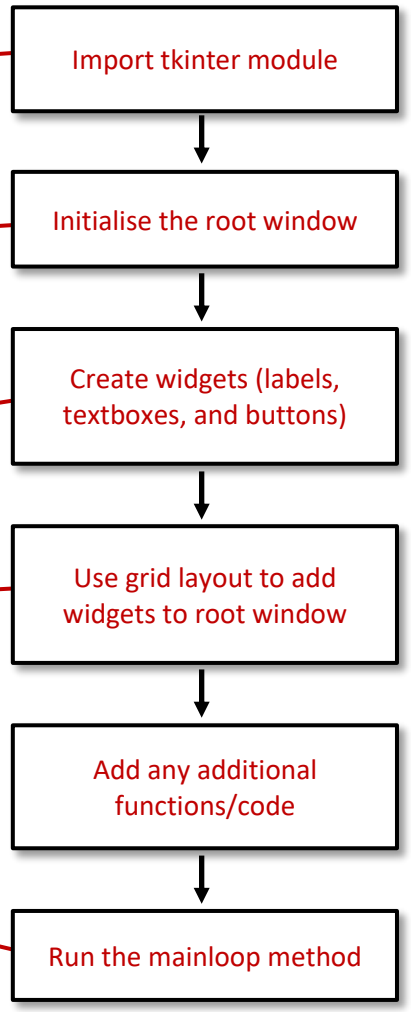
15: SIMPLE GUI USING TKINTER

There are several applications that can be used with Python to create a GUI (graphical user interface). This short tutorial uses the standard version that comes with Python 3. It is very simple to use to create basic functional interfaces with text boxes, buttons and forms.

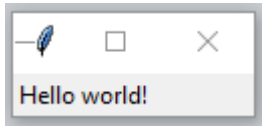
15.1 CREATING THE GUI

All the code examples below will follow the same basic structure:

```
1       from tkinter import *
2
3       # create the basic window first
4       root = Tk()
5
6       # create a label
7       my_label = Label(root, text="Hello world!")
8
9       # add the label to the root window
10      my_label.grid(row=0, column=1)
11
12      # the loop runs the screen and reacts to user actions
13      root.mainloop()
```

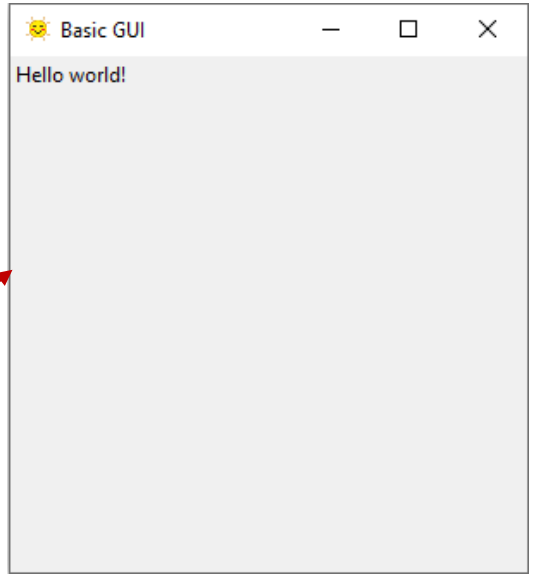


When the code is executed, this small window appears with the label:



