Name: Evan Leglar	**Lab Time** T 12:00 PM

Names of people you worked with:
• n/a

Websites you used:
• https://www.learnpyqt.com/courses/start/basic-widgets/

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	7

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

Evan Leglar

Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.

Turn Python code to Gradescope or your assignment will not be graded

BEFORE YOU BEGIN

1) Install PyQt5 using conda
2) Grab gui,py, slider.py, and random-graph.py from files
    a. Check that they all run
3) Take a look at the code, and see if you can figure out what the various pieces of it do. For the remainder of this lab, you should build on this code, since it contains all of the boilerplate code that you'll need.

Learning Objectives:

You should be able to do the following:

- Create a simple GUI using PyQt5
- Know how to set callbacks
- Know how the event manager communicates with your code

Thoughts (come back and read these after you've read the problems):

1. When you create a `SliderDisplay`, you should be able to specify a name ('foo' in the example), low value, a high value, and an optional argument saying how many ticks (individual values) you want to slider to have. The slider should display values to three decimal places. As an example, if you have a low of 0, a high of 1, and 1,000 ticks, the smallest increment the slider will register is 0.001.

2. `QSliders` only deal with integers. You're going to have to do some simple math to make them work with floating point numbers, like in the example.

3. Your interface doesn't have to look exactly like this. It should have the same number of sliders, labeled in the same way, and the same buttons, but the layout and details can differ. We're really looking for the same functionality, rather than the exact same look.

4. For the second interface, you should use your `SliderDisplay` class. Make the first three sliders range from 0 to 10, time from 0 to 100, and time step from 0.001 to 0.1. You should probably wrap up your code in a class, to help keep things encapsulated.

Grading Checkpoints

1. Successfully added a label. [1 point]
2. Successfully added a slider (1 point) with the correct bounds (1 point), that changes the value of the label (1 point). [3 points]
3. SliderDisplay displays a slider and a label (1 point), in a horizontal layout (1 point), and works as expected (1 point). [3 points]
4. Second GUI has the correct number of working sliders, using `SliderDisplay` (1 point), a Simulate button that works (1 point), and a quit button that works (1 point). [3 points]
5. **Extra Credit:** Adds a graphing area to the interface (1 point), Produces Correct plot (3 points). Displays system type (1 point). [5 points]

Remember to hand in the the python files and this file (as a PDF) to Gradescope.

Problem 1 Edit gui.py

1. Starting with gui.py:
    1. Add a label, above the quit button, and have it initially display the value zero.
    2. Add a horizontal slider, with a range of 0 to 10, and have it change the value of the label.

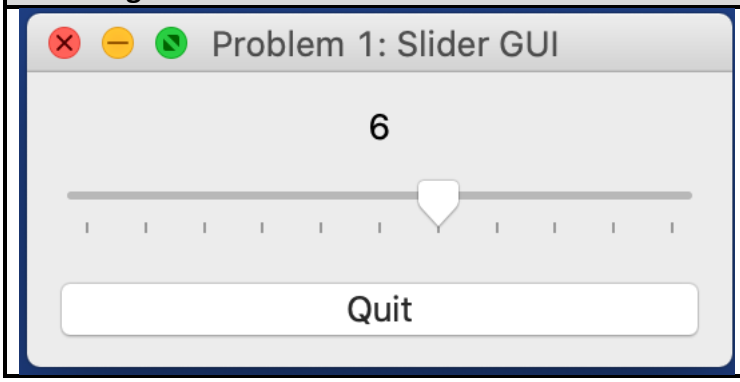
Comments for grader/additional information (if any)

