2.2 A small lexicon of widgets - Part 2

Non-clickable widgets

The next four widgets fall into the **non-clickable** category. They're designed to present **textual** information and don't have a `command` property, although you can use `bind()` to simulate similar behavior.

The `Label` widget displays some lines of text inside the window:

```
label = Label(master, option, ...)
```

The `Label` widget contains two usable properties, but you need to remember that they are mutually exclusive.

Here you are:

Label property	Property meaning
`text`	a string which will be **shown** within the `Label`; note: newline characters (`\n`) are interpreted in the `usual way`
`textvariable`	the same as for `text`, but makes use of an observable `StringVar` variable, so if you change the variable's alteration, it will be immediately visible on the screen.

The Label widget has no usable methods – sorry!

The sample in the editor shows how the `textvariable` accompanied by an observable variable can be used to **continuously** update the `Label`'s contents.

```
import tkinter as tk


def to_string(x):
    return "Current counter\nvalue is:\n" + str(x)


def plus():
    global counter
    counter += 1
    text.set(to_string(counter))


counter = 0
window = tk.Tk()
button = tk.Button(window, text="Go on!", command=plus)
button.pack()
text = tk.StringVar()
label = tk.Label(window, textvariable=text, height=4)
text.set(to_string(counter))
label.pack()
window.mainloop()
```

The `Message` widget is very similar to the `Label` (among other things, it has the **same properties**) but is able to **format** the presented text by fitting it automatically to the widget's size.

```
message = Message(master, option, ...)
```

The sample code will tell you more.

```
import tkinter as tk


def do_it_again():
    text.set(text.get() + "and again...")


window = tk.Tk()
button = tk.Button(window, text="Go ahead!", command=do_it_again)
button.pack()
text = tk.StringVar()
message = tk.Message(window, textvariable=text, width=400)
text.set("You did it again... ")
message.pack()
window.mainloop()
```

Run it and see how the `Message` widget reacts to the `tx` variable updates.

The `Frame` widget is, in fact, a `container` designed to store other widgets. This means that the `Frame` can be used to separate a **rectangular part** of the window and to treat it as a kind of *local window*. Such a *window works* as a **master widget** for all the widgets embedded within it. Moreover, the `Frame` has its **own coordinate system**, so when you place a widget inside a Frame, you measure its location relative to the Frame's **upper-left corner**, not the window's one. It also means that if you move the `Frame` to a new position, all its inner widgets will go with it.

Note: the `Frame` can grasp virtually any widget – including another `Frame`.

The `Frame` has one interesting property:

Frame property	Property meaning
`takefocus`	normally, the `Frame` **doesn't take the focus** (which would seem to be obvious) but if you really want it to behave in this way, you can set the property to `1`.

Take a look at the example in the editor.

```
import tkinter as tk

window = tk.Tk()

frame_1 = tk.Frame(window, width=200, height=100, bg='white')
frame_2 = tk.Frame(window, width=200, height=100, bg='yellow')

button_1_1 = tk.Button(frame_1, text="Button #1 inside Frame #1")
button_1_2 = tk.Button(frame_1, text="Button #2 inside Frame #1")
button_2_1 = tk.Button(frame_2, text="Button #1 inside Frame #2")
button_2_2 = tk.Button(frame_2, text="Button #2 inside Frame #2")

button_1_1.place(x=10, y=10)
button_1_2.place(x=10, y=50)
button_2_1.grid(column=0, row=0)
```

```
button_2_2.grid(column=1, row=1)

frame_1.pack()
frame_2.pack()

window.mainloop()
```

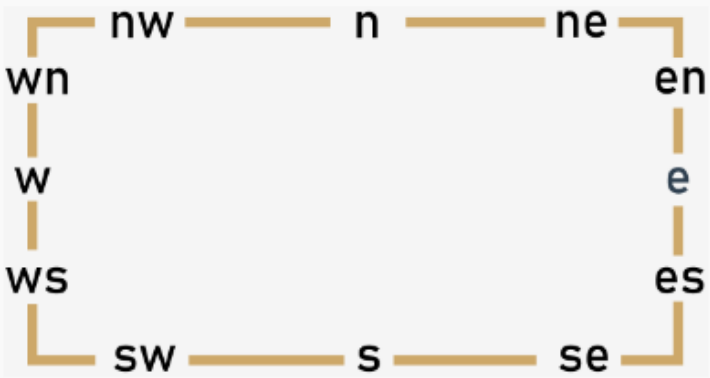
We've defined two separate frames and filled them with two buttons each. Note: we've used different geometry managers for both `Frames`. This is another advantage of the `Frame` – it helps you arrange the window in the most convenient way.

Pay attention to all four of the `Buttons`' constructors – how have we described a master widget there?

The `LabelFrame` widget is a `Frame` enriched with a **visible border** and a **title** (also visible). The title may be located at one of 12 possible places on the border line.

```
lfrm = LabelFrame(master, option, ...)
```

Some of the usable `LabelFrame` properties are gathered here:

LabelFrame property	Property meaning
`takefocus`	the same as for the `Frame`
`text`	the `LabelFrame`'s title
`labelanchor`	the **title's location**, defined as a string containing a quasi-compass coordinate (as shown by the image) 

We've rebuilt our previous example to employ a `LabelFrame` instead of a `Label` – look at the updated code we've provided in the editor.

```
import tkinter as tk

window = tk.Tk()
label_frame_1 = tk.LabelFrame(window, text="Frame #1",
                              width=200, height=100, bg='white')
label_frame_2 = tk.LabelFrame(window, text="Frame #2",
                              labelanchor='se', width=200, height=100, bg='yellow')

button_1_1 = tk.Button(label_frame_1, text="Button #1 inside Frame #1")
button_1_2 = tk.Button(label_frame_1, text="Button #2 inside Frame #1")
```

```
button_2_1 = tk.Button(label_frame_2, text="Button #1 inside Frame #2")
button_2_2 = tk.Button(label_frame_2, text="Button #2 inside Frame #2")

button_1_1.place(x=10, y=10)
button_1_2.place(x=10, y=50)
button_2_1.grid(column=0, row=0)
button_2_2.grid(column=1, row=1)

label_frame_1.pack()
label_frame_2.pack()
window.mainloop()
```

Run it and find the differences.