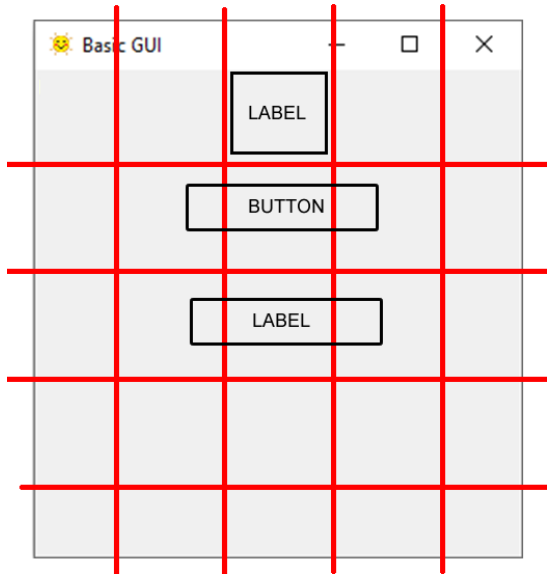
The size of the root window can be adjusted and a title and title bar image can be added:

```
1   from tkinter import *
2
3   # create the basic window first
4   root = Tk()
5
6   # edit the basic window
7   root.title("Basic GUI")
8   root.geometry("300x300")
9   root.iconphoto(False, PhotoImage(file='images/smiley.png'))
10
11  # create a label
12  my_label = Label(root, text="Hello world!")
13
14  # add the label to the root window
15  my_label.grid(row=0, column=1)
16
17  # the loop runs the screen and reacts to user actions
18  root.mainloop()
```



In this example, it is easier to see that the label is in row 0 and column 1, column 0 is empty.

In the next example, the GUI will have an image on a label and a button that displays text on a second label when the button is clicked. This shows how the window will appear →



By default, each widget (label, text entry, button, etc.) takes up one column or row. We can use some additional features of the geometry manager to improve the look of the widgets on the GUI.


```
1   from tkinter import *
2
3   # create the basic window first
4   root = Tk()
5
6   # edit the basic window
7   root.title("Basic GUI")
8   root.geometry("300x300")
9   root.iconphoto(False, PhotoImage(file='images/smiley.png'))
10
11  # set the image for the label
12  logo = PhotoImage(file='images/globe.png')
13  # create a label
14  logo_label = Label(root, image=logo)
15
16
17  # create a function to control clicking the button
18
19  def btn_click():
20      txt_label = Label(root, text="Hello world!")
21      txt_label.grid(row=3, column=1, columnspan=3, padx=20, pady=20)
22
23
24  # create a button
25  my_btn = Button(root, text="Click me!", command=btn_click)
26  my_btn.grid(row=2, column=1, columnspan=2, padx=20, pady=20)
27
28  # add the label to the root window
29  logo_label.grid(row=0, column=2, padx=80, pady=20)
30
31
32  # the loop runs the screen and reacts to user actions
33
34  root.mainloop()
```

Globe Image

Lines 11–14

The variable logo is created on line 12, using the PhotoImage() function used in Line 9 to add the icon to the root window. The image is then applied to the label in line 14.

Lines 28–29

The Label widget is placed onto the window using the geometry manager, and additional padding has been added in an X and Y direction around the label to ensure it appears towards the centre of the window.

