

# Python Mini Project Report

## Password Manager

### Members:

Roll Number	Name
Shivom Karnad	16010121081
Ayush Khandare	16010121084
Kedar Kulkarni	16010121096

### Description:

Password Manager is a simple but very handy tool to generate strong random passwords with a user defined criteria and store them in an encrypted hidden file. It will be only accessible if master username and password is entered correctly.

## System Architecture:

In total the program is divided into two parts to make it easier to bug fix and understand. The parts being main.py and UI.py (UI.py cannot be renamed without changing main.py).

main.py contains the inner working of the code. File handling, encryption, decryption, password generation and management is handled in main. It consists of a total of 7 functions, each tasked with one aspect of the program.

UI.py is a User Interface file. It has a total of 6 major functions which handle the 6 different types of UI windows that the program cycles through as needed. The entire UI is made with the help of tkinter and serves as a module to be imported into main.

main-UI integration is handled in the main.py file in the various functions. It can be said that the UI.py is an extension of main.py as in, UI.py functions take the inputs from the user and then pass it onto the main.py functions. For every main.py function that requires a user input, there is a UI.py function that is called to do the same.

## Working:

On first boot, the program takes master username and password from the user. These are to be then encrypted and stored in a hidden file "master.txt" for verifying subsequent launches of the program.

On reruns, the program then collects the master username and password entered during the first boot from the "master.txt" file and then decrypt it. It then collects the user-input username and password and verifies with the master. If they do not match, the program will give a small alert to notify the user of the wrong credentials.

Upon verifying the credentials, the user is greeted with a display with 3 options. The options being:

1. To create and store a new password
2. To display all the passwords created previously
3. Save and Exit

“Save and Exit” being the simplest of the three shuts down the program but only after displaying a “Have a good day!” message. It must be noted that this message will pop up if the user shuts down the program with the “X” button on the any of the windows as well.

Choosing the “Create a new password” option takes the user to another window which takes the requirements of the password. Then, a character set is generated according to the user requirement. One letter of the password is picked from every character type set to ensure that the final password contains all the types.

The character type sets being:

1. ascii lowercase characters
2. ascii uppercase characters
3. ascii digits
4. Some special characters

Rest of the password is randomly picked from the entire character set. This is then scrambled to create a truly (artificially) random password.

The password is then displayed onto a new window where the user gets the choice to keep the current generated password or to generate a new one. If the user decides to keep the password, user then has to input a caption for the password. The password along with its caption is then encrypted and stored in “random\_passwords.txt” file which is then hidden. Otherwise, the user is sent back to enter requirements to regenerate the password.

Choosing the “Show saved passwords” option will display all the passwords in a new window. The contents of the “random\_passwords.txt” are scanned and compiled into a list of passwords and their respective captions. This list is then color coded and displayed in a listbox, complete with a scroll bar.

**For a more technical working, comments are added in the code to help explain some of the intricate details.**

## Features:

1. Exception Handling
2. Random Password Generator
3. Random Password Manager
4. Simple Encryption (Fixed Key)
5. Login Verification
6. Crash and bug resistant

