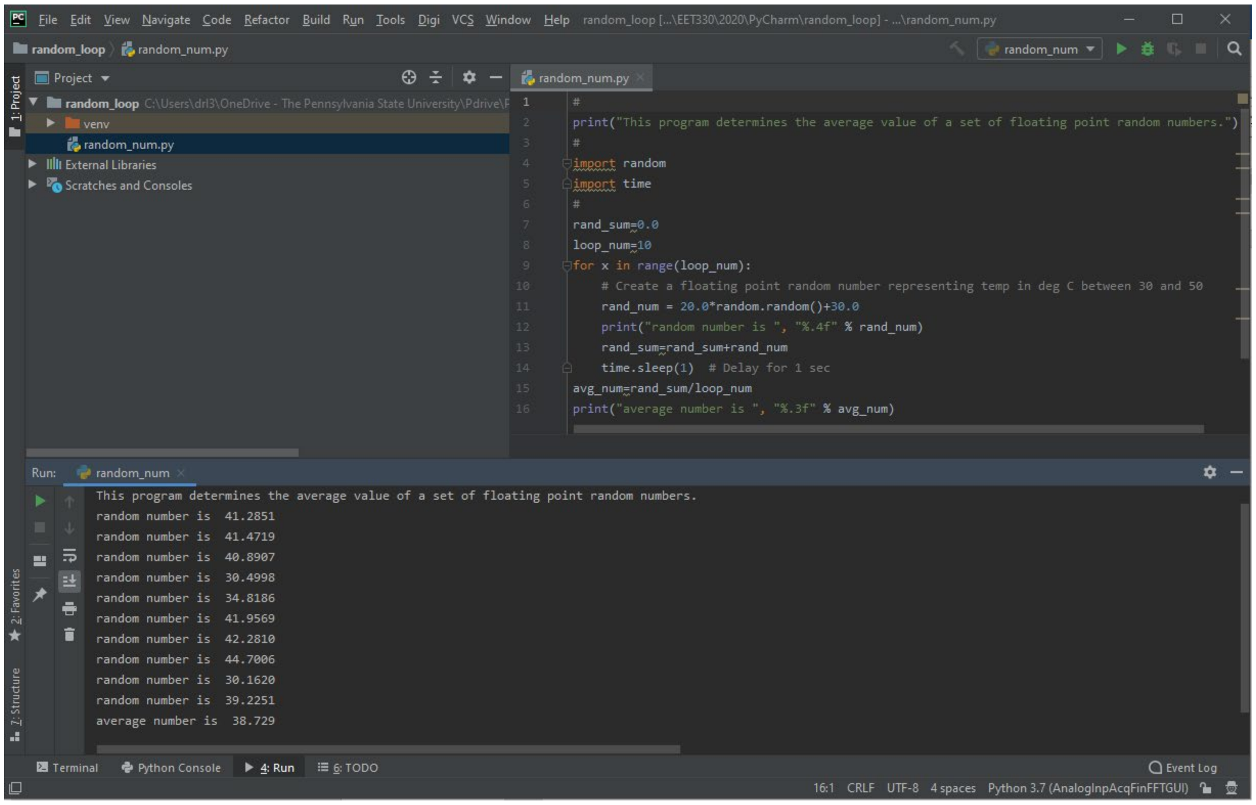
A variety of courses in both electrical and computer disciplines involve data acquisition. One such course is a Measurements and Instrumentation course [1]. This reference describes a junior-level course, which uses the myDAQ device for data acquisition and LabVIEW as the programming language, for designing and implementing measurement systems. Other courses that can include data acquisition software programming, computer networking, communications systems, and project-based courses.

Embedded systems are an alternative to PC-based data acquisition systems [2]. Embedded systems are used to read/analyze sensor data to perform data acquisition at a higher sample rate than what the PC is able to do, and adhere to a fixed sample rate requirement, which may be necessary for data processing and analysis. Courses that can use embedded systems include digital design, introduction to microprocessors, embedded systems design, intermediate microprocessors, and advanced microprocessors. Also, there are additional courses that can use embedded systems as an integral component to the course. These courses include wireless communications systems, control systems, and senior-level project-based courses. The inexpensive Raspberry Pi Pico W is an example of a hardware device for embedded systems. MicroPython (a small subset of the Python standard library) can be used for software programming, and Thonny can be used as the software development environment for writing code and downloading it to the device.

Due to the free and open-source software availability of Python and with the growing interest in graduates having Python experience, Python can be used for software development [3]. The goal of this paper is to utilize Python with the myDAQ to develop graphical user interface (GUI) programs suitable for lab projects for electrical and computer engineering and engineering technology programs.

Introduction to Python Coding Using PyCharm

For the software development environment, the Community Edition of PyCharm is a free download [4]. Various online resources are available for learning PyCharm [5-6]. Additionally, an open educational resource (OER) text is available online for students that are new to Python [7]. An example of a Python project within PyCharm is shown in Figure 1 [8]. This program calculates the average value of 10 temperature readings, where each temperature reading is a floating-point number generated from a random number that is scaled to represent temperature in °C between 30 and 50. A delay for one second is used after each number is generated. After all 10 temperatures have been displayed, the average temperature is displayed.



```
#
print("This program determines the average value of a set of floating point random numbers.")
#
import random
import time
#
rand_sum=0.0
loop_num=10
for x in range(loop_num):
    # Create a floating point random number representing temp in deg C between 30 and 50
    rand_num = 20.0*random.random()+30.0
    print("random number is ", "%.4f" % rand_num)
    rand_sum=rand_sum+rand_num
    time.sleep(1)  # Delay for 1 sec
avg_num=rand_sum/loop_num
print("average number is ", "%.3f" % avg_num)
```

```
This program determines the average value of a set of floating point random numbers.
random number is  41.2851
random number is  41.4719
random number is  40.8907
random number is  30.4998
random number is  34.8186
random number is  41.9569
random number is  42.2810
random number is  44.7006
random number is  30.1620
random number is  39.2251
average number is  38.729
```

Figure 1. Average Temperature Reading Python Project

Introduction to GUI Programming Using *tkinter*

A module named *tkinter* (short for "Tk interface") allows users the ability to create simple GUI programs [3]. The *tkinter* module provides various graphical widgets (e.g., Button, Entry, Label, Radiobutton, etc.) with which the user can interact or view. An example of a simple Python GUI program that converts temperature from degrees C to degrees F is shown in Figures 2-3. This GUI utilizes the following widgets.

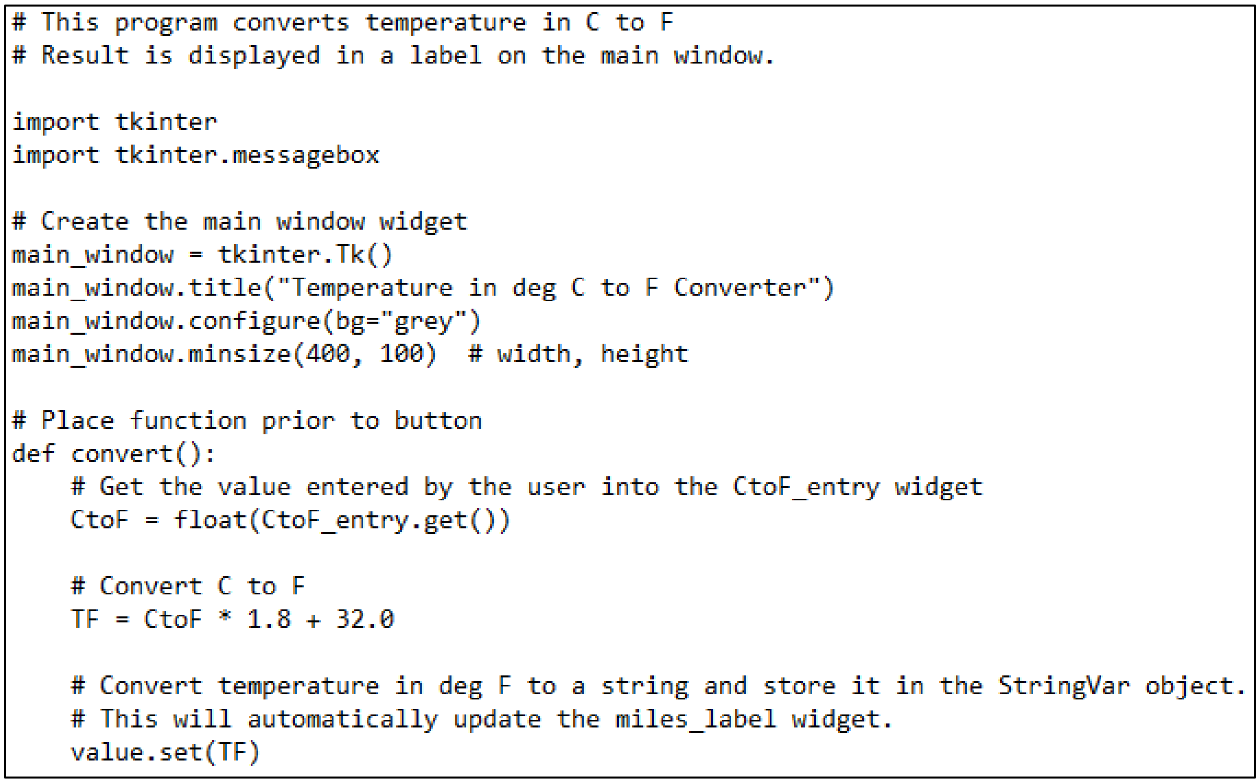
- Frame – Container to hold the other widgets.
- Entry – Area in which the user enters the temperature in degrees C.
- Label – Area that displays one line of text (e.g., Enter a temperature in deg. C:)
- Button – Causes an action to occur when selected.

