Worksheet 8 – GUI Layout & Widgets

In this worksheet we are going to learn how to take better control of the layout of our GUI elements, and discover how to add a few new widgets such as entry boxes, images, radio buttons, and checkboxes.

Entry boxes

Below is a program that creates an entry box that the user can type in to. When the user clicks the button that is created on line 8, whatever the user wrote in the entry box is printed to the output window:

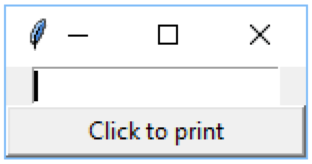
```python
1   from tkinter import *
2
3   def main():
4       global yourname
5       root = Tk()
6       yourname = Entry(root)
7       yourname.pack()
8       Button(root, text="Click to print", width=20, command=nameprint).pack()
9       root.mainloop()
10
11  def nameprint():
12      print (yourname.get())
13
14  main()
```

This program works by defining a text entry box with the code `yourname = Entry(root)` (line 6), and then getting the contents of the entry box so that it can be printed using the `yourname.get()` command (line 12).

Note that the entry box is assigned to the variable `yourname`. This variable could have been named anything, and the reason we have assigned it to a variable is so we can later use the `get()` method to get the contents of the box.

The `get()` method simply returns a value from a variable, in this case, the entry box.

You can see what the running program looks like below:



Putting elements on the canvas

The program above is missing a canvas. Let’s add one, then change the code so that our entry box and button are on the canvas instead of above it.

```
 1  from tkinter import *
 2
 3  def main():
 4      global yourname
 5      root = Tk()
 6      canvas = Canvas(root, width=450, height=300, bg = "grey")
 7      yourname = Entry(root)
 8      entrybutton = (Button(root, text="Click to print", width=20, command=nameprint))
 9      canvas.create_window(100, 70, width=100, window=yourname)
10      canvas.create_window(100, 100, width=100, window=entrybutton)
11      canvas.pack()
12      root.mainloop()
13
14  def nameprint():
15      print (yourname.get())
16
17  main()
```

There are some key differences in this program that allow elements to be created on the canvas. Let’s explore these below:

Line 6: We create the canvas in the same way that we have in previous programs we have written – by creating a canvas variable that holds a Canvas with specified size and background parameters.

Line 7: Notice that in the first program on line 7, we used `yourname.pack`, however in this program this is no longer needed. The reason for this is that this and any other elements are going to be on the canvas and will be ‘packed’ along with it, so they <u>do not need to be packed individually</u>.

Line 8: We have now assigned our button to a variable called “`entrybutton`”. This is important as placing the button inside of a variable makes it much easier to place on the canvas. Also note how `.pack` has been removed from the end of the line, and an extra set of brackets has been added to allow the button to be assigned to the new `entrybutton` variable.

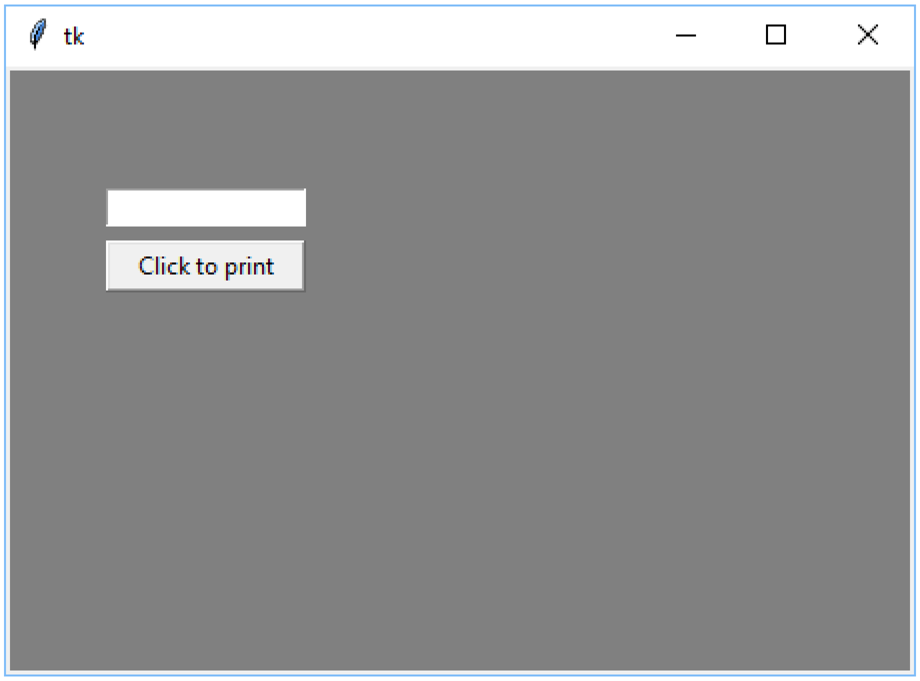
Note also that `width=20` could also be removed, as this will be overridden by the width we specify for the canvas window object (explained below).