## Packages Used:

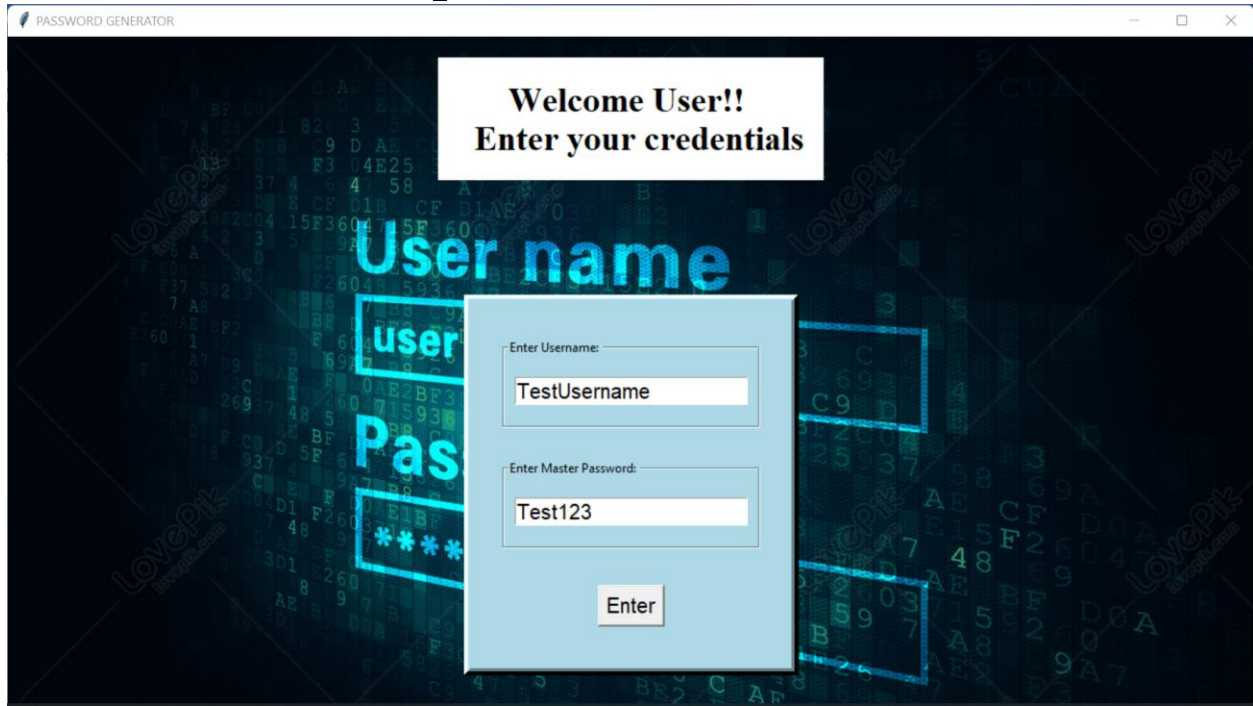
1. **tkinter** : to create all the UI elements
2. **pillow(PIL)** : to add jpg background image
3. **random** : to generate random passwords
4. **string** : for ascii character sets and digits
5. **os** : for file handling
6. **base64** : for simple encryption

## Contributions:

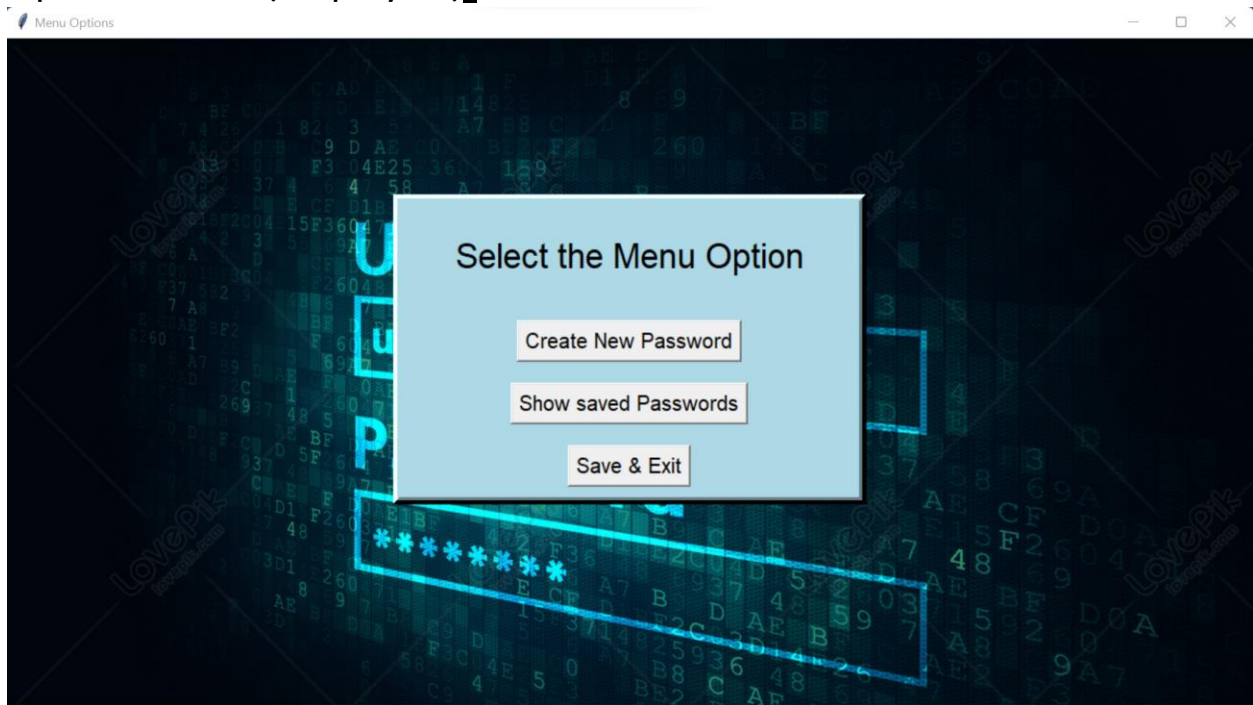
1. **Shivom Karnad** : Worked on Encryption, file handling and UI
2. **Ayush Khandare** : Worked on Frontend UI and adding features in system
3. **Kedar Kulkarni** : Worked on logic, exception handling and integration

