Chapter 9: Python GUI

Contents

Red light: No AI

Time required: 120 minutes

DRY

Don't **R**epeat **Y**ourself

Tutorials

- https://realpython.com/python-gui-tkinter/

Python GUI - Tkinter

Until now, our Python programs have interacted with users through the keyboard using the **input()** function. This type of interaction is simple and follows a fixed sequence. The program decides when to ask for input and what happens next.

Many programs you use—like web browsers, games, and apps—use windows, buttons, text boxes, and menus. These are all part of what's called a **Graphical User Interface**, or **GUI**.

What Is a GUI?

A GUI (Graphical User Interface) lets the user interact with the program using visual components called **widgets**—such as buttons, labels, and text entries. Unlike text-based programs, GUI programs are **event-driven**. This means the user controls the program flow by clicking, typing, or selecting something, and the program responds to those actions.

Tkinter: Python's Built-in GUI Module

Python includes a built-in module for creating GUI applications called **tkinter** (all lowercase). To use it, you just need to import it at the top of your program:

```
from tkinter import *
```

There are two common ways to write tkinter programs:

- Using only functions
- Using **object-oriented programming** (OOP) with classes

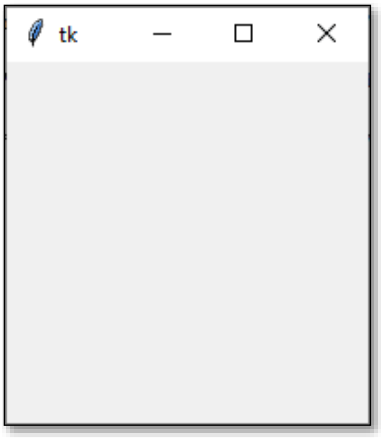
In this chapter, we will use the OOP approach. It helps us organize code better, especially as our programs grow larger. Each GUI widget (like a button or label) becomes easier to manage, and we avoid using global variables.

Tutorial 9.1: Hello, Tkinter!

Let's start with the most basic example—a window with nothing in it. No OOP at the beginning.

```python
# Import the tkinter library
from tkinter import *

# Create the main window
root = Tk()

# Start the event loop
root.mainloop()
```

Example run:



Not very exciting yet—but it's a start!

Let’s modify it to show a label that says “Hello, World!”

```python
from tkinter import *

root = Tk()
root.geometry("300x100")  # Set the window size

label = Label(root, text="Hello, World!")
label.pack()

root.mainloop()
```

Grid Layout Manager

Most GUIs are arranged in a **grid** of rows and columns. Tkinter’s grid() method places widgets in this layout.

(row=0, column=0)	(row=0, column=1)	(row=0, column=2)
(row=1, column=0)	(row=1, column=1)	(row=1, column=2)
(row=2, column=0)	(row=2, column=1)	(row=2, column=2)

```
from tkinter import *

# Create the main window
root = Tk()
root.title("Simple Grid Example")

# Create and place buttons using grid
btn1 = Button(root, text="Button 1")
btn1.grid(row=0, column=0)

btn2 = Button(root, text="Button 2")
btn2.grid(row=0, column=1)

btn3 = Button(root, text="Button 3")
btn3.grid(row=1, column=0)

# Start the GUI event loop
root.mainloop()
```

Layout Explanation

- **Button 1** is in **row 0, column 0**
- **Button 2** is in **row 0, column 1**
- **Button 3** is in **row 1, column 0**

Example run:

Entry Boxes

An **Entry box** is a widget that lets the user type text into your program. It's useful for getting input like a name, age, or other data.

Creating an Entry Box

Here is a simple example that creates an Entry box and places it in a window using the grid() layout:

```
from tkinter import *

root = Tk()

entry = Entry()
entry.grid(row=0, column=0)

root.mainloop()
```

Syntax

```
user_input = Entry(root, option=value, ... )
```

Getting Text from an Entry Box

Use the .get() method to read the text entered by the user:

```
text_value = entry.get()        # Returns a string
```

If you expect a number, you can convert it using int() or float():

```
number = int(entry.get())       # Converts to integer
```

Clearing the Entry Box

To delete all the text in the entry box, use:

```
entry.delete(0, END)
```

- **0**: Start at the first character
- **END**: Go to the end

Inserting text into the Entry Box

To insert text into an entry box:

```
entry.insert(0, 'hello')
```

- 0: Insert at the beginning

Buttons and the command Argument

The **Button** widget in Tkinter is used to create clickable buttons. Buttons can show text, images, or both. More importantly, buttons can perform an action when the user clicks on them.