Line 9: `canvas.create_window` is used to place something inside of the canvas using a *canvas window* object. A window is a rectangular area that can hold **one** Tkinter widget. In this case, we are using it to hold the `yourname` Entry box.

The `100` and `70` values are the respective x and y placement coordinates, `width` is the size of the window, and `window=` tells Python which widget to place in the window.

Line 10: As mentioned, canvas window objects can only hold one widget at a time, so here we create a second window to hold the clickable button that we have created (`entrybutton`).

You can see what the running program looks like below:



Now let's add a quit button to this program, as it is a good idea to include one on any GUI programs that we write:

```python
from tkinter import *

def main():
    global root
    global yourname
    root = Tk()
    canvas = Canvas(root, width=450, height=300, bg = "grey")
    yourname = Entry(root)
    entrybutton = (Button(root, text="Click to print", command=nameprint))
    quitbutton = (Button(root, text="Quit", command=quit))
    canvas.create_window(100, 70, width=100, window=yourname)
    canvas.create_window(100, 100, width=100, window=entrybutton)
    canvas.create_window(100, 130, width=100, window=quitbutton)
    canvas.pack()
    root.mainloop()

def nameprint():
    print (yourname.get())

def quit():
    root.destroy()

main()
```

The additions to the program we have made are:

Line 4: root needs to be specified as global, as we will be interacting it with the new quit function that we create on lines 20 and 21.

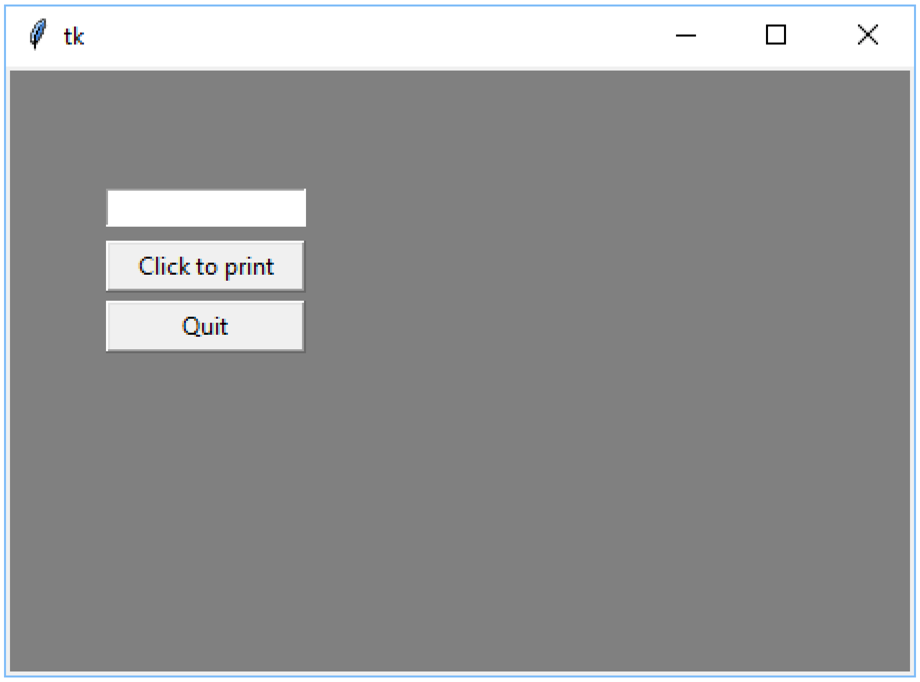
Line 10: Creates a new button that calls a function called `quit`, and assigns it to the variable `quitbutton`.

Note that this button does not have a width, and the width for the button on line 9 has also been removed. As mentioned, the button widths are no longer required as they are overridden by the canvas window specified on lines 12 and 13.

Line 13: Here we are adding another canvas window object to place `quitbutton` on the canvas.

Lines 20 & 21: This is the function that will run when the quit button is clicked.

