Python Fundamentals

by Viktor Grozdev

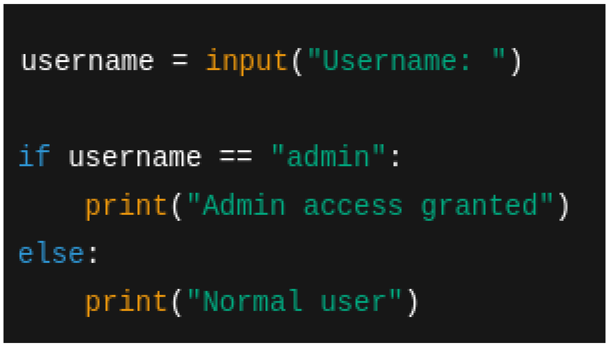
1. Conditions & Variables

Program:

A set of instructions executed **top to bottom** by the computer.

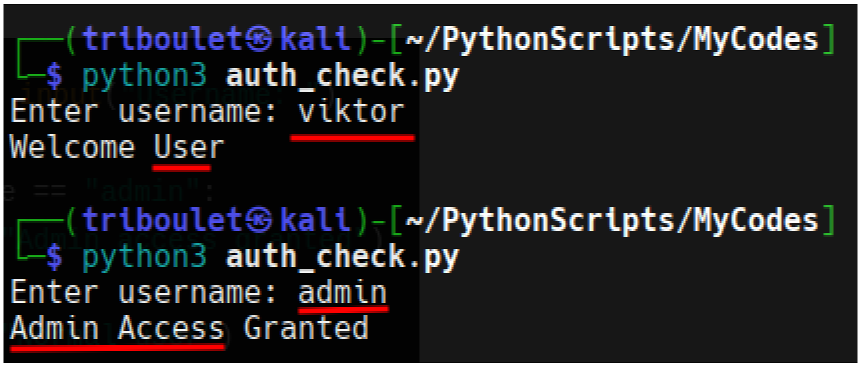
Mini Authorization process:

1. **Code v1**



```
username = input("Username: ")

if username == "admin":
    print("Admin access granted")
else:
    print("Normal user")
```

This is a very mini example of an authentication page. We can see that if we do not enter specifically “admin” we will not get the admin access.

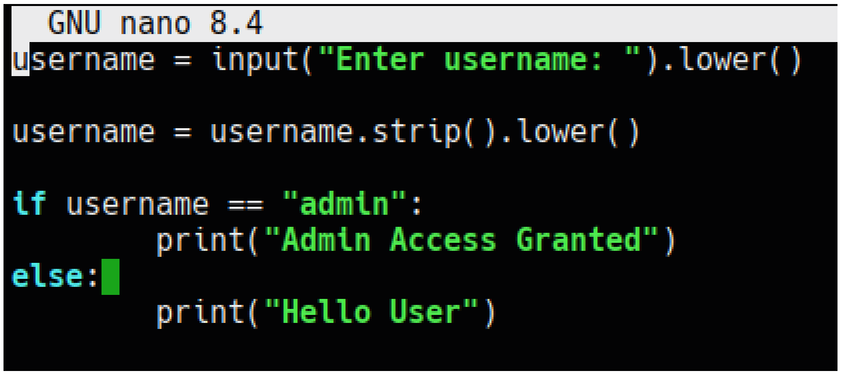


```
┌──(triboulet㉿kali)-[~/PythonScripts/MyCodes]
└─$ python3 auth_check.py
Enter username: viktor
Welcome User

┌──(triboulet㉿kali)-[~/PythonScripts/MyCodes]
└─$ python3 auth_check.py
Enter username: admin
Admin Access Granted
```

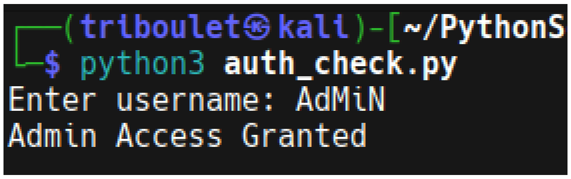
Everything besides **admin** gives us the normal user. We can type **AdMin** and we still will not get permission.