Figure 2 shows the program to create the main window and the function to convert the temperature to degrees F. The main window contains a title, configures it for a grey background color, and sets the size for width and height.

Figure 3 creates the top, middle, and bottom frames for the main window.

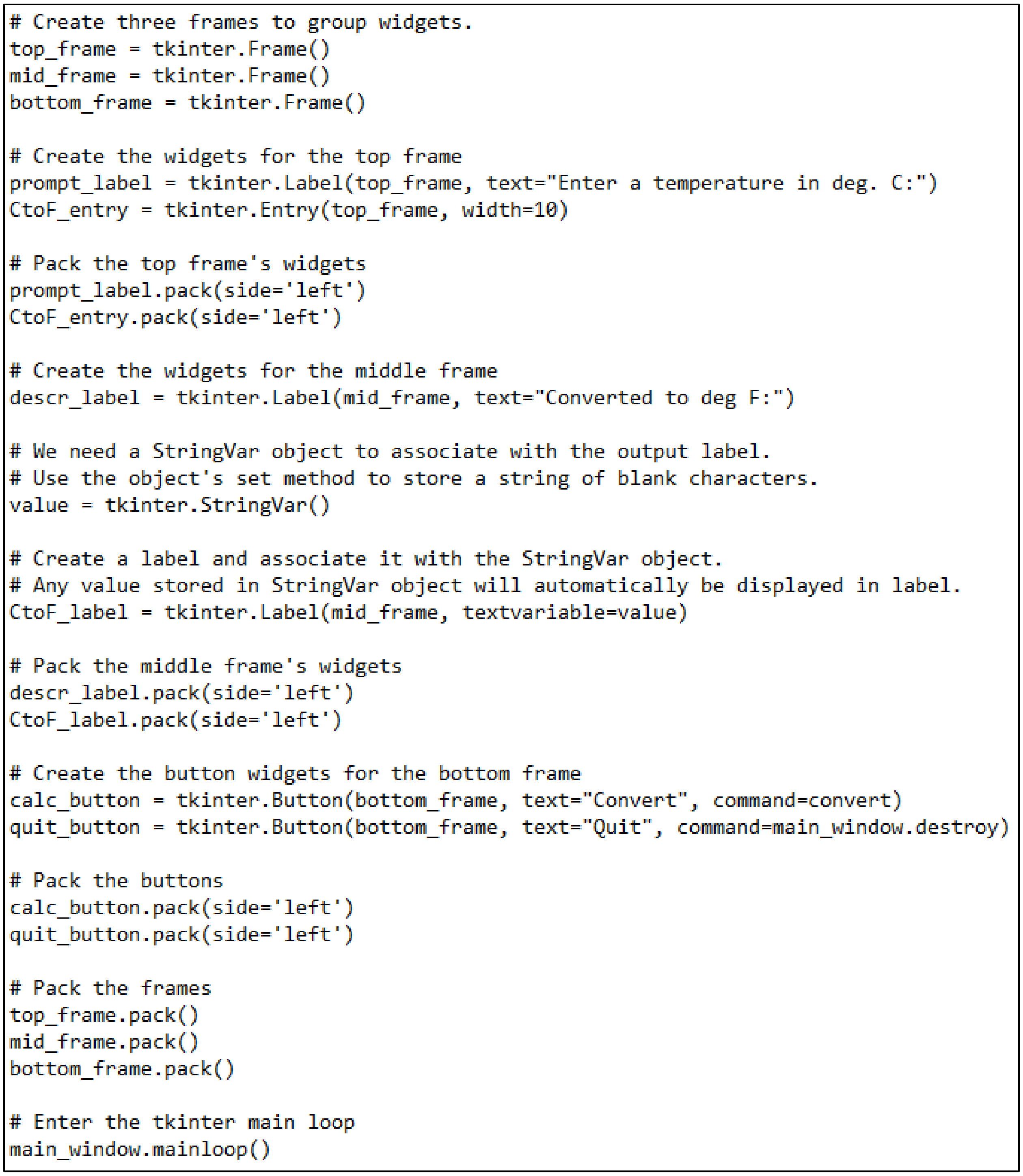
- Top frame contains the label specifying that the user enter the temperature in degrees C and an entry field for the user to enter the value.
- Middle frame contains the label and value for the temperature converted to degrees F.
- Bottom frame contains the convert and quit buttons. When the convert button is selected, the function shown in Figure 2 is executed. When the quit button is selected, the main window closes.

Figure 4 shows the GUI for the application.



```
# This program converts temperature in C to F
# Result is displayed in a label on the main window.

import tkinter
import tkinter.messagebox

# Create the main window widget
main_window = tkinter.Tk()
main_window.title("Temperature in deg C to F Converter")
main_window.configure(bg="grey")
main_window.minsize(400, 100)  # width, height

# Place function prior to button
def convert():
    # Get the value entered by the user into the CtoF_entry widget
    CtoF = float(CtoF_entry.get())

    # Convert C to F
    TF = CtoF * 1.8 + 32.0

    # Convert temperature in deg F to a string and store it in the StringVar object.
    # This will automatically update the miles_label widget.
    value.set(TF)
```

Figure 2. Convert C to F Python Project for Main Window and Callback Function



```
# Create three frames to group widgets.
top_frame = tkinter.Frame()
mid_frame = tkinter.Frame()
bottom_frame = tkinter.Frame()

# Create the widgets for the top frame
prompt_label = tkinter.Label(top_frame, text="Enter a temperature in deg. C:")
CtoF_entry = tkinter.Entry(top_frame, width=10)

# Pack the top frame's widgets
prompt_label.pack(side='left')
CtoF_entry.pack(side='left')

# Create the widgets for the middle frame
descr_label = tkinter.Label(mid_frame, text="Converted to deg F:")

# We need a StringVar object to associate with the output label.
# Use the object's set method to store a string of blank characters.
value = tkinter.StringVar()

# Create a label and associate it with the StringVar object.
# Any value stored in StringVar object will automatically be displayed in label.
CtoF_label = tkinter.Label(mid_frame, textvariable=value)

# Pack the middle frame's widgets
descr_label.pack(side='left')
CtoF_label.pack(side='left')

# Create the button widgets for the bottom frame
calc_button = tkinter.Button(bottom_frame, text="Convert", command=convert)
quit_button = tkinter.Button(bottom_frame, text="Quit", command=main_window.destroy)

# Pack the buttons
calc_button.pack(side='left')
quit_button.pack(side='left')

# Pack the frames
top_frame.pack()
mid_frame.pack()
bottom_frame.pack()

# Enter the tkinter main loop
main_window.mainloop()
```

Figure 3. Convert C to F Python Project to create Frames

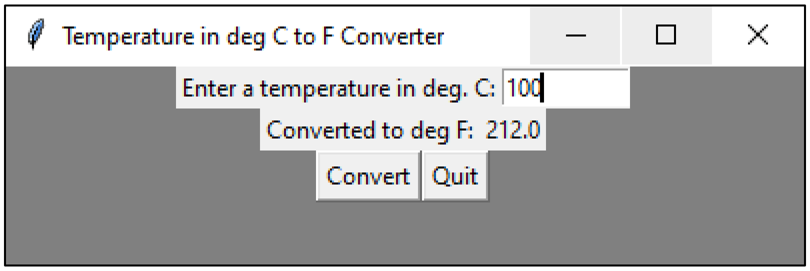

Temperature in deg C to F Converter
Enter a temperature in deg. C: 100
Converted to deg F: 212.0
Convert
Quit

Figure 4. Convert C to F Python GUI

Python GUI Projects Using *tkinter*

There are two Python GUI projects using *tkinter* shown in this paper. The first project is a Python GUI program for simulating various waveforms (sinusoidal, square, and triangular), determining the spectrum of these signals, and displaying both the time-domain and frequency-domain representation of these waveforms. The GUI allows the user to select the waveform type, amplitude characteristics (peak amplitude, DC offset), frequency, signal duration, and sample rate. The second lab project introduces students to Python GUI development for data collection using the myDAQ and plotting the time-domain and frequency-domain representation of the data collected. The waveforms are plotted using matplotlib, a Python library for creating static, animated, and interactive visualizations [9].

<u>Waveform Simulation GUI</u>

The engineering requirements for the waveform simulation project are shown below.

- GUI development using Python with windows for user inputs and outputs
- Selection for sinusoidal, square, triangular waveform simulation
- Inputs for waveform peak amplitude, DC offset, and frequency
- Inputs for sample rate and signal duration
- Buttons for selection and quit
- Output window showing both the time-domain and frequency-domain plots

The waveform simulation project imports the following libraries.

