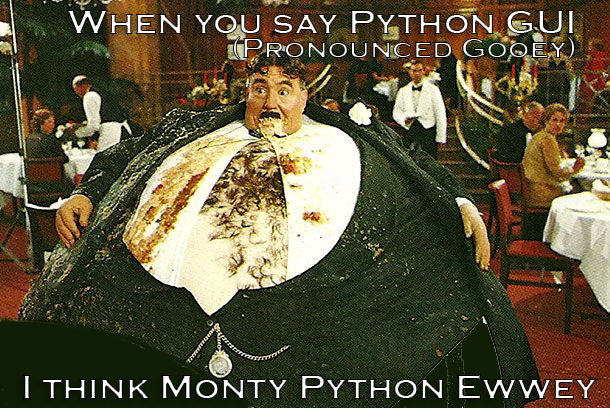
| Unit 10 Graphical User Interface with Tkinter [Learning Plan Index - Python](https://docs.google.com/document/d/1B5yWb6wCSRhqD42iWxCi7bmLPY2EqvU6pbiEQT0zs20/edit?usp=sharing)    *Unit 09 of Python Programming - Unit 10 Graphical User Interface with Tkinter* | |
| --- | --- |
| Learning Targets  This unit we will…  Learn how to create and work with GUI objects using Tkinter.  I can…   * Create a GUI window and a Canvas Window with Tkinter. * Pack or Grid different elements into a GUI object. * Create GUI windows in class files and associate methods with different GUI objects. * Use the move and after methods of the Canvas class to create animations. * Bind button clicks to a window so that programmed actions can happen on a click.   Vocabulary: GUI, pack, grid, class, method, event binding, Tkinter, labels, buttons, window, Canvas, animation, widgets, frames, widget variables, menus, scrollbars, dialog boxes. | |
| Learn About It!  *You can explore some, or all of these resources. If you want to see a resource again, go for it!*  [Learning Plan Index - Python](https://docs.google.com/document/d/1B5yWb6wCSRhqD42iWxCi7bmLPY2EqvU6pbiEQT0zs20/edit?usp=sharing) *These Collab documents review the concepts of each unit with code you can run and modify.* | |
| Non Collab Learning Plan  Collab documents do not allow for the creation of GUI objects. Repl.it does allow for the creation of GUI objects, so all the sample code in this learning plan will link to sample repl.it's that you can run. You can also create GUI objects on your computer with Pycharm. | |
| GUI with Tkinter   1. Graphical User Interfaces are referred to as GUI (pronounced gooey). In Python the package that contains all the code for creating GUI objects is called Tkinter, so to work with GUI the first thing you have to do is import the Tkinter package:  from tkinter import \* 2. The next thing you need to do is to create a window object using the TK class:  window = Tk() 3. Finally, and this is the **minimum for a GUI window**, you need to have a mainloop() to keep the window active so that is is actively waiting for interaction from the user, using: window.mainloop(), you can [check out this repl.it](https://replit.com/@MrReynolds/GUI01#main.py) **(GUI01)** to see this super basic GUI code. In this case it is window.mainloop() because the object that we created was named window because of, window = TK(). We could have given any name we want to our object, but window makes the most sense since that is what we are making.   Widgets & Frames   1. [Widgets](https://www.tutorialspoint.com/python/python_gui_programming.htm) are the objects like [buttons](https://www.tutorialspoint.com/python/tk_button.htm), and [labels](https://www.tutorialspoint.com/python/tk_label.htm) that you can load into a window. [Frames](https://www.tutorialspoint.com/python/tk_frame.htm) are a widget that are used to contain other widgets. So you will start with frames to act as containers for all of your other widgets. 2. Frames should be your primary container for your other widgets. You should create frames to hold the other widgets that should be grouped together, like a group of buttons. As you create widgets you assign them to a frame you have created. Check out these sites that show examples of how to create frames and load widgets into them [Pythonbasics.org](https://pythonbasics.org/tkinter-frame/) and [Tutorialspoint.com](https://www.tutorialspoint.com/python/tk_frame.htm). The [Tutorialspoint.com](https://www.tutorialspoint.com/python/tk_frame.htm) page lists the different parameters you can pass into a Frame when you create it to set things like height, color, cursor type, and border, to name a few. Another Frame resource that also demonstrates some use of grid() can be [found here at pythonguides.com](https://pythonguides.com/python-tkinter-frame/) 3. After you have created an item you need to load it into the window. If a widget has been assigned to a frame it will be within that fram when you load it.   [pack()](https://www.activestate.com/resources/quick-reads/how-to-use-pack-in-tkinter/) & [grid()](https://riptutorial.com/tkinter/example/29713/grid--#:~:text=tkinter%20grid()&text=The%20grid()%20geometry%20manager,%2C%20row%20%2C%20rowspan%20and%20sticky%20.) & (place())   1. To load things in the window you need to use one of three methods that will set where in the window the widget will go. **Pack()** is a very simple loader with limited commands as far as placing things within the window. Check out this [repl.it example of a simple pack()](https://replit.com/@MrReynolds/GUIPack) **(GUIPack)** example. For a simple window it is fine, but for anything complex you are better off with grid().  **Grid()** uses a row and column system (row & column start at 0) that will give you greater control on laying things out within your window. Check out this [simple repl.it that loads a bunch of buttons using grid()](https://replit.com/@MrReynolds/GUIGrid) **(GUIGrid)**. **Place()** uses absolute values to place things, which can cause layout issues between different operating systems and resolutions. We will not look at place() for this course. 2. Check out [GeeksforGeeks.org for a quick overview](https://www.geeksforgeeks.org/python-grid-method-in-tkinter/) of grid(), **notice the warning at the bottom of the page that says never to mix pack() and grid() when loading widgets into a window** (of course looking at some of the sample code on the sites they seem to mix the two? In some examples it seems like they pack() frames into the window where they want them and then use grid() in the frame to position things where they want). For a much more in depth look at grid() [check out this page at tkdocs.com](https://tkdocs.com/tutorial/grid.html) (keep in mind this site is showing how to use tkinter for four different languages, not just python, the python samples have the python logo next to it). Finally, to see an overview of the three methods used for placing objects in a window pack(), place(), and grid() [check out this page at zetcode.com](https://zetcode.com/tkinter/layout/).   Examples   1. Button & Label Pack **(GUI02)** - If you [look at this simple repl.it](https://replit.com/@MrReynolds/GUI02#main.py) **(GUI02)** you will see that both a button and label widget have been created and packed into the window. This example does not use a frame. 2. Adding actions to buttons **(GUI03)** - this [repl.it shows a function](https://replit.com/@MrReynolds/GUI03) has been added to a button with the line command=buttonClick where buttonClick() is a function that has a simple print command that happens when the button is clicked. You will also notice that some parameters have been added to the buttons and labels, changing the background color of the button and the font color of the label. You can look at the [widget reference](https://www.tutorialspoint.com/python/python_gui_programming.htm) to see all the parameters of the different widgets. 3. Using classes and methods **(GUI04)** - This [repl.it will show how all the code has been moved into a class declaration](https://replit.com/@MrReynolds/GUI04#main.py) and the function is now a method of the class. This code will not happen until a creation of the class object is called at the very bottom of the repl.it code. You will also notice that the keyword self comes into play especially with the line: command=buttonClick which becomes: command=self.buttonClick. 4. More advanced Class and method with variables **(GUI05)** - This [repl.it shows all the code for a window moved into a class with three methods](https://replit.com/@MrReynolds/GUI05#main.py). There are three groups of things, there is a checkbox and two radio buttons loaded into a frame, frame1. There is padding around frame1 so you can clearly see it. The checkbox has been assigned a variable and a method to perform an action when it is clicked. The two radio buttons have also been given a variable and method for action when they are checked. Since the radio buttons share a variable they work with each other as radio buttons, but notice they both have a different value for that variable when active. Next we have a label, entry, button, and message in frame2. The entry has been given a variable and the button has been given a method, when the button is clicked it will print whatever is in the entry. Finally we have a text area with an output message. 5. Another advanced Class and method with variables example **(GUI06)** - [This example is very similar to the last one](https://replit.com/@MrReynolds/GUI06). We have some frames containing other widgets. The widgets have variables and methods assigned to them for added functionality.   Widget Variables   1. Widget variables in Tkinter need to be created and assigned differently than we have been doing so far. First we are going to enclose the variables in objects and we have to decide what data type the variable will hold when we create them. Check out this [reference website](https://www.geeksforgeeks.org/python-setting-and-retrieving-values-of-tkinter-variable/). There are four different types of variables we can use:    1. BooleanVar()    2. StringVar()    3. IntVar()    4. DoubleVar() 2. Variables need to be created and assigned to a widget in order for you to retrieve and manipulate the variables and values associated with a widget. [Check this site for more details](https://programming.vip/docs/tkinter-variable-category-of-python.html). 