You can see what the running program looks like below (note the newly added Quit button):



If you want to reduce the amount of code your program uses, it is possible to combine a line that creates a button with the line that creates the canvas window object. For example:

```
entrybutton = (Button(root, text="Click to print", command=nameprint))

canvas.create_window(100, 100, width=100, window=entrybutton)
```

Becomes:

```
canvas.create_window(100, 100, width=100, window=(Button(root, text="Click to
print", command=nameprint)))
```

Using grids

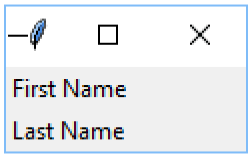
Grids are a powerful alternative to placing buttons on a canvas which let you specify where items are placed on the GUI using an easy system of **rows** and **columns**. To use grids, we use `.grid` instead of `.pack`.

Let's begin by making a simple grid that creates two rows with labels in our root window. Note that labels are used to display text that does not do anything (unlike entry boxes, which can be interacted with).

Program code

```
from tkinter import *

def main():
    root = Tk()
    Label(root, text="First Name").grid(row=0)
    Label(root, text = "Last Name").grid(row=1)
    root.mainloop()

main()
```

Program running



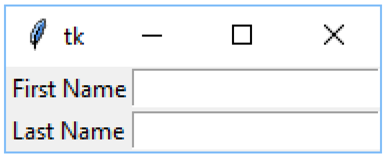
```
First Name
Last Name
```

Now let's expand the program by adding a second column which contains data entry boxes next to our existing labels:

Program code

```
from tkinter import *

def main():
    root = Tk()
    Label(root, text="First Name").grid(row=0, column=0)
    Label(root, text = "Last Name").grid(row=1, column=0)

    entry1 = Entry(root)
    entry2 = Entry(root)

    entry1.grid(row=0, column=1)
    entry2.grid(row=1, column=1)

    root.mainloop()

main()
```

Program running



First Name

Last Name

Let’s look at the differences in the second program:

Lines 5 & 6: Note that both of these lines now specify both the rows *and* the column to place the labels in (one in the first row, one in the second row, both in the first column). Although this is not necessarily required, specifying the column to use is a good habit to get in to if you plan to use more than one column as we have in this program.

Lines 8 & 9: Create two entry boxes and assign them to variables (`entry1` and `entry2`).

Lines 11 & 12: Specify the position for the two entry boxes (one in the first row, one in the second row, both in the second column).

By looking at the examples above, you may have noticed that rows and columns in Python are similar to tables or spreadsheets – the main differences are that there are no lines, and the row and column numbers start at ‘0’ instead of ‘1’.

Task

Add a third row to the program above, and put a **quit** button in it.

The answer is shown on the next page, but try it yourself before having a look!

Task 1 answer

```
1   from tkinter import *
2
3   def main():
4       global root
5       root = Tk()
6       Label(root, text="First Name").grid(row=0, column=0)
7       Label(root, text = "Last Name").grid(row=1, column=0)
8
9       entry1 = Entry(root)
10      entry2 = Entry(root)
11
12      entry1.grid(row=0, column=1)
13      entry2.grid(row=1, column=1)
14
15      Button(root, text="Quit", command=quit).grid(row=2, column=0)
16
17      root.mainloop()
18
19  def quit():
20      root.destroy()
21
22  main()
```

The additions to the program for this task are:

Line 4: `root` needs to be global if the `quit` function is going to be able to see it to close it.

Line 15: Adding of a button to run the `quit` function. Note that `.grid` is used to specify where the button will appear.

Lines 19 & 20: The `quit` function which will close the `root` window when it is run.

Adding a Print button

The code on the following page adds a print button to the third row, and uses the `.get` method to make it work. An explanation of the code is also provided underneath:

Program code

```
1   from tkinter import *
2
3   def main():
4       global root
5       global entry1
6       global entry2
7       root = Tk()
8       Label(root, text="First Name").grid(row=0, column=0)
9       Label(root, text = "Last Name").grid(row=1, column=0)
10
11      entry1 = Entry(root)
12      entry2 = Entry(root)
13
14      entry1.grid(row=0, column=1)
15      entry2.grid(row=1, column=1)
16
17      Button(root, text="Quit", command=quit).grid(row=2, column=0)
18      Button(root, text="Print", command=printname).grid(row=2, column=1)
19
20      root.mainloop()
21
22  def quit():
23      root.destroy()
24
25  def printname():
26      firstname = entry1.get()
27      lastname = entry2.get()
28      print("Your name is {} {}".format(firstname, lastname))
29
30  main()
```

Lines 5 & 6: `entry1` and `entry2` are made global so that the newly added `printname` function can see them.

Line 18: Creates a button to run the `printname` function and places it in row 3, column 2.