- tkinter
- matplotlib
- numpy
- scipy

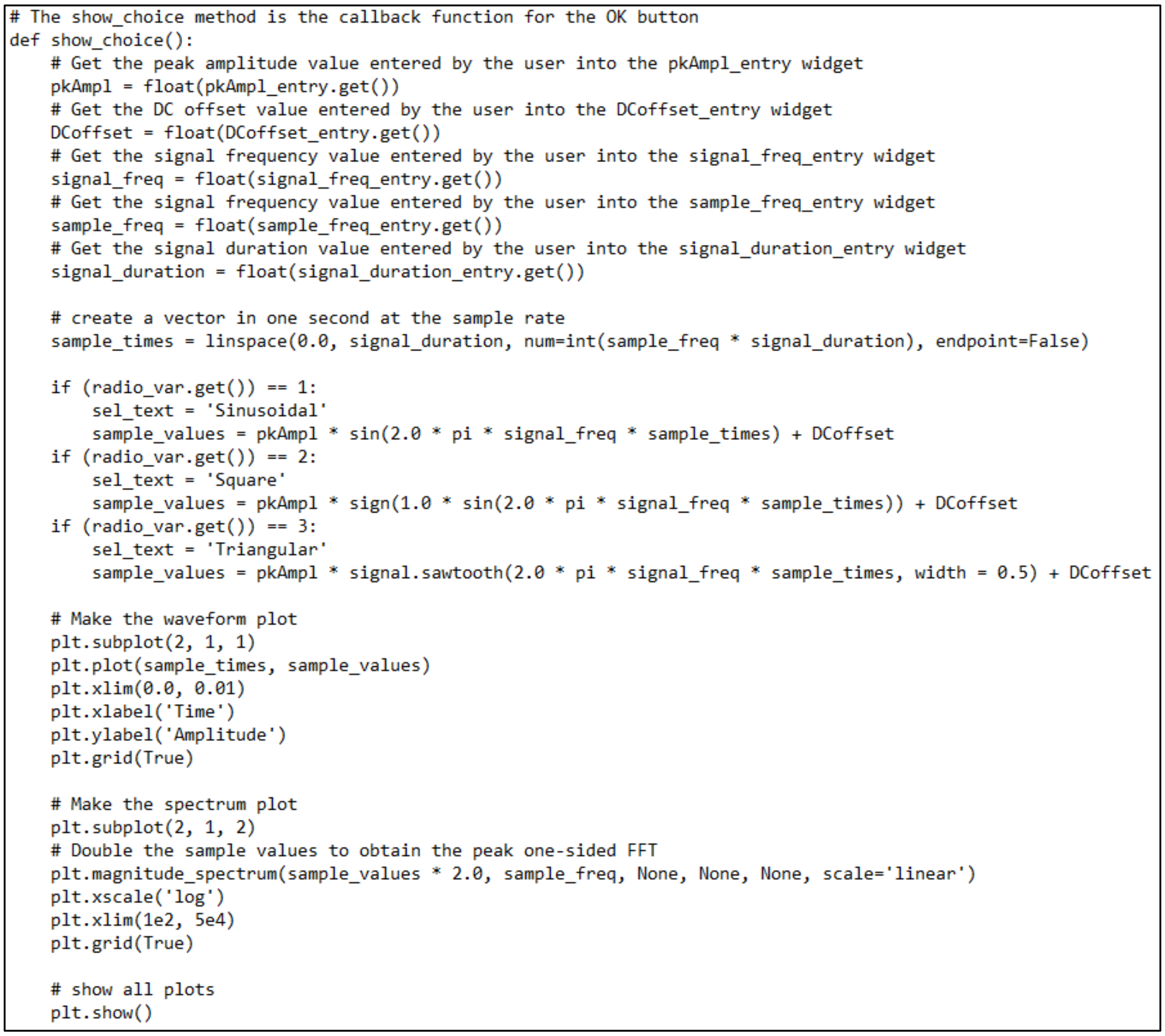
Figure 5 shows the primary Python GUI code for this project. It includes the program code to create the main window and the callback function. The main window contains a title and sets the size for width and height. The callback function includes the following program code.

- User entry values for waveform parameters (e.g., peak amplitude, DC offset, frequency)
- User entry values for sampling (e.g., rate, signal duration)
- Radio button user selection for waveform type (e.g., sinusoidal, square, triangular)
- Plotting for both the time-domain and frequency-domain outputs

Figure 6 shows the user interface window. When a 1 KHz, 5V peak squarewave is selected, Figure 7 shows the output window for the squarewave from this project. The time-domain x-axis plot is limited to 10mS to show only 10 cycles of the waveform. The frequency-domain plot (amplitude and frequency content of the spectrum) is consistent with the theoretical values from the Fourier Series shown in Equation 1 [10].

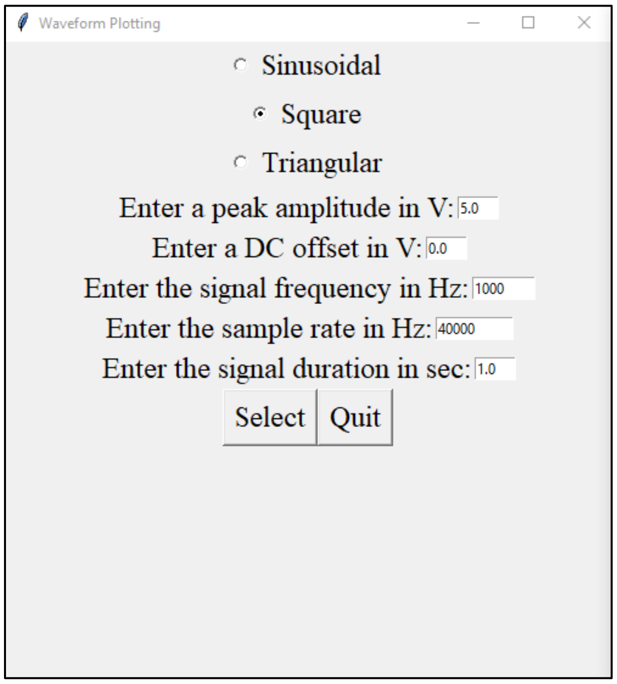
Figure 8 shows the output window for a triangular waveform (same parameters as for the squarewave from Figure 6) from this project. The frequency-domain plot (amplitude and frequency content of the spectrum) is consistent with the theoretical values from the Fourier Series shown in Equation 2.

Table 1 shows a summary of the spectrum (frequency content through the 5th harmonic) of the three waveforms.



```
# The show_choice method is the callback function for the OK button
def show_choice():
    # Get the peak amplitude value entered by the user into the pkAmpl_entry widget
    pkAmpl = float(pkAmpl_entry.get())
    # Get the DC offset value entered by the user into the DCoffset_entry widget
    DCoffset = float(DCoffset_entry.get())
    # Get the signal frequency value entered by the user into the signal_freq_entry widget
    signal_freq = float(signal_freq_entry.get())
    # Get the signal frequency value entered by the user into the sample_freq_entry widget
    sample_freq = float(sample_freq_entry.get())
    # Get the signal duration value entered by the user into the signal_duration_entry widget
    signal_duration = float(signal_duration_entry.get())

    # create a vector in one second at the sample rate
    sample_times = linspace(0.0, signal_duration, num=int(sample_freq * signal_duration), endpoint=False)

    if (radio_var.get()) == 1:
        sel_text = 'Sinusoidal'
        sample_values = pkAmpl * sin(2.0 * pi * signal_freq * sample_times) + DCoffset
    if (radio_var.get()) == 2:
        sel_text = 'Square'
        sample_values = pkAmpl * sign(1.0 * sin(2.0 * pi * signal_freq * sample_times)) + DCoffset
    if (radio_var.get()) == 3:
        sel_text = 'Triangular'
        sample_values = pkAmpl * signal.sawtooth(2.0 * pi * signal_freq * sample_times, width = 0.5) + DCoffset

    # Make the waveform plot
    plt.subplot(2, 1, 1)
    plt.plot(sample_times, sample_values)
    plt.xlim(0.0, 0.01)
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.grid(True)

    # Make the spectrum plot
    plt.subplot(2, 1, 2)
    # Double the sample values to obtain the peak one-sided FFT
    plt.magnitude_spectrum(sample_values * 2.0, sample_freq, None, None, None, scale='linear')
    plt.xscale('log')
    plt.xlim(1e2, 5e4)
    plt.grid(True)

    # show all plots
    plt.show()
```

