Python Tkinter

Checkboxes

Introduction

Checkboxes, also known as tickboxes or tick boxes or check boxes, are widgets that permit the user to make multiple selections from a number of different options. This is different to a radio button, where the user can make only one choice.

Usually, checkboxes are shown on the screen as square boxes that can contain white spaces (for false, i.e not checked) or a tick mark or X (for true, i.e. checked).

A caption describing the meaning of the checkbox is usually shown adjacent to the checkbox. The state of a checkbox is changed by clicking the mouse on the box. Alternatively it can be done by clicking on the caption, or by using a keyboard shortcut, for example the space bar.

A Checkbox has two states: on or off.

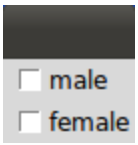
The Tkinter Checkbutton widget can contain text, but only in a single font, or images, and a button can be associated with a Python function or method. When a button is pressed, Tkinter calls the associated function or method. The text of a button can span more than one line.

Simple Example

The following example presents two checkboxes "male" and "female". Each checkbox needs a different variable name (IntVar()).

```
from tkinter import *

master = Tk()

var1 = IntVar()

Checkbutton(master, text="male", variable=var1).pack()

var2 = IntVar()

Checkbutton(master, text="female", variable=var2).pack()

mainloop()

print(var1.get())

print(var2.get())
```

If we start this script, we get the following window:

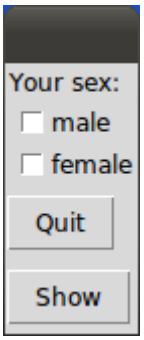


We can improve this example a little bit. First we add a Label to it. Furthermore we add two Buttons, one to leave the application and the other one to view the values var1 and var2.

```
from tkinter import *

master = Tk()


def var_states():
```

```
   print("male: %d, female: %d" % (var1.get(), var2.get()))


Label(master, text="Your sex:").pack()
var1 = IntVar()
Checkbutton(master, text="male", variable=var1).pack()
var2 = IntVar()
Checkbutton(master, text="female", variable=var2).pack()
Button(master, text='Quit', command=master.quit).pack()
Button(master, text='Show', command=var_states).pack()
mainloop()
```

The result of the previous script looks like this:



If we check "male" and click on "Show", we get the following output:

```
male: 1, female: 0
```

Entry Widgets

Introduction

Entry widgets are the basic widgets of Tkinter used to get input, i.e. text strings, from the user of an application. This widget allows the user to enter a single line of text. If the user enters a string, which is longer than the available display space of the widget, the content will be scrolled. This means that the string cannot be seen in its entirety. The arrow keys can be used to move to the invisible parts of the string. If you want to enter multiple lines of text, you have to use the text widget. An entry widget is also limited to single font.

The syntax of an entry widget looks like this:

w = Entry(master, option, ...)

"master" represents the parent window, where the entry widget should be placed. Like other widgets, it's possible to further influence the rendering of the widget by using options. The comma separated list of options can be empty.

The following simple example creates an application with two entry fields. One for entering a last name and one for the first name. We use Entry without options.

```
from tkinter import *

master = Tk()
Label(master, text="First Name").grid(row=0)
Label(master, text="Last Name").grid(row=1)
e1 = Entry(master).grid(row=0, column=1)
e2 = Entry(master).grid(row=1, column=1)

mainloop()
```

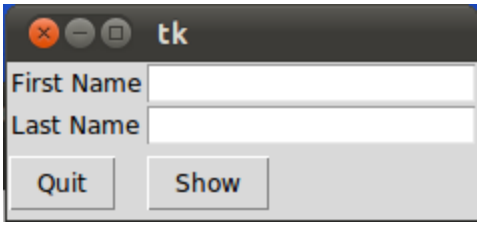
The window created by the previous script looks like this:

Okay, we have created Entry fields, so that the user of our program can put in some data. But how can our program access this data? How do we read the content of an Entry?