## Outputs:

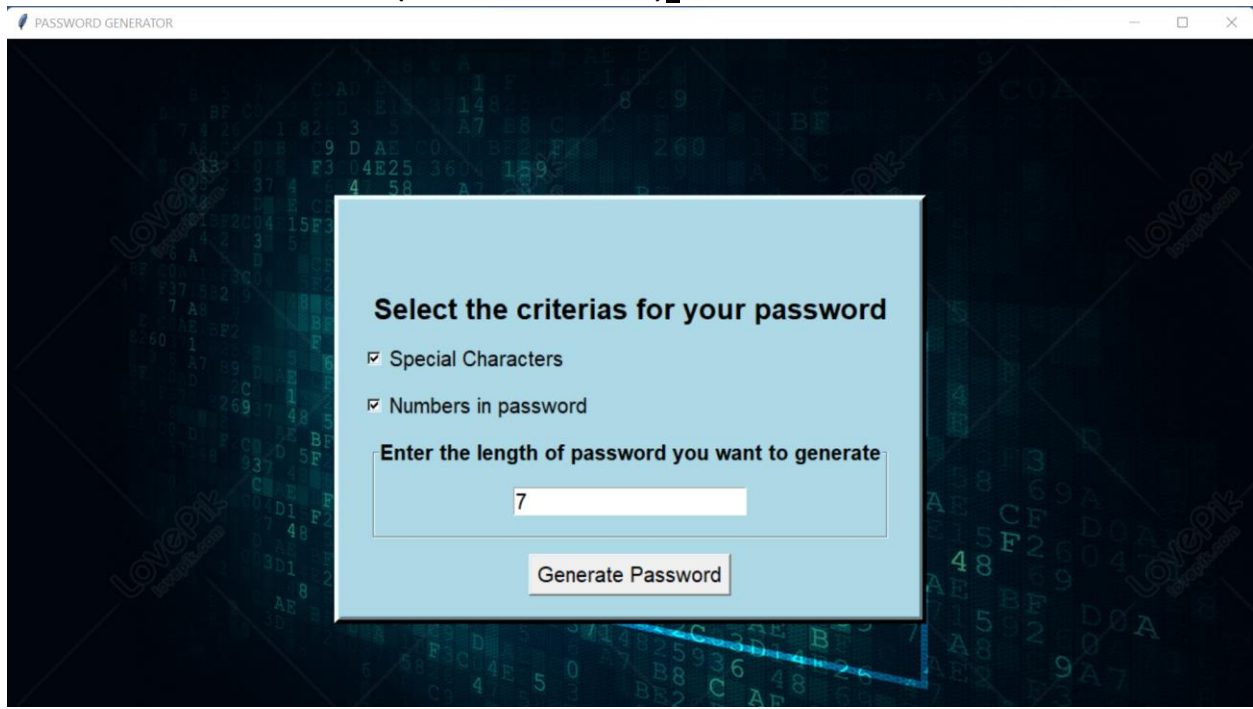
### Login Screen (Master UI):



### Options Menu (Display UI):



## Password Parameters (Parameters UI):



A screenshot of a web application titled "PASSWORD GENERATOR". The background is dark with a pattern of green and blue hexadecimal characters. A light blue dialog box is centered on the screen. Inside the dialog, the text "Select the criterias for your password" is at the top. Below it are two checked checkboxes: "Special Characters" and "Numbers in password". Underneath these is the text "Enter the length of password you want to generate" followed by a text input field containing the number "7". At the bottom of the dialog is a button labeled "Generate Password".

