PROGRAMMING WITH PYTHON

By Randal Root

Module 10

So far, we have spent all our time creating Console applications, but millions of users find the command shell to be intimidating and unintuitive. In this lesson, we look at how to create a windowed user interface!

Creating a UI with TK Interface

TK Interface, or Tkinter, *"… is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python."* (**https://en.wikipedia.org/wiki/Tkinter,2019**)

You create a Tk interface using **a collection of** "**Widgets**." Common widgets include buttons, textboxes, labels, and tabs. (Figure 1)

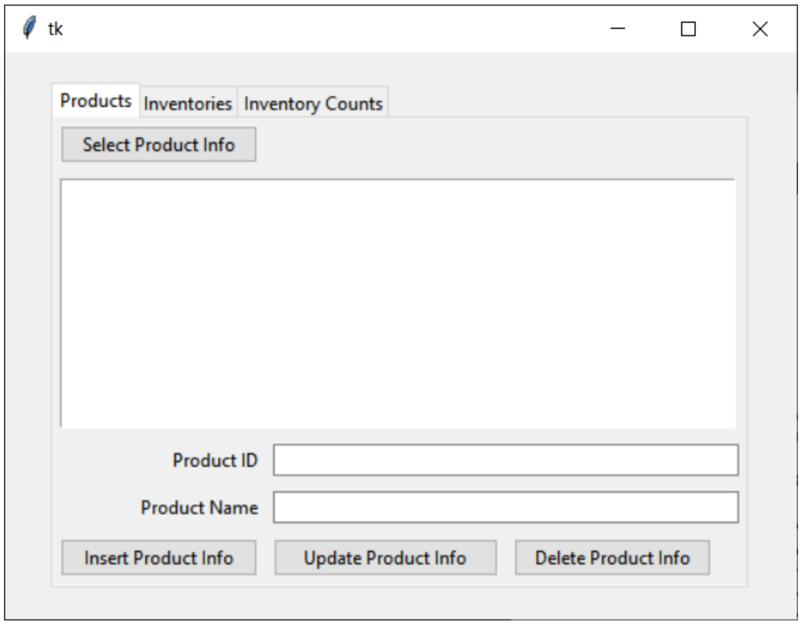


Figure 1. A common windowed interface

Window Root

To create a windowed interface, you need **a Window**! The window **is the "root" of your user interface (UI)** and forms the top of an object Hierarchy. You can document the hierarchy using simple comments like this:

```
# -- Window object -- root of the UI
#   -- Button
#   -- Textbox
```

To make a window, you first import the "**tkinter**" **module** and then **create a window** object using the tkinter.**Tk() constructor** method. Afterward, you use the object's **mainloop()** method to **display the window**.

```
# ---------------------------------------------------------- #
# Title: Listing01
# Description: Common components of a Windowed UI
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ---------------------------------------------------------- #
import tkinter as tk

application_window = tk.Tk()  # creates a root node
application_window.mainloop()  # displays window UI
```

Listing 1

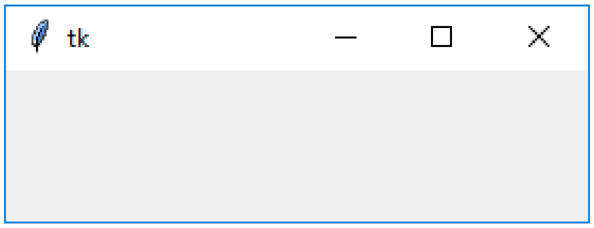


Figure 2: The results of Listing 1

Widgets

The window is a **container for widgets**. The window includes a simple, invisible, rectangular frame.

*"**Widgets** are **basically images on a computer** screen and they have a "look-and-feel" depending on the details of how the image is drawn. The "look-and-feel" of a widget is typically controlled by the operating system. For example, **GUI** programs on a **Macintosh computer typically look different** from programs on a Microsoft **Windows computer**."*
(**http://interactivepython.org/runestone/static/thinkcspy/GUIandEventDrivenProgramming/03_widgets.html**, 2019) [1] (External Link)

```
# ---------------------------------------------------------- #
# Title: Listing02
# Description: Creates the following UI objects
#  -- window_root (tk.TK)
#   -- lbl_math_info (tk.label)
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ---------------------------------------------------------- #
import tkinter as tk

# Create a root node Window object
application_window = tk.Tk()
application_window.geometry("400x100")  # Define the size of the window

# Create a label object in the Window and get a reference
lbl_math_results = tk.Label(application_window, text="Math Results")

# add to the root container using the pack "layout Manager" method
lbl_math_results.pack()

# Display the window
application_window.mainloop()
```

Listing 2

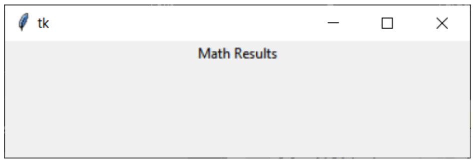


Figure 3: The results of Listing 2

When this code runs, the hierarchy of objects in memory now look like this:

```
#  -- window_root (tk.TK)
#   -- lbl_math_info (tk.label)
```