2. Code v2

Now let’s examine a logic change that can be seen:



```
  GNU nano 8.4
username = input("Enter username: ").lower()

username = username.strip().lower()

if username == "admin":
        print("Admin Access Granted")
else:
        print("Hello User")
```



```
┌──(triboulet㉿kali)-[~/PythonS
└─$ python3 auth_check.py
Enter username: AdMiN
Admin Access Granted
```

As we can see here typing **AdMiN** gives us access even though we didn’t type **admin** as set in the code. That’s because of this line `username = username.strip().lower()` telling python to normalize characters `.lower()` and to remove spaces `.strip()`

3. Summary

This section shows how authentication depends entirely on logic decisions made by the developer. Small changes in input handling can either strengthen or weaken security. Attackers look for cases where input is not properly validated or normalized.

2.Functions & Loops

[🧠] Core Idea

Functions = reusable logic

Loops = repeated attempts

–

Every brute-force tool looks like this:

loop → try input → check result → repeat

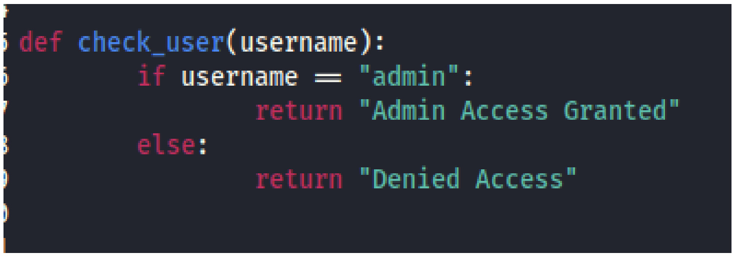
2.1. Functions

What is a function?

A function is **a named block of code** that:

- Receives input
- Does something with it
- Returns a result

Example #1:



```python
def check_user(username):
        if username == "admin":
                return "Admin Access Granted"
        else:
                return "Denied Access"
```

In this picture we can see that we have defined a function!

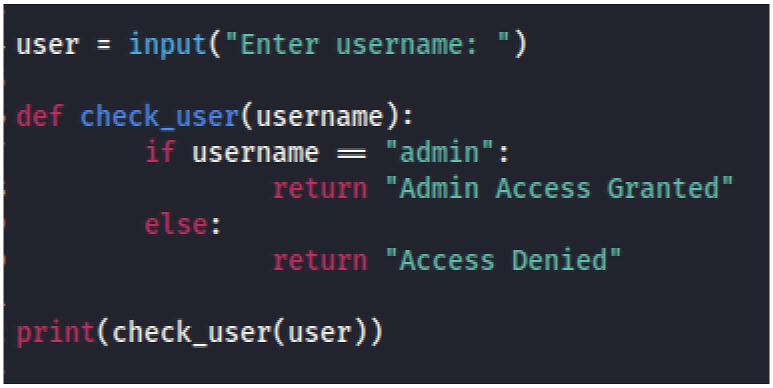
`def` -> defines the function

`check_user` -> this is the function name (this is optional)

`username` -> this is the output we are going to follow. (from previous code!

`return` -> output

This is how the full code should look like!



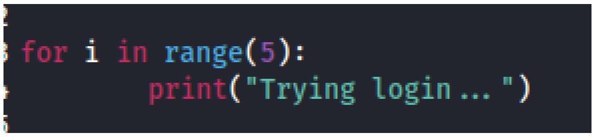
```python
user = input("Enter username: ")

def check_user(username):
        if username == "admin":
                return "Admin Access Granted"
        else:
                return "Access Denied"

print(check_user(user))
```

When we run the code we can see that typing **admin** will give us **Admin Access Granted**. Anything else would give us **Access Denied**.

2.2. Loops

What is a loop?

A loop repeats code **until a condition is met**.