To put it in a nutshell: The get() method is what we are looking for. We extend our little script by two buttons "Quit" and "Show". We bind the function show_entry_fields(), which is using the get() method on the Entry objects, to the Show button. So, every time this button is clicked, the content of the Entry fields will be printed on the terminal from which we had called the script.

```
from tkinter import *

def show_entry_fields():
   print("First Name: %s\nLast Name: %s" % (e1.get(), e2.get()))

master = Tk()
Label(master, text="First Name").grid(row=0)
Label(master, text="Last Name").grid(row=1)
e1 = Entry(master)
e1.grid(row=0, column=1)
e2 = Entry(master)
e2.grid(row=1, column=1)
Button(master, text='Quit', command=quit).grid(row=3, column=0, pady=4)
Button(master, text='Show', command=show_entry_fields).grid(row=3, column=1,pady=4)

mainloop()
```

The complete application looks now like this:



Let's assume now that we want to start the Entry fields with default values, e.g. we fill in "Miller" or "Baker" as a last name, and "Jack" or "Jill" as a first name. The new version of our Python program gets the following two lines, which can be appended after the Entry definitions, i.e. "e2 = Entry(master)":

```
e1.insert(10,"Miller")
e2.insert(10,"Jill")
```

What about deleting the input of an Entry object, every time, we are showing the content in our function show_entry_fields()? No problem! We can use the delete method. The delete() method has the format delete(first, last=None). If only one number is given, it deletes the character at index. If two are given, the range from "first" to "last" will be deleted. Use delete(0, END) to delete all text in the widget.

```
from tkinter import *

def show_entry_fields():
   print("First Name: %s\nLast Name: %s" % (e1.get(), e2.get()))
   e1.delete(0,END)
   e2.delete(0,END)

master = Tk()
Label(master, text="First Name").grid(row=0)
Label(master, text="Last Name").grid(row=1)

e1 = Entry(master)
e1.grid(row=0, column=1)
e2 = Entry(master)
e2.grid(row=1, column=1)

e1.insert(0,"Miller")
e2.insert(0,"Jill")

Button(master, text='Quit', command=master.quit).grid(row=3, column=0, pady=4)
Button(master, text='Show', command=show_entry_fields).grid(row=3, column=1,pady=4)

mainloop()
```

Entry Example

This example adds two numbers.

```
from tkinter import *

root = Tk()
label1 =Label(root, text ="first")
label1.grid(row =1, column =0)

num1_txtbx =Entry(root)
num1_txtbx.grid(row =1, column =1)

label2 =Label(root, text ="second")
label2.grid(row =2, column =0)

num2_txtbx =Entry(root)
num2_txtbx.grid(row =2, column =1)

answer_label =Label(root, text ="---")
answer_label.grid(row =3, column =0)

def addF():
    if (num1_txtbx.get() and num2_txtbx.get() != ""):
        num1 =float(num1_txtbx.get())
        num2 =float(num2_txtbx.get())
        answer= num1 + num2
        answer_label.configure(text =answer)

calculate_button =Button(root, text="calculate", command= addF)
calculate_button.grid(row =3, column =0, columnspan =2)
root.mainloop()
```

Canvas Widgets

Introduction

The Canvas widget supplies graphics facilities for Tkinter. Among these graphical objects are lines, circles, images, and even other widgets. With this widget it's possible to draw graphs and plots, create graphics editors, and implement various kinds of custom widgets.

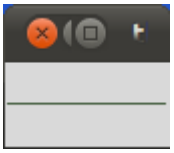
We demonstrate in our first example, how to draw a line. The method create_line(coords, options) is used to draw a straight line. The coordinates "coords" are given as four integer numbers: x1, y1, x2, y2 This means that the line goes from the point (x1, y1) to the point (x2, y2) After these coordinates follows a comma separated list of additional parameters, which may be empty. We set, for example, the colour of the line to the special green of our website: fill="#476042"

We kept the first example intentionally very simple. We create a canvas and draw a straight horizontal line into this canvas. This line vertically cuts the canvas into two areas.