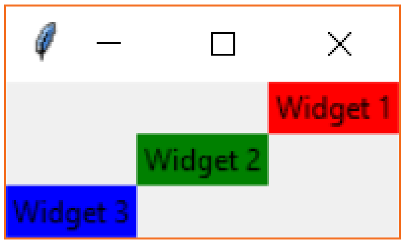
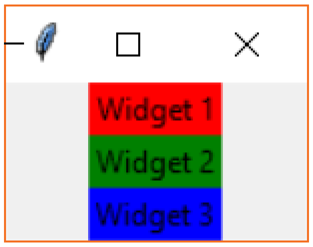
Layout Managers

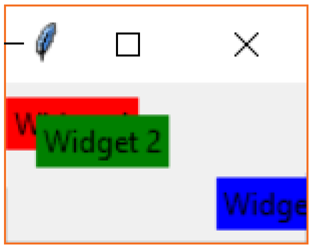
You **control where widgets are placed using a layout manager**. These are **bound to the widget using a layout manager method** like the one in the last **example** called "**pack**()."

```
lbl_math_results.pack()
```

*"**Just instantiating (creating) a widget does not necessarily mean that it will appear on the screen** (with the exception of Toplevel(), which automatically creates a new window). **To get the widget to appear, we need to tell the parent widget where to put it**. To do that, we **use** one of Tkinter's three geometery managers (also known as **layout managers**)."* (**https://learn.sparkfun.com/tutorials/python-gui-guide-introduction-to-tkinter/all#tkinter-overview**, 2019) *[2]* (External Link)

*There are **three layout managers*** in the Tkinter module. Here is a good visualization and the three types of Layout managers *from the learn.Sparkfun.com website tutorial [2].*

Manager	**Description**	**Example**
Grid	The grid manager places widgets in a **table format with rows and columns**. It **will avoid overlapping** widgets and will resize rows/columns as necessary to fit the widgets.	
Pack	Pack is often the easiest geometry manager to use, as it just **puts widgets in a single row or column** (default). It "packs" the widgets by putting them **side-by-side (or top-to-bottom)**.	

Place	The place geometry manager offers the most control but can be the most difficult to use. It allows you to **specify the absolute or relative positions of the widgets** in a window (or parent widget).	

" The three geometry managers are: grid, pack, and place. You should never mix geometry managers within the same hierarchy, but you can embed different managers within each other (for example, you can lay out a frame widget with grid in a Toplevel and then use pack to put different widgets within the frame)."(**https://learn.sparkfun.com/tutorials/python-gui-guide-introduction-to-tkinter/all#tkinter-overview**, 2019)[2] (External Link)

Note: We use the Grid layout manager for our labs and demos.

Grids

*"All widgets have a **.grid(row=r, column=c) method that will place a widget in the cell (r,c) of a 2-D table**. By **default** the widget is rendered in **the middle of the cell** and as small as possible. This can be changed using other optional parameters. The **sticky option allows you to move or** stretch a widget to the borders of a cell. There are 4 options, **tk.E, tk.W, tk.N, and tk.S**, which are abbreviations for east, west, north, and south respectively. These options are **'bit flags'** that can be **combined in any combination** using simple addition." **[1]***

```
# ---------------------------------------------------------- #
# Title: Listing 03
# Description: Creates the following UI objects
#  -- window_root (tk.TK)
#   -- lbl_math_info (tk.label)
#   -- btn_add(tk.button)
#   -- btn_subtract(tk.button)
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ---------------------------------------------------------- #
import tkinter as tk

# Create a root node Window object
application_window = tk.Tk()
application_window.geometry("400x100")
application_window.title("Simple Math")   # Add a titles to the Window

# Create a label and add it to a grid container
lbl_math_results = tk.Label(application_window, text="Math Info")
lbl_math_results.grid(row=1, column=1, sticky=tk.NW, padx=10, pady=5)

# Create a button and add it to a grid container
```

```
btn_add = tk.Button(application_window, text="Add", width=10)
btn_add.grid(row=2, column=1, sticky=tk.NW, padx=10, pady=5)

# Create a button and add it to a grid container
btn_subtract = tk.Button(application_window, text="Subtract", width=10)
btn_subtract.grid(row=2, column=2, sticky=tk.NW, padx=5, pady=5)

# Display the window
application_window.mainloop()
```

Listing 3

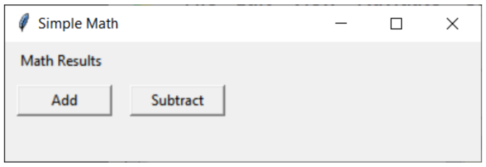


Figure 4. The result of listing 3

Lab 10-1

Create a Window interface that looks like the following image:

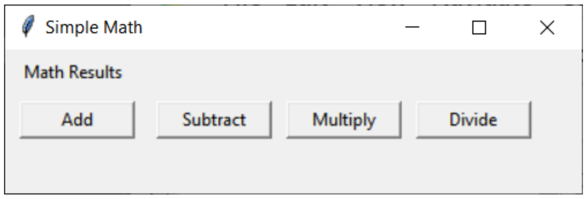


Figure 5. The result of Lab 10-1

1. Create a User Interface that looks like Figure 5.

Note: *Try to get the look and feel close to the picture, but do not worry about minor differences!*