You do this by attaching a **function** to the button using the command option. This function is called a **callback**—because Tkinter *calls back* the function when the button is clicked.

Syntax:

```
button = Button(master, text="Click Me", command=function_name)
```

- **root**: The parent widget (usually the main window)
- **text**: The label on the button
- **command**: The function to call when the button is clicked

Important:
Do **not** use parentheses after the function name when assigning it to command.

Correct: command=my_function
Incorrect: command=my_function()

Example Using Entry Input with a Button

```
from tkinter import *

def show_input():
    name = entry.get()
    display_label.config(text="You entered: " + name)

root = Tk()
root.title("Entry and Label Example")

# Entry widget
entry = Entry(root, width=30)
entry.grid(row=0, column=0, padx=10, pady=10)

# Button that updates the label
btn = Button(root, text="Submit", command=show_input)
btn.grid(row=0, column=1, padx=10, pady=10)

# Label to display output
display_label = Label(root, text="")
display_label.grid(row=1, column=0, columnspan=2, pady=10)

root.mainloop()
```

Example run:

Assignment 9.1: Contact Form UI

Objective:

Create a Tkinter application that accepts user input (name and email), displays the entered information, and includes a "Clear" button to reset the form.

Concepts Covered:

- Tkinter window and widgets
- Entry widgets for user input

- Labels to display messages
- Buttons with command
- grid() layout
- .get(), .delete(), .insert(), and .config()

Starter Code:

```
from tkinter import *

root = Tk()
root.title("Contact Form")
root.geometry("300x200")  # Set window size

# TODO: Step 2 - Create Labels and Entry widgets

# TODO: Step 3 - Create Submit and Clear buttons


# TODO: Step 4 - Create a label to display the output

# TODO: Step 5 - Define the submit_function
# This function should:
#  - Get text from entries
#  - Update the display label with the input

# TODO: Step 6 - Define the clear_function
# This function should:
#  - Clear the entry fields
#  - Reset the display label to blank

# Start the main event loop
root.mainloop()
```

Step 2: Add Labels and Entry Boxes

Add two **labels** and **entry boxes** side by side using grid():

Label	**Entry**
Name:	[_____]

Email:	[_____]

Step 2: Add Submit and Clear Buttons

Add two buttons below the entry boxes:

- **Submit**: When clicked, it reads the name and email and displays them in a label below the buttons.
- **Clear**: When clicked, it clears both entry boxes and the display label.

Place buttons in the third row.

Step 3: Add a Display Label

Below the buttons, add a label to show the submitted information.

Place this label in row=3, and make it span both columns using columnspan

Step 4: Add Functionality

- Use **.get()** to retrieve values from the entries
- Use **.config()** to update the display label
- Use **.delete(0, END)** to clear the entries
- Test both buttons

Example run:

Tutorial 9.2: Square Calculator

Time for OOP. This will be the coding method we will use for all Tkinter programs from this point forward.

We are going to create a Tkinter GUI program that calculates the square of the number entered by the user.

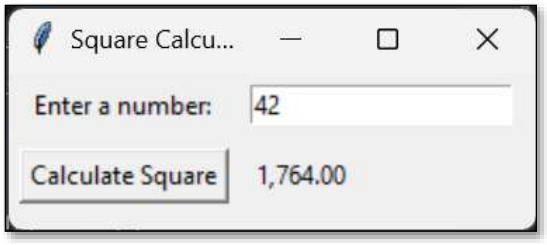
```
"""
Name: square_calculator.py
Author:
Created:
Purpose: Calculate the square of a number entered by the user
"""

from tkinter import *
```

```
class SquareCalculator:
    def __init__(self):
        # Create the main tkinter window
        self.root = Tk()
        # Set the size of the window
        self.root.geometry("250x75")
        # Set the title of the window
        self.root.title("Square Calculator")
        # Call the method to create widgets
        self.create_widgets()
        # Start the tkinter event loop
        self.root.mainloop()
```

```python
    # ------------------------ CREATE WIDGETS ---------------------------- #
    def create_widgets(self):
        # Create a label to instruct the user to enter a number
        self.lbl_instruction = Label(self.root, text="Enter a number:")
        self.lbl_instruction.grid(row=0, column=0)

        # Create an entry widget for the user to input a number
        self.entry_number = Entry(self.root)
        self.entry_number.grid(row=0, column=1)

        # Create a button to calculate the square of the entered number
        self.btn_calculate = Button(
            self.root, text="Calculate Square", command=self.calculate_square
        )
        self.btn_calculate.grid(row=1, column=0)

        # Create a label to display the result of the calculation
        self.lbl_result = Label(self.root, text="", anchor="w")
        self.lbl_result.grid(row=1, column=1, sticky="w")

        # Set padding for all widgets inside the frame
        for widget in self.root.winfo_children():
            widget.grid_configure(padx=5, pady=5)
```

```python
    # ------------------------ CALCULATE SQUARE -------------------------- #
    def calculate_square(self):
        # Try to get the input, calculate the square, and display the result
        try:
            # Retrieve the number entered by the user
            number = float(self.entry_number.get())
            # Calculate the square of the number
            square = number**2
            # Update the result label with the calculated square
            self.lbl_result.config(text=f"{square:,.2f}")
        except ValueError:
            # If the input is invalid, display an error message
            self.lbl_result.config(text="Please enter a valid number!")


# Create an instance of the SquareCalculator class to run the application
square_calculator = SquareCalculator()
```

