LEARNING tkinter

#tkinter

Table of Contents

About 1

Chapter 1: Getting started with tkinter 2

Remarks 2

Differences between python 2 and 3 2

Importing in python 2.x 2

Importing in python 3.x 2

Further Reading 2

Versions 3

Tcl 3

Python 3

Examples 4

Installation or Setup 4

Hello, World! (minimal) 5

Hello, World! (modular, object-oriented) 6

Chapter 2: Adding Images To Label/Button 8

Introduction 8

Examples 8

File Formats Supported By Tkinter 8

Usage of .GIF formats. 8

Chapter 3: Customize ttk styles 9

Introduction 9

Examples 9

Customize a treeview 9

Chapter 4: Delaying a function 11

Syntax 11

Parameters 11

Remarks 11

Examples 11

.after() 11

Chapter 5: Multiple windows (TopLevel widgets) 13

Examples 13
Difference between Tk and Toplevel 13
arranging the window stack (the .lift method) 14
Chapter 6: Scrolling widgets 16
Introduction 16
Syntax 16
Parameters 16
Remarks 16
Examples 16
Connecting a vertical scrollbar to a Text widget 16
Scrolling a Canvas widget horizontally and vertically 16
Scrolling a group of widgets 17
Chapter 7: The Tkinter Entry Widget 18
Syntax 18
Parameters 18
Remarks 18
Examples 18
Creating an Entry widget and setting a default value 18
Getting the value of an Entry widget 18
Adding validation to an Entry widget 19
Getting int From Entry Widget 19
Chapter 8: The Tkinter Radiobutton widget 20
Syntax 20
Parameters 20
Remarks 20
Examples 21
Here's an example of how to turn radio buttons to button boxes: 21
Create a group of radiobuttons 21
Chapter 9: Tkinter Geometry Managers 22
Introduction 22
Examples 22
pack() 22

grid() 23

place() 24

Chapter 10: Ttk widgets 27

Introduction 27

Syntax 27

Parameters 27

Remarks 27

Examples 27

Treeview: Basic example 27

Create the widget 27

Definition of the columns 27

Definition of the headings 28

Insert some rows 28

Packing 28

Progressbar 29

Function updating the progressbar 29

Set the maximum value 29

Create the progress bar 29

Initial and maximum values 29

Emulate progress each 0.5 s 29

Credits 31

About

Chapter 1: Getting started with tkinter

Remarks

Tkinter ("**Tk Inter**face")is python's standard cross-platform package for creating graphical user interfaces (GUIs). It provides access to an underlying Tcl interpreter with the Tk toolkit, which itself is a cross-platform, multilanguage graphical user interface library.

Tkinter isn't the only GUI library for python, but it is the one that comes standard. Additional GUI libraries that can be used with python include wxPython, PyQt, and kivy.

Tkinter's greatest strength is its ubiquity and simplicity. It works out of the box on most platforms (linux, OSX, Windows), and comes complete with a wide range of widgets necessary for most common tasks (buttons, labels, drawing canvas, multiline text, etc).

As a learning tool, tkinter has some features that are unique among GUI toolkits, such as named fonts, bind tags, and variable tracing.

Differences between python 2 and 3

Tkinter is largely unchanged between python 2 and python 3, with the major difference being that the tkinter package and modules were renamed.

Importing in python 2.x

In python 2.x, the tkinter package is named `Tkinter`, and related packages have their own names. For example, the following shows a typical set of import statements for python 2.x:

```
import Tkinter as tk
import tkFileDialog as filedialog
import ttk
```

Importing in python 3.x

Although functionality did not change much between python 2 and 3, the names of all of the tkinter modules have changed. The following is a typical set of import statements for python 3.x:

```
import tkinter as tk
from tkinter import filedialog
from tkinter import ttk
```

Further Reading

- Tkinter questions on Stackoverflow
- Official Python 3 tkinter documentation
- Official Python 2 tkinter documentation
- Tkdocs.com - multiplatform tk documentation
- Effbot introduction to tkinter
- Tkinter reference guide, New Mexico Tech

Versions

Tcl

Version	Release Date
8.6	2016-07-27
8.5	2016-02-12
8.4	2013-06-01
8.3	2002-10-18
8.2	1999-12-16
8.1	1999-05-26
8.0	1999-03-09

Python

Version	Release Date
3.6	2016-12-23
3.5	2015-09-13
3.4	2014-03-17
3.3	2012-09-29
3.2	2011-02-20
3.1	2009-06-26
3.0	2008-12-03
2.7	2010-07-03

Version	Release Date
2.6	2008-10-02
2.5	2006-09-19
2.4	2004-11-30
2.3	2003-07-29
2.2	2001-12-21
2.1	2001-04-15
2.0	2000-10-16

Examples

Installation or Setup

Tkinter comes pre-installed with the Python installer binaries for Mac OS X and the Windows platform. So if you install Python from the official binaries for Mac OS X or Windows platform, you are good to go with Tkinter.

For Debian versions of Linux you have to install it manually by using the following commands.

For Python 3

> sudo apt-get install python3-tk

For Python 2.7

> sudo apt-get install python-tk

Linux distros with yum installer can install tkinter module using the command:

> yum install tkinter

Verifying Installation

To verify if you have successfully installed Tkinter, open your Python console and type the following command:

```
import tkinter as tk # for Python 3 version
```

or

```
import Tkinter as tk # for Python 2.x version
```

You have successfully installed Tkinter, if the above command executes without an error.

To check the Tkinter version, type the following commands in your Python REPL:

For python 3.X

```
import tkinter as tk
tk._test()
```

For python 2.X

```
import Tkinter as tk
tk._test()
```

Note: Importing `Tkinter as tk` is not required but is good practice as it helps keep things consistent between version.