Tk vs. Ttk

*"The tkinter module implements **two versions of widgets: one is "generic,"** which makes widgets look the same regardless of what computer your program is running on, and the other implements widgets that emulate a computer's "look-and-feel". How you import the tkinter module determines which widgets are defined. Using the import statements shown below, the standard convention uses the name **tk to access the "generic" widgets** and the name **ttk** to access the **stylized**, "look-and-feel" widgets**.** **You always need to import the tk functionality** because that allows you to create an application window. You **can import the ttk functionality if you want "look-and-feel" widgets**. You can inter-mix the tk and ttk widgets in an interface if you so choose." **[1]***

```
# ---------------------------------------------------------- #
# Title: Listing 04
# Description: Creates the following UI objects
#  -- window_root (tk.TK)
#   -- lbl_math_info (ttk.label)
#   -- btn_add(ttk.button)
#   -- btn_subtract(ttk.button)
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ---------------------------------------------------------- #
import tkinter as tk
from tkinter import ttk  # Updated look and feel

# Create a root node Window object
application_window = tk.Tk()
application_window.geometry("400x100")
application_window.title("Simple Math")  # Add a titles to the Window

# Create a label and add it to a grid container
lbl_math_results = ttk.Label(application_window, text="Math Results")
lbl_math_results.grid(row=1, column=1, sticky=tk.NW, padx=10, pady=5)

# Create a button and add it to a grid container
btn_add = ttk.Button(application_window, text="Add", width=10)
btn_add.grid(row=2, column=1, sticky=tk.NW, padx=10, pady=5)

# Create a button and add it to a grid container
btn_subtract = ttk.Button(application_window, text="Subtract", width=10)
btn_subtract.grid(row=2, column=2, sticky=tk.NW, padx=5, pady=5)

# Display the window
application_window.mainloop()
```

Listing 4

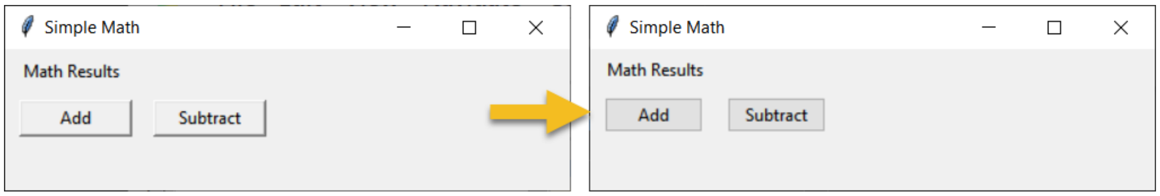


Figure 6. The results of listing 4

Organizing your Code

Currently, there is not much code in our example, but that changes shortly. So, we should start **organizing our code using custom classes and methods**!

Creating a UI Class

Let's **create a class to hold the code for our main window** (which runs in our main module.)

```
# ---------------------------------------------------------- #
# Title: Listing 05
# Description: Creates a Windowed UI
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ---------------------------------------------------------- #
import tkinter as tk
from tkinter import ttk  # Updated look and feel

class MainWindow(object):
    """  Description: Creates the following UI objects:

    window_root (tk.TK):
        lbl_math_info (ttk.label):
        btn_add (ttk.button):
        btn_subtract (ttk.button):
    """
    @staticmethod
    def create_main_window():
        application_window = tk.Tk()
        application_window.geometry("400x100")
        application_window.title("Simple Math")

        lbl_math_results = ttk.Label(application_window, text="Math Results")
        lbl_math_results.grid(row=1, column=1, sticky=tk.NW, padx=10, pady=5)

        btn_add = ttk.Button(application_window, text="Add", width=10)
        btn_add.grid(row=2, column=1, sticky=tk.NW, padx=10, pady=5)

        btn_subtract = ttk.Button(application_window, text="Subtract", width=10)
        btn_subtract.grid(row=2, column=2, sticky=tk.NW, padx=5, pady=5)

        return application_window
# End class

if __name__ == '__main__':
    mw = MainWindow.create_main_window()
    mw.mainloop()
```
Listing 5

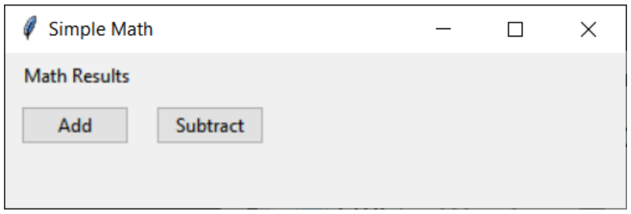


Figure 7. the result of Listing 6

Lab 10-2

Use the demonstrated code to create a UI that looks like this:

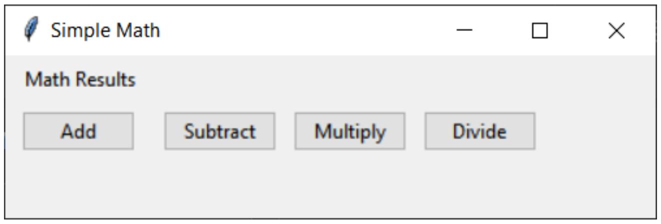


Figure 8. The result of Lab 10-2

1. Create a UI class with the indicated object names and hierarchy.

```
class MainWindow(object):
    """  Description: Creates the following UI objects:
    window_root (tk.TK)
        lbl_math_info (ttk.label)
        btn_add (ttk.button)
        btn_subtract (ttk.button)
        btn_multiply (ttk.button)
        btn_divide (ttk.button)
    """
```