3. The type of variable you create depends on the widget you are using. If you look at this [checkbox widget](https://www.tutorialspoint.com/python/tk_checkbutton.htm) you will see under the list of parameters in the table that #25 it says that for this widget an IntVar would be the correct sort of variable to use and it has some default values.  If you look at this [entry widget](https://www.tutorialspoint.com/python/tk_entry.htm) you will see that it has something called a textvariable for a parameter at #16 and this widget should use a StringVar() to store any values associated with it.  You can use this [widget reference](https://www.tutorialspoint.com/python/python_gui_programming.htm) to look up any widget to see what sort of variable type should be employed with it. 4. Once you have created a variable it will be available throughout your program like they have been previously. If you are creating your windows in Class declarations, which you should, then those variables should be declared with the self keyword in the init method so that they become class variables that are available throughout the Class object.   Canvas   1. The [Canvas](https://www.tutorialspoint.com/python/tk_canvas.htm) is just one of the widgets that you can add to a window. The canvas is used for drawing on the screen, much like the screen we used for turtle projects. The canvas has a number of methods for drawing different shapes and text to the screen. You can check out [this website](https://pythonbasics.org/tkinter-canvas/) with some sample Canvas code. You can also look at this [repl.it that has a simple Canvas window](https://replit.com/@MrReynolds/GUICanvas) with buttons that allow you to draw different shapes to the screen. Information about the different drawing methods like line, arc, polygon, image, and text can be found at these sites:    1. [Tutorialspoint.com](https://www.tutorialspoint.com/python/tk_canvas.htm)    2. [Pythonguides.com](https://pythonguides.com/python-tkinter-canvas/)    3. [python-course.eu](https://www.python-course.eu/tkinter_canvas.php) 2. The big difference between a turtle window and a Canvas window is the location of the origin point for the X & Y axis. In Canvas the origin (0,0) is in the upper left hand corner with X increasing as it moves to the right and the Y **increasing** as it moves down the page. The Y moving in a positive value as it moves down is the biggest difference and adjustment to the corner origin. The Y moves in a positive direction going down because it makes it easier to work with, you don’t have to think about negative numbers. You will find Canvas in many other languages and it has always worked with the origin being in the upper left-hand corner in every other language where I have seen it.   Animation with Canvas   1. Check out [this sample animation repl.it](https://replit.com/@MrReynolds/GUIAnimationsRect) that has an animation on Canvas that you can control. When you run it you can click on the window and then use the arrow keys to change the direction of the square and the space bar will stop it. This animation uses key bindings to get the arrow keys to work and that is covered later in this document. What I want you to look at is the movement method on line 35. In there is the canvas move method (self.canvas.move(self.rectangle, self.x, self.y). The move method takes an object and two numbers for parameters, every time the move method is called it will move the object on the screen by the two number parameters. The first number will be how much the X will change and the second number will be how much the Y will change. If both numbers are 0 the object will not move. The other method to notice is the after method (self.canvas.after(100, self.movement)). The after method takes a number in milliseconds as the first parameter and that is the delay it will take before calling the method self.movement. In essence this after method causes the movement method to be called over and over so that the square keeps moving on the screen. If you comment the after method out the square will only move once and then stop. The original code for this example came from [geeksforgeeks.org](https://www.geeksforgeeks.org/python-tkinter-moving-objects-using-canvas-move-method/) and I made some modifications. 