Hello, World! (minimal)

Let's test our basic knowledge of tkinter by creating the classic "Hello, World!" program.

First, we must import tkinter, this will vary based on version (see remarks section about "Differences between Python 2 and 3")

In Python 3 the module `tkinter` has a lowercase t:

```
import tkinter as tk
```

In Python 2 the module `Tkinter` has a uppercase T:

```
import Tkinter as tk
```

Using `as tk` isn't strictly necessary but we will use it so the rest of this example will work the same for both version.

now that we have the tkinter module imported we can create the root of our application using the `Tk` class:

```
root = tk.Tk()
```

This will act as the window for our application. (note that *additional* windows should be `Toplevel` instances instead)

Now that we have a window, let's add text to it with a `Label`

```
label = tk.Label(root, text="Hello World!") # Create a text label
label.pack(padx=20, pady=20) # Pack it into the window
```

Once the application is ready we can start it (enter the *main* event *loop*) with the `mainloop` method

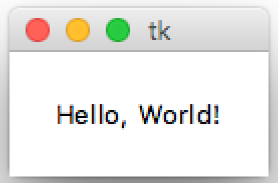
```
root.mainloop()
```

This will open and run the application until it is stopped by the window being closed or calling exiting functions from callbacks (discussed later) such as `root.destroy()`.

Putting it all together:

```
import tkinter as tk # Python 3.x Version
#import Tkinter as tk # Python 2.x Version

root = tk.Tk()

label = tk.Label(root, text="Hello World!") # Create a text label
label.pack(padx=20, pady=20) # Pack it into the window

root.mainloop()
```

And something like this should pop up:



Hello, World! (modular, object-oriented)

```
import tkinter as tk

class HelloWorld(tk.Frame):
    def __init__(self, parent):
        super(HelloWorld, self).__init__(parent)

        self.label = tk.Label(self, text="Hello, World!")
        self.label.pack(padx=20, pady=20)

if __name__ == "__main__":
    root = tk.Tk()

    main = HelloWorld(root)
    main.pack(fill="both", expand=True)

    root.mainloop()
```

Note: It's possible to inherit from just about any tkinter widget, including the root window. Inheriting from `tkinter.Frame` is at least arguably the most flexible in that it supports multiple document interfaces (MDI), single document interfaces (SDI), single page applications, and multiple-page applications.

Read Getting started with tkinter online: https://riptutorial.com/tkinter/topic/987/getting-started-with-tkinter

Chapter 2: Adding Images To Label/Button

Introduction

This shows the proper usage of images and how to correctly display images.

Examples

File Formats Supported By Tkinter

Tkinter support .ppm files from PIL(Python Imaging Library), .JPG, .PNG and .GIF.

To import and image you first need to create a reference like so:

```
Image = PhotoImage(filename = [Your Image here])
```

Now, we can add this image to Button and Labels like so using the "img" callback:

```
 Lbl = Label (width=490, img=image)
```

Usage of .GIF formats.

In order to display a gif, you need to show it frame by frame sort of like an animation.

An animated gif consists of a number of frames in a single file. Tk loads the first frame but you can specify different frames by passing an index parameter when creating the image. For example:

```
frame2 = PhotoImage(file=imagefilename, format="gif -index 2")
```

If you load up all the frames into separate PhotoImages and then use timer events to switch the frame being shown (label.configure(image=nextframe)). The delay on the timer lets you control the animation speed. There is nothing provided to give you the number of frames in the image other than it failing to create a frame once you exceed the frame count.

Read Adding Images To Label/Button online: https://riptutorial.com/tkinter/topic/9746/adding-images-to-label-button

Chapter 3: Customize ttk styles

Introduction

The style of the new ttk widgets is one of the most powerful aspects of ttk. Besides the fact that it is a completely different way of working than the traditional tk package, it enables to perform a huge degree of customization on your widgets.

Examples

Customize a treeview

By taking Treeview: Basic example, it can be shown how to customize a basic treeview.

In this case, we create a style "mystyle.Treeview" with the following code (see the comments to understand what each line does):

```
style = ttk.Style()
style.configure("mystyle.Treeview", highlightthickness=0, bd=0, font=('Calibri', 11)) # Modify
the font of the body
style.configure("mystyle.Treeview.Heading", font=('Calibri', 13,'bold')) # Modify the font of
the headings
style.layout("mystyle.Treeview", [('mystyle.Treeview.treearea', {'sticky': 'nswe'})]) # Remove
the borders
```

Then, the widget is created giving the above style:

```
tree=ttk.Treeview(master,style="mystyle.Treeview")
```