2. Test your UI.

Note: *Try to get the look and feel close to the picture, but do not worry about minor differences!*

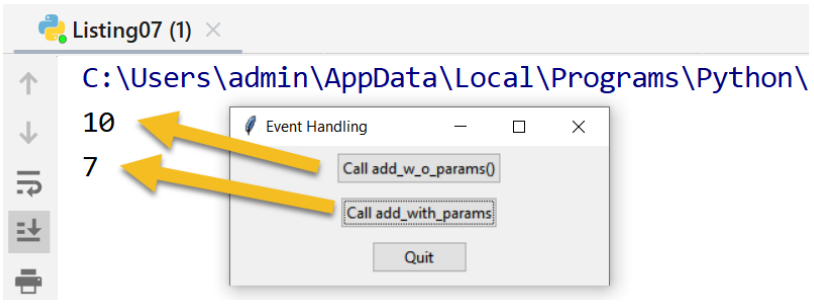
Working with Buttons

To make your buttons work you need to **create a event handing function**. These functions can have zero or more parameters, but they do not return data. When **someone clicks the button the code inside the function runs**.

Important: You **use a lambda syntax for** the event handling method **when** there are **parameters as shown in Listing 6**!

```
# ---------------------------------------------------------- #
# Title: Listing 06
# Description: Demonstrate using Event Handlers
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ---------------------------------------------------------- #
import tkinter as tk
from tkinter import ttk

class MathProcessor(object):
    @staticmethod
    def add_w_o_params(): print(5 + 5)
    @staticmethod
    def add_with_params(n1, n2): print(n1 + n2)

class MainWindow(object):

    def __init__(self):
        self.window = tk.Tk()
        self.window.title("Event Handling")
        self.window.geometry("300x110")

        # Set Event handler for custom method w/o parameters
        btn1 = ttk.Button(self.window, text="Call add_w_o_params()")
        btn1.pack(pady=5)
        btn1['command'] = MathProcessor.add_w_o_params  # don't use parentheses ()

        # Set Event handler for custom method with parameters
        btn5 = ttk.Button(self.window, text="Call add_with_params")
        btn5.pack(pady=5)
        btn5['command'] = lambda: MathProcessor.add_with_params(5, 2)

        #  Set Event handler to a built-in method
        btn3 = ttk.Button(self.window, text="Quit")
        btn3.pack(pady=5)
        btn3['command'] = self.window.destroy  # don't use parentheses ()

if __name__ == '__main__':
    mw = MainWindow()
    mw.window.mainloop()
```

Listing 6

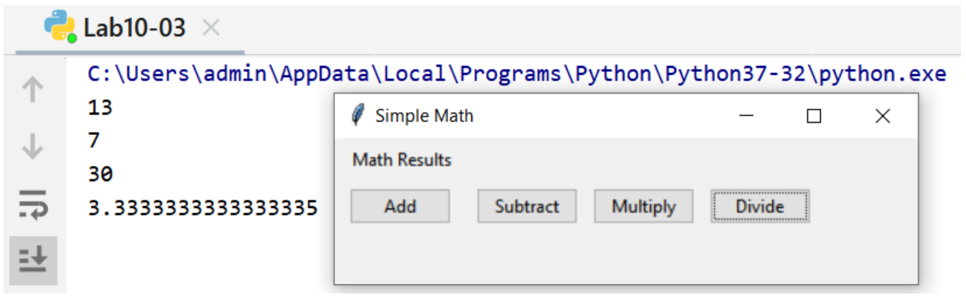


```
C:\Users\admin\AppData\Local\Programs\Python\
10
7
```

Figure 9. the results of Listing 6

Lab 10-3

Modify the code you created in Lab 10-2 to click events. The outcome should look like this:



```
C:\Users\admin\AppData\Local\Programs\Python\Python37-32\python.exe
13
7
30
3.3333333333333335
```

Figure 10. The results of lab 10-3

1. Create the following class:

```
class MathProcessor(object):
    @staticmethod
    def add(n1, n2): print(n1 + n2)

    @staticmethod
    def subtract(n1, n2): print(n1 - n2)

    @staticmethod
    def multiply(n1, n2): print(n1 * n2)

    @staticmethod
    def divide(n1, n2): print(n1 / n2)
```

2. Connect buttons to each method using a lambda syntax like this:

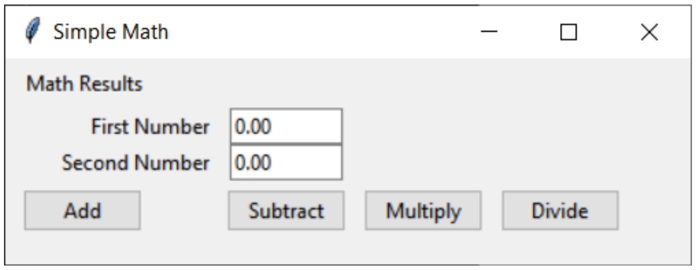
```
btn_add = ttk.Button(application_window, text="Add", width=10)
btn_add.grid(row=2, column=1, sticky=tk.NW, padx=10, pady=5)
btn_add['command'] = lambda: MathProcessor.add(10, 3)
```