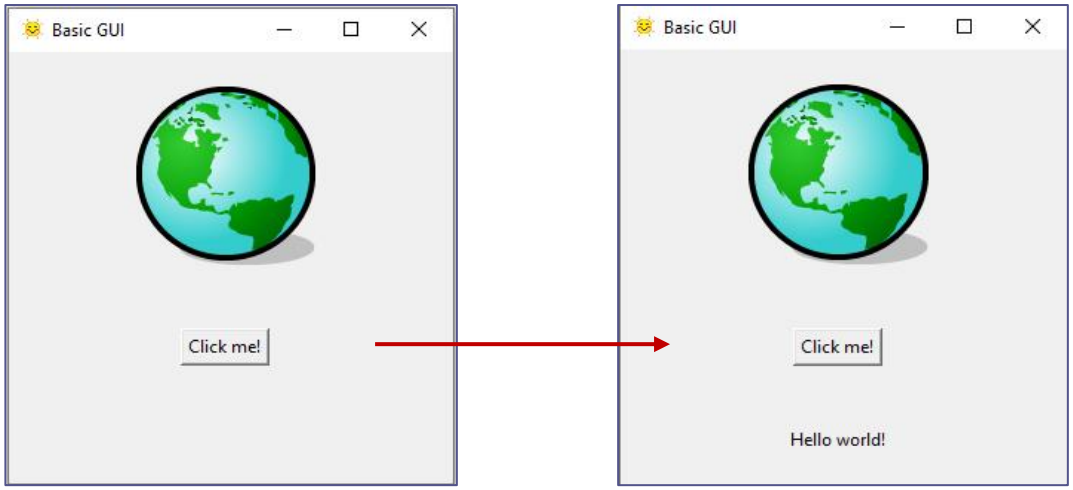
Lines 19–21

This function controls what happens when the button created on line 25 is clicked. The function must appear BEFORE the button is created as the button references the function. This label widget is allowed to cover three columns by using the geometry manager feature 'columnspan' and setting the number of columns.

Lines 25–26

The button is created with the text and the function is linked using the command.

When the code is executed, the window on the left is shown; clicking the button makes the text appear in the label below.



The next improvement is to add a text input box to ask for a name and then include the name in the message shown in the label when the button is clicked.

Here is a section of the code showing the additions, which also involved amending which row numbers to fit in the extra label and text entry.

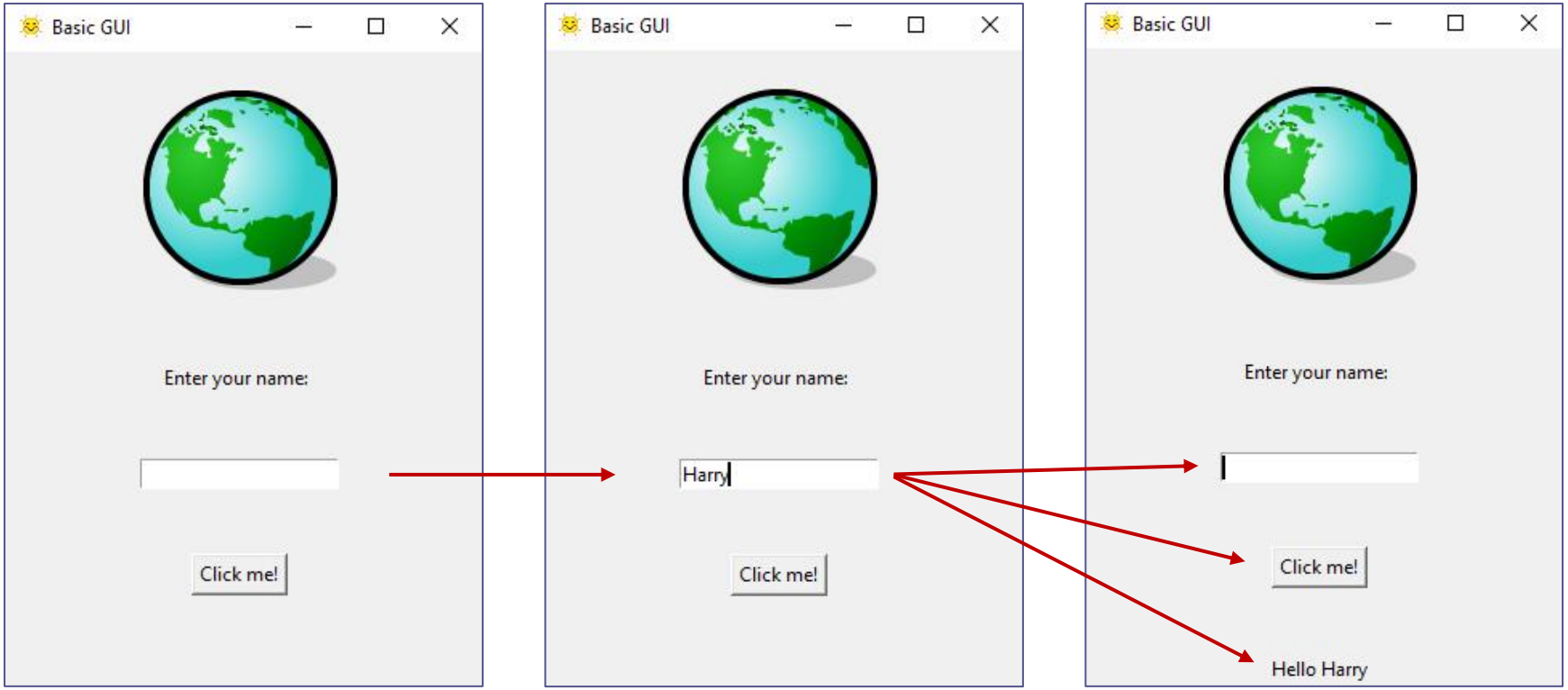
```
16      # add a text entry
17      e = Entry(root)
18      e.grid(row=4, column=1, padx=20, pady=20)
19
20      # add label for text entry
21      e_label = Label(root, text="Enter your name: ")
22      e_label.grid(row=3, column=1, padx=20, pady=20)
```