Figure 5. Waveform Simulation Project for Main Window and Callback Function



Sinusoidal [✓ Square] Triangular

Enter a peak amplitude in V: 5.0

Enter a DC offset in V: 0.0

Enter the signal frequency in Hz: 1000

Enter the sample rate in Hz: 40000

Enter the signal duration in sec: 1.0

Figure 6. Waveform Simulation User Interface Window

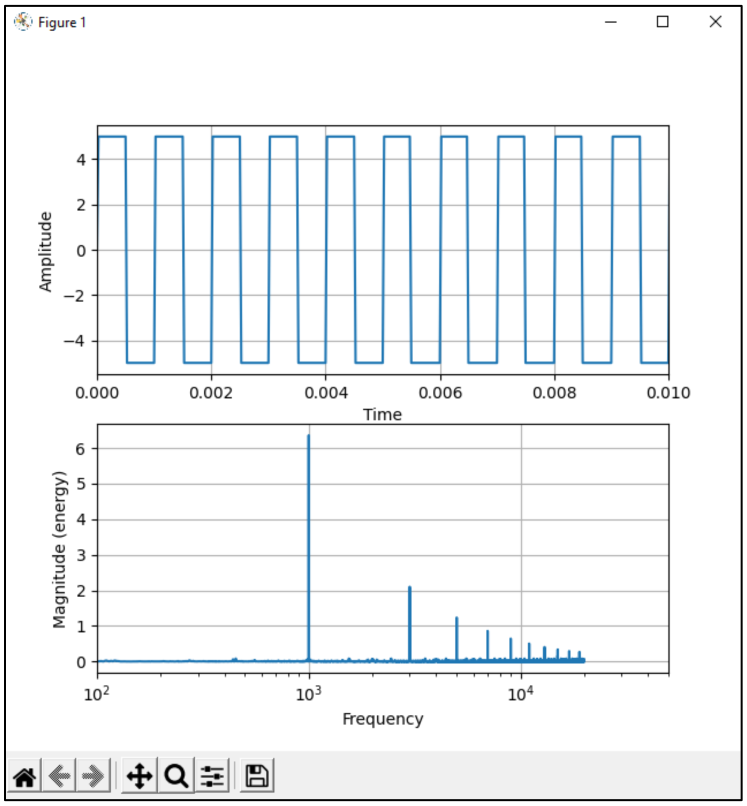


Figure 7. Square Waveform Simulation Output Window

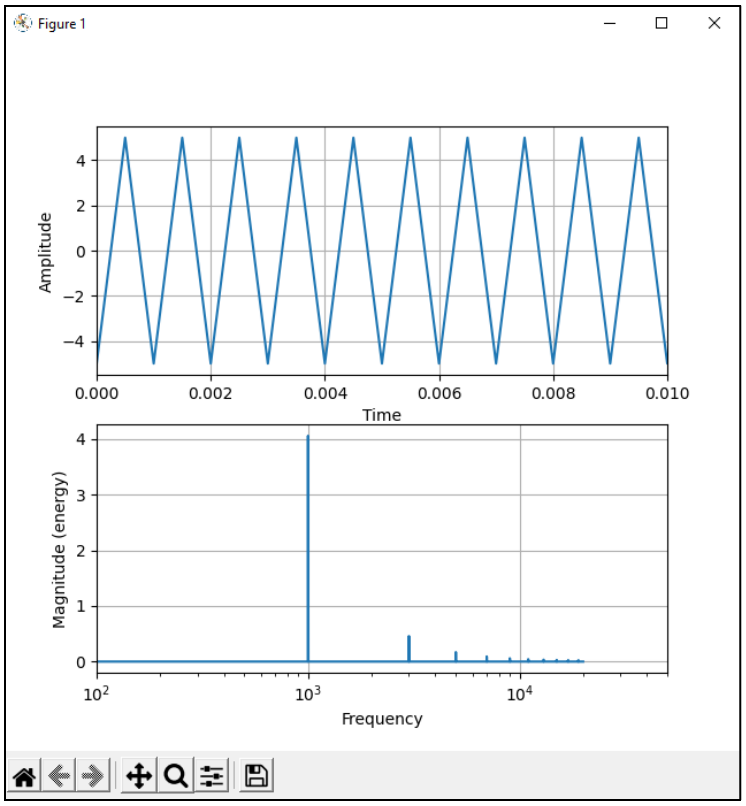


Figure 8. Triangular Waveform Simulation Output Window

$$\sum_{n=odd}^{\infty} \frac{2*V_{p-p}}{n\pi} \sin(n\omega t) \qquad \text{Eq. 1}$$

$$\sum_{n=odd}^{\infty} \frac{-4*V_{p-p}}{(n\pi)^2} \cos(n\omega t) \qquad \text{Eq. 2}$$

Table 1. Waveform Spectrum Results

Waveform	**Magnitude of Peak Voltage (V) and Frequency (Hz)**		
	1st Harmonic	**3rd Harmonic**	**5th Harmonic**
Sinusoidal	5, 1000	N/A	N/A
Square	$^{20}/_{\pi}$, 1000	$^{20}/_{3\pi}$, 3000	$^{20}/_{5\pi}$, 5000
Triangle	$^{40}/_{\pi^2}$, 1000	$^{40}/_{(3\pi)^2}$, 3000	$^{40}/_{(5\pi)^2}$, 5000

Data Collection GUI Using the myDAQ

The engineering requirements for the data collection project are shown below.

- GUI development using Python with windows for user inputs and outputs
- Inputs for sample rate and number of sample
- Buttons for select and quit
- Output window showing both the time-domain and frequency-domain plots

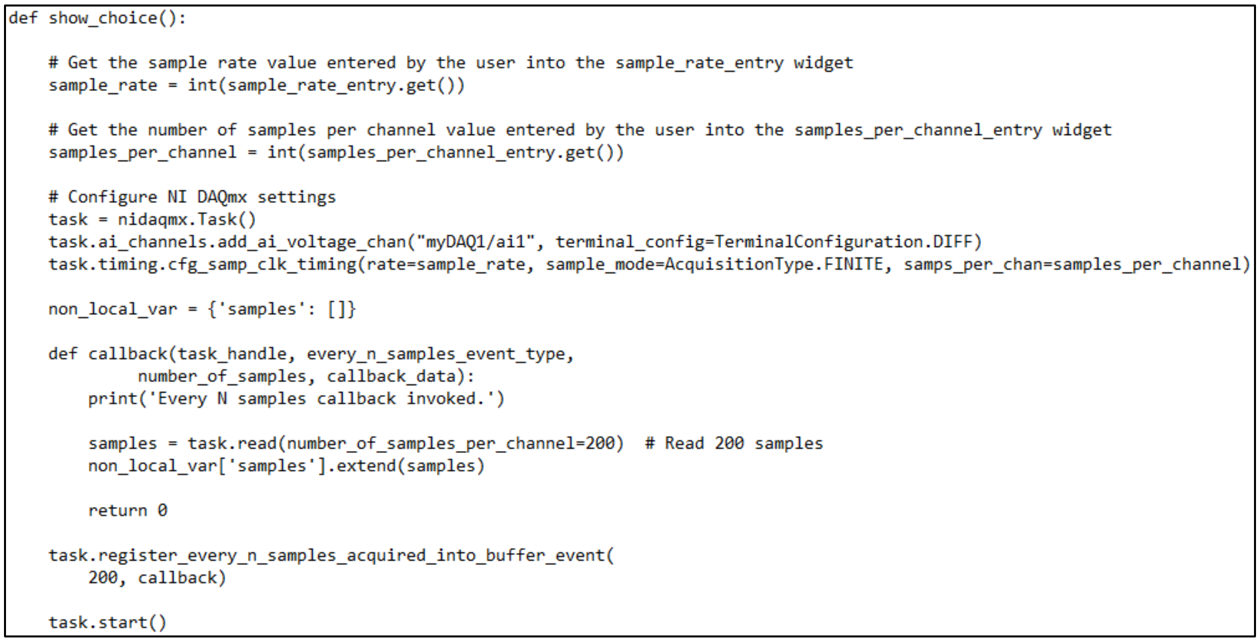
The waveform simulation project imports the following libraries.

- tkinter
- matplotlib
- numpy
- nidaqmx

The nidaqmx package allows the development of data acquisition applications using the myDAQ device in Python. Instructions for installing nidaqmx and for downloading example programs for analog input are available online [11]. Figure 9 shows the callback function that performs the data acquisition [11].

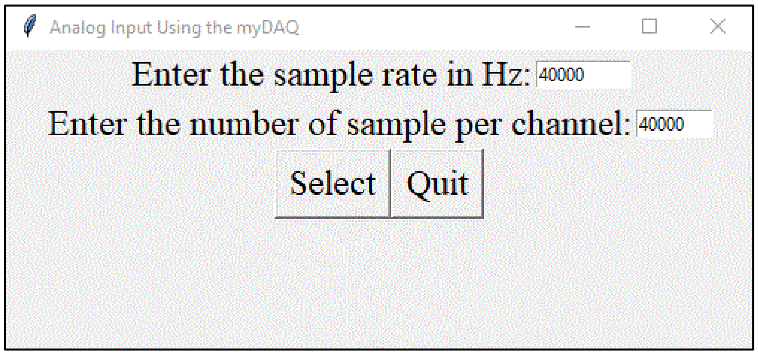
For performing the data acquisition, sinusoidal, square, and triangular waveforms are applied to channel 1 of the myDAQ. Each waveform was generated using an Agilent 33120A Function/Arbitrary Waveform Generator. The frequency was set a 1KHz with a peak voltage of 5V.

Figure 10 shows the user interface window. The sample rate was set at 40k samples/sec and 40k were collected, resulting in a time record of 1 sec. Figures 11-13 show the output windows for each waveform. The time-domain x-axis plots are limited to 10mS to show only 10 cycles of the waveform. The frequency-domain plots show the voltage magnitude in dB.



```
def show_choice():

    # Get the sample rate value entered by the user into the sample_rate_entry widget
    sample_rate = int(sample_rate_entry.get())

    # Get the number of samples per channel value entered by the user into the samples_per_channel_entry widget
    samples_per_channel = int(samples_per_channel_entry.get())

    # Configure NI DAQmx settings
    task = nidaqmx.Task()
    task.ai_channels.add_ai_voltage_chan("myDAQ1/ai1", terminal_config=TerminalConfiguration.DIFF)
    task.timing.cfg_samp_clk_timing(rate=sample_rate, sample_mode=AcquisitionType.FINITE, samps_per_chan=samples_per_channel)

    non_local_var = {'samples': []}

    def callback(task_handle, every_n_samples_event_type,
             number_of_samples, callback_data):
        print('Every N samples callback invoked.')

        samples = task.read(number_of_samples_per_channel=200)  # Read 200 samples
        non_local_var['samples'].extend(samples)

        return 0

    task.register_every_n_samples_acquired_into_buffer_event(
        200, callback)

    task.start()
```

