# **Programming with Python**

# Module 10:



In this class we have used Python to learn the foundations of programing. In this module I want to extend that knowledge to other technologies in an effort to show you how these basic concepts can be applied.

As your programs become more complex you may need to add more people or use different technologies to complete your projects. This often entails the use of technologies and tools outside of what you presently know.

* Windowed UIs
* Other Dev Environments
* Other Languages
* GitHub

# Windowed UIs

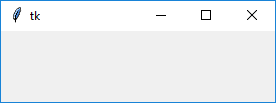
So far, we have spent all our time creating Console applications, but millions of users find the command shell to be intimidating and unintuitive. In this lesson we will look at how to create a windowed user interface!

## Window Root

To create a windowed interface, you need **a Window**! This **is the "root" of your UI** and will form the top of an object Hierarchy.

To make a window you first import the "tkinter" module and then **create and display** a window root with this code:

**import** tkinter **as** tk  
  
application\_window = tk.Tk() **# creates a root node**  
application\_window.mainloop() **# displays window UI**



## 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)

**import** tkinter **as** tk  
**from** tkinter **import** ttk  
  
application\_window = tk.**Tk**()lbl\_product\_info = tk.**Label**(application\_window, text=**"Product Info"**)  
lbl\_product\_info.**pack**() *# add to the root container using the* ***pack*** *layout manager method*application\_window.mainloop()

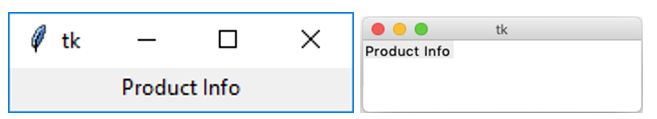


Figure: A window with a label on both Windows and Mac OS

Now, we have created the following hierarchy of objects:

**-- application\_window tk.TK  
 -- lbl\_product\_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\_product\_info.**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. | Tkinter grid layout example |
| 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**). | Tkinter pack layout example |
| 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). | Tkinter place layout example |

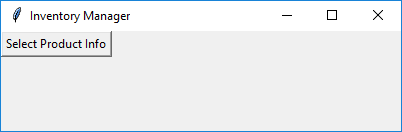
" The three geometry managers are: grid, pack, and place. You **should never mix geometry managers within the same hierarchy, but you can embed different managers within each other** (for example, you can lay out a frame widget with grid in a Toplevel and then use pack to put different widgets within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

**Note:** We will 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]**

import tkinter as tk  
  
"""  
Desc: Creates the following UI objects  
 -- window\_root tk.TK  
 -- btn\_sel\_product\_info tk.button  
"""  
  
window = tk.Tk() *# creates an root node*window.title("Inventory Manager")  
window.geometry("400x100+20+20")  
  
btn\_sel\_product\_info = tk.Button(window, text="Select Product Info")  
btn\_sel\_product\_info.grid(row=2, column=2, sticky=tk.N ) *# add to a grid container*window.mainloop()



# Entry (Textbox)

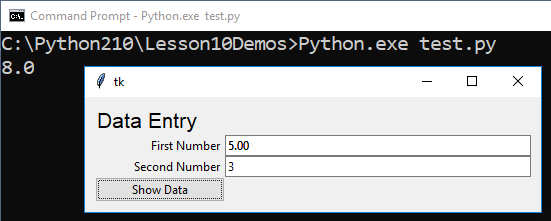
The **Entry** widget is a standard Tkinter widget used to enter or display a **single line of text**.