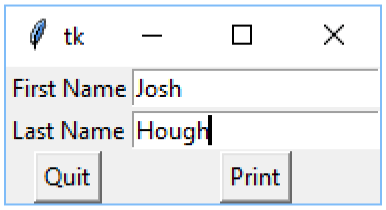
Line 25: Creates a new function called `printname`

Line 26: Creates a new variable called `firstname`, and assigns user input to it by “getting” it from the `entry1` box.

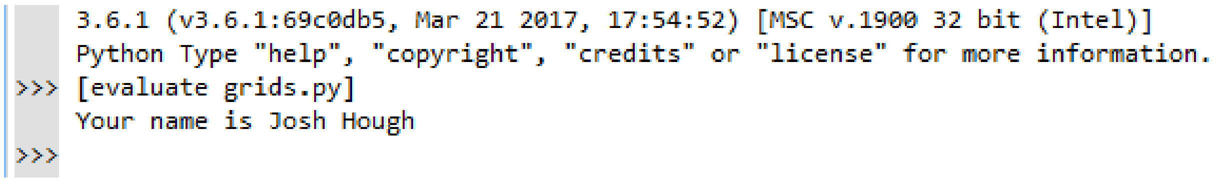
Line 27: Repeats the process as above, only creates a variable called `lastname` and gets the input from the `entry2` box.

Line 28: Prints out a string of text with the values of `firstname` and `lastname` inserted after the text. Note that string formatting is used here to ensure that the output displays properly. For more on string formatting, see **this link**.

Program running



Program output



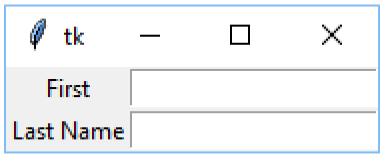
```
    3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
    Python Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate grids.py]
    Your name is Josh Hough
>>>
```

Sticky and Pad

"Sticky" lets you control the alignment of text (and pictures) within the grid, and "Pad" lets you put padding (empty space) around things. The following sections explain how these can be used to improve the formatting of our grid.

Sticky

By default, elements in a grid are placed in the 'middle' of the row. For example, if we wrote the above program but changed the text of the first label from "First Name" to just "First", the program would display as follows:

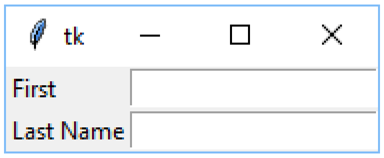


Notice how "First" is centre aligned in the row. "Last Name" appears left aligned, but only because it fills the width of the row (it is in actuality also centre aligned).

Using sticky, we can modify the code as shown on the next page:

```
1   from tkinter import *
2
3   def main():
4       root = Tk()
5       Label(root, text="First").grid(row=0, column=0, sticky=W)
6       Label(root, text = "Last Name").grid(row=1, column=0, sticky=W)
7
8       entry1 = Entry(root)
9       entry2 = Entry(root)
10
11      entry1.grid(row=0, column=1)
12      entry2.grid(row=1, column=1)
13
14      root.mainloop()
15
16  main()
```

Notice how `sticky=W` has been added to the label grid properties on lines 5 and 6. Adding this causes the program to display as follows:



Note how “First” is now left aligned in the row.

Although “Last Name” looks the same as the example on the previous page, it has also been left aligned. This is a good habit to get in to, as even though we can’t see a difference with the alignment of this particular label, we would if the column was made wider elsewhere in the program.

Note: If you wanted to align the label to the right, you would use “E” instead of “W”.

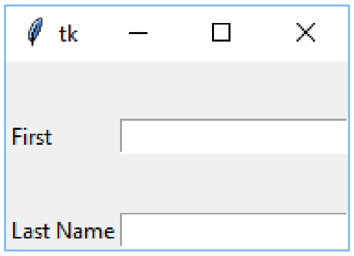
Pad

`padx` and `pady` are properties which can be used to place padding (space) around elements on our GUI. `padx` adds padding to the right and left, and `pady` adds padding above and below.

For example, note how line 5 of the program above has been modified in the example below to include `pady` of 30:

```
5       Label(root, text="First").grid(row=0, column=0, sticky=W, pady=30)
```

This causes the Label element to display with 15 padding above and 15 padding below the first label element, which causes the program to display as follows:



Note that 30 padding is rather too much, and causes the GUI to look unbalanced. Use trial and error and your own sense of design when deciding how much padding to apply to elements on your interface.