Figure 9. Data Collection Project for Callback Function



Analog Input Using the myDAQ

Enter the sample rate in Hz: 40000

Enter the number of sample per channel: 40000

Figure 10. Data Collection User Interface Window

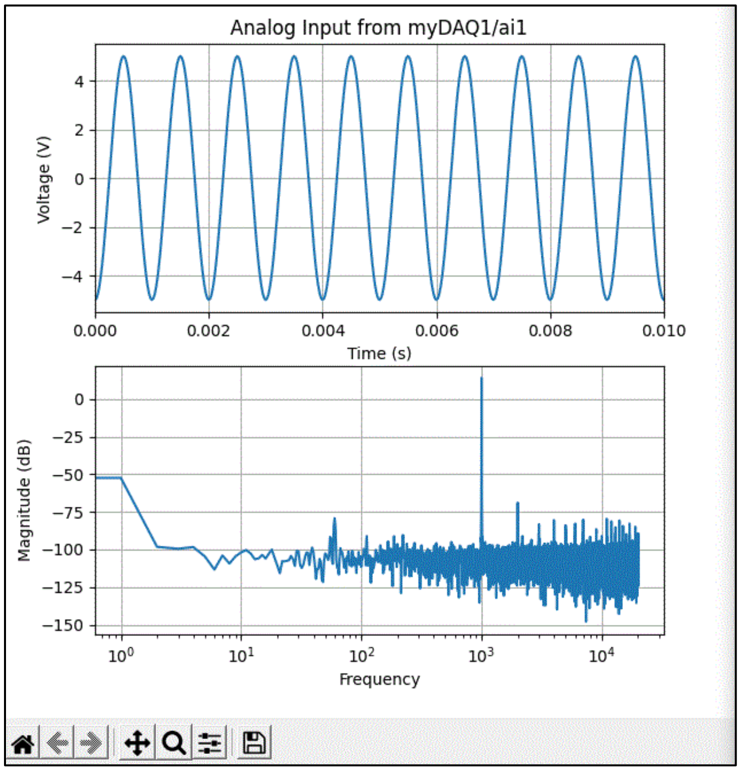


Figure 11. Sinusoidal Waveform Data Collection Output Window

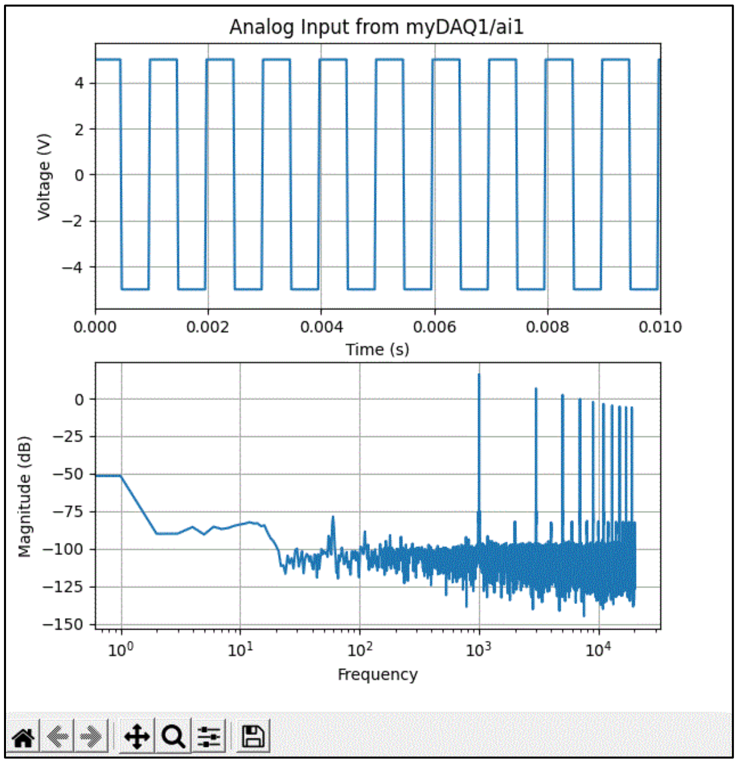


Figure 12. Square Waveform Data Collection Output Window

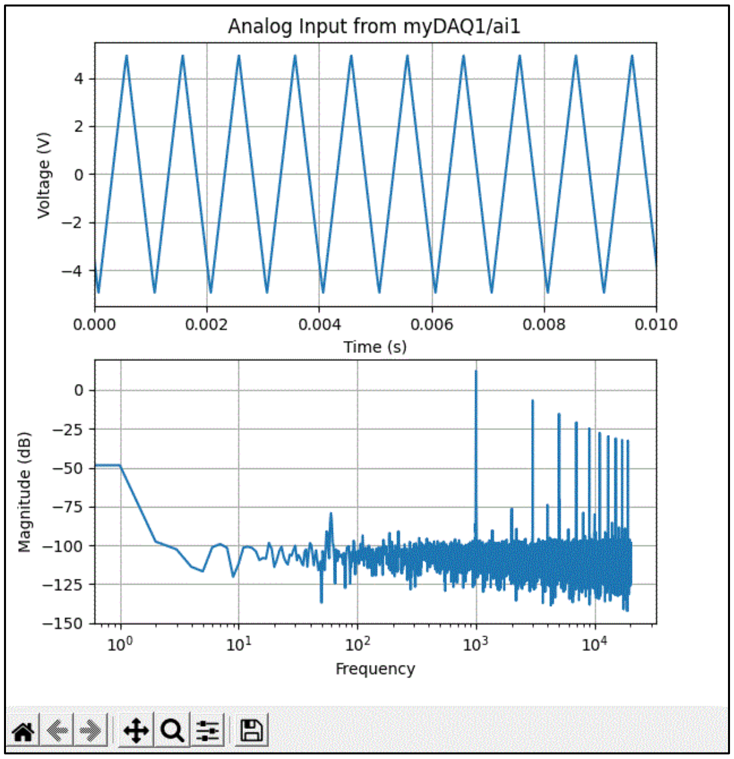


Figure 13. Triangular Waveform Data Collection Output Window

Introduction to GUI Programming Using PySide6 and Qt Designer

The Python *tkinter* package, included with Python installations, offers a quick and simple way to create desktop GUI applications. However, as the demand for more sophisticated GUI design and intuitive visual layout tools with real-time inspection grows, a more advanced framework like Qt for Python becomes a logical choice [12]. The current version, called PySide6, is a Python library for developing GUI applications using the Qt toolkit. PySide6 enables developers to build rich, interactive desktop applications in Python. This is suitable for more sophisticated projects in the upper-level engineering, engineering technology, and computer science courses.

A key component of PySide6 is the Qt Designer, a visual drag-and-drop tool for designing GUIs [13-14]. Qt Designer allows users to create interfaces without writing code, streamlining the design process. Once designed, the interface can be exported as a .ui file and loaded into a PySide6 application, or the .ui file can be compiled into a Python file for integration into the project. The latter method is preferred, as it offers more detailed information about the widget components used and greater flexibility.

Designer is automatically installed when PySide6 is installed via pip. After installation, Qt Designer can be launched from the command line using the built-in launcher within the project's virtual environment [15].

When launched, Figure 14 appears, prompting the selection of a form style.

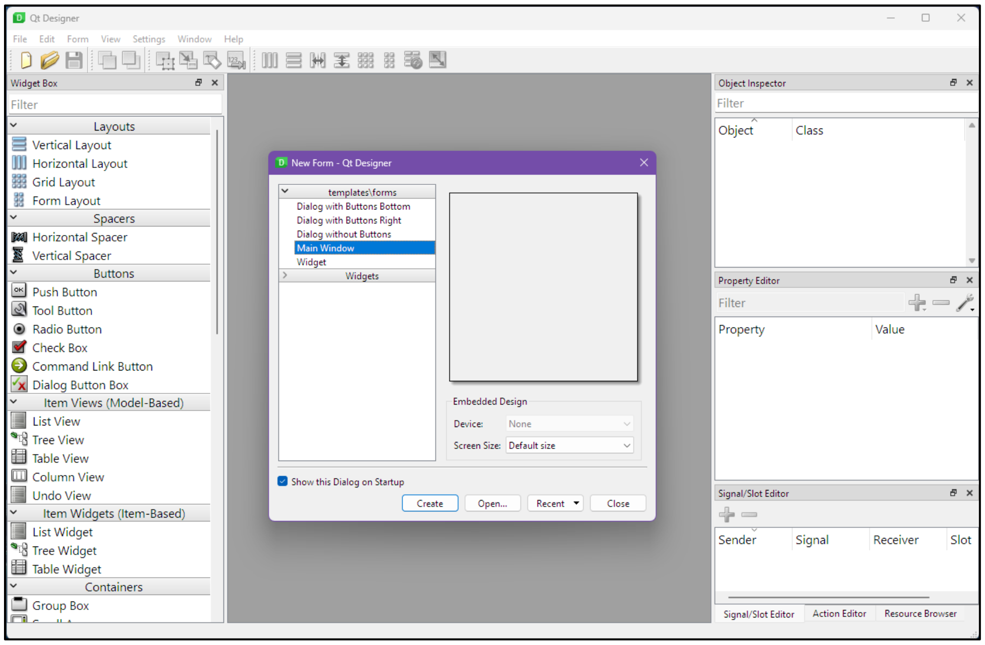
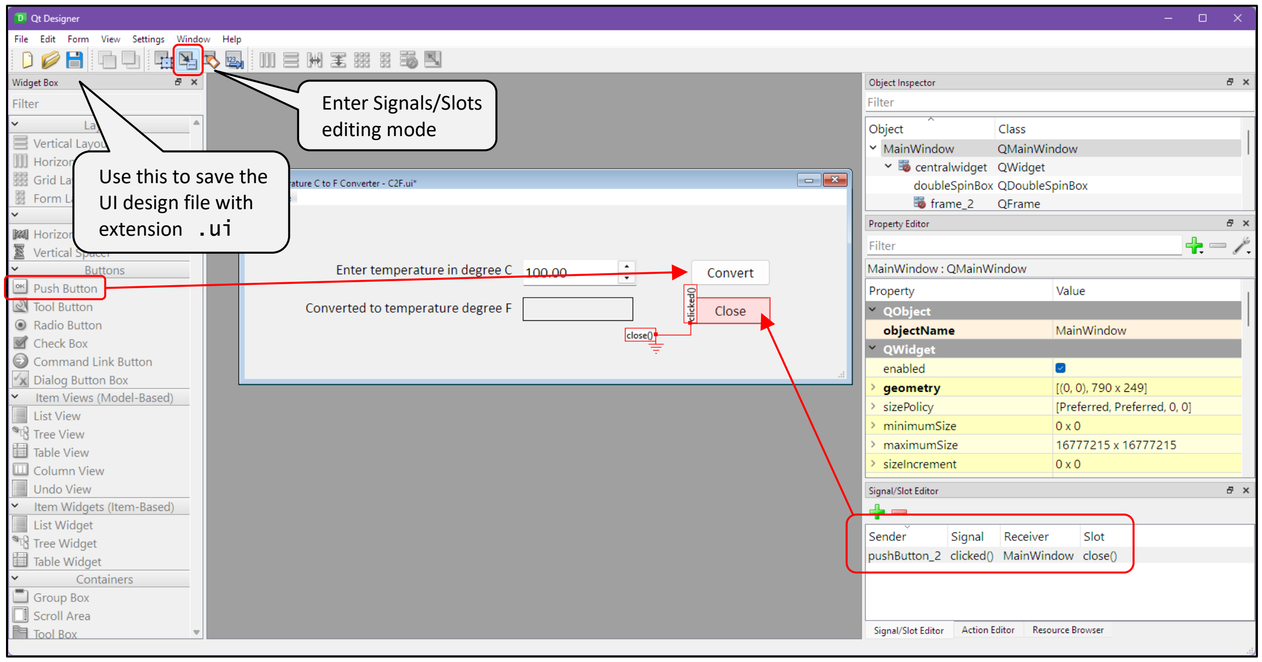