Example run:



ttk Themed Widgets (Improved Tkinter Widgets)

The **tkinter.ttk** module provides **themed widgets**, which are modern, platform-native versions of Tkinter's standard widgets. The name **ttk** stands for **Tk Themed Widgets**.

These widgets improve the **look and feel** of your GUI to better match the operating system (Windows, macOS, Linux).

How to Use ttk

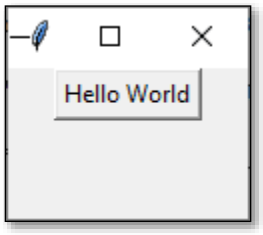
To use ttk widgets in your program, **import them after importing Tkinter**

```
from tkinter import *
from tkinter.ttk import *
```

This will override the default Tkinter widgets with their **themed** versions. For example, **ttk.Button** replaces the older **tk.Button**.

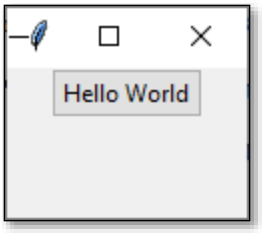
Using the **ttk** widgets give the application an improved look and feel.

tk code for Hello World:




```
from tkinter import *
root = Tk()
btn_hello = Button(root, text="Hello World").grid()
mainloop()
```

ttk code for Hello World:




```
from tkinter import *
from tkinter.ttk import *
root = Tk()
btn_hello = Button(root, text="Hello World").grid()
mainloop()
```

Styling Note

Themed widgets use a style system. You style them with the Style class:

```
style = Style()
style.configure("TButton", foreground="blue", font=("Arial", 12))
```

Styling ttk widgets is more flexible but slightly more complex.

Grid Options

You can control spacing and alignment using additional options with grid():

Spacing

- padx: Space **outside** the widget (left and right)
- pady: Space **outside** the widget (top and bottom)
- ipadx: Space **inside** the widget (horizontal padding)
- ipady: Space **inside** the widget (vertical padding)

```
display_label = Label(root, text="")
display_label.grid(row=3, column=0, columnspan=2, pady=10)
```

Spanning Rows and Columns

Use **rowspan** or **columnspan** to let a widget span across multiple cells.

```
self.output_label = Label()
self.output_label.grid(row=1, column=0, columnspan=3)
```

Tutorial 9.3: Temperature Converter

Let's create a tkinter ttk version of our old friend, the temperature converter.

If you are programming in Windows → find a free thermometer icon. The extension for Windows icon files is ico.

NOTE: If you have a Mac or Linux, do not try to add ico files. They are Windows only.

Go to IconArchive for free ico files.

If you wish to put in a degree symbol,:

1. https://www.degreesymbol.net/
2. Click the Copy button.
3. Paste in your code. °