2. If you [look at this second animation repl.it](https://replit.com/@MrReynolds/GUIAnimations#main.py) you’ll see one major difference. This repl.it has the move method, self.canvas.move("text", self.dx, 0), in a loop on line 76 and the after method is just used as a delay. This version of the move method is using the “**tags**” attribute of the text object to pass into the move method. In this case a variable could also have set equal to = self.canvas.create\_text(self.x, 30, text = "Message moving?", tags = "text") and then that variable could have been passed into the move method. The loop has a boolean trigger to start and stop the animation and the after method takes a single parameter, instead of two like the previous example, so it acts as a delay. The time delay that is passed into it is a class variable which allows for the change of speed functionality self.canvas.after(self.sleepTime).   Images   1. You can add images to your GUI windows, but they should be gif images, although I got a png image to work in this [repl.it example](https://replit.com/@MrReynolds/GUIImages#main.py), the jpg images caused an error. Images need to be declared as variables, which can then be added to a frame, canvas, or button. When you declare your variables you have to specify where the image can be found in your file structure in relation to your program file. In the repl.it example my pictures are in an images folder, which I have to include in my variable declaration. If you look at the example, there are no real actions because there are no assigned methods, but it does demonstrate adding pictures to different elements. You will also notice in the example that there is an added line of code that allows me to resize the images.   Menus   1. GUI windows need file menus at the top. This [sample repl.it shows how to add menus](https://replit.com/@MrReynolds/GUIMenu#main.py) to a window. This example has redundant commands, the add command from the file menu, button with the + picture, and the button at the bottom that says “Add”, all have the same functionality. This is just demonstrating how you could add a file menu and how you could call a method from that menu or a button to execute the command. 2. This site [lists all the parameters and options](http://tutorialspoint.com/python/tk_menu.htm) with file menus and has some sample code. 3. This site also has a [quick overview with some basic samples](https://pythonbasics.org/tkinter-menu/) of creating and working with menus. 4. Here is one more website with some [basic menu examples](https://www.geeksforgeeks.org/python-menu-widget-in-tkinter/).   Popup Menus   1. You can create popup menus that function like regular menus. Check out this [repl.it that has a popup menu](https://replit.com/@MrReynolds/GUIMenuPopup) when you right click the window. The popup menu works very similar to a regular menu. If you look at the code you will see that the right click button has a binding with the popup method so that the menu window will open when you right click the screen.   Scrollbars   1. [Checkout this repl.it](https://replit.com/@MrReynolds/GUIScrollBar#main.py) to see the code needed to add a scrollbar to a gui window. You will notice that this repl.it has a second file which is just another set of code that adds a scrollbar. Check out this [site for a reference](https://www.tutorialspoint.com/python/tk_scrollbar.htm) for the different parameters and methods associated with scrollbars.   Dialog Boxes   1. Check out this [repl.it with a selection of dialog boxes](https://replit.com/@MrReynolds/GUIDialog#main.py). The original code for these boxes came from [geeksforgeeks.org](https://www.geeksforgeeks.org/python-tkinter-messagebox-widget/). All the boxes return a value, the simple boxes with only an “OK” button always return ok. The others will return a value depending on the button that is clicked. You can turn the repl.it to see what values the different buttons return. You could use these values to make decisions in your program based on the user input. At the bottom are three types of input boxes, (string, integer, float), that you can use to get user input in a GUI program. There is also a color chooser, you could also open a file upload window, although you would have to set file settings to allow file uploads, not sure you could make a program that allows you to upload to a repl.it subfile. [Check out this site for details.](https://runestone.academy/runestone/books/published/thinkcspy/GUIandEventDrivenProgramming/02_standard_dialog_boxes.html)   