Figure 14. Qt Designer Launch Window

As an example, a simple temperature converter program, shown in Figure 15, is used to illustrate the basic operations of Qt Designer. The Main Window form can be selected and resized by grabbing one of its edges using the mouse to a desired dimension. The components (e.g., buttons, text fields, combo boxes, etc.) from the left pane are placed in the window. The right pane shows the component properties (e.g., window title, programming linked code to a component's event, etc.) that can be changed or manipulated. The 'Close' button has an associated close window event. This can be achieved by selecting the Signals/Slots Editor mode, with no coding required. The designer has many standard event handlers for deployment.



Object	Class
MainWindow	QMainWindow
centralwidget	QWidget
doubleSpinBox	QDoubleSpinBox
frame_2	QFrame

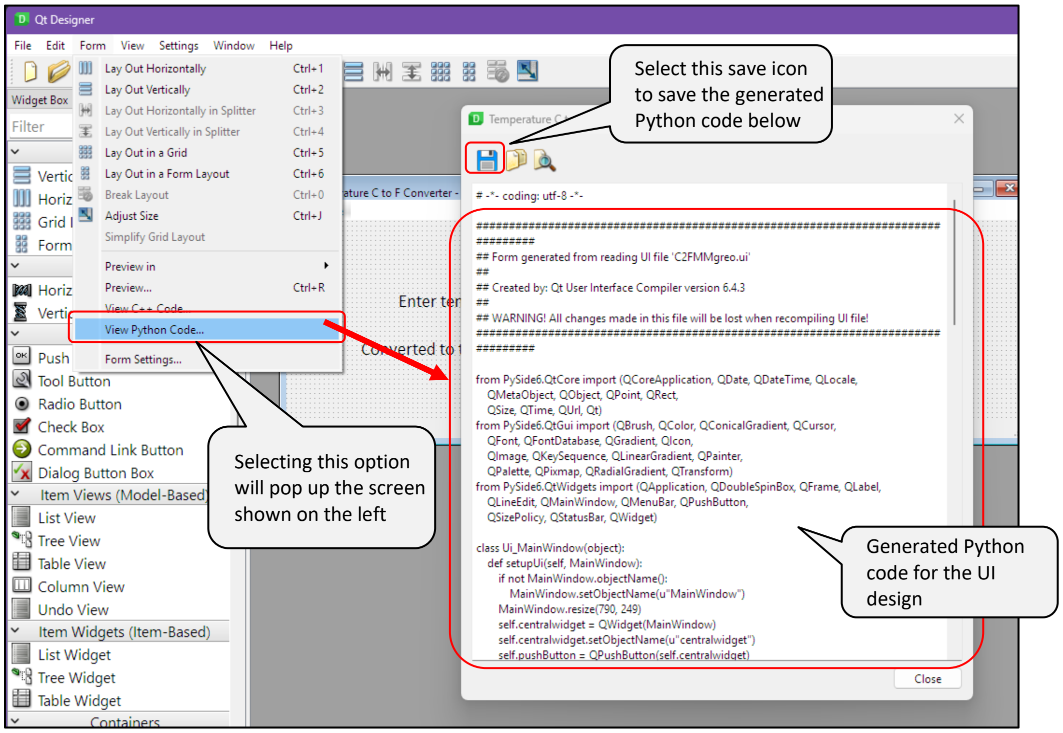
MainWindow : QMainWindow

Property	Value
QObject	
objectName	MainWindow
QWidget	
enabled	✓
geometry	[(0, 0), 790 x 249]
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 x 0
maximumSize	16777215 x 16777215
sizeIncrement	0 x 0

Sender	Signal	Receiver	Slot
pushButton_2	clicked()	MainWindow	close()

Figure 15. Layout of Temperature Converter Program

To preview the window design, shortcut key ctrl-r can be used. The preview can reveal certain behaviors like the numeric entry behaviors of the Celsius entry box, including minimum/maximum values, decimal points, etc. The Python code of this UI design can be exported be selecting Form option → View Python Code. A popup screen showing the Python code generated is shown in Figure 16. The save icon is selected to save the code to the desired project folder to be integrated into the main project code.


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 C++ Code
View Python Code...
Form Settings...
Widget Box
Filter
Push
Tool Button
Radio Button
Check Box
Command Link Button
Dialog Button Box
Item Views (Model-Based)
List View
Tree View
Table View
Column View
Undo View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Containers
Temperature C to F Converter
Enter ter
Converted to
Select this save icon to save the generated Python code below
Selecting this option will pop up the screen shown on the left
Generated Python code for the UI design
Close

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

################################################################################
## Form generated from reading UI file 'C2FMMgreo.ui'
##
## Created by: Qt User Interface Compiler version 6.4.3
##
## WARNING! All changes made in this file will be lost when recompiling UI file!
################################################################################

from PySide6.QtCore import (QCoreApplication, QDate, QDateTime, QLocale,
    QMetaObject, QObject, QPoint, QRect,
    QSize, QTime, QUrl, Qt)
from PySide6.QtGui import (QBrush, QColor, QConicalGradient, QCursor,
    QFont, QFontDatabase, QGradient, QIcon,
    QImage, QKeySequence, QLinearGradient, QPainter,
    QPalette, QPixmap, QRadialGradient, QTransform)
from PySide6.QtWidgets import (QApplication, QDoubleSpinBox, QFrame, QLabel,
    QLineEdit, QMainWindow, QMenuBar, QPushButton,
    QSizePolicy, QStatusBar, QWidget)

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        if not MainWindow.objectName():
            MainWindow.setObjectName(u"MainWindow")
        MainWindow.resize(790, 249)
        self.centralwidget = QWidget(MainWindow)
        self.centralwidget.setObjectName(u"centralwidget")
        self.pushButton = QPushButton(self.centralwidget)
```

Figure 16. Python Code for UI Design

Since the UI Python code will be generated every time there are changers made, the user should not modify this code manually. Finally, this Python file (ui_C2F.py for this example) is integrated with the project main code. The project code for the temperature converter is shown in Figure 17.

```
import sys
from functools import partial     # See reference [16]

from PySide6 import QtWidgets     # PySide6 library
from ui_C2F import Ui_MainWindow  # from generated UI Python code

class MainWindow(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setupUi(self)

def convert_c2f(mw: MainWindow) -> None:
    fahrenheit = round(32.0 + 9.0/5.0 * mw.doubleSpinBox.value(), 2)
    mw.lineEdit.setText(f'{fahrenheit:.2f}')

def show_window(mw: MainWindow) -> None:
    mw.doubleSpinBox.setValue(100)
    mw.pushButton.clicked.connect(partial(convert_c2f, mw))
    mw.show()