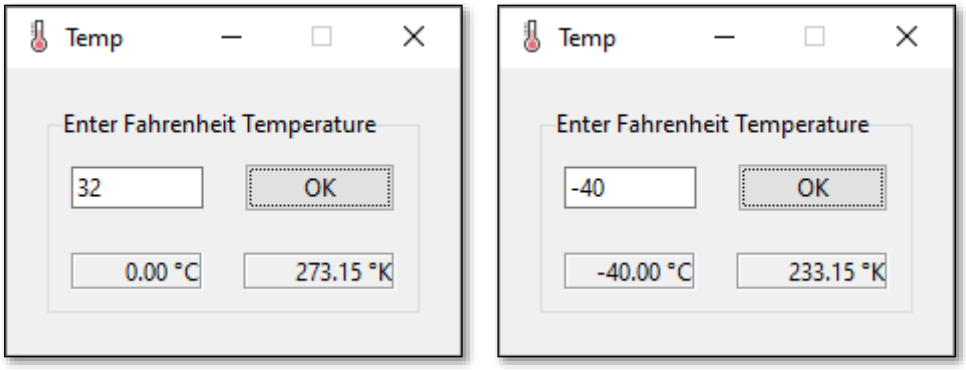
Create a Python program named: **temperature_converter_gui.py**

```python
"""
    Name: temperature_converter_gui.py
    Author:
    Created:
    Purpose: Convert Fahrenheit to Celsius and Kelvin
"""
# Import the tkinter module with tk standard widgets
from tkinter import *
# Override tk widgets with themed ttk widgets if available
from tkinter.ttk import *


class TemperatureConverter:
    def __init__(self):
        # Create the root window
        self.root = Tk()
        self.root.title("Temp")
        # Add icon to window corner
        # Search the web for free thermometer.ico files
        self.root.iconbitmap("thermometer.ico")
        # Prevent window from resizing
        self.root.resizable(False, False)
        # Create the GUI widgets in a separate method
        self.create_widgets()
        # Call the mainloop method to start program
        mainloop()

# ---------------------- CONVERT TEMPERATURE ----------------------------#
    def convert_temperature(self, *args):
        """Method called by the button click event
           Convert fahrenheit to celsius and kelvin.
        """
        # Get input from user as a float
        self.fahrenheit = float(self.entry.get())

        # Convert Fahrenheit to Celsius and Kelvin
        self.celsius = ((self.fahrenheit - 32) * 5.0) / 9.0
        self.kelvin = (((self.fahrenheit - 32) * 5.0) / 9.0) + 273.15

        # Display output in labels using the configure method
        self.lbl_celsius.configure(text=f"{self.celsius:,.2f} °C")
        self.lbl_kelvin.configure(text=f"{self.kelvin:,.2f} °K")

        # 0 starts the selection at the beginning of the entry widget text
        # END finishes the selection at the end of the entry widget text
        self.entry.selection_range(0, END)
```

```python
    # ---------------------- CREATE WIDGETS ---------------------------------#
    def create_widgets(self):
        """Create and grid widgets."""
        # Create main label frame to hold widgets
        self.main_frame = LabelFrame(
            self.root,                          # Assign to parent window
            text="Enter Fahrenheit Temperature",  # Text for the frame
            relief=GROOVE                       # Decorative border
        )

        # Create entry widget in the frame to get input from user
        self.entry = Entry(
            self.main_frame,  # Assign to parent frame
            width=10          # Width in characters
        )

        # Create button in the frame to call calulate method
        self.btn_calculate = Button(
            self.main_frame,     # Assign to parent frame
            text="OK",           # Text shown on button
            # Connect convert method to button click
            command=self.convert_temperature
        )

        # Create label in the frame to show celsius
        self.lbl_celsius = Label(
            self.main_frame,     # Assign to parent frame
            width=10,            # Width in characters
            relief=GROOVE,       # Decorative border
            anchor=E             # Anchor text to the East
        )

        # Create label in the frame to show output
        self.lbl_kelvin = Label(
            self.main_frame,
            width=10,
            relief=GROOVE,
            anchor=E
        )
```

```
        # Use Grid layout manager to place widgets in the frame
        self.entry.grid(row=0, column=0)
        self.btn_calculate.grid(row=0, column=1)

        # sticky uses cardinal directions to stick labels
        # to sides of the column
        # sticky=EW expands the label to fit the column
        # based on the largest widget
        self.lbl_celsius.grid(
            row=1,
            column=0,
            sticky=EW
        )
        self.lbl_kelvin.grid(
            row=1,
            column=1,
            sticky=EW
        )

        # Set padding between frame and window
        self.main_frame.grid_configure(padx=20, pady=20)

        # Set padding for all widgets inside the frame
        for widget in self.main_frame.winfo_children():
            widget.grid_configure(padx=10, pady=10)

        # Start the program with focus on the entry widget
        self.entry.focus_set()

        # Bind both enter key to the convert method
        # When either Enter key is pressed,
        # the convert method will be fired
        # <Return> - Enter key on main keyboard
        # <KP_Enter> - Enter key on number pad/key pad
        self.root.bind("<Return>", self.convert_temperature)
        self.root.bind("<KP_Enter>", self.convert_temperature)


# -------------------- MAIN PROGRAM -------------------------------------#
"""Create program object from the program class to run the program."""
temperature_converter = TemperatureConverter()
```

Example run:



Explanation Using Frames and Widgets with OOP

In the Temperature Converter, we used OOP.