Adding images

The code below can be used to add an image and control it with the grid:

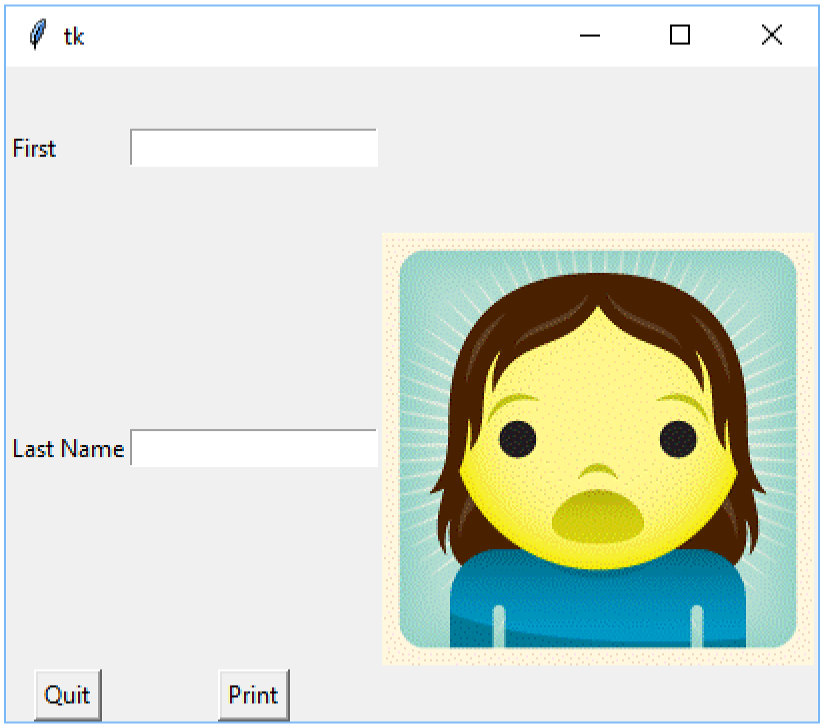
```
1   from tkinter import *
2
3   def main():
4       global root
5       global entry1
6       global entry2
7       root = Tk()
8       Label(root, text="First").grid(row=0, column=0, sticky=W, pady=30)
9       Label(root, text = "Last Name").grid(row=1, column=0, sticky=W)
10
11      photo = PhotoImage (file="E:\Ara\person2.gif")
12      imageplace = Label(root, image=photo)
13      imageplace.grid(column=3, row=1)
14
15      entry1 = Entry(root)
16      entry2 = Entry(root)
17
18      entry1.grid(row=0, column=1)
19      entry2.grid(row=1, column=1)
20
21      Button(root, text="Quit", command=quit).grid(row=2, column=0)
22      Button(root, text="Print", command=printname).grid(row=2, column=1)
23
24      root.mainloop()
25
26  def quit():
27      root.destroy()
28
29  def printname():
30      firstname = entry1.get()
31      lastname = entry2.get()
32      print("Your name is {} {}".format(firstname, lastname))
33
34  main()
```

Line 11: Creates a variable called `photo` that has assigned to it the specified image. Note that while the variable name could be changed, `PhotoImage` is a command and cannot be substituted for other wording. Also note that the file path `"E:\Ara\person2.gif"` should be modified depending on the location of the image file.

Line 12: Labels are used to insert both text **and** images onto the GUI, so a label called `imageplace` is created, and the `photo` variable is assigned to it.

Line 13: The `imageplace` label that contains the image we want to insert is now placed on the grid in column 3, row 1.

Program running



It is important to note that the image displayed is the actual size of the image, so you will need to size images you wish to insert correctly first. Also worth noting is that your images must be in .gif format, as this is all PhotoImage supports!

Image spanning and padding

The program above doesn't look very well designed as the image has extended the height of the second row by a significant amount.

To get around this, we can use a few techniques:

- Reducing the size of the image
- Using the `rowspan` command so the image can take up more than one row (like merging cells in a table)
- Using padding to move the image away from the input boxes to make it look neater

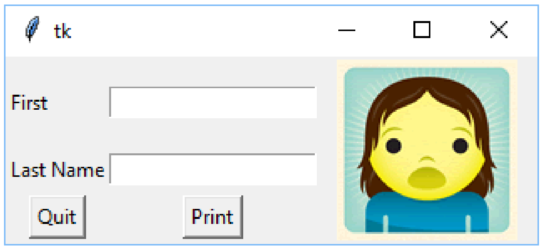
All three techniques have been applied in the code shown below:

```
11        photo = PhotoImage (file="E:\Ara\person2_half_size.gif")
12        imageplace = Label(root, image=photo)
13        imageplace.grid(column=3, row=0, rowspan=3, padx=10)
```

Line 11: The filepath has been changed to point to an image file that is smaller than the one previously used.