**import** tkinter **as** tk  
**from** tkinter **import** ttk  
  
**class MathProcessor**():

@staticmethod  
 **def** add\_values(n1, n2): **return** n1 + n2

**class IOProcessor**():

@staticmethod  
 **def** write\_results(n1, n2):  
 text = str.format(**'The Sum of {n1} and {n2} is {result}\n'**,  
 n1=n1, n2=n2, result=MathProcessor.add\_values(n1, n2))  
 print(text)  
  
window = tk.Tk()  
window[**'padx'**] = 10  
window[**'pady'**] = 10  
  
lbl\_data\_entry = ttk.Label(window, text=**"Data Entry"**, font=(**"Arial"**, 16), anchor=tk.W)  
lbl\_data\_entry.grid(row=1, column=1, sticky=tk.W)  
  
lbl\_first\_name = ttk.Label(window, text=**"First Number "**, width=20, anchor=tk.E)  
lbl\_first\_name.grid(row=2, column=1, sticky=tk.E)  
txt\_first\_number = ttk.**Entry**(window, width=50)  
txt\_first\_number.grid(row=2, column=2)  
txt\_first\_number.**insert**(0, **"0.00"**)  
  
lbl\_second\_name = ttk.Label(window, text=**"Second Number "**, width=20, anchor=tk.E)  
lbl\_second\_name.grid(row=3, column=1, sticky=tk.E)  
txt\_second\_ number = ttk.**Entry**(window, width=50)  
txt\_second\_ number.grid(row=3, column=2)  
txt\_second\_ number.**insert**(0, **"0.00"**)  
  
btn\_show\_data = ttk.Button(window, text=**"Show Data"**, width=20)  
btn\_show\_data.grid(row=5, column=1, sticky=tk.E)  
btn\_show\_data[**'command'**] = **lambda**: **IOProcessor.write\_results**(float(txt\_first\_number.**get**()),  
 float(txt\_second\_ number.**get**()))  
  
window.mainloop()

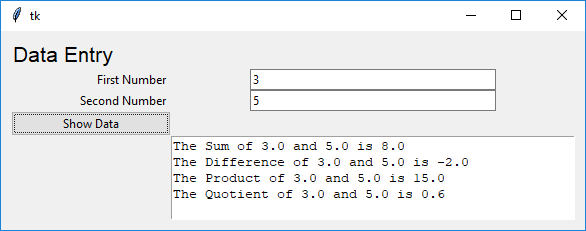


# Text (Multi-line Textbox)

"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]

**import** tkinter **as** tk  
**from** tkinter **import** ttk  
  
**class** MathProcessor():  
 **def** add\_values(n1, n2): **return** n1 + n2  
  
 **def** subtract\_values(n1, n2): **return** n1 - n2  
  
 **def** multiply\_values(n1, n2): **return** n1 \* n2  
  
 **def** divide\_values(n1, n2): **return** n1 / n2  
  
**class** IOProcessor():  
 **def** write\_results\_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\_values(n1,n2))  
 text += str.format(**'The Difference of {n1} and {n2} is {result}\n'**,  
 n1=n1, n2=n2, result=MathProcessor.subtract\_values(n1,n2))  
 text += str.format(**'The Product of {n1} and {n2} is {result}\n'**,  
 n1=n1, n2=n2, result=MathProcessor.multiply\_values(n1,n2))  
 text += str.format(**'The Quotient of {n1} and {n2} is {result}\n'**,  
 n1=n1, n2=n2, result=MathProcessor.divide\_values(n1,n2))  
 textbox.delete(1.0, tk.END)  
 textbox.insert(tk.END, text)  
 textbox[**'state'**] = **'disabled'**

window = tk.Tk()  
window[**'padx'**] = 10  
window[**'pady'**] = 10  
  
lbl\_data\_entry = ttk.Label(window, text=**"Data Entry"**, font=(**"Arial"**, 16), anchor=tk.W)  
lbl\_data\_entry.grid(row=1, column=1, sticky=tk.W)  
  
lbl\_first\_num = ttk.Label(window, text=**"First Number "**, width=20, anchor=tk.E)  
lbl\_first\_num.grid(row=2, column=1, sticky=tk.E)  
txt\_first\_num = ttk.Entry(window, width=50)  
txt\_first\_num.grid(row=2, column=2)  
txt\_first\_num.insert(0, **"0.00"**)  
  