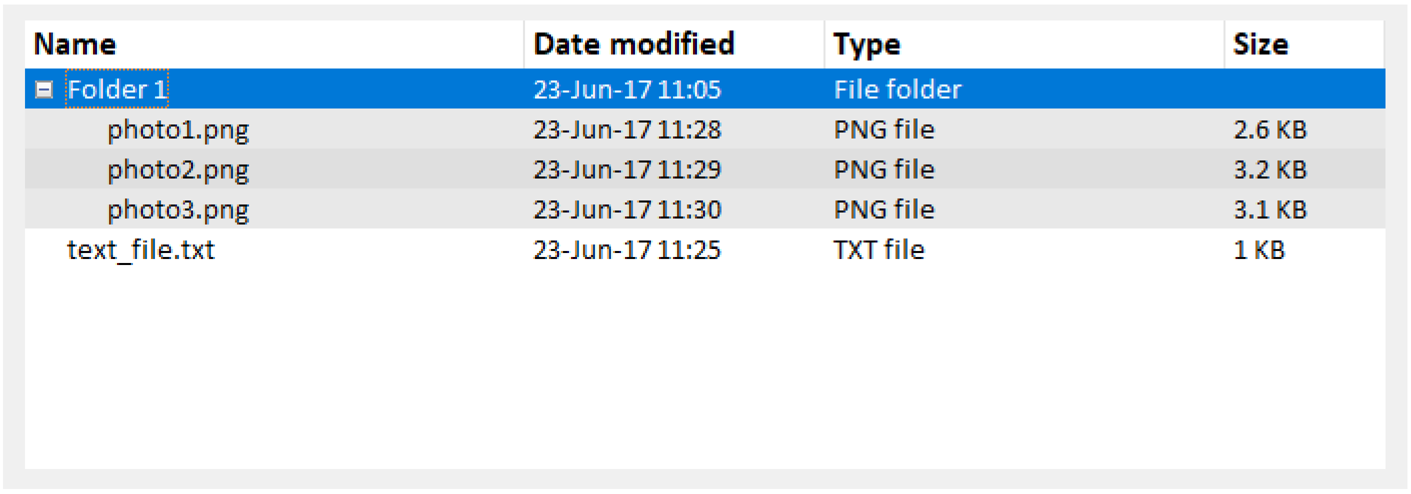
If you would like to have a different format depending on the rows, you can make use of `tags`:

```
tree.insert(folder1, "end", "", text="photo1.png", values=("23-Jun-17 11:28","PNG file","2.6
KB"),tags = ('odd',))
tree.insert(folder1, "end", "", text="photo2.png", values=("23-Jun-17 11:29","PNG file","3.2
KB"),tags = ('even',))
tree.insert(folder1, "end", "", text="photo3.png", values=("23-Jun-17 11:30","PNG file","3.1
KB"),tags = ('odd',))
```

Then, for instance, a background color can be associated to the tags:

```
tree.tag_configure('odd', background='#E8E8E8')
tree.tag_configure('even', background='#DFDFDF')
```

The result is a treeview with modified fonts on both the body and headings, no border and different colors for the rows:



Name	Date modified	Type	Size
⊟ Folder 1	23-Jun-17 11:05	File folder	
photo1.png	23-Jun-17 11:28	PNG file	2.6 KB
photo2.png	23-Jun-17 11:29	PNG file	3.2 KB
photo3.png	23-Jun-17 11:30	PNG file	3.1 KB
text_file.txt	23-Jun-17 11:25	TXT file	1 KB

Note: To generate the above picture, you should add/change the aforementioned lines of code in the example Treeview: Basic example.

Read Customize ttk styles online: https://riptutorial.com/tkinter/topic/10624/customize-ttk-styles

Chapter 4: Delaying a function

Syntax

- widget.after(delay_ms, callback, *args)

Parameters

Parameter	Description
delay_ms	Time (milliseconds) which is delayed the call to the function `callback`
callback	Function that is called after the given `delay_ms`. If this parameter is not given, `.after` acts similar to `time.sleep` (in milliseconds)

Remarks

Syntax assumes a `widget` accepted by the method `.after` has been previously created (i.e `widget=tk.Label(parent)`)

Examples

.after()

`.after(delay, callback=None)` is a method defined for all tkinter widgets. This method simply calls the function `callback` after the given `delay` in ms. If no function is given, it acts similar to `time.sleep` (but in milliseconds instead of seconds)

Here is an example of how to create a simple timer using `after`:

```
# import tkinter
try:
    import tkinter as tk
except ImportError:
    import Tkinter as tk

class Timer:
    def __init__(self, parent):
        # variable storing time
        self.seconds = 0
        # label displaying time
        self.label = tk.Label(parent, text="0 s", font="Arial 30", width=10)
        self.label.pack()
        # start the timer
        self.label.after(1000, self.refresh_label)

    def refresh_label(self):
        """ refresh the content of the label every second """
        # increment the time
        self.seconds += 1
        # display the new time
        self.label.configure(text="%i s" % self.seconds)
        # request tkinter to call self.refresh after 1s (the delay is given in ms)
        self.label.after(1000, self.refresh_label)

if __name__ == "__main__":
    root = tk.Tk()
    timer = Timer(root)
    root.mainloop()
```

Read Delaying a function online: https://riptutorial.com/tkinter/topic/6724/delaying-a-function

Chapter 5: Multiple windows (TopLevel widgets)

Examples

Difference between Tk and Toplevel

`Tk` is the absolute root of the application, it is the first widget that needs to be instantiated and the GUI will shut down when it is destroyed.

`Toplevel` is a window in the application, closing the window will destroy all children widgets placed on that window{1} but will not shut down the program.

```
try:
    import tkinter as tk #python3
except ImportError:
    import Tkinter as tk #python2

#root application, can only have one of these.
root = tk.Tk()

#put a label in the root to identify the window.
label1 = tk.Label(root, text="""this is root
closing this window will shut down app""")
label1.pack()

#you can make as many Toplevels as you like
extra_window = tk.Toplevel(root)
label2 = tk.Label(extra_window, text="""this is extra_window
closing this will not affect root""")
label2.pack()

root.mainloop()
```

If your python program only represents a single application (which it almost always will) then you should have only one `Tk` instance, but you may create as many `Toplevel` windows as you like.

```
try:
    import tkinter as tk #python3
except ImportError:
    import Tkinter as tk #python2

def generate_new_window():
    window = tk.Toplevel()
    label = tk.Label(window, text="a generic Toplevel window")
    label.pack()

root = tk.Tk()

spawn_window_button = tk.Button(root,
                                text="make a new window!",
                                command=generate_new_window)
```

```
spawn_window_button.pack()

root.mainloop()
```

{1}: if a Toplevel (`A = Toplevel(root)`) is the parent of another Toplevel (`B = Toplevel(A)`) then closing window A will also close window B.

arranging the window stack (the .lift method)