Entry (Textbox)

"The **Entry** widget is a standard Tkinter widget used to enter or display a **single line of text**.

```
# ---------------------------------------------------------- #
# Title: Listing 07
# Description: Demonstrate using Entry textboxes
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ---------------------------------------------------------- #
import tkinter as tk
from tkinter import ttk

class MathProcessor(object):
    @staticmethod
    def add(n1, n2): print(n1 + n2)

    @staticmethod
    def subtract(n1, n2): print(n1 - n2)

    @staticmethod
    def multiply(n1, n2): print(n1 * n2)

    @staticmethod
    def divide(n1, n2): print(n1 / n2)

class MainWindow(object):
    """  Description: Creates the following UI objects:
    window_root (tk.TK)
        lbl_math_info (ttk.label)
        txt_first_number (ttx.entry)
         txt_second_number (ttx.entry)
        btn_add (ttk.button)
        btn_subtract (ttk.button)
        btn_multiply (ttk.button)
        btn_divide (ttk.button)
    """

    @staticmethod
    def create_main_window():
        """ Create  and configure a root node Window object"""
        application_window = tk.Tk()
        application_window.geometry("400x120")
        application_window.title("Simple Math")

        lbl_math_results = ttk.Label(application_window, text="Math Results")
        lbl_math_results.grid(row=1, column=1, sticky=tk.NW, padx=10, pady=5)

        lbl_first_number = ttk.Label(
            application_window,
            text="First Number ",
            width=20,
            anchor=tk.E
        )
        lbl_first_number.grid(row=2, column=1, sticky=tk.E)
        txt_first_number = ttk.Entry(application_window, width=10)
        txt_first_number.grid(row=2, column=2)
        txt_first_number.insert(0, "0.00")

        lbl_second_number = ttk.Label(
            application_window,
            text="Second Number ",
            width=20,
            anchor=tk.E
        )
        lbl_second_number.grid(row=3, column=1, sticky=tk.E)
        txt_second_number = ttk.Entry(application_window, width=10)
        txt_second_number.grid(row=3, column=2)
        txt_second_number.insert(0, "0.00")

        btn_add = ttk.Button(application_window, text="Add", width=10)
        btn_add.grid(row=4, column=1, sticky=tk.NW, padx=10, pady=5)
        btn_add['command'] = lambda: MathProcessor.add(10, 3)

        btn_subtract = ttk.Button(application_window, text="Subtract", width=10)
        btn_subtract.grid(row=4, column=2, sticky=tk.NW, padx=5, pady=5)
        btn_subtract['command'] = lambda: MathProcessor.subtract(10, 3)

        btn_multiply = ttk.Button(application_window, text="Multiply", width=10)
        btn_multiply.grid(row=4, column=3, sticky=tk.NW, padx=5, pady=5)
        btn_multiply['command'] = lambda: MathProcessor.multiply(10, 3)

        btn_divide = ttk.Button(application_window, text="Divide", width=10)
        btn_divide.grid(row=4, column=4, sticky=tk.NW, padx=5, pady=5)
        btn_divide['command'] = lambda: MathProcessor.divide(10, 3)

        return application_window
# End class

if __name__ == '__main__':
    mw = MainWindow.create_main_window()
    mw.mainloop()
```

Listing 7


Simple Math

Math Results

First Number 0.00

Second Number 0.00

Figure 11. The results of Listing 7

Spanning Multiple Cells

"Widgets can ***take up more than a single cell in the grid****; to do this, you'll use the "columnspan" and "rowspan"options when gridding the widget. These are analogous to the "colspan" and "rowspan" attribute of HTML tables."*(**https://tkdocs.com/tutorial/grid.html**, 2019)

If you look at Figure 11, you can see a grid of 4 by 4 cells, with each control is in its own cell. Figure 12 shows a visualization of these cells.

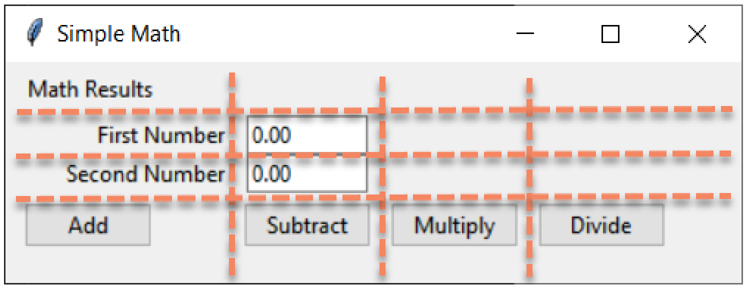


Figure 12. Imaging the control grid

However, sometimes you want a control to cover more than one cell in the grid. When that happens, you can use the columnspan parameter. For example, the following code will cause the txt_first_number textbox (Entry) control to span 3 cells in the grid as shown in Figure 14. To make the UI look the way you want you might also have to adjuct the xpad and ypad values.

```
#txt_second_number.grid(row=3, column=2)
txt_second_number.grid(row=3, column=2, columnspan=3)
txt_second_number.insert(0, "0.00")

btn_add = ttk.Button(application_window, text="Add", width=10)
btn_add.grid(row=4, column=1, sticky=tk.NE, padx=5, pady=5)
btn_add['command'] = lambda: MathProcessor.add(10, 3)
```