PASSWORD GENERATOR

Select the criterias for your password

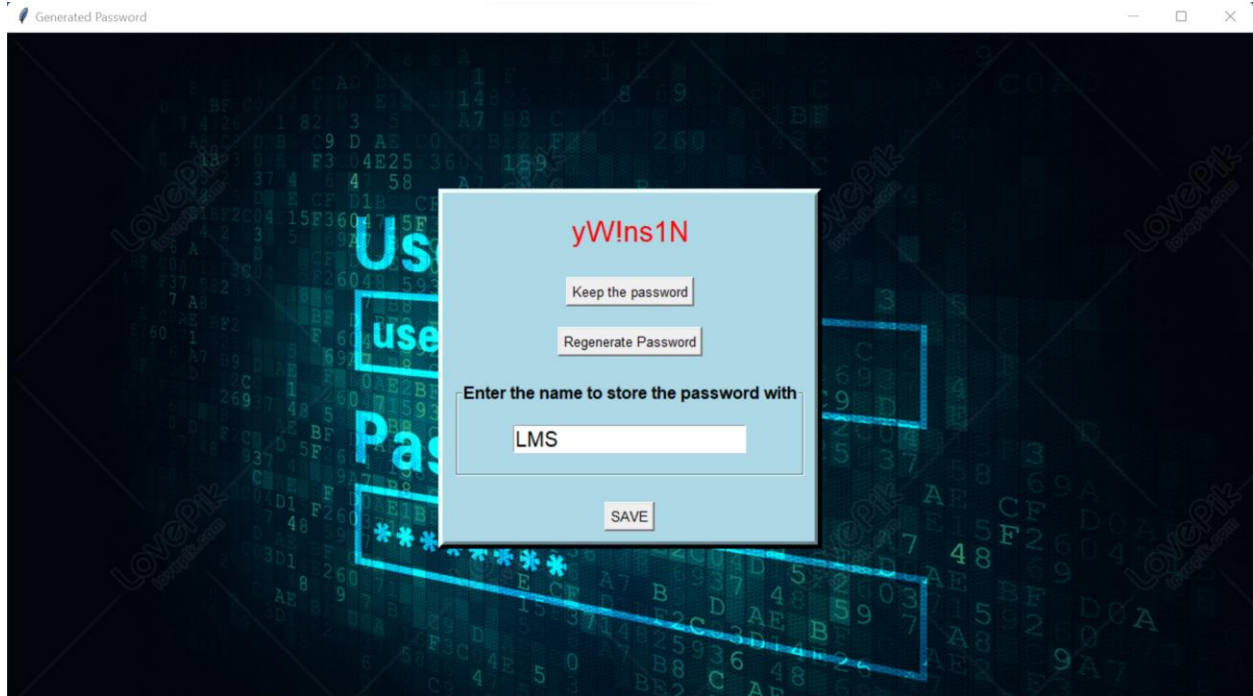
- ☒ Special Characters
- ☒ Numbers in password

Enter the length of password you want to generate

7

Generate Password

## Generated Password (Password Maker UI):



A screenshot of a web application titled "Generated Password". The background is dark with a pattern of green and blue hexadecimal characters. A light blue dialog box is centered on the screen. Inside the dialog, the generated password "yWIns1N" is displayed in red text at the top. Below it are two buttons: "Keep the password" and "Regenerate Password". Underneath these is the text "Enter the name to store the password with:" followed by a text input field containing the text "LMS". At the bottom of the dialog is a button labeled "SAVE".