The most basic case to lift a particular window above the others, just call the `.lift()` method on that window (either `Toplevel` or `Tk`)

```
import tkinter as tk #import Tkinter as tk #change to commented for python2

root = tk.Tk()

for i in range(4):
    #make a window with a label
    window = tk.Toplevel(root)
    label = tk.Label(window,text="window {}".format(i))
    label.pack()
    #add a button to root to lift that window
    button = tk.Button(root, text = "lift window {}".format(i), command=window.lift)
    button.grid(row=i)

root.mainloop()
```

However if that window is destroyed trying to lift it will raise an error like this:

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "/.../tkinter/__init__.py", line 1549, in __call__
    return self.func(*args)
  File "/.../tkinter/__init__.py", line 785, in tkraise
    self.tk.call('raise', self._w, aboveThis)
_tkinter.TclError: bad window path name ".4385637096"
```

Often when we are trying to put a particular window in front of the user but it was closed a good alternative is to recreate that window:

```
import tkinter as tk #import Tkinter as tk #change to commented for python2

dialog_window = None

def create_dialog():
    """creates the dialog window
  ** do not call if dialog_window is already open, this will
     create a duplicate without handling the other
if you are unsure if it already exists or not use show_dialog()"""
    global dialog_window
    dialog_window = tk.Toplevel(root)
    label1 = tk.Label(dialog_window,text="this is the dialog window")
    label1.pack()
    #put other widgets
    dialog_window.lift() #ensure it appears above all others, probably will do this anyway

def show_dialog():
    """lifts the dialog_window if it exists or creates a new one otherwise"""
    #this can be refactored to only have one call to create_dialog()
    #but sometimes extra code will be wanted the first time it is created
    if dialog_window is None:
        create_dialog()
        return
    try:
        dialog_window.lift()
    except tk.TclError:
        #window was closed, create a new one.
        create_dialog()


root = tk.Tk()

dialog_button = tk.Button(root,
                          text="show dialog_window",
                          command=show_dialog)
dialog_button.pack()
root.mainloop()
```

This way the function `show_dialog` will show the dialog window whether it exists or not, also note that you can call `.winfo_exists()` to check if it exists before trying to lift the window instead of wrapping it in a `try:except`.

There is also the `.lower()` method that works the same way as the `.lift()` method, except lowering the window in the stack:

```
import tkinter as tk #import Tkinter as tk #change to commented for python2

root = tk.Tk()
root.title("ROOT")
extra = tk.Toplevel()
label = tk.Label(extra, text="extra window")
label.pack()

lower_button = tk.Button(root,
                         text="lower this window",
                         command=root.lower)
lower_button.pack()

root.mainloop()
```

You will notice that it lowers even below other applications, to only lower below a certain window you can pass it to the `.lower()` method, similarly this can also be done with the `.lift()` method to only raise a window above another one.

Read Multiple windows (TopLevel widgets) online:
https://riptutorial.com/tkinter/topic/6439/multiple-windows--toplevel-widgets-

Chapter 6: Scrolling widgets

Introduction

Scrollbars can be added to Listbox, Canvas, and Text widgets. In addition, Entry widgets can be scrolled horizontally. To be able to scroll other type of widgets, you need to put them inside a Canvas or a Text widget.

Syntax

- scrollbar = tk.Scrollbar(parent, **kwargs)

Parameters

Parameter	Description
parent	tkinter widgets exist in a hierarchy. Except for the root window, all widgets have a parent. Some online tutorials call this "master". When the widget is added to the screen with pack, place or grid, it will appear inside this parent widget
orient	Orientation of the scrollbar, either `"vertical"` (default value) or `"horizontal"`

Remarks

These examples assume that tkinter has been imported with either `import tkinter as tk` (python 3) or `import Tkinter as tk` (python 2).

Examples

Connecting a vertical scrollbar to a Text widget

The connection between the widget and the scrollbar goes both ways. The scrollbar needs to be expanded vertically so that it has the same height as the widget.

```
text = tk.Text(parent)
text.pack(side="left")

scroll_y = tk.Scrollbar(parent, orient="vertical", command=text.yview)
scroll_y.pack(side="left", expand=True, fill="y")

text.configure(yscrollcommand=scroll_y.set)
```

Scrolling a Canvas widget horizontally and vertically

The principle is essentially the same as for the Text widget, but a `Grid` layout is used to put the scrollbars around the widget.

```
canvas = tk.Canvas(parent, width=150, height=150)
canvas.create_oval(10, 10, 20, 20, fill="red")
canvas.create_oval(200, 200, 220, 220, fill="blue")
canvas.grid(row=0, column=0)

scroll_x = tk.Scrollbar(parent, orient="horizontal", command=canvas.xview)
scroll_x.grid(row=1, column=0, sticky="ew")

scroll_y = tk.Scrollbar(parent, orient="vertical", command=canvas.yview)
scroll_y.grid(row=0, column=1, sticky="ns")

canvas.configure(yscrollcommand=scroll_y.set, xscrollcommand=scroll_x.set)
```

Unlike for the Text widget, the scrollable region of the Canvas is not updated automatically when its content is modified, so we need to define it and update it manually using the `scrollregion` argument:

```
canvas.configure(scrollregion=canvas.bbox("all"))
```

`canvas.bbox("all")` returns the coordinates of the rectangle fitting the whole canvas content.

Scrolling a group of widgets

When a window contains many widgets, they might not all be visible. However, neither a window (Tk or Toplevel instance) nor a Frame are scrollable. One solution to make the window content scrollable is to put all the widgets in a Frame, and then, embed this Frame in a Canvas using the `create_window` method.

