**Tkinter** **notes**

While watching the educational video about Tkinter, I’ll take notes on any valuable details or concepts worth adding to this notes file.

**from tkinter import \***

* This line means **import everything** (all public functions, classes, and constants) from the tkinter module.
* It allows you to use Tkinter elements directly without writing tkinter. before them.

**Creating the Main Window**

* Tk() creates the main (root) window of the app.
* Use Tk() if you did from tkinter import \*, or tk.Tk() if you did import tkinter as tk.
* Only **one** Tk() instance should exist at a time.

from tkinter import \*

myframe = Tk()

**mainloop()**

* Keeps the window open and responsive to user actions.
* Must be called after creating the window.
* Use Toplevel() for extra windows if needed.

myframe.mainloop()

**.mainloop() Placement**

* .mainloop() should always be the **last line** in your Tkinter program.
* It starts the event loop that keeps the window **running and responsive**.
* Any code written **after** .mainloop() will only execute **after the window is closed**.

**Creating a Button**

* In Tkinter, Button is a **widget** (a GUI element like a label, entry box, etc.).
* It’s a **class (constructor)** used to create button objects.
* Example:
* mybutton = Button(myframe, text='Click me')
  + myframe → tells Tkinter which window the button belongs to.
  + text='Click me' → defines what appears on the button.
* After creating it, you must place it using a **geometry manager** like .pack() to make it visible:

mybutton.pack()

**Button Colors and Font**

* To change the **text color**, use fg='color', and for the **background**, use bg='color'.

mybutton = Button(myframe, text='Click Me', fg='white', bg='blue')

* You can also use **hex color codes** (e.g. #FF5733). Designers can get these codes from tools like color pickers, Adobe apps, or even Google.
* To change the **font**, use the font option:

font='FontName 20 bold'

* The font name refers to fonts **already installed** on your system — if a chosen font isn’t installed, Tkinter automatically switches to a default one.
* You can view all fonts available in Tkinter using the tkinter.font module:

import tkinter.font as font

print(font.families())

* This prints a list of all installed fonts that Tkinter can use on your system.
* Use any of these names inside the font=('FontName', size, style) option.

**Button Size**

* You can set a button’s **exact size** using width and height:

mybutton = Button(myframe, text='Click Me', width=10, height=2)

* Another way to make the button **bigger than its default size** is by adding internal padding:

mybutton.pack(padx=20, pady=10)

* padx is for width (horizontal space), and pady is for height (vertical space).
* They add extra space **around** the button inside the window.

**Button Command**

* The command option defines **what happens when the button is clicked**.
* You can link it to a built-in action or a custom function.

mybutton = Button(myframe, text='Exit', command=exit)

→ Clicking this button will close the program.

* You can also connect it to your own function:

def say\_hello():

print("Hello!")

mybutton = Button(myframe, text='Greet', command=say\_hello)

as you can see here, hello is without (), and that is because if we add them the function will work on its own, that why we write just the function without anything else.

* The command is one of the most important options in the Button widget because it controls the button’s **behavior**.

**.pack() Method**

* + .pack() displays the widget in the window.
  + You can control its position, e.g.:

mybutton.pack(side=LEFT)

It’s one of several **geometry managers** (others include .grid() and .place()).

**Window Title and Size**

* + Change the window name with:

myframe.title('Cipher Vault')

* + Set the window size with:

myframe.geometry('W x H') # Example: '400x300'

**webbrowser.open('link')**

* Opens a **web page** directly from Python using the default browser.
* Example:

import webbrowser

webbrowser.open('https://www.python.org')

* The 'link' is any valid URL you want to open.
* Often used inside a button’s command to open websites when clicked.

Button(myframe, text='Open Site', command=lambda:webbrowser.open('https://example.com')

**Label Widget**

* A **Label** is a Tkinter widget used to **display text or images** inside a window.
* It’s mainly for showing information, not for user interaction.

mylabel = Label(myframe, text='Welcome to CIPHER VAULT!')

mylabel.pack()

* You can customize it with options like fg, bg, font, and padx / pady.
* Labels are often used for **titles, instructions, or status messages** in the interface.

now I want to ask you if we print (mylabel = Label(myframe, text='Welcome to CIPHER VAULT!', pady=10) what pady will do here, while we have another element that is the button

**pady Inside a Label**

* pady inside a Label adds **vertical space inside the widget**.
* Example:

mylabel = Label(myframe, text='Welcome', pady=10)

mylabel.pack()

* The **label widget** becomes taller, giving the text "Welcome" more breathing room.
* This does **not** move other widgets (like a button) below it.

**pady in .pack()**

* When pady is used in .pack(), it adds **space outside the widget**, between it and other widgets.
* Example:
* mylabel.pack(pady=10)
* This pushes the label **down from the top** and also **adds space above the button** below.
* Use this to **control spacing between widgets** in the window.

**anchor Option**

* The anchor option controls **where the widget’s content is positioned** **inside its area**.
* Example:
* mylabel.pack(anchor='ne')
* Direction values use compass points:
  + n, s, e, w → north, south, east, west
  + ne, nw, se, sw → corners
  + center → default
* Example: anchor='nw' places the widget **at the top-left corner** of the window.
* Used with .pack() or .place() to fine-tune widget positioning.

**Positioning Widgets with side**

* The side option in .pack() controls **where the widget is placed** inside the window.
* Example:

mybutton.pack(side=LEFT)

* Common values: TOP, BOTTOM, LEFT, RIGHT.
* It’s not only for buttons — you can use it with **any widget** (labels, entries, etc.) to control their position in the window.

ipad is an option to add an internal space in components

**Entry Widget**

* The **Entry** widget creates a small text box for the user to **type input**.
* Example:

myentry = Entry(myframe)

myentry.pack()

**ipadx and ipady**

* ipadx and ipady add **internal padding** (extra space **inside** the widget’s borders).
* ipadx → adds horizontal space (width).
* ipady → adds vertical space (height).
* Example:

mybutton.pack(ipadx=10, ipady=5)

* This makes the widget’s content (like text or icons) look less tight and more centered.

**fill and expand in pack()**

* fill allows a widget to **stretch** when the window is resized.
* It makes the widget grow **horizontally (fill='x')**, **vertically (fill='y')**, or **both (fill='both')**.
* Example:

mybutton.pack(fill='x')

→ The button stretches across the window width.

* However, when you use side (like LEFT or RIGHT), fill might not behave as expected.
* To make it work better, you can add expand=True, which lets the widget take up **extra empty space** and resize more freely.

Example:

mybutton.pack(fill='both', expand=True)

→ The widget now grows in all directions when the window is resized.