The function has been amended to get the text entry (line 28) and concatenate the text variable with the label text on line 29. Finally, the data in the text entry box is deleted on line 31.

In addition, the height of the window has been changed.

```
27      def btn_click():
28          name = e.get()
29          txt_label = Label(root, text="Hello " + name)
30          txt_label.grid(row=6, column=1, columnspan=3, padx=20, pady=20)
```

Here is the result of the code additions:



EXERCISE 40: SIMPLE GUI QUIZ

Create a GUI that will:

1. Include a suitable title and icon in the GUI window
2. Display a picture of a famous landmark, e.g. the Eiffel Tower
3. Ask the user to enter the name of the city where the landmark is situated
4. If the answer is correct, display suitable text message
5. If the answer is wrong, ask the user to try again

Hint: remember to store your images in the same folder as your Python file. You will need to amend row positions, column positions and padding to correctly display your GUI.

15.2 LINKING THE GUI TO A SIMPLE DATABASE

This example will demonstrate how easy it is to combine a simple database with Tkinter to create a useable interface.

```
1   from tkinter import *
2   import sqlite3
3
4   root = Tk()
5   root.title("Address Book")
6   root.iconphoto(False, PhotoImage(file='contact-list.png'))
7   root.geometry("400x600")
8
9   # create a database
10
11  conn = sqlite3.connect("address_book.db")
12
13  # create a cursor
14  c = conn.cursor()
15
16  # create a table
17
18  c.execute("""CREATE TABLE IF NOT EXISTS address_book
19  ( f_name TEXT, s_name TEXT, mob TEXT, email TEXT)""")
```

In addition to Tkinter, you will also need to import sqlite3 to run the

Create the database connection as detailed in 16.4 SQL and

The CREATE TABLE code is slightly different as the program will be run several times. It will cause an error if we try to create two tables with the same name.

```
24      def submit():
25          # Connect to the db
26          conn = sqlite3.connect("address_book.db")
27          # create a cursor
28          c = conn.cursor()
29          # Insert into table
30          c.execute("INSERT INTO address_book VALUES (:f_name,:s_name,:mob,:email)",
31                    {"f_name": f_name.get(),
32                     "s_name": s_name.get(),
33                     "mob": mob.get(),
34                     "email": email.get()
35                     })
36
37          conn. commit()
38          conn.close()
39
40          # clear text boxes
41          f_name.delete(0, END)
42          s_name.delete(0, END)
43          mob.delete(0, END)
44          email.delete(0, END)
```

Each function for the buttons MUST connect to the database – Lines 9–14 are repeated here.

The data for each database field is obtained from the ENTRY text box for each.