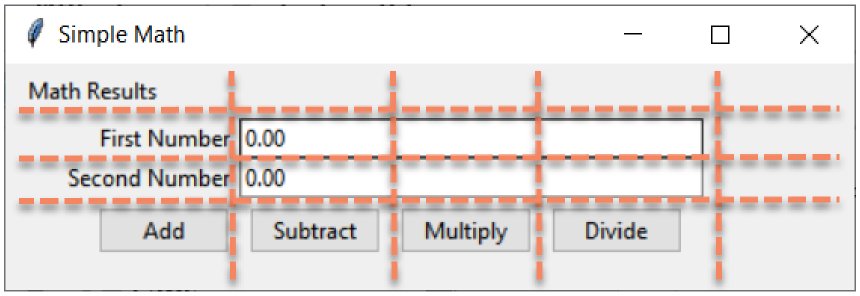


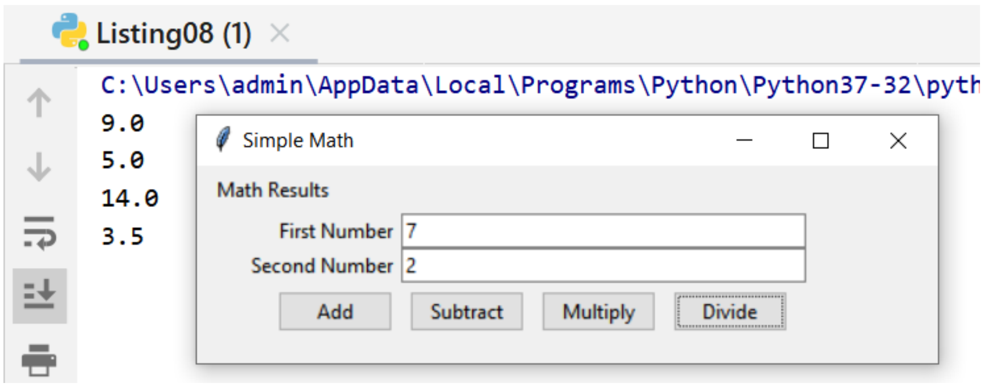
Figure 13. The results of not using columnspan and adjusted padding

Getting Entry Data

When you want to access that data of an Entry textbox you use the get() method of the Entry object (Listing 8).

```
# ---------------------------------------------------------- #
# Title: Listing 08
# Description: Demonstrate getting data from Entry textboxes
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ---------------------------------------------------------- #
import tkinter as tk
from tkinter import ttk

class MathProcessor(object):
    @staticmethod
    def add(n1, n2): print(n1 + n2)

    @staticmethod
    def subtract(n1, n2): print(n1 - n2)

    @staticmethod
    def multiply(n1, n2): print(n1 * n2)

    @staticmethod
    def divide(n1, n2): print(n1 / n2)

class MainWindow(object):
    """  Description: Creates the following UI objects:
    window_root (tk.TK)
        lbl_math_info (ttk.label)
        txt_first_number (ttk.entry)
        txt_second_number (ttk.entry)
        btn_add (ttk.button)
        btn_subtract (ttk.button)
        btn_multiply (ttk.button)
        btn_divide (ttk.button)
    """

    @staticmethod
    def create_main_window():
        """ Create  and configure a root node Window object"""
        application_window = tk.Tk()
        application_window.geometry("450x120")
        application_window.title("Simple Math")

        lbl_math_results = ttk.Label(application_window, text="Math Results")
        lbl_math_results.grid(row=1, column=1, sticky=tk.NW, padx=10, pady=5)

        lbl_first_number = ttk.Label(
            application_window,
            text="First Number ",
            width=20,
            anchor=tk.E
        )
        lbl_first_number.grid(row=2, column=1, sticky=tk.E)

        txt_first_number = ttk.Entry(application_window, width=40)
        txt_first_number.grid(row=2, column=2, columnspan=3)
        txt_first_number.insert(0, "0.00")

        lbl_second_number = ttk.Label(
            application_window,
            text="Second Number ",
            width=20,
            anchor=tk.E
        )
        lbl_second_number.grid(row=3, column=1, sticky=tk.E)

        txt_second_number = ttk.Entry(application_window, width=40)
        txt_second_number.grid(row=3, column=2, columnspan=3)
        txt_second_number.insert(0, "0.00")

        btn_add = ttk.Button(application_window, text="Add", width=10)
        btn_add.grid(row=4, column=1, sticky=tk.NE, padx=5, pady=5)
        btn_add['command'] = lambda: MathProcessor.add(
            float(txt_first_number.get()), float(txt_second_number.get())
        )

        btn_subtract = ttk.Button(application_window, text="Subtract", width=10)
        btn_subtract.grid(row=4, column=2, sticky=tk.NW, padx=5, pady=5)
        btn_subtract['command'] = lambda: MathProcessor.subtract(7,2))

        btn_multiply = ttk.Button(application_window, text="Multiply", width=10)
        btn_multiply.grid(row=4, column=3, sticky=tk.NW, padx=0, pady=5)
        btn_multiply['command'] = lambda: MathProcessor.multiply(7,2))

        btn_divide = ttk.Button(application_window, text="Divide", width=10)
        btn_divide.grid(row=4, column=4, sticky=tk.NW, padx=5, pady=5)
        btn_divide['command'] = lambda: MathProcessor.divide(7,2)

        return application_window
# End class

if __name__ == '__main__':
    mw = MainWindow.create_main_window()
    mw.mainloop()
```

Listing 8



```
C:\Users\admin\AppData\Local\Programs\Python\Python37-32\pyth
9.0
5.0
14.0
3.5
```

Math Results

First Number 7

Second Number 2

Figure 14. The results of Listing 8

Lab 10-4

Modify the code you created in Lab 10-3 to capture data from a first and second number using Entry textboxes, then adds, subtracts, multiplies, and divides the values from the textboxes. Format the UI to look like Figure 14.