gui.py

```
#!/usr/bin/env python3

from PyQt5.QtGui import *
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget,
QPushButton, QVBoxLayout, QLabel, QSizePolicy, QSlider
from PyQt5.QtCore import *
import slider


class Interface(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        self.setWindowTitle('Problem 1: Slider GUI')

        # A widget to hold everything
        widget = QWidget()
        self.setCentralWidget(widget)

        # A layout
        layout = QVBoxLayout()
        widget.setLayout(layout)

        # Problem 1 Label
        self.label_slider = QLabel()
        self.label_slider.setText('0')
        self.label_slider.setSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Expanding)
        self.label_slider.setAlignment(Qt.AlignCenter)

        # A button
        quit_button = QPushButton('Quit')
        quit_button.clicked.connect(app.exit)

        # Problem 1 Slider
        self.slide_button = QSlider(Qt.Horizontal)
        self.slide_button.setTickPosition(QSlider.TicksBelow)
        self.slide_button.setMinimum(0)
        self.slide_button.setMaximum(10)
        self.slide_button.setTickInterval(1)
```

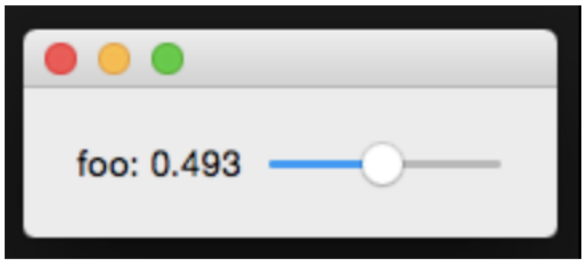
```
        self.slide_button.valueChanged.connect(self.value)

        # Widget Layouts
        layout.addWidget(self.label_slider)
        layout.addWidget(self.slide_button)
        layout.addWidget(quit_button)

        # Add other widgets to the layout here.  Possibly other
layouts.
    def value(self):
        """Return the current value of the slider"""
        slider_value = self.slide_button.value()
        self.label_slider.setText(str(slider_value))


if __name__ == '__main__':
    app = QApplication([])

    interface = Interface()

    interface.show()

    app.exec_()
```

GUI image



Problem 2 Slider

1. Edit `slider.py`. This file has a class, called `SliderDisplay`. Edit this file to encapsulate setting the min, max, and starting values and have it include a label that displays the value. When you're done, you should get something that looks like this:



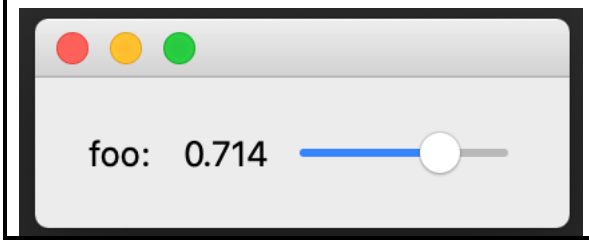
Comments for grader/additional information (if any)

slider.py

```
#!/bin/usr/env python3

from PyQt5.QtWidgets import QApplication, QWidget, QSlider, QLabel,
QHBoxLayout
from PyQt5.QtCore import *


class SliderDisplay(QWidget):
    def __init__(self, name, low, high, ticks=1000):
        QWidget.__init__(self)
        layout = QHBoxLayout()
        self.setLayout(layout)
        self.ticks = ticks

        # Slider
        self.slider = QSlider(Qt.Horizontal)
        self.slider.setMinimum(low * self.ticks)
        self.slider.setMaximum(high * self.ticks)
        self.slider.valueChanged.connect(self.value)

        # Slider Label
        self.label_slider = QLabel()
        self.label_slider.setText(f'{name}:')

        # Slider Value
        self.slider_value = QLabel()
        self.slider_value.setText('0.000')

        # Widget Layouts
        layout.addWidget(self.label_slider)
        layout.addWidget(self.slider_value)
        layout.addWidget(self.slider)
```