if __name__ == '__main__':
    # Create the Qt application and main window
    app = QtWidgets.QApplication(sys.argv)
    mainWindow = MainWindow()
    show_window(mainWindow)
    sys.exit(app.exec())
```

A UI window class to setup the entire window layout via calling **self.setupUi()**

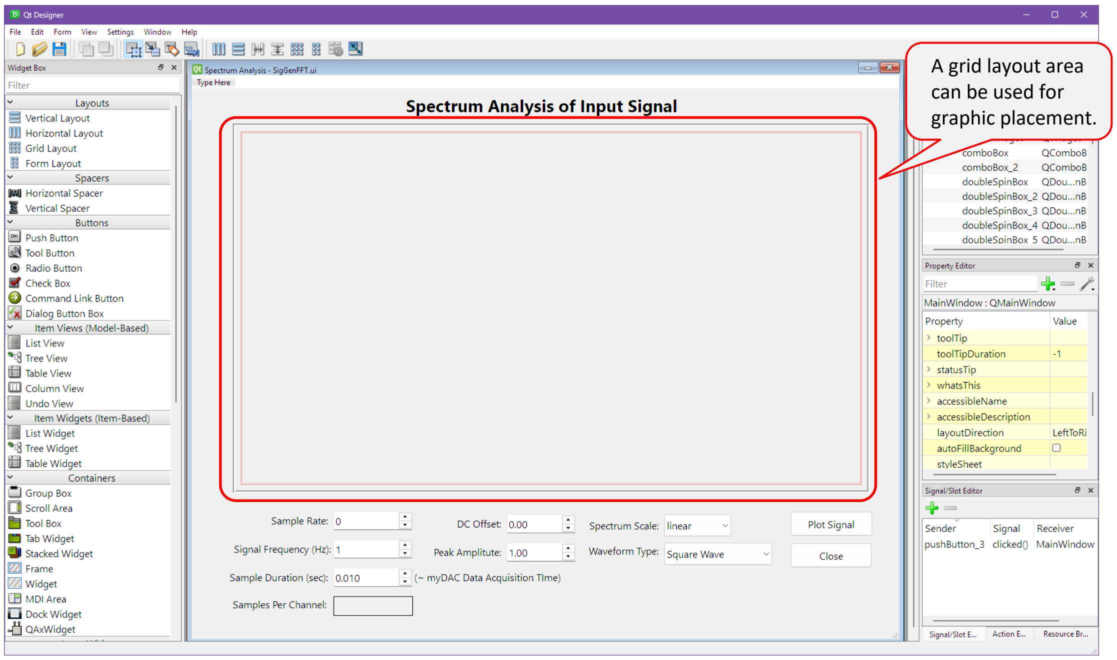
This code is linked to the click event of the 'Convert' button on the UI. The event will invoke the function call to **convert_c2f()[16]**

Standard boiler plate template code for PySide6 integration

Figure 17. Python Code for Temperature Converter Project

Python GUI Project Using PySide6 and Qt Designer

In the following section, a more sophisticated screen layout is presented which provides similar functionality as compared to the previous two *tkinter* GUI projects. A grid layout component for the graphical output simulation of various waveforms (sinusoidal, square, and triangular), and for displaying both the time-domain and frequency-domain representation of these waveforms, is shown in Figure 18.



Sample Rate: 0
Signal Frequency (Hz): 1
Sample Duration (sec): 0.010 (~ myDAC Data Acquisition TIme)
Samples Per Channel:
DC Offset: 0.00
Peak Amplitute: 1.00
Spectrum Scale: linear
Waveform Type: Square Wave

comboBox	QComboB
comboBox_2	QComboB
doubleSpinBox	QDou...nB
doubleSpinBox_2	QDou...nB
doubleSpinBox_3	QDou...nB
doubleSpinBox_4	QDou...nB
doubleSpinBox_5	QDou...nB

MainWindow : QMainWindow

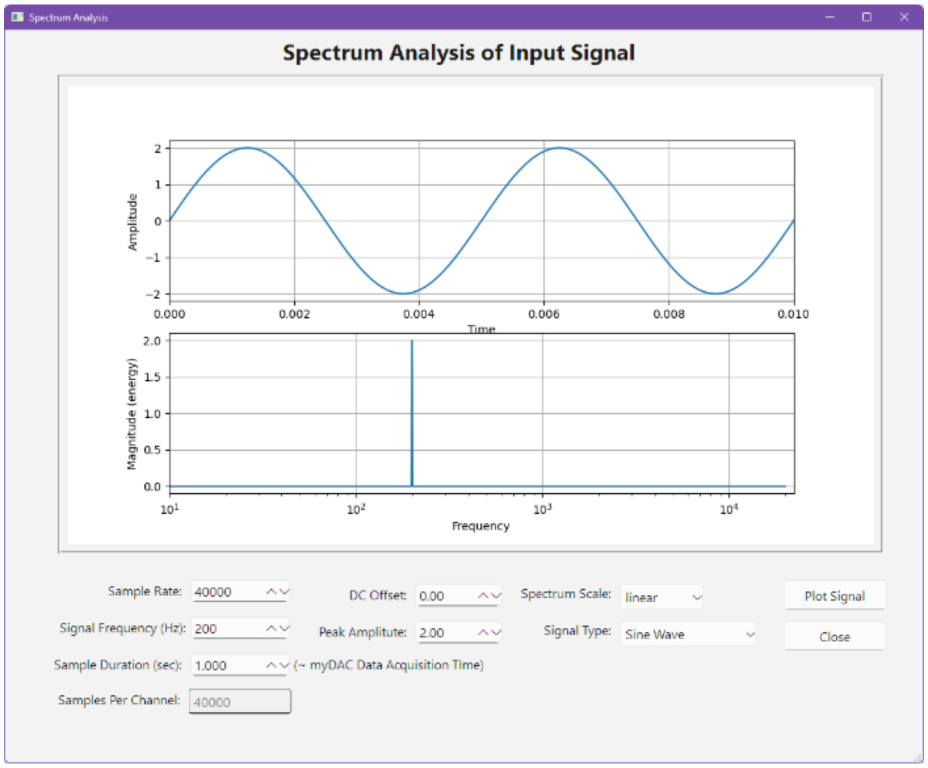
Property	Value
toolTip	
toolTipDuration	-1
statusTip	
whatsThis	
accessibleName	
accessibleDescription	
layoutDirection	LeftToRi
autoFillBackground	☐
styleSheet	

Sender	Signal	Receiver
pushButton_3	clicked()	MainWindow

Figure 18. Grid Layout for Waveform Simulation and Data Acquisition Project

Similar to the previous *tkinter* project, this GUI gives the user the ability to select the signal type (sinusoidal, square, triangular, including the myDAQ for data acquisition), frequency, signal duration, sample rate, peak amplitude, DC offset, and spectrum scale (linear or dB). As software development becomes more sophisticated, students are required to learn object-oriented programming concepts [17]. Since the notion of 'Classes' is permeated in almost all programming languages, Python and PySide6 heavily use such programming constructs.

Figures 19-22 show various waveform outputs for the Waveform Simulation and Data Acquisition Project. Figures 19-21 show the simulated waveform spectrums plotted with a linear scale. Figure 22 shows the sinewave output captured from the myDAQ data acquisition device, and the spectrum is plotted using the dB scale. All waveforms had a 200Hz frequency with a peak voltage of 2V.



Sample Rate: 40000

Signal Frequency (Hz): 200

Sample Duration (sec): 1.000 (~ myDAC Data Acquisition Time)

Samples Per Channel: 40000

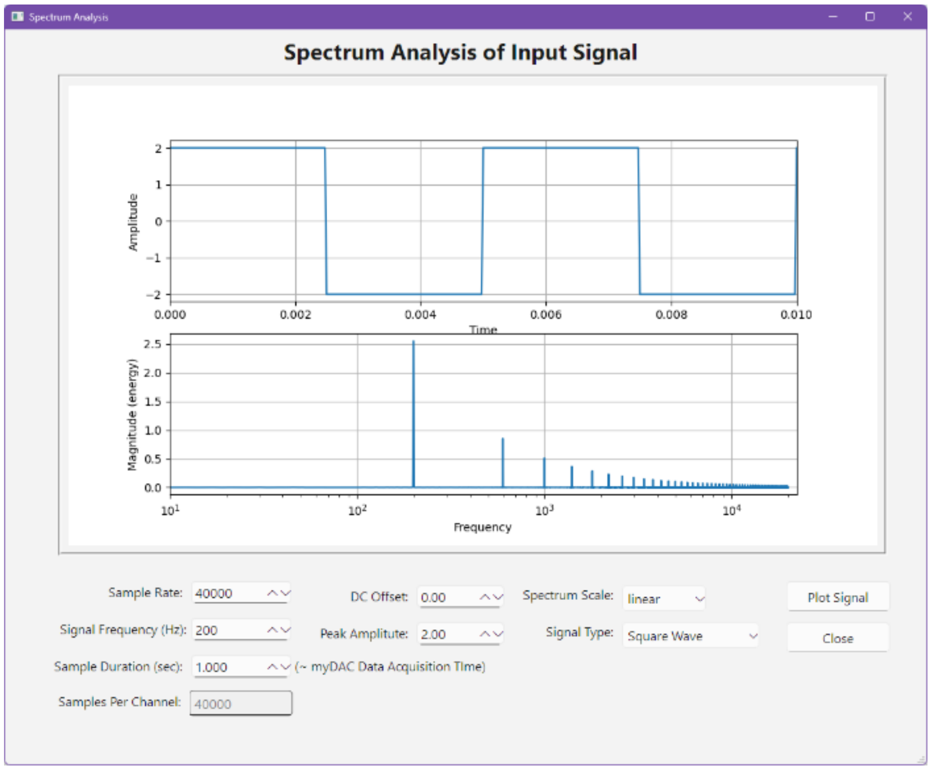
DC Offset: 0.00

Peak Amplitute: 2.00

Spectrum Scale: linear

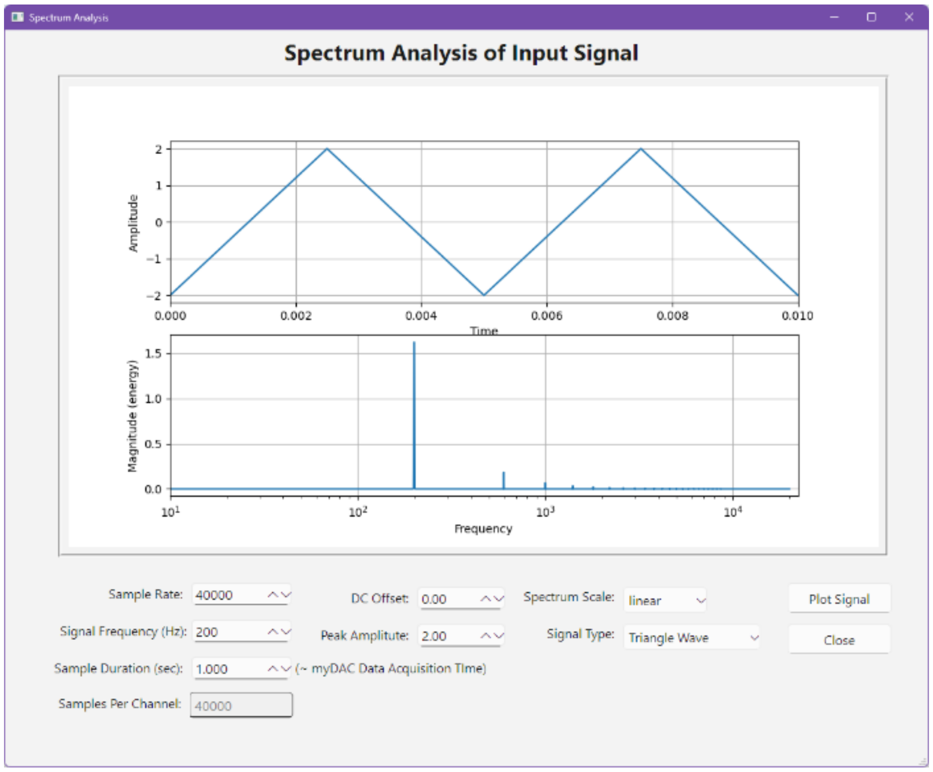
Signal Type: Sine Wave

Figure 19. Sinusoidal Waveform for Waveform Simulation and Data Acquisition Project



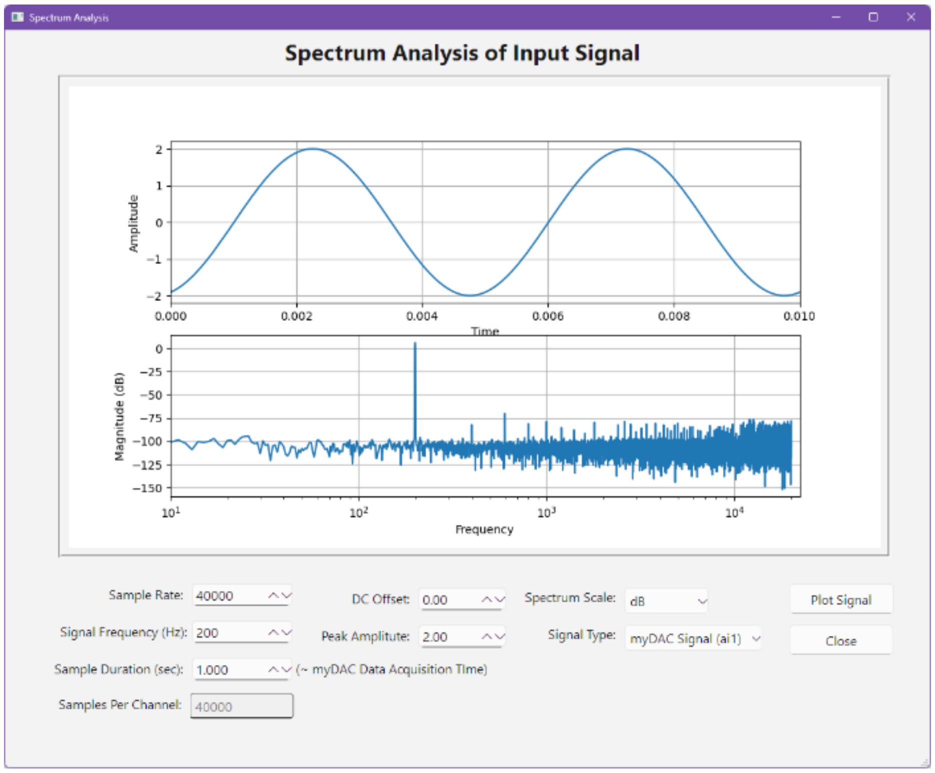
Sample Rate: 40000
DC Offset: 0.00
Spectrum Scale: linear
Signal Frequency (Hz): 200
Peak Amplitute: 2.00
Signal Type: Square Wave
Sample Duration (sec): 1.000 (~ myDAC Data Acquisition Time)
Samples Per Channel: 40000

Figure 20. Square Waveform for Waveform Simulation and Data Acquisition Project



Sample Rate: 40000
DC Offset: 0.00
Spectrum Scale: linear
Signal Frequency (Hz): 200
Peak Amplitute: 2.00
Signal Type: Triangle Wave
Sample Duration (sec): 1.000 (~ myDAC Data Acquisition Time)
Samples Per Channel: 40000

Figure 21. Triangular Waveform for Waveform Simulation and Data Acquisition Project



Sample Rate: 40000
Signal Frequency (Hz): 200
Sample Duration (sec): 1.000 (~ myDAC Data Acquisition Time)
Samples Per Channel: 40000
DC Offset: 0.00
Peak Amplitute: 2.00
Spectrum Scale: dB
Signal Type: myDAC Signal (ai1)

Figure 22. Data Collection for Waveform Simulation and Data Acquisition Project

Student Assessment

Student assessment was performed with an electrical and computer engineering technology senior design project team. The goal of the project was the design of an acoustical data acquisition system that used Python for GUI development. As part of standard classroom practice and instructional improvement efforts, an informal debriefing discussion was conducted with student participants near the conclusion of their senior design project. This project involved extensive use of Python-based GUI development tools and system-level integration of hardware and software components. The purpose of the debriefing was to gather student perspectives on the educational value of the project experience and to identify areas for pedagogical enhancement.