Line 13: The row has been changed to 0 so the image is inserted on the first row instead of the second, `rowspan=3` allows the image to span all 3 rows of the grid, and `padx` adds 5 spacing to the left and right of the image.

Although not shown in the code sample above, the label `pady` on line 8 has also been changed to 15 instead of 30.



Radio buttons and checkboxes

As mentioned in worksheet 7, radio boxes allow the user to select only one from a number of options, and check boxes allow the user to select more than one option at the same time.

For example, radio buttons could be used for questions such as “Are you a student?”, and checkboxes could be used for questions such as “Choose your ice cream toppings: sprinkles, chocolate sauce, bananas”.

Radio buttons

When creating radio buttons, you use a line for each option and call a subroutine to pass a value. For example:

```
1   from tkinter import *
2
3   def select():
4       print ("You chose {}".format(var.get()))
5
6   def main():
7       global var
8       root = Tk()
9       var = IntVar()
10      Radiobutton(root, text="Button One", variable=var, value=1, command=select).pack(anchor = W)
11      Radiobutton(root, text="Button Two", variable=var, value=2, command=select).pack(anchor = W)
12      Radiobutton(root, text="Button Three", variable=var, value=3, command=select).pack(anchor = W)
13
14      root.mainloop()
15
16  main()
```

Lines 3 & 4: Define a function which, when run, gets the value of the `var` variable and prints it.

Line 7: Makes the `var` variable global so it can be seen by the `select` function.

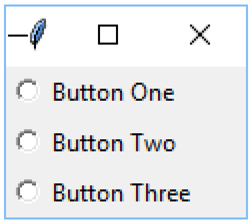
Line 9: Creates a variable called `var`, and assigns it the type `IntVar` which is a **variable class**. We use a variable class here as the Checkbutton and Radiobutton widgets require variable classes to work properly.

Note that if this radio button was being set up to work with text values rather than number values, we could have assigned the type `StringVar` rather than `IntVar`.

Line 10: Creates a Radiobutton widget that points to the `var` variable that was set up on line 9. The Radiobutton contains a `value` command which specifies the value that will be set when the Radiobutton is clicked (in this case, the value is set to 1). `command=select` specifies the function that will be run when the Radiobutton is clicked (in this program, the select function).

Lines 11 & 12: Create two more Radiobutton widgets that specify values of 2 and 3. Note how all three Radiobuttons point to the same variable (`variable=var`) – this is done as to get proper radio behavior, all buttons in a group must point to the same variable.

Program running



Program output

```
   3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
   Python Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate radiobuttons.py]
   You chose 1
```

Displaying radio button output

Below is another version of the program that displays the value you click in the GUI instead of printing it to the output window:

Program code

```
from tkinter import *

def select():
    selection = "You selected option " + var.get()
    buttons.config(text = selection)

def main():
    global var
    global buttons
    root = Tk()
    var = StringVar()
    Radiobutton(root, text="Button One", variable=var, value=1, command=select, indicatoron=0, width=20).pack(anchor = W)
    Radiobutton(root, text="Button Two", variable=var, value=2, command=select, indicatoron=0, width=20).pack(anchor = W)
    Radiobutton(root, text="Button Three", variable=var, value=3, command=select, indicatoron=0, width=20).pack(anchor = W)

    buttons = Label(root)
    buttons.pack()

    root.mainloop()

main()
```

Line 4: Defines a variable called `selection` that contains a string of text, and the value of the `var` variable.

Line 5: Formats the content of the `buttons` variable (which is a label) to contain the text specified in the `selection` variable.

Line 9: Makes the `buttons` variable global so it can be seen by the `select` function.

Line 11: Note that `var` is now `StringVar` rather than `IntVar`. This is because the contents of the variable will be added to a string, so we want the variable to be a string to begin with to avoid any errors. If for any reason we wanted to keep the `var` variable as an integer, we could have left `var` as `IntVar`, and changed line 4 to:

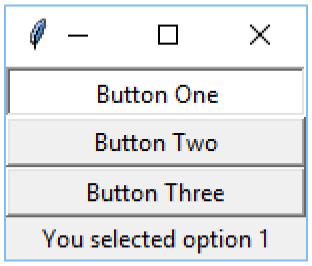
```
    selection = "You selected option " + str(var.get())
```

Line 12: Creates the Radiobutton widgets as before, only now `indicatoron=0` is used to change the style of the buttons from the default style, and the buttons have a width of 20 applied to them.

For Mac users, note that the indicatoron style does not always work on OSX depending on the version used.

Lines 17 & 18: Create the buttons label and pack it.