```
canvas = tk.Canvas(parent)
scroll_y = tk.Scrollbar(parent, orient="vertical", command=canvas.yview)

frame = tk.Frame(canvas)
# group of widgets
for i in range(20):
    tk.Label(frame, text='label %i' % i).pack()
# put the frame in the canvas
canvas.create_window(0, 0, anchor='nw', window=frame)
# make sure everything is displayed before configuring the scrollregion
canvas.update_idletasks()

canvas.configure(scrollregion=canvas.bbox('all'),
                 yscrollcommand=scroll_y.set)

canvas.pack(fill='both', expand=True, side='left')
scroll_y.pack(fill='y', side='right')
```

Read Scrolling widgets online: https://riptutorial.com/tkinter/topic/8931/scrolling-widgets

Chapter 7: The Tkinter Entry Widget

Syntax

- entry = tk.Entry(*parent*, ***kwargs*)
- entry.get()
- entry.insert(index, "value")
- entry.delete(start_index, end_index)
- entry.bind(event, callback)

Parameters

Parameter	Description
parent	tkinter widgets exist in a hieararchy. Except for the root window, all widgets have a parent. Some online tutorials call this "master". When the widget is added to the screen with `pack`, `place` or `grid`, it will appear inside this parent widget
width	The width specifies the *desired* width of the widget based on an average character width. For variable width fonts, this is based on the width of the zero character (`0`). The default is 20. Note that the actual width could be larger or smaller depending on how it is added to the screen.

Remarks

These examples assume that tkinter has been imported with either `import tkinter as tk` (python 3) or `import Tkinter as tk` (python 2).

Examples

Creating an Entry widget and setting a default value

```
entry = tk.Entry(parent, width=10)
entry.insert(0, "Hello, World!")
```

Getting the value of an Entry widget

The value of an entry widget can be obtained with the `get` method of the widget:

```
name_entry = tk.Entry(parent)
...
name = name_entry.get()
```

Optionally, you may associate an instance of a `StringVar`, and retrieve the value from the `StringVar` rather than from the widget:

```
name_var = tk.StringVar()
name_entry = tk.Entry(parent, textvariable=name_var)
...
name = name_var.get()
```

Adding validation to an Entry widget

To restrict the characters that can be typed into an entry widget, only numbers for instance, a validate command can be added to the entry. A validate command is a function that return `True` if the change is accepted, `False` otherwise. This function will be called each time the content of the entry is modified. Various arguments can be passed to this function, like the type of change (insertion, deletion), the inserted text, ...

```
def only_numbers(char):
    return char.isdigit()

validation = parent.register(only_numbers)
entry = Entry(parent, validate="key", validatecommand=(validation, '%S'))
```

The `validate` option determines the type of event that triggers the validation, here, it's any keystroke in the entry. The `'%S'` in the validatecommand option means that the inserted or deleted character is passed in argument to the `only_numbers` function. The full list of possibilities can be found here.

Getting int From Entry Widget

When using the .get() method whatever is in the entry widget will be converted into a string. For example, regardless of the type of input(It can be a number or sentence), the resulting outcome will be a string. If the user types 4 the output will be "4" as in a string. To get an int from an Entry Widget, first, call the .get() method.

```
What_User_Wrote = Entry.get()
```

Now we convert that string into an int like so:

```
Convert_To_Int = int(What_User_Wrote)
```

Likewise, if you want to save time you can simply do:

```
Convert_To_Int = int(Entry.get())
```

You can use the above method if you don't want to convert str to int.

Read The Tkinter Entry Widget online: https://riptutorial.com/tkinter/topic/4868/the-tkinter-entry-widget

Chapter 8: The Tkinter Radiobutton widget

Syntax

- radiobutton = tk.Radiobutton(parent, **kwargs)

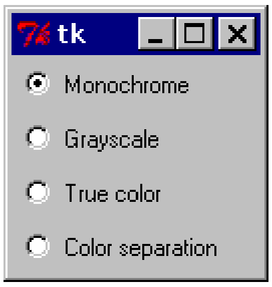
Parameters

Parameter	Description
parent	tkinter widgets exist in a hierarchy. Except for the root window, all widgets have a parent. Some online tutorials call this "master". When the widget is added to the screen with pack, place or grid, it will appear inside this parent widget.
command	function called each time the user changes the state of the radiobutton
indicatoron	1 or True for radio buttons, 0 or False for button boxes
text	Text to display next to the radiobutton.
value	When the radiobutton is selected, the associated control variable is set to value.
variable	Control variable the radiobutton shares with the other radiobutton of the group.

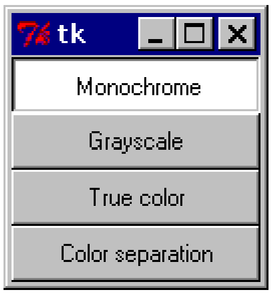
Remarks

These examples assume that tkinter has been imported with either `import tkinter as tk` (python 3) or `import Tkinter as tk` (python 2).

Reference:



To turn the above example into a “button box” rather than a set of radio buttons, set the indicatoron option to 0. In this case, there’s no separate radio button indicator, and the selected button is drawn as SUNKEN instead of RAISED:



-effbot

Examples

Here's an example of how to turn radio buttons to button boxes:

```
import tkinter as tk
root = tk.Tk()

rbvar = StringVar()
rbvar.set(" ")

rb1 = tk.Radiobutton(root, text="Option 1", variable=rbvar, value='a', indicatoron=0)
rb1.pack()

rb2 = tk.Radiobutton(root, text="Option 2", variable=rbvar, value='b', indicatoron=0)
rb2.pack()
```

Create a group of radiobuttons

Such a group is made of radiobuttons that share a control variable so that no more than one can be selected.