Generated Password

yWIns1N

Keep the password

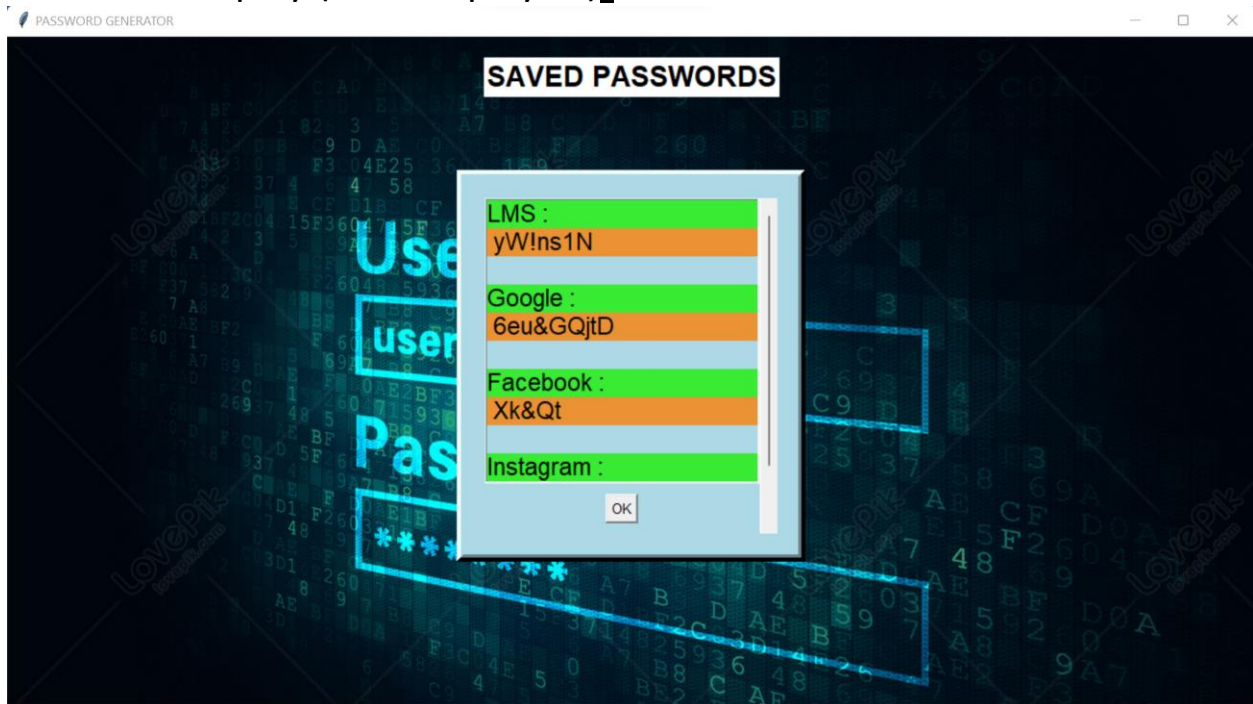
Regenerate Password

Enter the name to store the password with:

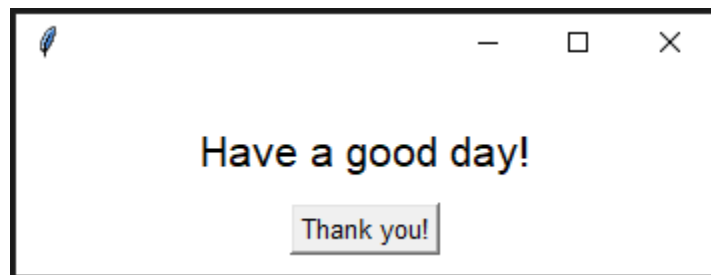
LMS

SAVE

## Password Display (Pwd Display UI):



## Save and Exit (Alert UI):





## Code:

### #main.py

```
import string #for ascii letters and digits
import random #for smhfrandom password generation
import base64 #for base64 encoding and decoding
import os #for deleting unrequired files and hiding certain others
import UI #User Interface

def pwd_gen(length, special, digits, toggle):
    pwd= [] #temporary var to store password
    #Entire character set
    chars = "".join(string.ascii_letters) #all aplhabets upper and lower
    if digits: #toggle for digits
        chars = chars.join(string.digits) #add digits to character set
        pwd.append(random.choice(string.digits)) #choose one random digit
    if special: #toggle for special characters
        chars = chars.join("!@#$$%^&*?_") #add special characters to character set
        pwd.append(random.choice("!@#$$%^&*?_")) #choose one random special
character
        pwd.append(random.choice(string.ascii_uppercase)) #picking one lowercase
Letter
        pwd.append(random.choice(string.ascii_lowercase)) #picking one uppercase
Letter

    #Making random password
    for i in range(length-toggle): #picking the rest of the password
        pwd.append(random.choice(chars))
    #Shuffling password
    pwd_shuff = "" #temporary var to store scrambled password
    for i in range(length):
        n = random.randint(0,length-i)-1 #randomly pick one index
        pwd_shuff += pwd[n] #add the character at that index
        del(pwd[n]) #delete that index
    return pwd_shuff

def parameters():
    #Exception Handling format
    while True:
        toggle = 0 #a variable to set character set
```



```

    try:
        #Toggle for special characters
        special, digits, length = UI.parameters_UI() #get requirements for
password
        if special: #if user wants special characters
            toggle += 1 #increase types of characters used by 1
        if digits: #if user wants digits
            toggle += 1 #increase types of characters used by 1
        length = int(length) #to check if length is an integer
        if length > 20: #if length is greater than 20
            UI.alert_UI("Length cannot be greater than 20!")
            continue #restarts loop
        if length <= toggle+2: #is length is less than minimum length to
cover all character types
            UI.alert_UI(f"Length should be greater than {toggle+2}!")
            continue #restarts loop
    except ValueError: #if anything other than an integer is input
        UI.alert_UI("Please Enter an Integer value!")
        continue #restarts loop
    break #if everything is correct
    return length, special, digits, toggle

def pwd_maker():
    while True:
        length, special, digits, toggle = parameters() #taking password criteria
        password = pwd_gen(length, special, digits, toggle+2) #generating
password
        caption = UI.pwd_maker_UI(password) #get caption and display password
        if type(caption) == str: #if caption is entered
            if caption.find(":") != -1: #if ":" in caption
                UI.alert_UI("' ':' is an Invalid Input!")
                continue #since ":" can break display as it uses .split(":")
            break #correct caption
        final_password = f"{caption} : {password}" #save format
        final_password = encrypt(final_password) #encryption
        os.system("attrib -h random_passwords.txt")
        with open(f"random_passwords.txt", "a") as f:
            f.write(f"{final_password}\n") #storing excrpyted password
        os.system("attrib +h random_passwords.txt")

def pwd_display(): #displaying passwords
    try:
        caption, password = [], [] #temporary lists