When using a frame to place our widgets, we need to do two things: create the widget itself and then place it onscreen.

When we create a widget, we need to specify its **parent**. That is the widget that the new widget will be placed inside. In this case, we want our entry placed inside the content frame. Our entry, and other widgets we'll create shortly, are said to be **children** of the content frame. In Python, the **parent** is passed as the first parameter when instantiating a widget object as shown below.

```
50          # Create entry widget in the frame to get input from user
51          self.entry = Entry(
52              self.main_frame,
53              text='',
54              width=10)
```

When we create a widget, we can provide it with certain **configuration options**. In the example above, we specify how wide we want the entry to appear, i.e., 10 characters.

When widgets are created, they don't automatically appear on the screen; **tk** doesn't know where you want them placed relative to other widgets. That's what the **grid** part does. Widgets are placed in the appropriate column (0 or 1) and row (also 0 or 1).

```
77          # Use Grid layout manager to place widgets in the frame
78          self.entry.grid(row=0, column=0)
79          self.btn_calculate.grid(row=0, column=1)
80          self.lbl_celsius.grid(row=1, column=0)
81          self.lbl_kelvin.grid(row=1, column=1, sticky=E)
```

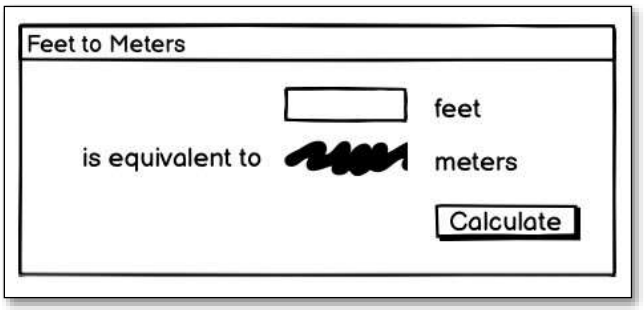
The **sticky** option to grid describes how the widget should line up within the grid cell, using compass directions. **E** (east) means to anchor the widget to the right side of the cell, **WE**

(west-east) anchors the widget to both the left and right sides, and so on. Python defines constants for these directional strings, which you can provide as a list, e.g., **W** or **WE**.

GUI Design with a Pencil

Say we want to create a GUI for a meters to feet calculator.

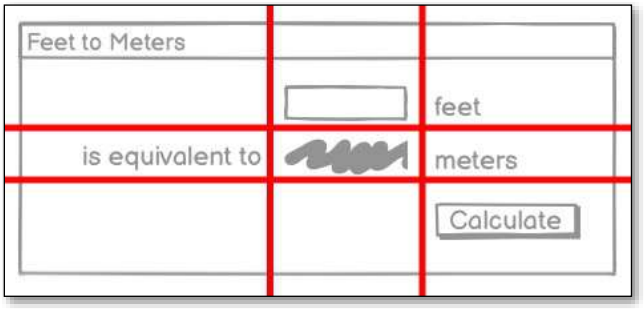
Sketch your interface on paper. It might look something like this:



Here are the widgets we will need.

1. Text entry to type in the number of feet.
2. Calculate button will get the value out of that entry, perform the calculation, and put the result in a label below the entry.
3. Three static labels ("feet," "is equivalent to," and "meters"), which help our user figure out how to run the application.

Let's look at the layout. The grid layout is very flexible for this type of GUI design. The widgets we've included seem to be naturally divided into a grid with three columns and three rows. In terms of layout, things seem to naturally divide into three columns and three rows, as illustrated below:



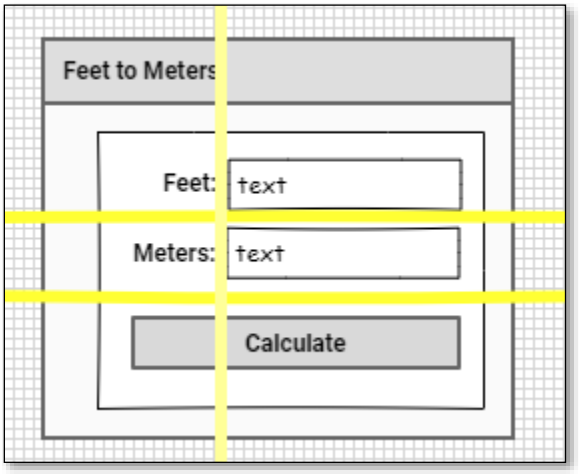
The layout of our user interface with a 3 x 3 grid.