lbl\_second\_num = ttk.Label(window, text=**"Second Number "**, width=20, anchor=tk.E)  
lbl\_second\_num.grid(row=3, column=1, sticky=tk.E)  
txt\_second\_num = ttk.Entry(window, width=50)  
txt\_second\_num.grid(row=3, column=2)  
txt\_second\_num.insert(0, **"0.00"**)  
  
btn\_show\_data = ttk.Button(window, text=**"Show Data"**, width=20)  
btn\_show\_data.grid(row=5, column=1, sticky=tk.E)  
btn\_show\_data[**'command'**] = **lambda**: IOProcessor.write\_results\_to\_textbox(float(txt\_first\_num.get()),  
 float(txt\_second\_num.get()),  
 mtx)  
mtx = tk.Text(width=40,height=10)  
mtx.grid(row=6, column=2, sticky=tk.W)  
  
window.mainloop()

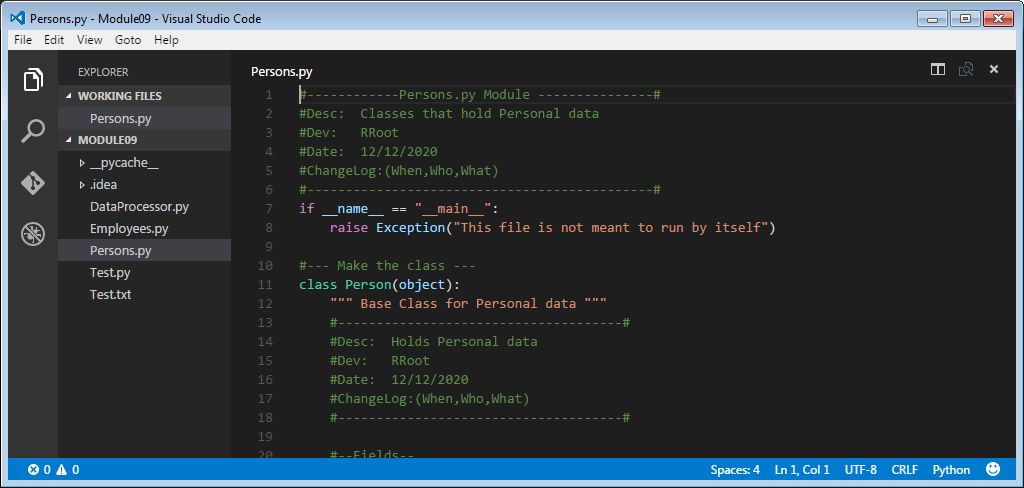


**NOTE:** To learn more about how to create Windowed applications continue reading your book!

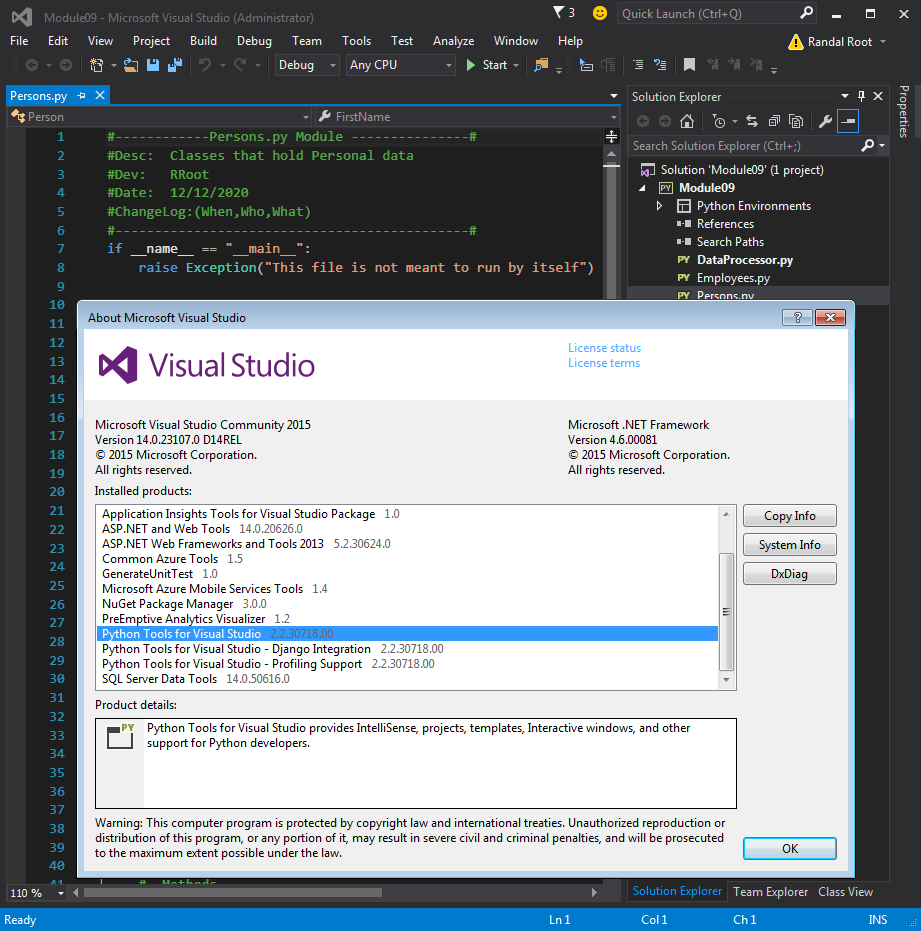
# Other Dev Environments

Microsoft has long used Visual Studio as its development environment. Recently this environment is being moved to an Open Source version in addition to its free version called Community Edition.

The Open Source version is called Visual Studio Code (beta) and can be installed on Mac and Linux.



The Community Edition is currently only running on Windows, but will be available in the future on Mac and Linux in the future.



# Other Languages

Once you have learned one language you will find that learning another is often quite easy.

For example, compare the difference between a Python Script file and C# code file. Here is an example of a python class

#### Python

#--- Make the class ---

class Person(object):

""" Base Class for Personal data """

#--Fields--

\_\_Counter = 0

#--Constructor--