```

```

        with open("random_passwords.txt","r") as f:
            for line in f.readlines(): #taking single line from file
                line = line.replace("\n","") #to avoid errors
                line = decrypt(line)
                contents = line.split(":") #split with respect to ":"
                caption.append(contents[0]) #before ":" = username
                password.append(contents[1]) #after ":" = password
            UI.pwd_display_UI(caption, password) #display
        except FileNotFoundError: #if not passwords were created thusfar
            UI.alert_UI("No passwords were created!")

def encrypt(string_sample): #base64 encryption
    string_bytes = string_sample.encode("ascii") #convert string into its ascii
    value
    base64_bytes = base64.b64encode(string_bytes) #convert ascii value into
    base64 binary
    base64_string = base64_bytes.decode("ascii") #convert base64 binary back into
    encoded ascii
    return base64_string #return encoded base64

def decrypt(base64_string): #base64 decryption
    base64_bytes = base64_string.encode("ascii") #convert encoded ascii into
    base64 binary
    string_bytes = base64.b64decode(base64_bytes) #convert base64 binary into
    ascii value
    string_sample = string_bytes.decode("ascii") #convert ascii value into string
    return string_sample #return decoded string

def master():
    try:
        with open("master.txt","r") as f:
            master = [] #temporary list
            for line in f.readlines():
                master.append(decrypt(line)) #adds lines of master.txt in master
    list

    try:
        username = master[0].replace("\n","") #retrieves username
        master_password = master[1].replace("\n","") #and password
    except IndexError: #incase master file contents were deleted
        os.remove("random_passwords.txt")
        os.remove("master.txt")
        quit()

```

```

        try:
            input_username, input_password = UI.master_UI() #take input
credentials
        except NameError: #if user closes window without any inputs
            UI.alert_UI("Have a good day!")
            quit()
        if master_password == input_password and username == input_username:
#valid credentials
            return False, username #stop loop
        else: #input password is not the same as master
            UI.alert_UI("Incorrect Credentials!")
            return True, username #reloop

except FileNotFoundError:
    with open("master.txt","w") as f: #on first boot/ master file was deleted
        username, master_password = UI.master_UI()
        f.write(f"{encrypt(username)}\n") #write username
        f.write(encrypt(master_password)) #write password
        os.system("attrib +h master.txt") #hides master file
        if os.path.exists("random_passwords.txt"): #incase master file was
deleted
            os.remove("random_passwords.txt") #passwords will be deleted
        return False, username #reloop

run = True #variable to control while loop
while run:
    run, username = master() #run master until correct credentials are entered

while True: #loop until program is closed

    u = UI.display_UI() #call display
    if u == 0: #if window was closed without any inputs
        UI.alert_UI("Have a good day!")
        quit()
    if u==1: #create new password
        pwd_maker()
    elif u==2: #display saved passwords
        pwd_display()
    elif u==3: #save and exit
        UI.alert_UI("Have a good day!")
        quit()