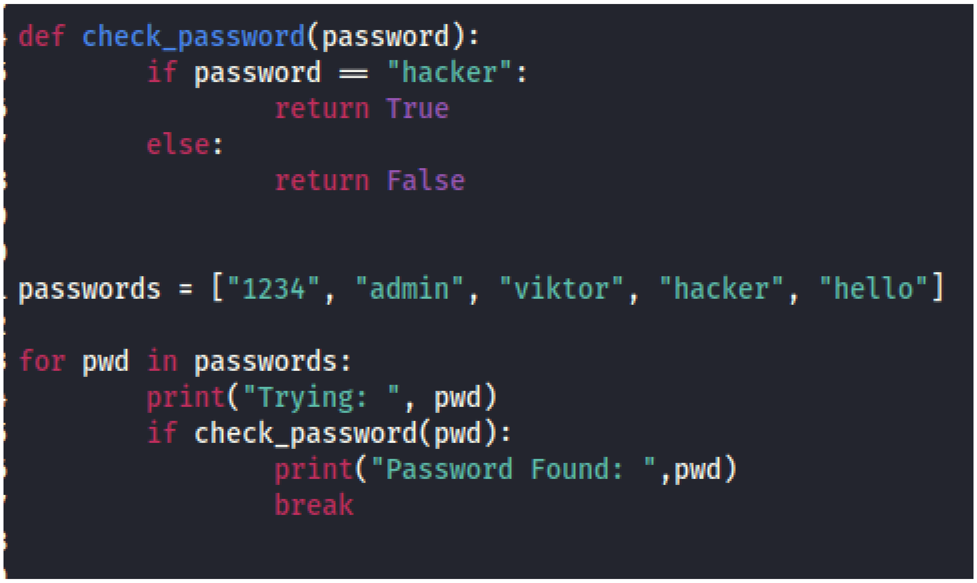
```python
for i in range(5):
        print("Trying login...")
```

This is an example of a very simple loop.

`range(5)` -> 0,1,2,3,4 -> the function runs 5 times starting from 0

Example #2

Let’s create a simple brute-forcing tool using functions and loops!

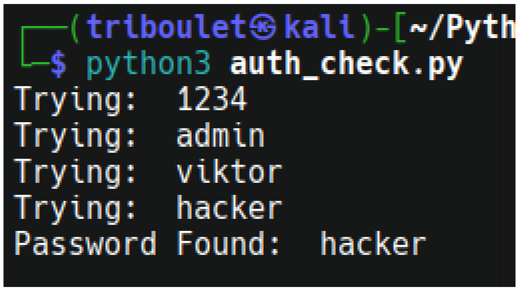


```
def check_password(password):
        if password == "hacker":
                return True
        else:
                return False


passwords = ["1234", "admin", "viktor", "hacker", "hello"]

for pwd in passwords:
        print("Trying: ", pwd)
        if check_password(pwd):
                print("Password Found: ",pwd)
                break
```

`pwd` -> **This is a variable that stands for passwords.** It is not strict. You can do whatever you want

As you can see in this code we first defined a function where if the password is **hacker** it must return **True**. Then we set the **passwords** list and after that we created a ***loop*** where we tell python to go through all of the passwords following the rules set in the function and if it found a match to output **Password Found**.

Output:



```
┌──(triboulet㉿kali)-[~/Pyth
└─$ python3 auth_check.py
Trying:  1234
Trying:  admin
Trying:  viktor
Trying:  hacker
Password Found:  hacker
```

Important Keywords:

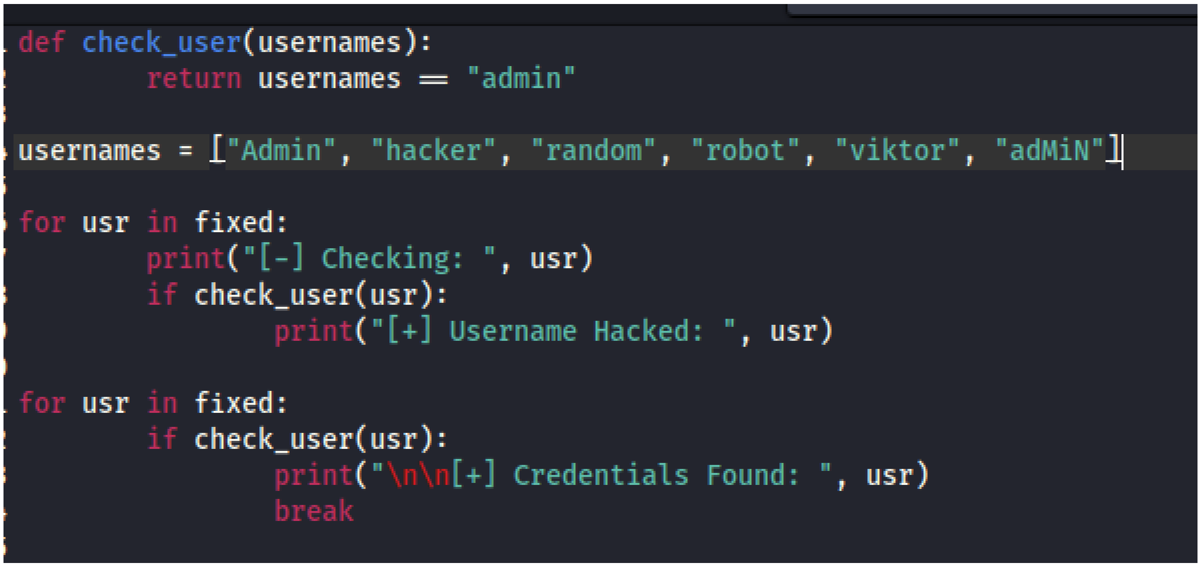
Keyword	Meaning
def	define function
return	send result back
for	loop
break	stop loop
True / False	boolean logic

EXERCISE

ChatGPT gave me a task to create auth_bruteforce.py which has a function called check_user. There needs to be a loop and also the correct username is **admin**. The function must also be allowing **aDmiN, AdMiN, ADMin etc.** Let’s begin!

SOLUTION

First I started with the basics. I defined a function and then made it return the username **admin**:

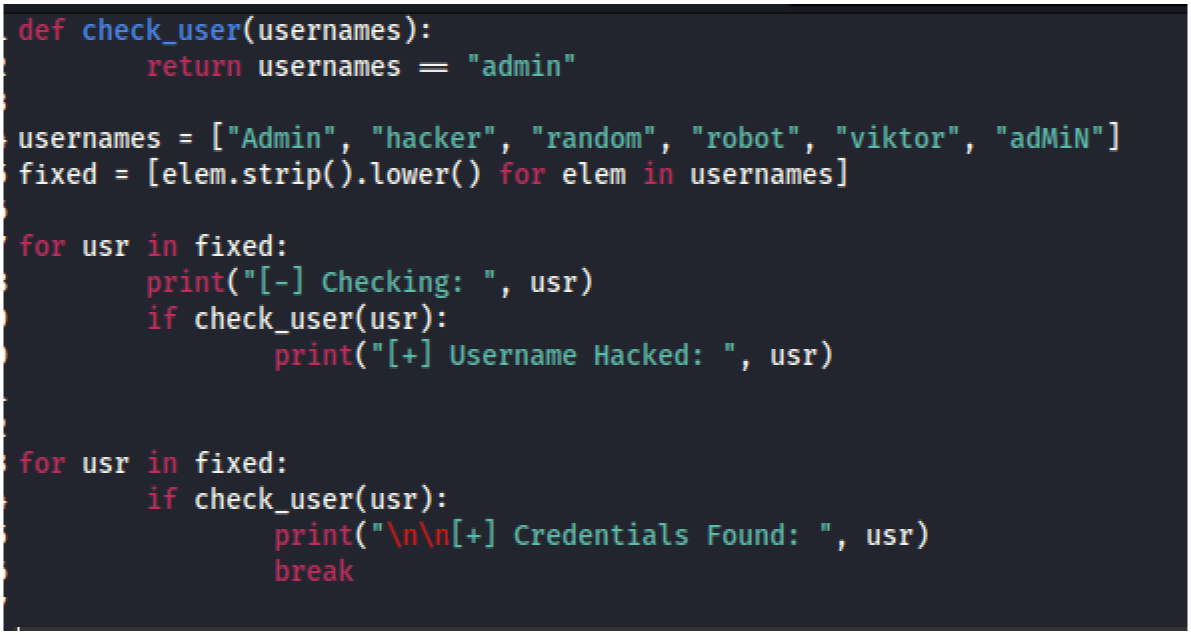


```
def check_user(usernames):
        return usernames == "admin"

usernames = ["Admin", "hacker", "random", "robot", "viktor", "adMiN"]

for usr in fixed:
        print("[-] Checking: ", usr)
        if check_user(usr):
                print("[+] Username Hacked: ", usr)

for usr in fixed:
        if check_user(usr):
                print("\n\n[+] Credentials Found: ", usr)
                break
```