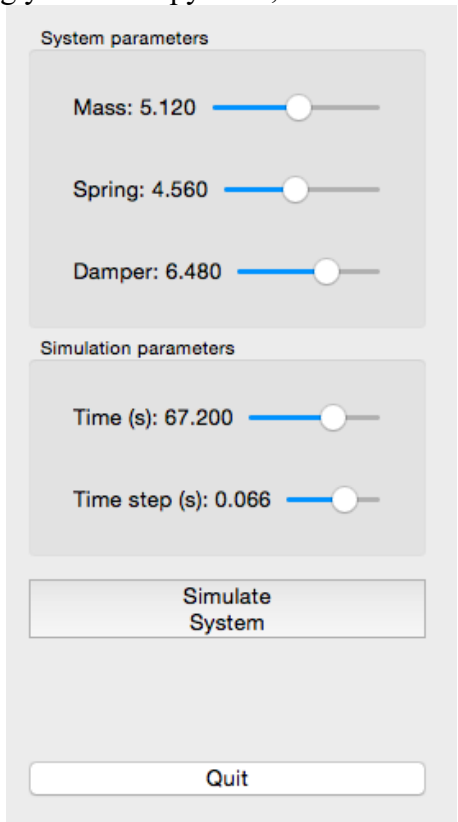
```
    def value(self):
        """Return the current value of the slider"""
        slider_value = self.slider.value()/ self.ticks
        self.slider_value.setText('{0: .3f}'.format(slider_value))


if __name__ == '__main__':
    app = QApplication([])

    slider = SliderDisplay('foo', 0, 1)

    slider.show()

    app.exec_()
```

Gui image



Problem 3 lots of sliders

1. Make a new file, gui2.py
    1. Copy gui.py into it
2. Using your slider.py code, create an interface that looks like this:



3. The Quit button should end the application, and the Simulate System button should print out the values of Mass, Spring, Damper, Time, and Time Step to the console (ie using the print function).
    1. Remember, you don't have to make it look **exactly** the same, just make sure you have the right number of sliders

Comments for grader/additional information (if any)

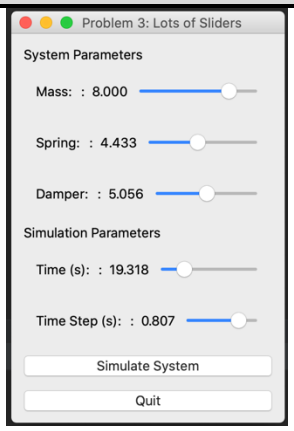
Gui2.py

```
#!/usr/bin/env python3


from PyQt5.QtGui import *
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget,
QPushButton, QVBoxLayout, QLabel, QSizePolicy
from PyQt5.QtCore import *
```

```
from slider import SliderDisplay


class Interface(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        self.setWindowTitle('Problem 3: Lots of Sliders')

        # A widget to hold everything
        widget = QWidget()
        self.setCentralWidget(widget)

        # A layout
        layout = QVBoxLayout()
        widget.setLayout(layout)

        # System Parameter Sliders
        self.sys_params = QLabel('System Parameters')
        self.mass = SliderDisplay("Mass", 0, 10)
        self.spring = SliderDisplay("Spring", 0, 10)
        self.damper = SliderDisplay("Damper", 0, 10)

        # Simulation Parameter Sliders
        self.sim_params = QLabel('Simulation Parameters')
        self.Time = SliderDisplay("Time (s)", 0, 100)
        self.step = SliderDisplay("Time Step (s)", 0, 1)

        # Interface buttons
        simulate_button = QPushButton('Simulate System')
        simulate_button.clicked.connect(self.simulate)
        quit_button = QPushButton('Quit')
        quit_button.clicked.connect(app.exit)

        # Widget Layouts
        layout.addWidget(self.sys_params)
        layout.addWidget(self.mass)
        layout.addWidget(self.spring)
        layout.addWidget(self.damper)
        layout.addWidget(self.sim_params)
        layout.addWidget(self.Time)
        layout.addWidget(self.step)
        layout.addWidget(simulate_button)
        layout.addWidget(quit_button)

    def simulate(self):
        print("Mass: ", self.mass.value)
        print("Spring: ", self.spring.value)
        print("Damper: ", self.damper.value)
        print("Time: ", self.Time.value)
        print("Time Step: ", self.step.value)

    # def value(self):



if __name__ == '__main__':
```

```
    app = QApplication([])

    interface = Interface()

    interface.show()

    app.exec_()
```