Program running



You selected option 1

Checkboxes

Checkboxes need to be handled a little differently than radio buttons, as we don't want to run a command each time we check a box. Instead, it is better to work out what was checked afterwards.

Take a look at the program on the next page – it includes two checkbox buttons, a Next button, and *IF* statements to control what happens if the boxes are checked when the Next button is pressed:

```
 1  from tkinter import *
 2  
 3  def nextbutton():
 4      if check1.get()==1:
 5          print ("You selected Music.")
 6      if check2.get()==1:
 7          print ("You selected Video.")
 8  
 9  def main():
10      global check1
11      global check2
12      root = Tk()
13  
14      check1 = IntVar()
15      check2 = IntVar()
16  
17      c1 = Checkbutton(root, text="Music", variable=check1, width=20).pack()
18      c2 = Checkbutton(root, text="Video", variable=check2, width=20).pack()
19      Button(root, text="Next", width=8, command=nextbutton).pack()
20  
21      root.mainloop()
22  
23  main()
```

Line 3: Defines a function which will run when the Next button is clicked

Lines 4 & 5: Create an IF statement that triggers if the value of `check1` is equal to 1 (in other words, if the `check1` value is ticked. Look below for what this variable is assigned to).

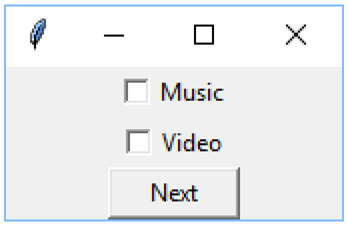
Lines 6 & 7: The same as lines 4 & 5, only this IF statement triggers if the value of `check2` is 1.

Lines 10 & 11: Make the `check1` and `check2` variables global so they can be seen by the `nextbutton` function.

Lines 14 & 15: Set up two variables, `check1` and `check2`, as variable classes (remember that both radio buttons and checkboxes need these to work correctly).

Lines 17 & 18: Creates two checkboxes in two newly created variables (`c1` & `c2`), and assigns the `check1` and `check2` variables to them. This allows `check1` and `check2` to be tested to see if they are checked or not by the IF statements on lines 4 to 7.

Program running



Output

```
You selected Music.
```

Combining and using multiple boxes

With checkboxes you can have as many boxes as you want, so there is no issue about having multiple sets of boxes. With radio buttons however, you can only select one box from the set of options, so there needs to be a way of having more than one set at a time if you wish to display multiple groups of radio buttons.

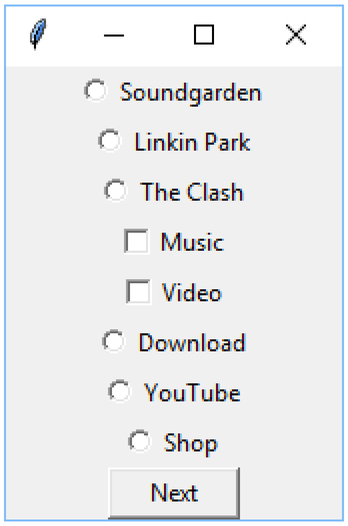
The solution to this is simple – change the variable name that is assigned to the radio buttons.

The example below shows a program that has two sets of radio buttons, and one set of checkboxes. What's important to note here is that each set of radio buttons has a different variable (`artist` and `media`) that are applied to them:

Program code

```
from tkinter import *

def main():
    global artist, media, check1, check2
    root = Tk()
    root.title("Music Selection")

    #Sets up radiobutton group 1
    artist = IntVar()
    Radiobutton(root, text="Soundgarden", variable=artist, value=1, command=selecta, width=20).pack(anchor = W)
    Radiobutton(root, text="Linkin Park", variable=artist, value=2, command=selecta, width=20).pack(anchor = W)
    Radiobutton(root, text="The Clash", variable=artist, value=3, command=selecta, width=20).pack(anchor = W)

    #Sets up checkbox group
    check1 = IntVar()
    check2 = IntVar()
    c1 = Checkbutton(root, text="Music", variable=check1, width=20).pack()
    c2 = Checkbutton(root, text="Video", variable=check2, width=20).pack()

    #Sets up radiobutton group 2
    media = IntVar()
    Radiobutton(root, text="Download", variable=media, value=1, command=selectb, width=20).pack(anchor = W)
    Radiobutton(root, text="YouTube", variable=media, value=2, command=selectb, width=20).pack(anchor = W)
    Radiobutton(root, text="Shop", variable=media, value=3, command=selectb, width=20).pack(anchor = W)

    #Next button to interact with checkbox
    Button(root, text="Next", width=8, command=nextbutton).pack()

    root.mainloop()

#Function to interact with checkbox
def nextbutton():
    if check1.get()==1:
        print ("You selected Music.")
    if check2.get()==1:
        print ("You selected Video.")

#Function to interact with radiobutton group 1
def selecta():
    print ("You selected artist option " + str(artist.get()))

#Function to interact with radiobutton group 2
def selectb():
    print ("You selected video option " + str(media.get()))

main()
```

Program running:



A bit of customising

On the next page is a code example which shows the above program recreated using the grid instead of .pack.