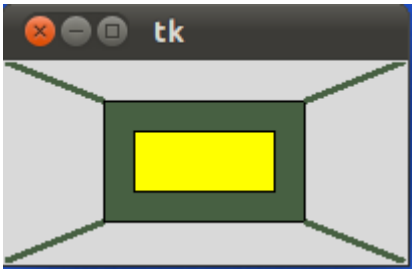
The casting to an integer value in the assignment "y = int(canvas_height / 2)" is superfluous, because create_line can work with float values as well. They are automatically turned into integer values. In the following you can see the code of our first simple script:

```
from tkinter import *

master = Tk()

canvas_width = 80

canvas_height = 40

w = Canvas(master, width=canvas_width, height=canvas_height)

w.pack()

y = int(canvas_height / 2)

w.create_line(0, y, canvas_width, y, fill="#476042")

mainloop()
```

If we start this program, using Python 3, we get the following window:



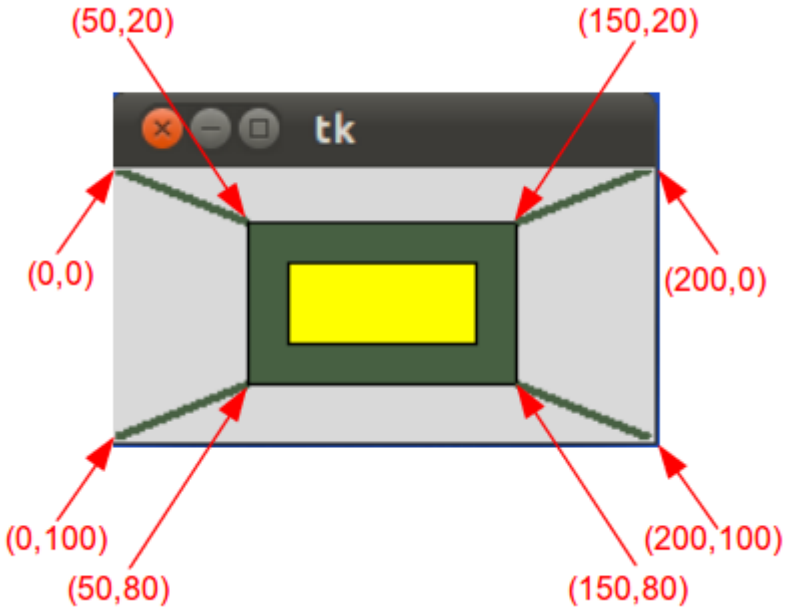
For creating rectangles we have the method create_rectangle(coords, options). Coords is again defined by two points, but this time the first one is the top left point and the bottom right point of the rectangle.



The window, you see above, is created by the following Python tkinter code:

```
from tkinter import *

master = Tk()

w = Canvas(master, width=200, height=100)

w.pack()

w.create_rectangle(50, 20, 150, 80, fill="#476042")

w.create_rectangle(65, 35, 135, 65, fill="yellow")

w.create_line(0, 0, 50, 20, fill="#476042", width=3)

w.create_line(0, 100, 50, 80, fill="#476042", width=3)

w.create_line(150,20, 200, 0, fill="#476042", width=3)

w.create_line(150, 80, 200, 100, fill="#476042", width=3)

mainloop()
```

The following image with the coordinates will simplify the understanding of application of create_lines and create_rectangle in our previous example.