```

## #UI.py

```
from tkinter import *
from PIL import Image, ImageTk #background image

def parameters_UI():
    ###main window
    root = Tk()
    root.geometry('1200x750')
    root.title("PASSWORD GENERATOR")
    root.minsize(550, 550)
    root.configure(bg = 'white')
    root.minsize(1200,750)
    root.maxsize(1200,750)

    ###Bg Image
    photo = Image.open('Password generator/bg image.jpg')
    picture = ImageTk.PhotoImage(photo)
    pict = Label(image = picture)
    pict.place(x = 0, y = 0, relwidth=1, relheight=1)

    ###Frame for criterias
    frame = LabelFrame(root, padx = 20, pady = 20, bg = '#ADD8E6', relief =
RAISED, borderwidth=7)
    frame.pack(pady=150)

    ###Heading for criterias
    criteria = Label(frame, text = '\n\nSelect the criterias for your password',
font = 'helvetica 20 bold', fg = 'black', bg = '#ADD8E6')
    criteria.pack()

    ###Checkbox Command
    def checkbox():
        response_special.config(text = var.get())
        response_digits.config(text = var1.get())
        global inp
        inp = len_entry.get(1.0, "end-1c")
        root.destroy()

    ###Checkbox variable
```

```

var = StringVar(value = 'No Characters')
var1 = StringVar(value = 'No digits')

###Creating Checkboxes
special_char = Checkbutton(frame, text = 'Special Characters', variable=var,
onvalue="Include Char",offvalue='No Characters', font = 'helvetica 15', bg =
'#ADD8E6')
special_char.pack(anchor = 'nw', pady = 10)

digits = Checkbutton(frame, text = 'Numbers in password', onvalue="Include
digits", offvalue="No digits", variable=var1, font = 'helvetica 15', bg =
'#ADD8E6')
digits.pack(anchor = 'nw')

response_special = Label(root)
response_digits = Label(root)

#Creating Length Criteria
length = LabelFrame(frame, text = 'Enter the length of password you want to
generate', bg = '#ADD8E6', font = 'helvetica 15 bold')
length.pack(anchor = 'nw', pady = 15, padx = 10)
len_entry = Text(length, height = 1, width = 20, font = 20)
len_entry.pack(pady = 20, padx = 10)

###Creating Generate Button
generate = Button(frame, text = 'Generate Password', font = 'helvetica 15',
command = checkbox)
generate.pack()
root.mainloop()

if var.get() == 'No Characters':
    spchar = False
else:
    spchar = True
if var1.get() == 'No digits':
    num = False
else:
    num = True
return spchar, num, inp

def master_UI():

```

```

#widget is created
window = Tk()
window.geometry('1200x750')
window.title("PASSWORD GENERATOR")
window.minsize(550, 550)
window.configure(bg='white')
window.minsize(1200,750)
window.maxsize(1200,750)

###Bg Image
photo = Image.open('Password generator/bg image.jpg')
picture = ImageTk.PhotoImage(photo)
pict = Label(image = picture)
pict.place(x = 0, y = 0, relwidth=1, relheight=1)

# title
title = Label(window, text=''Welcome User!!
Enter your credentials  '',
font='Times 25 bold',bg='white', pady=20, fg='black')
title.pack(pady = 20)

frame = LabelFrame(window, padx=20, pady=20, bg='#ADD8E6',
                    relief=RAISED, borderwidth=7)
frame.pack(padx=30, pady=90)

def close():
    global usr
    global pwd
    usr = username.get()
    pwd = password.get()
    window.destroy()

# username
details = LabelFrame(frame, text='Enter Username: ', bg='#ADD8E6')
username = Entry(details, font='helvetica 15')
details.pack(pady=15, padx=10)
username.pack(pady=20, padx=10)

# master password
details = LabelFrame(frame, text='Enter Master Password: ', bg='#ADD8E6')
password = Entry(details, font='helvetica 15')
details.pack(pady=15, padx=10)
password.pack(pady=20, padx=10)

enter = Button(frame, text = 'Enter', font = 'helvetica 15', command = close)

```

```

enter.pack(pady = 20)

window.mainloop()
try:
    return usr, pwd
except NameError:
    alert_UI("Have a good day!")
    quit()

def display_UI():
    root = Tk()
    root.geometry('1200x750')
    root.title("Menu Options")
    root.configure(bg = 'white')
    root.minsize(1200,750)
    root.maxsize(1200,750)

    ###Bg Image
    photo = Image.open('Password generator/bg image.jpg')
    picture = ImageTk.PhotoImage(photo)
    pict = Label(image = picture)
    pict.place(x = 0, y = 0, relwidth=1, relheight=1)

    frame = LabelFrame(root, bg = '#ADD8E6', relief=RAISED, borderwidth=7)
    frame.pack(pady=150)

    global a
    a = 0

    def create():
        global a
        a = 1
        root.destroy()

    def show():
        global a
        a = 2
        root.destroy()

    def save_exit():
        global a
        a = 3
        root.destroy()