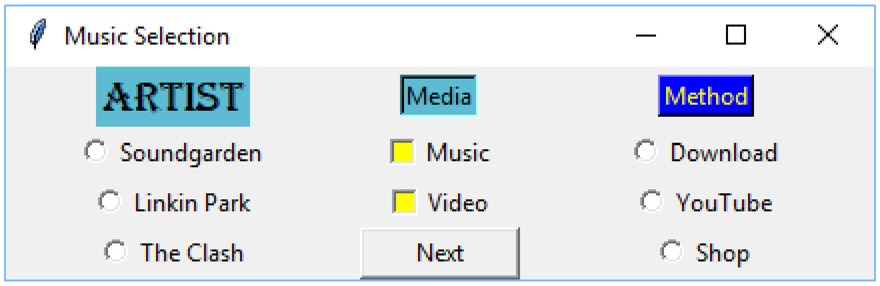
The radio buttons, checkboxes, and buttons have been customised throughout to change the look of the program.

Note that while this program is not very pretty, it does display a range of customisation options!

Program code

```python
from tkinter import *

def main():
    global artist, media, check1, check2
    root = Tk()
    root.title("Music Selection")

    #Sets up radiobutton group 1
    artist = IntVar()
    Label(root, text="Artist", bg="green", borderwidth=2, font="algerian 16").grid(row=0, column=0)
    Radiobutton(root, text="Soundgarden", variable=artist, value=1, command=selecta, width=20).grid(row=1, column=0, sticky=W)
    Radiobutton(root, text="Linkin Park", variable=artist, value=2, command=selecta, width=20).grid(row=2, column=0, sticky=W)
    Radiobutton(root, text="The Clash", variable=artist, value=3, command=selecta, width=20).grid(row=3, column=0, sticky=W)

    #Sets up checkbox group
    check1 = IntVar()
    check2 = IntVar()
    Label(root, text="Media", bg="green", relief="sunken").grid(row=0, column=1)
    c1 = Checkbutton(root, text="Music", variable=check1, width=10, selectcolor="yellow").grid(row=1, column=1, sticky=W)
    c2 = Checkbutton(root, text="Video", variable=check2, width=10, selectcolor="yellow").grid(row=2, column=1, sticky=W)

    #Sets up radiobutton group 2
    media = IntVar()
    Label(root, text="Method", bg="blue", relief="raised", fg="yellow").grid(row=0, column=2)
    Radiobutton(root, text="Download", variable=media, value=1, command=selectb, width=20).grid(row=1, column=2, sticky=W)
    Radiobutton(root, text="YouTube", variable=media, value=2, command=selectb, width=20).grid(row=2, column=2, sticky=W)
    Radiobutton(root, text="Shop", variable=media, value=3, command=selectb, width=20).grid(row=3, column=2, sticky=W)

    #Next button to interact with checkbox
    Button(root, text="Next", width=10, command=nextbutton, overrelief="sunken", activebackground="blue").grid(row=3, column=1)

    root.mainloop()

#Function to interact with checkbox
def nextbutton():
    if check1.get()==1:
        print ("You selected Music.")
    if check2.get()==1:
        print ("You selected Video.")

#Function to interact with radiobutton group 1
def selecta():
    print ("You selected artist option " + str(artist.get()))

#Function to interact with radiobutton group 2
def selectb():
    print ("You selected video option " + str(media.get()))

main()
```

Program running



Support with what you have learned

A variety of new tkinter techniques have been introduced in this worksheet. If you encounter some errors when applying these techniques – don't worry, it's pretty common when starting out!

Fortunately, there exists a wealth of information on the internet around tkinter and how its various features and widgets work that can be used to help us get our programs working correctly.

Although we encourage you to do your own research to find examples which make the most sense to you, below are a few links from the excellent **effbot.org An Introduction to Tkinter** that you may find useful.

Grid – explains the Grid geometry manager, and includes detailed information on the methods that can be applied.

Radio buttons – Goes into extensive detail on the Radiobutton widget, and shows how to use a loop to create a large number of radio buttons.

Checkboxes – Detailed information on checkboxes, with examples of how a class can be used with the checkbutton variable.