def \_\_init\_\_(self, FirstName = ""):

#Attributes

self.\_\_FirstName = FirstName # Private Attribute

Person.\_\_SetObjectCount() # Private Method

#--Properties--

#FirstName

@property #getter(accessor)

def FirstName(self):

return self.\_\_FirstName

@FirstName.setter #(mutator)

def FirstName(self, Value):

self.\_\_FirstName = Value

#--Methods--

def ToString(self):

"""Explictly returns field data"""

return self.FirstName

def \_\_str\_\_(self):

"""Implictly returns field data"""

return self.FirstName

@staticmethod

def GetObjectCount(): # You do not need the self keyword

return Person.\_\_Counter

@staticmethod

def \_\_SetObjectCount(): # This is a private and static method

Person.\_\_Counter += 1

#--End of class Person--

#### Now let’s compare it to Microsoft’s C# language (which is very similar to Java!).

#### Microsoft’s C#

class Person:Object

{

//--Fields--

static int \_\_Counter = 0;

string \_\_FirstName;

//--Constructor--

public Person(string FirstName = "")

{

//Attributes

this.\_\_FirstName = FirstName; // Private Attribute = Field in C#

Person.\_\_SetObjectCount(); // Private Static Method

}

//--Properties--

//FirstName

public string FirstName

{

get

{ return \_\_FirstName; }

set

{ \_\_FirstName = value; }

}

public override string ToString()

{

return this.FirstName;

}

static int GetObjectCount()

{ // You do not need the self keyword

return Person.\_\_Counter;

}

static void \_\_SetObjectCount()

{ // This is a private and static method

Person.\_\_Counter += 1;

}

}//--End of class Person--

## Demonstration

Your instructor will now demonstrate another comparison between Python and C# using a Visual Studio Solution.