```



```

    option = Label(frame, text = 'Select the Menu Option', font = 'helvetica 25',
fg = 'black', bg = '#ADD8E6')
    option.pack(padx = 50, pady = 30)

    create = Button(frame, text = 'Create New Password', font = 'helvetica 15',
command=create)
    create.pack(pady = 10)

    show = Button(frame, text='Show saved Passwords', font='helvetica 15',
command=show)
    show.pack(pady = 10)

    save_exit = Button(frame, text = 'Save & Exit', font = 'helvetica 15',
command = save_exit)
    save_exit.pack(pady = 10)

    root.mainloop()
    return a

def pwd_display_UI(caption, password):
    tags = caption
    passwords = password

    window = Tk()
    window.geometry('1200x750')
    window.title("PASSWORD GENERATOR")
    window.minsize(550, 550)
    window.configure(bg='white')
    window.minsize(1200,750)
    window.maxsize(1200,750)

    ###Bg Image
    photo = Image.open('Password generator/bg image.jpg')
    picture = ImageTk.PhotoImage(photo)
    pict = Label(image = picture)
    pict.place(x = 0, y = 0, relwidth=1, relheight=1)

    def ok_button():
        window.destroy()

    title = Label(window, text="SAVED PASSWORDS", bg = "white",
font="helvetica 20 bold")
    title.pack(pady=20)

```

```

frame = LabelFrame(window, padx=20, pady=20, bg='#ADD8E6',
                    relief=RAISED, borderwidth=7)
frame.pack(pady=50, padx=50)

scrollbar = Scrollbar(frame)
scrollbar.pack(side=RIGHT, fill=Y)

disp = Listbox(frame, bg='#ADD8E6', font='helvetica 17',
               yscrollcommand=scrollbar.set)

for i in range(len(passwords)):
    disp.insert(END, tags[i] + ':')
    disp.itemconfig(END, bg='#3aeb34')
    disp.insert(END, passwords[i])
    disp.itemconfig(END, bg='#eb9334')
    disp.insert(END, "")
    disp.pack()

    ok_button = Button(frame, text = 'OK', font = 'helvetica 10', command =
ok_button)
    ok_button.pack(pady = 10, padx=20)

window.mainloop()

def pwd_maker_UI(password):
    global regen
    regen = 0
    root = Tk()
    root.geometry('1200x750')
    root.title('Generated Password')
    root.configure(bg = 'white')
    root.minsize(1200,750)
    root.maxsize(1200,750)

    ###Bg Image
    photo = Image.open('Password generator/bg image.jpg')
    picture = ImageTk.PhotoImage(photo)
    pict = Label(image = picture)
    pict.place(x = 0, y = 0, relwidth=1, relheight=1)

    def keep_pw():
        def save_button():
            global input_tag
            input_tag = entrybox.get()

```

```

        if input_tag == "":
            input_tag = ""
        else:
            root.destroy()

        sitename = LabelFrame(frame, text = 'Enter the name to store the password',
with', bg = '#ADD8E6', font = 'helvetica 12 bold')
        sitename.pack(pady = 15, padx = 10)
        entrybox = Entry(sitename, font = 'helvetica 15')
        entrybox.pack(pady = 20, padx = 10)
        ok_button = Button(frame, text = 'SAVE', font = 'helvetica 10', command =
save_button)
        ok_button.pack(pady = 10, padx=20)

    def regenerate():
        global regen
        regen = 1
        root.destroy()

    frame = LabelFrame(root, bg = '#ADD8E6', relief=RAISED, borderwidth=7)
    frame.pack(pady = 150)

    pw_display = Label(frame, text = password, font = 'helvetica 20', bg =
'#ADD8E6', fg = 'red')
    pw_display.pack(pady = 15, padx = 50)

    keep = Button(frame, text = 'Keep the password', font = 'helvetica 10',
command = keep_pw)
    keep.pack(pady = 10)

    regenerate = Button(frame, text = 'Regenerate Password', font = 'helvetica
10', command = regenerate)
    regenerate.pack(pady = 10)

    root.mainloop()
    if regen == 1:
        return False
    if regen == 0:
        try:
            return input_tag
        except NameError:
            alert_UI("Have a good day!")
            quit()

```

```

def alert_UI(message):
    root = Tk()
    root.geometry('350x100')
    root.title("")
    root.configure(bg = "white")

    def ok_button():
        root.destroy()

    invalid = Label(text=f'\n{message}', bg = 'white', font = 'helvetica 16')
    invalid.pack()

    text = "OK"
    if message == "Have a good day!":
        text = "Thank you!"

    ok_button = Button(text = f"{text}", font = 'helvetica 10', command =
ok_button)
    ok_button.pack(pady = 10, padx=20)

    root.mainloop()

```

## References:

- <https://www.geeksforgeeks.org/encoding-and-decoding-base64-strings-in-python/>
- <https://www.geeksforgeeks.org/python-gui-tkinter/>

## Also Available on GitHub:

<https://github.com/Bad-Astronomer/FY-PythonProject>