Note: *Try to get the look and feel close to the picture, but do not worry about minor differences! If you get stuck you*

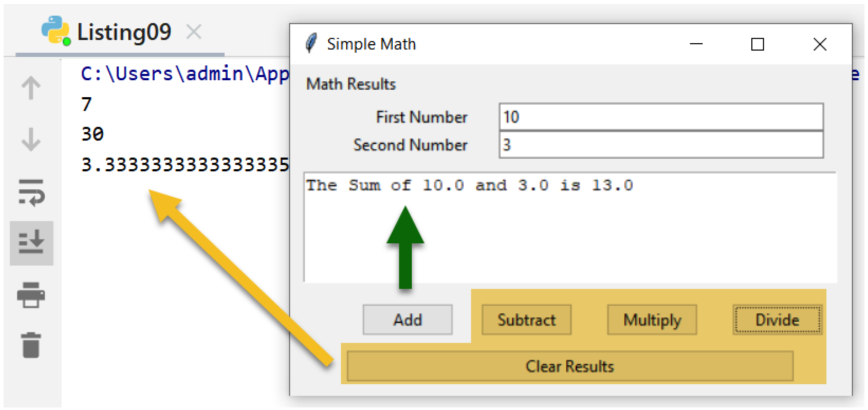
Text (Multi-line Textbox)

*"A text widget **manages a multi-line text area**. Like the canvas widget, Tk's text widget is an immensely flexible and powerful tool which can be used for a wide variety of tasks." (**https://tkdocs.com/tutorial/text.html**, 2019) [4]*

```
# ---------------------------------------------------------- #
# Title: Listing 09
# Description: Demonstrate using  multi-line textboxes
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# ---------------------------------------------------------- #
import tkinter as tk
from tkinter import ttk

class MathProcessor(object):
    @staticmethod
    def add(n1, n2): return n1 + n2

    @staticmethod
    def subtract(n1, n2): print(n1 - n2)

    @staticmethod
    def multiply(n1, n2): print(n1 * n2)

    @staticmethod
    def divide(n1, n2): print(n1 / n2)
```

```
class MathIO(object):

    @staticmethod
    def clear_textbox(textbox):
        textbox['state'] = 'normal'
        textbox.delete(1.0, tk.END)
        textbox['state'] = 'disabled'

    @staticmethod
    def write_sum_to_textbox(n1, n2, textbox):
        textbox['state'] = 'normal'
        text = str.format('The Sum of {n1} and {n2} is {result}\n',
                          n1=n1, n2=n2, result=MathProcessor.add(n1, n2))
        textbox.insert(tk.END, text)
        textbox['state'] = 'disabled'

    @staticmethod
    def write_difference_to_textbox(n1, n2, textbox):
        textbox['state'] = 'normal'
        text = str.format('The difference of {n1} and {n2} is {result}\n',
                          n1=n1, n2=n2, result=MathProcessor.subtract(n1, n2))
        textbox.insert(tk.END, text)
        textbox['state'] = 'disabled'

    @staticmethod
    def write_product_to_textbox(n1, n2, textbox):
        textbox['state'] = 'normal'
        text = str.format('The product of {n1} and {n2} is {result}\n',
                          n1=n1, n2=n2, result=MathProcessor.multiply(n1, n2))
        textbox.insert(tk.END, text)
        textbox['state'] = 'disabled'

    @staticmethod
    def write_quotient_to_textbox(n1, n2, textbox):
        textbox['state'] = 'normal'
        text = str.format('The quotient of {n1} and {n2} is {result}\n',
                          n1=n1, n2=n2, result=MathProcessor.divide(n1, n2))
        textbox.insert(tk.END, text)
        textbox['state'] = 'disabled'
# End class

class MainWindow(object):
    """  Description: Creates the following UI objects:
    window_root (tk.TK)
        lbl_math_info (ttk.label)
        txt_first_number (ttk.entry)
        txt_second_number (ttk.entry)
        mtx_results (ttk.textbox)
        btn_add (ttk.button)
        btn_subtract (ttk.button)
        btn_multiply (ttk.button)
        btn_divide (ttk.button)
    """
    @staticmethod
    def create_main_window():
        """ Create  and configure a root node Window object"""
        application_window = tk.Tk()
        application_window.geometry("425x250")
        application_window.title("Simple Math")

        lbl_math_results = ttk.Label(application_window, text="Math Results")
        lbl_math_results.grid(row=1, column=1, sticky=tk.NW, padx=10, pady=5)

        lbl_first_number = ttk.Label(
            application_window,
            text="First Number ",
            width=20,
            anchor=tk.E
        )
        lbl_first_number.grid(row=2, column=1, sticky=tk.E)
        txt_first_number = ttk.Entry(application_window, width=40)
        txt_first_number.grid(row=2, column=2, columnspan=3)
        txt_first_number.insert(0, "0.00")

        lbl_second_number = ttk.Label(
            application_window,
            text="Second Number ",
            width=20,
            anchor=tk.E
        )
        lbl_second_number.grid(row=3, column=1, sticky=tk.E)
        txt_second_number = ttk.Entry(application_window, width=40)
        txt_second_number.grid(row=3, column=2, columnspan=3)
        txt_second_number.insert(0, "0.00")

        # Adding a multi-line textbox
        mtx_results = tk.Text(width=50, height=5)
        mtx_results.grid(row=4,
            column=1,
            sticky=tk.N,
            columnspan=4,
            padx = 10,
            pady =10
            )
        btn_add = ttk.Button(application_window, text="Add", width=10)
        btn_add.grid(row=5, column=1, sticky=tk.E, padx=15, pady=5)
        btn_add['command'] = lambda: MathIO.write_sum_to_textbox(
            float(txt_first_number.get()),
            float(txt_second_number.get()),
            mtx_results)

        btn_subtract = ttk.Button(application_window, text="Subtract", width=10)
        btn_subtract.grid(row=5, column=2, sticky=tk.W, padx=5, pady=5)
        btn_subtract['command'] = \
            lambda: MathProcessor.subtract(10, 3)

        btn_multiply = ttk.Button(application_window, text="Multiply", width=10)
        btn_multiply.grid(row=5, column=3, sticky=tk.W, padx=5, pady=5)
        btn_multiply['command'] = \
            lambda: MathProcessor.multiply(10, 3)

        btn_divide = ttk.Button(application_window, text="Divide", width=10)
        btn_divide.grid(row=5, column=4, sticky=tk.W, padx=5, pady=5)
        btn_divide['command'] = \
            lambda: MathProcessor.divide(10, 3)

        btn_divide = ttk.Button(application_window, text="Clear Results", width=55)
        btn_divide.grid(row=6, column=1,  padx=15, pady=5, columnspan=4)
        btn_divide['command'] = lambda: MathIO.clear_textbox(mtx_results)
        return application_window
# End class

if __name__ == '__main__':
    mw = MainWindow.create_main_window()
    mw.mainloop()
```