The database must be saved and closed in each function

The data is also deleted from each ENTRY text box

```
48      def search():
49          # Connect to the db
50          conn = sqlite3.connect("address_book.db")
51          # create a cursor
52          c = conn.cursor()
53          c.execute("SELECT *,oid  FROM address_book")
54          records = c.fetchall()
55          # print(records)  # check that records are printing-commented out
56
57          # loop through records
58          print_records = ""
59          for record in records:
60              print_records += str(record[0]) + " " + str(record[1]) + " " + str(record[2]) + " " + \
61                               str(record[3]) + " " + str(record[4]) + "\n"
62          query_lbl = Label(root, text=print_records)
63          query_lbl.grid(row=8, column=0, columnspan=3, padx=30, pady=20)
64
65          conn.commit()
66          conn.close()
```

The OID (object identifier) is a set of integers that uniquely identify each row. This will be used to identify the record to be deleted.

Each field in the database record is concatenated into a string variable, print_records. The last field is the OID.

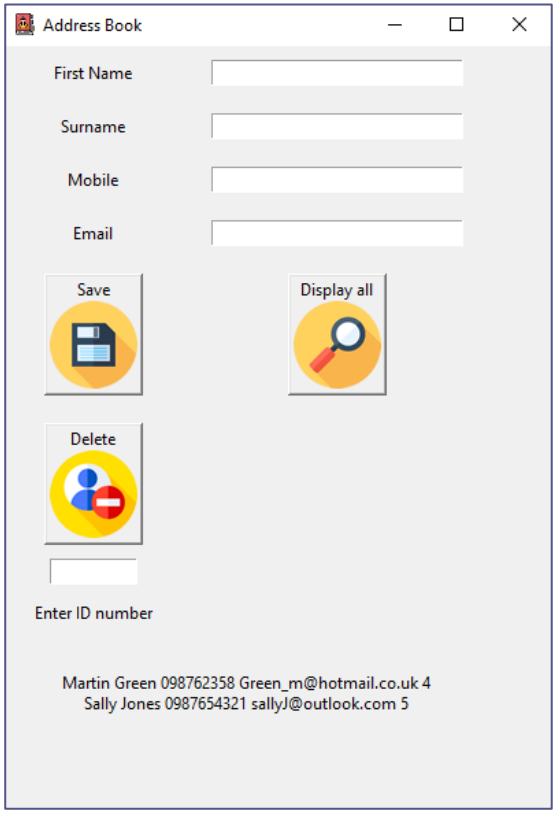
Line 62 displays the data in a label.

As you can see, the buttons now have both text and an image.

```
106     # create submit button
107     save_img = PhotoImage(file='save-file.png')
108     submit_btn = Button(root, text="Save", image=save_img, compound=BOTTOM,  command=submit)
109     submit_btn.grid(row=5, column=0, padx=30, pady=10)
```

The OID can now be used to identify the record to be deleted:

```
69      def delete_rec():
70          # Connect to the db
71          conn = sqlite3.connect("address_book.db")
72          # create a cursor
73          c = conn.cursor()
74
75          c.execute("DELETE FROM address_book WHERE oid = ?", (id_no.get()))
76          id_no.delete(0, END)
77          conn.commit()
78          conn.close()
```



EXERCISE 41: COMPLETE THE UPDATE FEATURE

The partially completed program now needs to be completed to include an UPDATE feature; one of your contacts may change their email, phone number or surname.

1. Add a function to perform this update using the OID field to identify the record
2. Attach the function to a new button and check it works

HINT: you will need to insert the field name into your UPDATE sql command but you do not know which the user will choose. The simplest method is to use the 'dot' format() string built-in function like this:

```
UPDATE database_name SET {0} = ? WHERE field_name = ?.format (update_col_name),(value1,value2)
```

For extra credit, improve the look or the functionality of this simple GUI.

All the images, database file and partially completed Python file are provided for you to use; download them from http://zzed.uk/10583-Ex41