GUI Design with Pencil

Pencil is a free and open-source GUI prototyping tool that people can easily install and use to create mockups in popular desktop platforms. This type of design is called a wireframe.

1. Go to https://pencil.evolus.vn/
2. Download and install Pencil for your OS.

Example wireframe of our evolving GUI:



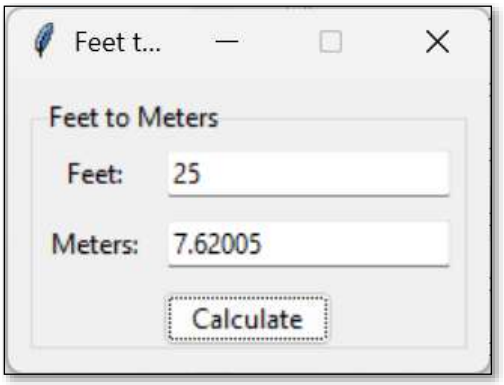
This example uses a grid layout. 2 columns, 3 rows inside a frame.

Assignment 9.2: Feet to Meters Converter

Use what we have learned about Tkinter in OOP to create a Feet to Meters Converter based on the previous OOP programming.

- Use Temperature Converter as a template.

Example run:

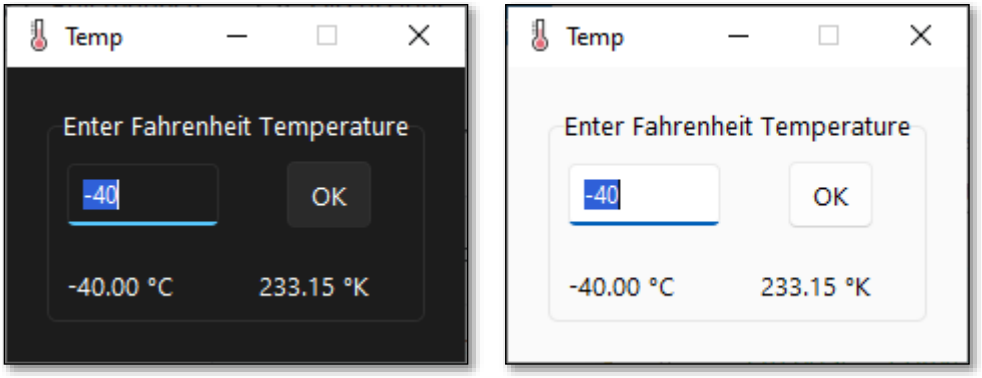


Tutorial 9.4: Temperature Converter with Sun Valley Theme

Tkinter isn't a very modern looking interface. One of the most interesting things about Python is how many open-source libraries can be added to Python to solve almost any problem.

This Tkinter themed example is created from **rdbende's Sun Valley ttk** theme library.

https://github.com/rdbende/Sun-Valley-ttk-theme



Temperature Converter Theme Code Changes

Save **temperature_converter.py** as **temperature_converter_theme.py**

```
"""
    Name: temperature_converter_theme.py
    Author:
    Created:
    Purpose: Convert Fahrenheit to Celsius and Kelvin
    https://github.com/rdbende/Sun-Valley-ttk-theme
"""
# Import the tkinter module with tk standard widgets
from tkinter import *
# Override tk widgets with themed ttk widgets if available
from tkinter.ttk import *
# pip install sv-ttk
# Import Sun Valley Theme
import sv_ttk
```

1. Install the sv-ttk library: **pip install sv-ttk**
2. Import the new library: **import sv_ttk**

```
31          # Create the GUI widgets in a separate method
32          self.create_widgets()
33
34          # Set the theme to light or dark
35          sv_ttk.set_theme("light")
36
37          # Call the mainloop method to start program
38          mainloop()
```

3. Set the theme to light or dark as shown. The above example is light.

Assignment Submission

1. Attach the pseudocode or create a TODO.
2. Attach all tutorials and assignments.
3. Attach screenshots showing the successful operation of each tutorial program.
4. Submit in Blackboard.