```
# control variable
var = tk.IntVar(parent, 0)

# group of radiobuttons
for i in range(1,4):
    tk.Radiobutton(parent, text='Choice %i' % i, value=i, variable=var).pack()

tk.Button(parent, text='Print choice', command=lambda: print(var.get())).pack()
```

Read The Tkinter Radiobutton widget online: https://riptutorial.com/tkinter/topic/6338/the-tkinter-radiobutton-widget

Chapter 9: Tkinter Geometry Managers

Introduction

There are three geometry managers to position widgets: `pack()`, `grid()` and `place()`.

Examples

pack()

The `pack()` geometry manager organizes widgets in blocks before placing them in the parent widget. It uses the options `fill`, `expand` and `side`.

Syntax

```
widget.pack(option)
```

Fill
Determines if the widget keeps the minimal space needed or takes up any extra space allocated to it. Attributes: NONE (default), X (fill horizontally), Y (fill vertically), or BOTH (fill both horizontally and vertically).

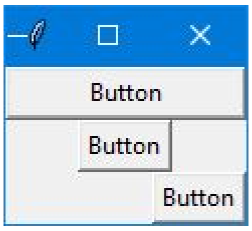
Expand
When set to YES, the widget expands to fill any space not used in widget's parent. Attributes: YES, NO.

Side
Determines which side of the widget's parent it packs to. Attributes: TOP (default), BOTTOM, LEFT, or RIGHT.

Example

```
from tkinter import *
root = Tk()
btn_fill = Button(root, text="Button")
btn_fill.pack(fill=X)

btn_expand = Button(root, text="Button")
btn_expand.pack(expand=YES)

btn_side = Button(root, text="Button")
btn_side.pack(side=RIGHT)

root.mainloop()
```

Result



grid()

The `grid()` geometry manager organises widgets in a table-like structure in the parent widget. The master widget is split into rows and columns, and each part of the table can hold a widget. It uses `column`, `columnspan`, `ipadx`, `ipady`, `padx`, `pady`, `row`, `rowspan` and `sticky`.

Syntax

```
widget.grid(options)
```

Column
The column to put widget in. The default column is 0, which is the leftmost column.

Columnspan
How many columns widget takes up. The default is 1.

Ipadx
How many pixels to pad widget horizontally inside the widget's borders.

Ipady
How many pixels to pad widget vertically inside the widget's borders.

Padx
How many pixels to pad widget horizontally outside the widget's borders.

Pady
How many pixels to pad widget vertically outside the widget's borders.

Row
The row to put widget in. The default row is 0, which is the topmost column.

Rowspan
How many rows the widget takes up. The default is 1.

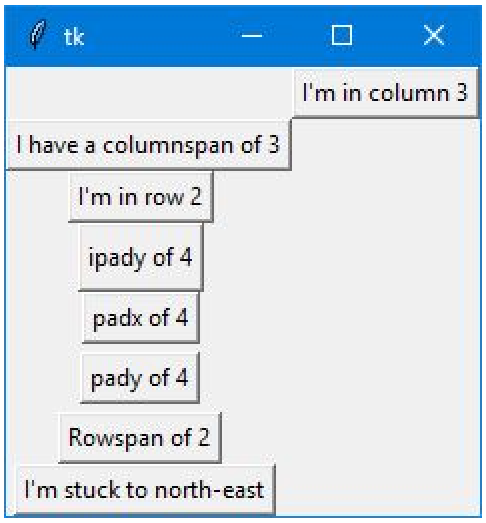
Sticky
When the widget is smaller than the cell, `sticky` is used to indicate which sides and corners of the cell the widget sticks to. The direction is defined by compass directions: N, E, S, W, NE, NW, SE, and SW and zero. These could be a string concatenation, for example, NESW make the widget take up the full area of the cell.

Example

```
from tkinter import *
root = Tk()
```

```
btn_column = Button(root, text="I'm in column 3")
btn_column.grid(column=3)

btn_columnspan = Button(root, text="I have a columnspan of 3")
btn_columnspan.grid(columnspan=3)

btn_ipadx = Button(root, text="ipadx of 4")
btn_ipadx.grid(ipadx=4)

btn_ipady = Button(root, text="ipady of 4")
btn_ipady.grid(ipady=4)

btn_padx = Button(root, text="padx of 4")
btn_padx.grid(padx=4)

btn_pady = Button(root, text="pady of 4")
btn_pady.grid(pady=4)

btn_row = Button(root, text="I'm in row 2")
btn_row.grid(row=2)

btn_rowspan = Button(root, text="Rowspan of 2")
btn_rowspan.grid(rowspan=2)

btn_sticky = Button(root, text="I'm stuck to north-east")
btn_sticky.grid(sticky=NE)