Students reported that the project required significant self-directed learning, including mastering new programming environments (e.g., PySide6), understanding hardware communication protocols (e.g., I2C, SPI), and learning practical engineering skills such as PCB design and soldering without formal coursework in these areas. This learning model emphasized situational problem-solving and adaptability—characteristics highly valued in engineering education.

In addition to technical skill development (e.g., coding, GUI creation, system integration), students cited growth in soft skills such as technical documentation, team communication, meeting management, delegation of tasks, and professional accountability. The emphasis on self-reliance and perseverance, particularly in overcoming hardware-software integration challenges, reinforced the importance of "learning how to learn" - an essential outcome for engineering graduates.

Students also reflected on the value of early exposure to hands-on tinkering, troubleshooting, and independent projects during undergraduate coursework. They recommended that future students cultivate a mindset focused on understanding the "why" behind technical content, rather than only procedural knowledge, to better transfer skills to novel scenarios.

Overall, the senior design project experience provided an authentic, open-ended learning environment that required persistence, critical thinking, and systems-level problem-solving. Insights from this debriefing will inform future enhancements to project scaffolding, support structures, and preparatory coursework to better prepare students for the demands of senior-level capstone projects and professional practice.

Summary

Data acquisition is a common topic in a variety of courses in electrical and computer engineering and engineering technology, software engineering, and computer science programs. With the increasing availability of free and open-source software package availability in Python and the increasing demand for Python skills among graduates, some courses are now incorporating Python for software development in practical applications. Python comes with a module called *tkinter* (short for "Tk interface") that allows users to create simple GUI programs. More sophisticated GUI designs can be created using PySide6 and Qt Designer.

There are two Python GUI projects using *tkinter* shown in this paper. The first project is a Python GUI program for simulating various waveforms (sinusoidal, square, and triangular), determining the spectrum of these signals, and displaying both the time-domain and frequency-domain representation of these waveforms. The second lab project introduces students to Python GUI development for data collection using the myDAQ and plotting the time-domain and frequency-domain representation of the data collected. Then, these programs are developed using PySide6 and Qt Designer.

Preliminary results for developing Python GUI projects are promising. More Python data acquisition projects will need to be developed and compared with using other graphical software development tools (e.g., LabVIEW). Additional student feedback is needed to confirm conclusions.

References

[1] D. Loker and S. Strom, "Innovative Laboratory Projects for a Measurements and Instrumentation Course," *Annual Meeting, American Society for Engineering Education,* 2019.

[2] D. Loker, "Data Acquisition Using the Raspberry Pi Pico W," *Annual Meeting, American Society for Engineering Education,* 2024.

[3] T. Gaddis, *Starting out with Python*, Pearson Education, 2018.

[4] JetBrains.com. [Online]. Available: https://www.jetbrains.com/pycharm/

[5] youtube.com. [Online]. Available: https://www.youtube.com/watch?v=SZUNUB6nz3g

[6] JetBrains.com. [Online]. Available: https://www.jetbrains.com/pycharm/learn/

[7] mvcc.edu. [Online]. Available: https://www2.mvcc.edu/users/faculty/jfiore/CP/labs/LaboratoryManualForComputerProgramming.pdf

[8] D. Loker, "MicroPython in a Wireless Communications Systems Course," *Annual Meeting, American Society for Engineering Education,* 2021.

[9] matplotlib.org. [Online]. Available: https://matplotlib.org/

[10] W. Tomasi, *Electronic Communications Systems, Fundamentals Through Advanced*, 5th ed., Pearson Education, 2004.

[11] GitHub.com. [Online]. Available: https://github.com/ni/nidaqmx-python/tree/master

[12] Qt.io. [Online]. Available: https://www.qt.io/qt-for-python

[13] pythonguis.com. [Online]. Available: https://www.pythonguis.com/pyside6-tutorial/

[14] youtube.com. [Online]. Available: https://www.youtube.com/watch?v=uzqDnB44qf4

[15] realpython.com. [Online]. Available: https://realpython.com/python-virtual-environments-a-primer/ -

[16] geeksforgeeks.com. [Online]. Available: https://www.geeksforgeeks.org/partial-functions-python/

[17] Irv Kalb, *Object-Oriented Python: Master OOP by Building Games and GUIs*, No Starch Press, 2022.