Listing 9



```
C:\Users\admin\App
7
30
3.3333333333333335
```

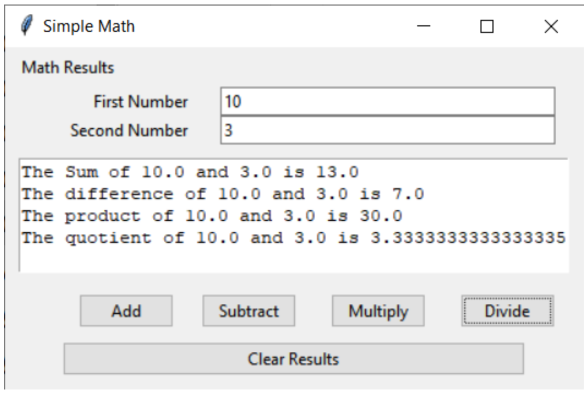
First Number: 10
Second Number: 3

```
The Sum of 10.0 and 3.0 is 13.0
```

Figure 15. The results of listing 9

Lab 10-5

Modify the code you created in Lab 10-3 to capture data from a first and second number using Entry textboxes, then adds, subtracts, multiplies, and divides the values from the Entry textboxes and add the results to a multi-line textbox. Format the UI to look like Figure 16.



```
The Sum of 10.0 and 3.0 is 13.0
The difference of 10.0 and 3.0 is 7.0
The product of 10.0 and 3.0 is 30.0
The quotient of 10.0 and 3.0 is 3.3333333333333335
```

Figure 16. Result of the lab 10-5

Note: *Try to get the look and feel close to the picture, but do not worry about minor differences! If you get stuck you*

Summary

In this module, we looked at how to create a windowed user interface. This process is substantially harder code than a console user interface, so you may want to continue to use a console UI for now, but as you become a more experienced programmer you will find that a Windowed use is much appreciated!

At this point, you should try and answer as many of the following questions as you can from memory, then review the subjects that you cannot. Imagine being asked these questions by a co-worker or in an interview to connect this learning with aspects of your life.

- What is the difference between a console and a windowed UI?
- What are the pros and cons of a windowed UI?
- What is Event Handling?
- What are Widgets?

When you can answer all of these from memory, it is time to complete the module's assignment and complete the course.

References

1. Runestone, GUI and Event Driven Programming

http://interactivepython.org/runestone/static/thinkcspy/GUIandEventDrivenProgramming/toctree.html, 2019, (External Link)

2. Python GUI Guide: Introduction to Tkinter

https://learn.sparkfun.com/tutorials/python-gui-guide-introduction-to-tkinter/all#tkinter-overview, 2019, (External Link)

3. Effbot.org, tkinterbook

https://effbot.org/tkinterbook/, 2019, (External Link)

4. TkDocs.org, tutorial

https://tkdocs.com/tutorial/text.html, 2019, (External Link)