root.mainloop()
```

Result



place()

The `place()` manager organises widgets by placing them in a specific position in the parent widget. This geometry manager uses the options `anchor`, `bordermode`, `height`, `width`, `relheight`, `relwidth`,`relx`, `rely`, `x` and `y`.

Anchor
Indicates where the widget is anchored to. The options are compass directions: N, E, S, W, NE, NW, SE, or SW, which relate to the sides and corners of the parent widget. The default is NW (the

upper left corner of widget)

Bordermode
Bordermode has two options: `INSIDE`, which indicates that other options refer to the parent's inside, (Ignoring the parent's borders) and `OUTSIDE`, which is the opposite.

Height
Specify the height of a widget in pixels.

Width
Specify the width of a widget in pixels.

Relheight
Height as a float between 0.0 and 1.0, as a fraction of the height of the parent widget.

Relwidth
Width as a float between 0.0 and 1.0, as a fraction of the width of the parent widget.

Relx
Horizontal offset as a float between 0.0 and 1.0, as a fraction of the width of the parent widget.

Rely
Vertical offset as a float between 0.0 and 1.0, as a fraction of the height of the parent widget.

X
Horizontal offset in pixels.

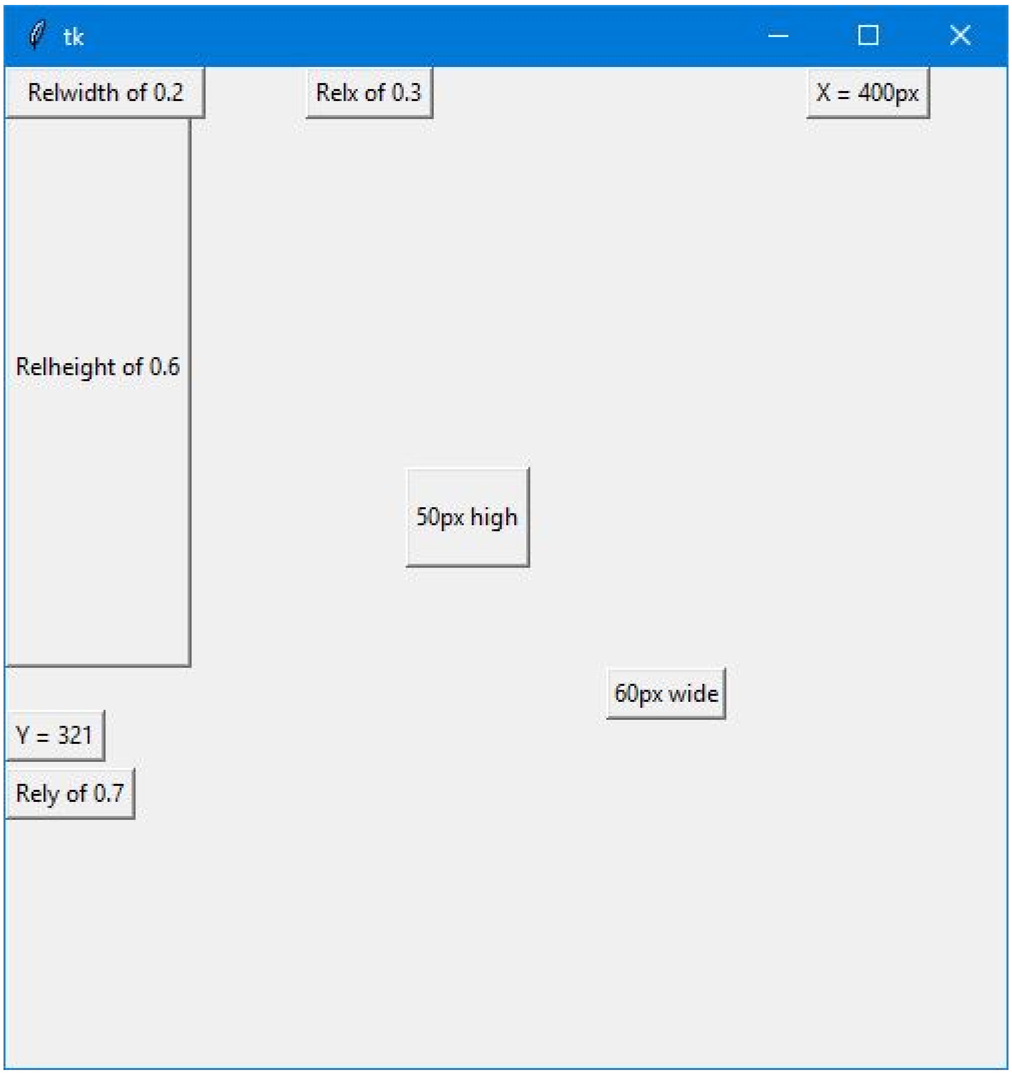
Y
Vertical offset in pixels.

Example

```
from tkinter import *
root = Tk()
root.geometry("500x500")

btn_height = Button(root, text="50px high")
btn_height.place(height=50, x=200, y=200)

btn_width = Button(root, text="60px wide")
btn_width.place(width=60, x=300, y=300)

btn_relheight = Button(root, text="Relheight of 0.6")
btn_relheight.place(relheight=0.6)

btn_relwidth= Button(root, text="Relwidth of 0.2")
btn_relwidth.place(relwidth=0.2)

btn_relx=Button(root, text="Relx of 0.3")
btn_relx.place(relx=0.3)

btn_rely=Button(root, text="Rely of 0.7")
btn_rely.place(rely=0.7)
```

```
btn_x=Button(root, text="X = 400px")
btn_x.place(x=400)

btn_y=Button(root, text="Y = 321")
btn_y.place(y=321)

root.mainloop()
```

Result



Read Tkinter Geometry Managers online: https://riptutorial.com/tkinter/topic/9620/tkinter-geometry-managers

Chapter 10: Ttk widgets

Introduction

Examples of the different ttk widgets. Ttk has a total of 17 widgets, eleven of which already existed in tkinter (tk).

Using ttk module gives your application a more modern and improved look.

Syntax

- tree=ttk.Treeview(master,**kwargs)

Parameters

Parameter	Description
master	tkinter widgets exist in a hieararchy. Except for the root window, all widgets have a parent (also called "master"). When the widget is added to the screen with pack, place or grid, it will appear inside this parent widget

Remarks

These examples assume that tkinter has been imported with either `import tkinter as tk` (python 3) or `import Tkinter as tk` (python 2).

It is also assumed that ttk has been imported with either `from tkinter import ttk` (python 3) or `import ttk` (python 2).

Examples

Treeview: Basic example

This widget is used to display items with hierarchy. For instance, windows explorer can be reproduced in this way. Some nice tables can be also done using `treeview` widget.

Create the widget

```
tree=ttk.Treeview(master)
```

Definition of the columns

You can define how many columns, their width and minimum width when the user tries to stretch it. By defining `stretch=tk.NO`, the user cannot modify the width of the column.