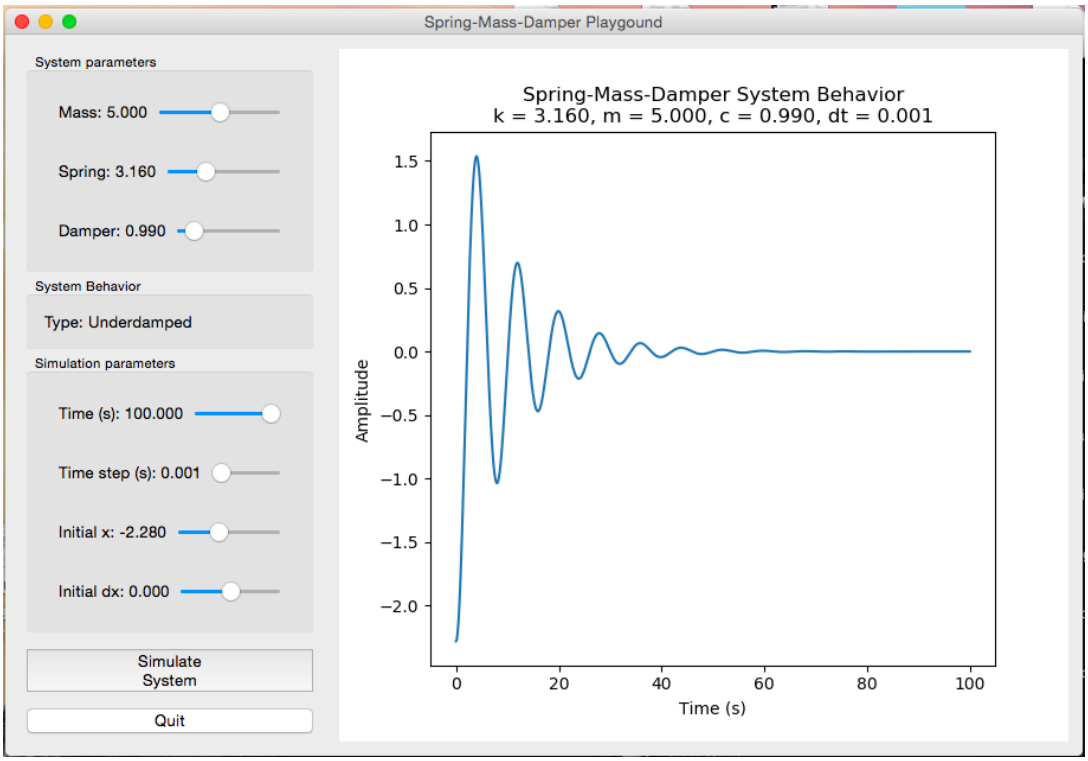
Image



Console

```
/usr/local/bin/python3.7
/Users/evanleglar/Desktop/ME499_Python/Lab_8/gui2.py
Mass:  8.0
Spring:  4.433
Damper:  5.056
Time:  19.318
Time Step:  0.807

Process finished with exit code 0
```

Problem 4 Extra credit



1. Add a graph of the actual system behavior to the GUI, using the spring-mass-damper simulation code from Lab 4. When you set the parameters for the system and the simulation in the GUI, then press the "Simulate System" button, it should draw the correct graph.
    1. Values for the simulator should come from your sliders
    2. Type is calculated as sliders change
    3. Clicking Simulate system should
        1. Change the title to reflect the current values of k, m, c, and dt
        2. Change the plot appropriately.
        3. Re-calculate the type of the system and set the label appropriately
2. Hint: Look at random-graph.py

Comments for grader/additional information (if any)

Gui2.py
`Copy your code here`

Image
% Put an image of your gui with a graph here