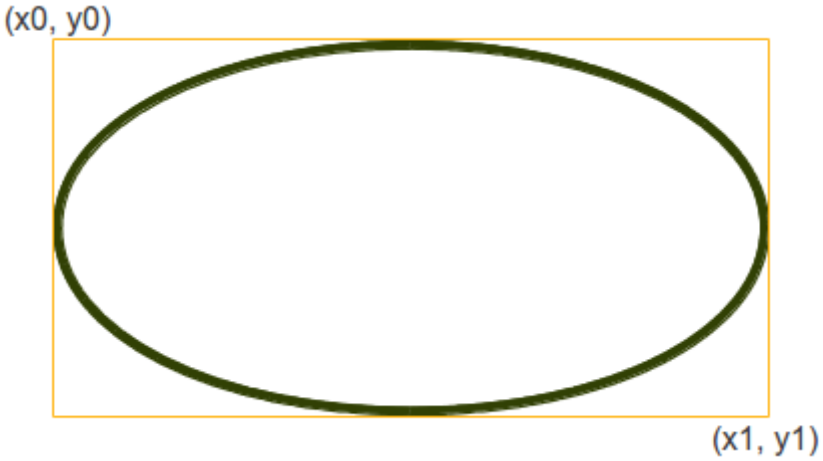


Oval Objects

An oval (or an ovoid) is any curve resembling an egg (ovum means egg in Latin). It resembles an ellipse, but it is not an ellipse. The term "oval" is not well-defined. Many different curves are called ovals, but they all have in common:

- They are differentiable, simple (not self-intersecting), convex, closed, plane curves
- They are very similar in shape to ellipses
- There is at least one axis of symmetry

The word oval stems from Latin ovum meaning "egg" and that's what it is: A figure which resembles the form of an egg. An oval is constructed from two pairs of arcs, with two different radii A circle is a special case of an oval.



We can create an oval on a canvas c with the following method:

```
id = C.create_oval ( x0, y0, x1, y1, option, ... )
```

This method returns the object ID of the new oval object on the canvas C.

The following script draws a circle around the point (75,75) with the radius 25:

```
from tkinter import *

canvas_width = 190
canvas_height =150

master = Tk()

w = Canvas(master,
           width=canvas_width,
           height=canvas_height)
w.pack()

w.create_oval(50,50,100,100)

mainloop()
```

We can define a small function drawing circles by using the create_oval() method.

```
def circle(canvas,x,y, r):
```

```
    id = canvas.create_oval(x-r,y-r,x+r,y+r)
    return id
```

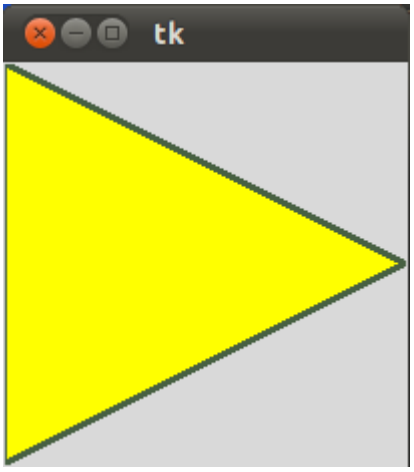
Drawing Polygons

If you want to draw a polygon, you have to provide at least three coordinate points:
create_polygon(x0,y0, x1,y1, x2,y2, ...)

In the following example we draw a triangle using this method:

```
from tkinter import *

canvas_width = 200
canvas_height =200
python_green = "#476042"

master = Tk()

w = Canvas(master,
           width=canvas_width,
           height=canvas_height)
w.pack()

points = [0,0,canvas_width,canvas_height/2, 0, canvas_height]
w.create_polygon(points, outline=python_green,
            fill='yellow', width=3)

mainloop()
```

It looks like this:



When you read this, there may or not be Christmas soon, but we present a way to improve your next Christmas with some stars, created by Python and Tkinter. The first star is straight forward with hardly any programming skills involved:

```
from tkinter import *

canvas_width = 200
```

```
canvas_height =200
python_green = "#476042"

master = Tk()

w = Canvas(master,
           width=canvas_width,
           height=canvas_height)
w.pack()

points = [100, 140, 110, 110, 140, 100, 110, 90, 100, 60, 90, 90, 60, 100, 90, 110]

w.create_polygon(points, outline=python_green,
            fill='yellow', width=3)

mainloop()
```