```
tree["columns"]=("one","two","three")
tree.column("#0", width=270, minwidth=270, stretch=tk.NO)
tree.column("one", width=150, minwidth=150, stretch=tk.NO)
tree.column("two", width=400, minwidth=200)
tree.column("three", width=80, minwidth=50, stretch=tk.NO)
```

Definition of the headings

```
tree.heading("#0",text="Name",anchor=tk.W)
tree.heading("one", text="Date modified",anchor=tk.W)
tree.heading("two", text="Type",anchor=tk.W)
tree.heading("three", text="Size",anchor=tk.W)
```

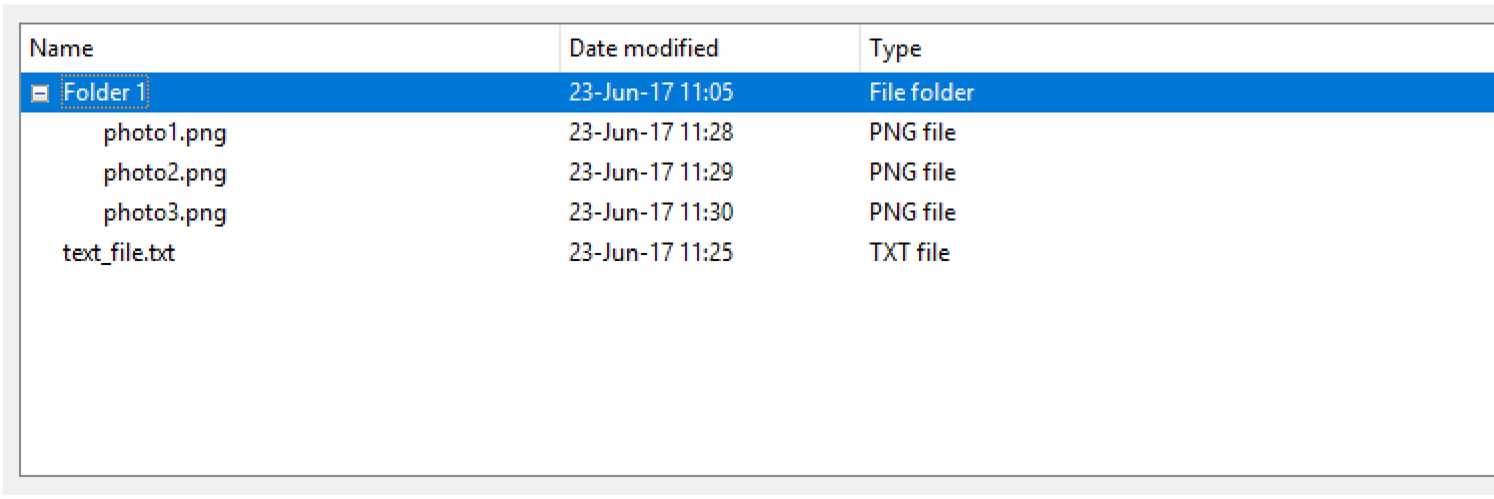
Insert some rows

```
# Level 1
folder1=tree.insert("", 1, "", text="Folder 1", values=("23-Jun-17 11:05","File folder",""))
tree.insert("", 2, "", text="text_file.txt", values=("23-Jun-17 11:25","TXT file","1 KB"))
# Level 2
tree.insert(folder1, "end", "", text="photo1.png", values=("23-Jun-17 11:28","PNG file","2.6
KB"))
tree.insert(folder1, "end", "", text="photo2.png", values=("23-Jun-17 11:29","PNG file","3.2
KB"))
tree.insert(folder1, "end", "", text="photo3.png", values=("23-Jun-17 11:30","PNG file","3.1
KB"))
```

Packing

```
tree.pack(side=tk.TOP,fill=tk.X)
```

On Windows, the following screenshot can be obtained from this example.



Name	Date modified	Type
Folder 1	23-Jun-17 11:05	File folder
photo1.png	23-Jun-17 11:28	PNG file
photo2.png	23-Jun-17 11:29	PNG file
photo3.png	23-Jun-17 11:30	PNG file
text_file.txt	23-Jun-17 11:25	TXT file

Progressbar

The widget `ttk.progress` is useful when dealing with long computations so that the user knows that the program is running. Following, an example updating a progressbar each 0.5 seconds is given:

Function updating the progressbar

```
def progress(currentValue):
    progressbar["value"]=currentValue
```

Set the maximum value

```
maxValue=100
```

Create the progress bar

```
progressbar=ttk.Progressbar(master,orient="horizontal",length=300,mode="determinate")
progressbar.pack(side=tk.TOP)
```

"determinate" mode is used when the progressbar is under control of the program.

Initial and maximum values

```
currentValue=0
progressbar["value"]=currentValue
progressbar["maximum"]=maxValue
```

Emulate progress each 0.5 s

```
divisions=10
for i in range(divisions):
    currentValue=currentValue+10
    progressbar.after(500, progress(currentValue))
    progressbar.update() # Force an update of the GUI
```

Read Ttk widgets online: https://riptutorial.com/tkinter/topic/10622/ttk-widgets

Credits

S. No	Chapters	Contributors
1	Getting started with tkinter	Billal BEGUERADJ, Bryan Oakley, Community, J.J. Hakala, j_4321, JGreenwell, Mike - SMT, Neil A., Nico Brubaker, Razik, ryneke, Tadhg McDonald-Jensen, tao, Yamboy1
2	Adding Images To Label/Button	Angrywasabi
3	Customize ttk styles	David Duran
4	Delaying a function	David Duran, Neil A., Tadhg McDonald-Jensen
5	Multiple windows (TopLevel widgets)	Tadhg McDonald-Jensen
6	Scrolling widgets	j_4321
7	The Tkinter Entry Widget	Angrywasabi, Bryan Oakley, double_j, j_4321
8	The Tkinter Radiobutton widget	j_4321, nbro, Parviz Karimli
9	Tkinter Geometry Managers	Henry
10	Ttk widgets	David Duran