Event Bindings   1. It is possible to bind events like the click of a mouse to some window element so some action happens when a mouse is clicked on that element. [Take a look at this binding example repl.it](https://replit.com/@MrReynolds/GUIBindings#main.py). When you run it you will see a very simple window, when you move your mouse in and out of the window you will see a print statement signifying when you enter and leave the window. If you left click, right click, mouse wheel click, or double click on the window a print statement will tell you the X & Y coordinates of where in the window you clicked. Also a key binding has been set with the window so if you mouseover the window anything you type will be output with a print statement. 2. Here is a link to [geeksforgeeks.com](https://www.pythontutorial.net/tkinter/tkinter-event-binding/) where this code came from with some more explanation of what is happening. Their example is not in a class, but I fixed that. 3. You can also look at this page at [pythontutorials.net](https://www.pythontutorial.net/tkinter/tkinter-event-binding/) where there is more in depth information about key bindings. 4. Finally - I have listed below the most common bindings with a brief description of what they are and how they work.   Common Bindings  <Button-1> - Button 1 is the leftmost button, button 2 is the middle button (where available), and button 3 the rightmost button. <Button-1>, <ButtonPress-1>, and <1> are all synonyms. For mouse wheel support under Linux, use Button-4 (scroll up) and Button-5 (scroll down)  <B1-Motion> - The mouse is moved, with mouse button 1 being held down (use B2 for the middle button, B3 for the right button).  <ButtonRelease-1> Button 1 was released. This is probably a better choice in most cases than the Button event, because if the user accidentally presses the button, they can move the mouse off the widget to avoid setting off the event.  <Double-Button-1> Button 1 was double clicked. You can use Double or Triple as prefixes.  <Enter> The mouse pointer entered the widget (this event doesn’t mean that the user pressed the Enter key!).  <Leave> The mouse pointer left the widget.  <FocusIn> Keyboard focus was moved to this widget, or to a child of this widget.  <FocusOut> Keyboard focus was moved from this widget to another widget.  <Return> The user pressed the Enter key. For an ordinary 102-key PC-style keyboard, the special keys are Cancel (the Break key), BackSpace, Tab, Return(the Enter key), Shift\_L (any Shift key), Control\_L (any Control key), Alt\_L (any Alt key), Pause, Caps\_Lock, Escape, Prior (Page Up), Next (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num\_Lock, and Scroll\_Lock.  <Key> The user pressed any key. The key is provided in the char member of the event object passed to the callback (this is an empty string for special keys).  a - The user typed an “a”. Most printable characters can be used as is. The exceptions are space (<space>) and less than (<less>). Note that 1 is a keyboard binding, while <1> is a button binding.  <Shift-Up> The user pressed the Up arrow, while holding the Shift key pressed. You can use prefixes like Alt, Shift, and Control.  <Configure> The widget changed size (or location, on some platforms). The new size is provided in the width and height attributes of the event object passed to the callback.  <Activate> A widget is changing from being inactive to being active. This refers to changes in the state option of a widget such as a button changing from inactive (grayed out) to active.  <Deactivate> A widget is changing from being active to being inactive. This refers to changes in the state option of a widget such as a radio button changing from active to inactive (grayed out).  <Destroy> A widget is being destroyed.  <Expose> This event occurs whenever at least some part of your application or widget becomes visible after having been covered up by another window.  <KeyRelease> The user let up on a key.  <Map> A widget is being mapped, that is, made visible in the application. This will happen, for example, when you call the widget's .grid() method.  <Motion> The user moved the mouse pointer entirely within a widget.  <MouseWheel> The user moved the mouse wheel up or down. At present, this binding works on Windows and MacOS, but not under Linux.  <Unmap> A widget is being unmapped and is no longer visible.  <Visibility> Happens when at least some part of the application window becomes visible on the screen. | |



Would you like a mint? It is only wafer thin.