Lets create a wifi grabber for windows

Class WifiPasswordExtractor:
↳ defines the class for encapsulation

Step 1

def __init__(Self, master):
↳ * dunder method init to make a Construc
↳ * takes root window as parameter
↳ * windows property

Initialize the main frame

Helps in further root. functionality

```
Self.master = master
Self.master.geometery("500x700")
Self.master.title("Wifi Password Extractor)
Self.master.resizable(False, False)
```

→ Sets window si

→ Sets ti

↳ not resizable

Step 2

Self.Create_widgets()
↳ will be the contents of the UI

```
Self.main_frame = CTkframe (Self.master)
```

↳ By the refrence of $\boxed{\text{master}}$ → root window a frame is created like a container widget (like a box) to hold other widgets!

```
Self.main_frame.pack (padx=20, pady=20
    fill="both", expand = True.)
```

place the frame in the window with padding (padx, pady) fill by both side and Expandable by window size if resizable would be true

Self.scroll_frame = CTKScrollableFrame(Self.main_fram, width=600, height=350)

↳ Scrollable fram under master Channing main_fram under main frame So main_frame becomes master for the scroll frame
+ gth Scrollable

Self.scroll_frame.pack(pady=20, padx=20, fill="both", expand=True)

↳ placing scrollable plain on the main_frame to be visible with padding + hieght width adjustment

CTKButton(~~Self~~
  Self.main_frame, → When to place
  text = "Show password with AP", → what should be written
  Command = Self.Show_~~passwords~~ profiles → what to do on button click
).pack(pady=5)
  ↓
  on the gap of 5px up and down

Same
CTkButton(
  Self.main_frame,
  text = "Clear result",
  Command = Self.clear_result,
).pack(pady=5)

CTk Button(
  Self.main_frame,
  text = "Exit app", Command = Self.exit_app
).pack(pady=5)

def show_profiles

try: #Exception handling

Self. clear_results() # clear previos data

profile_data = sb. check_output ("netsh wlan show profiles", shell = True). decode()

↳ sb. check_output will store the output to a variable, Output to what? to wifi command netsh wlan show profiles that is to be then decoded to human readable format.

profiles = re. findall (r"All users profile\s*:\s*(.*)", profile_data)

↳ apply regex operation on available string stored in profile_data and return a list of it

If not profiles:

If no profile found then label will be created

CTkLabel(
    self. scroll_frame,
    text = "No wifi profiles found.",
    font = ("Arial", 14)
). pack (anchor = "W", padx = 10, pady=5)

⟷ anchored to west

return

↳ will get out of the program.

for profiles in profiles:

try: → Exception Handling

Will Extract the password

```
details = sb.check_output(
        f'netsh wlan show profile name = "{
    profile}" Key=clear', shell=True).decode()
```

and will save the result 1 By 1 in details where further detailed program will be executed

```
password match = re.search(r"Key Content\s*:\s*
    (.*)", details)
```

↳ regex usage to extract Key Content

```
password = password match.group(1) if passwor-
     match else "No password (Open Network)"
```

↳ grouping of each password, displayed below

For password and A.P display

```
CTkLabel(
    self.scroll_frame, → where to add label
    text = f"Wifi Name: {profile}\n Password:
~~except~~ (Key Content): {password}\n\n",
    font=("Arial", 15),
    justify = "left",
    anchor = "w",
    wraplength=600
    ).pack(anchor="w", padx= 10, pady=5)
```

Text display formatting

except Exception ~~as e~~:
↳ Show general error if password Extraction fails

CTk label (

where to show, how the text formatting

{ Self. scroll_frame,
text = f" Error for {profile} for retrieving password",
font = ("Arial", 13),
justify = " left",
anchor = "W",
wraplength = 600

wrap text after 600 pixels!

). pack (anchor = "W", padx = 10, pady = 5)
↳ position of label!

except exception as e:
Show exception if profile retrieval fails

CTklabel (

where to put the Warning and about text formatting

{ Self. Scroll_frame
text = f " X Failed to retrieve profiles. \n {e}",
font = ("Arial", 13)
). pack (anchor = "W", padx = 10, pady = 5)
↳ display on the window (master)

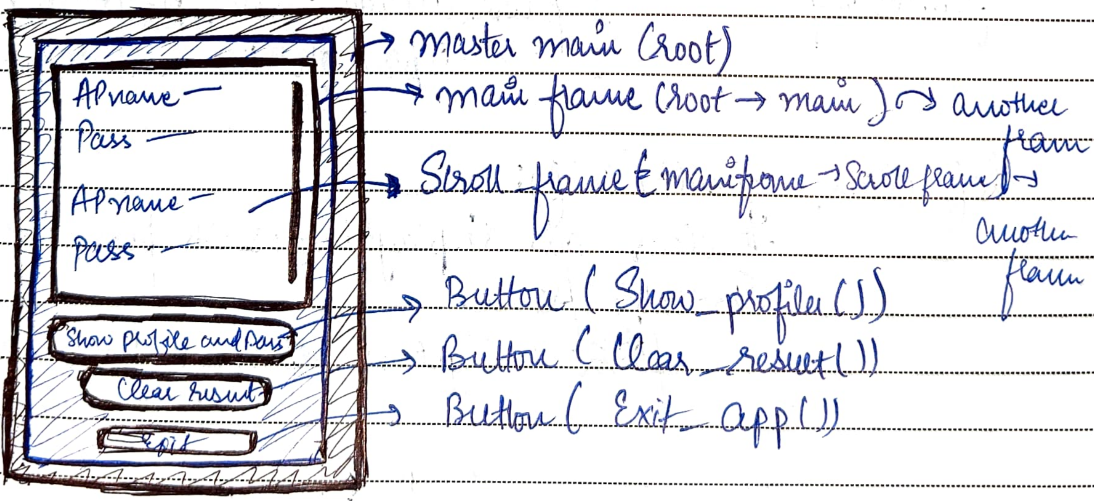
def Clear_result (Self):
# clear all displayed results from the Scrollable frame
~~from~~ for widget in Self. Scroll_frame . winfo_children ():
widget . destroy ()

→ widget is each GUI element inside the frame
→ in self.scroll_frame.winfo_children() method that returns a [list] of all child widgets inside a container (like scroll_frame)
→ .destroy() will remove/delete the widget

```
def exit_app(self):
    self.master.distroy()
```
↳ distroy function again to remove or delete the ~~widget~~ application

```
If __name__ == "__main__":
```
↳ dunder method for initializing the project
```
    root = CTk()
```
→ Creating the main window
```
    app = WiFiPasswordExtractor(root)
```
making object for class, passing root as parameter as master
```
    root.mainloop()
```
↳ last line for the finishing of the task

So the interface looks like



AP name
Pass
AP name
Pass
Show profile and pass
Clear result
Exit
master main (root)
main frame (root → main) → another frame
Scroll frame (main frame → Scroll frame) → another frame
Button (Show_profiles())
Button (Clear_result())
Button (Exit_app())