Then I created the list and put admin two times to examine what the output would look like.

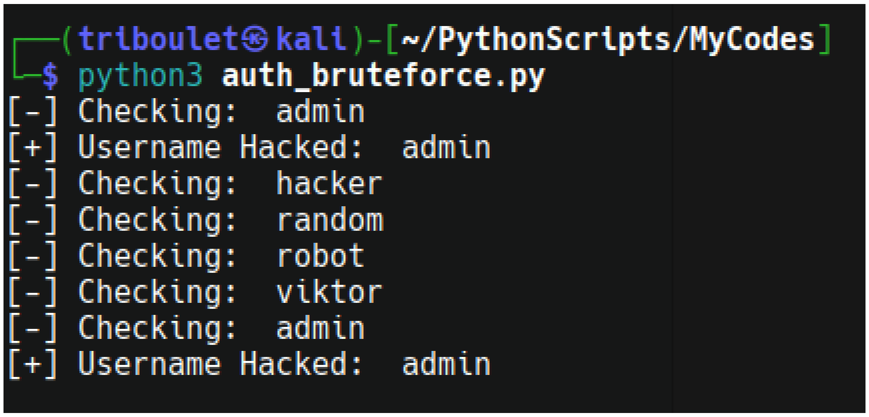
After that I started making the **loop**. It was also pretty simple.

The part with allowing other variations of the username **admin** was a bit tricky. Because I got an error. After a research I found that if I use **elem** for each element in the list it will work. So here is how the full code looks like:



```python
def check_user(usernames):
        return usernames == "admin"

usernames = ["Admin", "hacker", "random", "robot", "viktor", "adMiN"]
fixed = [elem.strip().lower() for elem in usernames]

for usr in fixed:
        print("[-] Checking: ", usr)
        if check_user(usr):
                print("[+] Username Hacked: ", usr)


for usr in fixed:
        if check_user(usr):
                print("\n\n[+] Credentials Found: ", usr)
                break
```

Output:

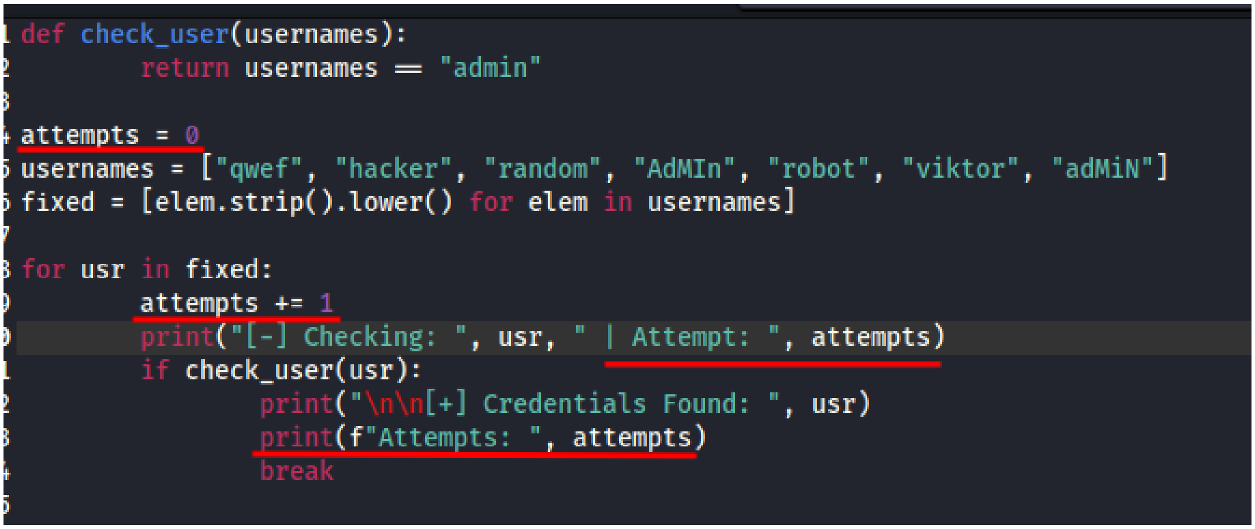


```
┌──(triboulet㉿kali)-[~/PythonScripts/MyCodes]
└─$ python3 auth_bruteforce.py
[-] Checking:  admin
[+] Username Hacked:  admin
[-] Checking:  hacker
[-] Checking:  random
[-] Checking:  robot
[-] Checking:  viktor
[-] Checking:  admin
[+] Username Hacked:  admin
```

We got two matches. And the task is done!

BONUS:

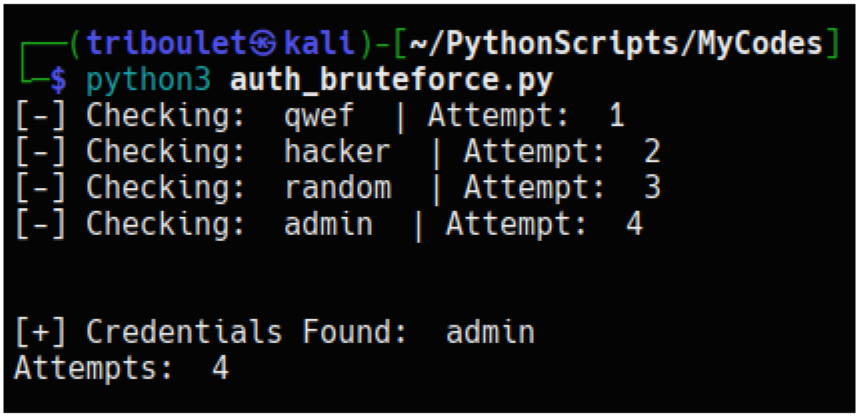
If we want to make an attempt counter this is how we would’ve proceeded!



```python
def check_user(usernames):
	return usernames == "admin"

attempts = 0
usernames = ["qwef", "hacker", "random", "AdMIn", "robot", "viktor", "adMiN"]
fixed = [elem.strip().lower() for elem in usernames]

for usr in fixed:
	attempts += 1
	print("[-] Checking: ", usr, " | Attempt: ", attempts)
	if check_user(usr):
		print("\n\n[+] Credentials Found: ", usr)
		print(f"Attempts: ", attempts)
		break

```

We first introduce the attempts variable. After that we add it in the loop and say that for every output we add 1 starting from 0. And then we add it in the output.

Here is how it should look:



```
┌──(triboulet㉿kali)-[~/PythonScripts/MyCodes]
└─$ python3 auth_bruteforce.py
[-] Checking:  qwef  | Attempt:  1
[-] Checking:  hacker  | Attempt:  2
[-] Checking:  random  | Attempt:  3
[-] Checking:  admin  | Attempt:  4


[+] Credentials Found:  admin
Attempts:  4
```

3. Files and wordlists

- Real pentesting tools **do not hardcode credentials**. They read massive wordlists from files, process them line by line, and test each entry.

Pattern:

```
open file
read one line
normalize input
send attempt
analyze response
repeat
```

Basic File reading:

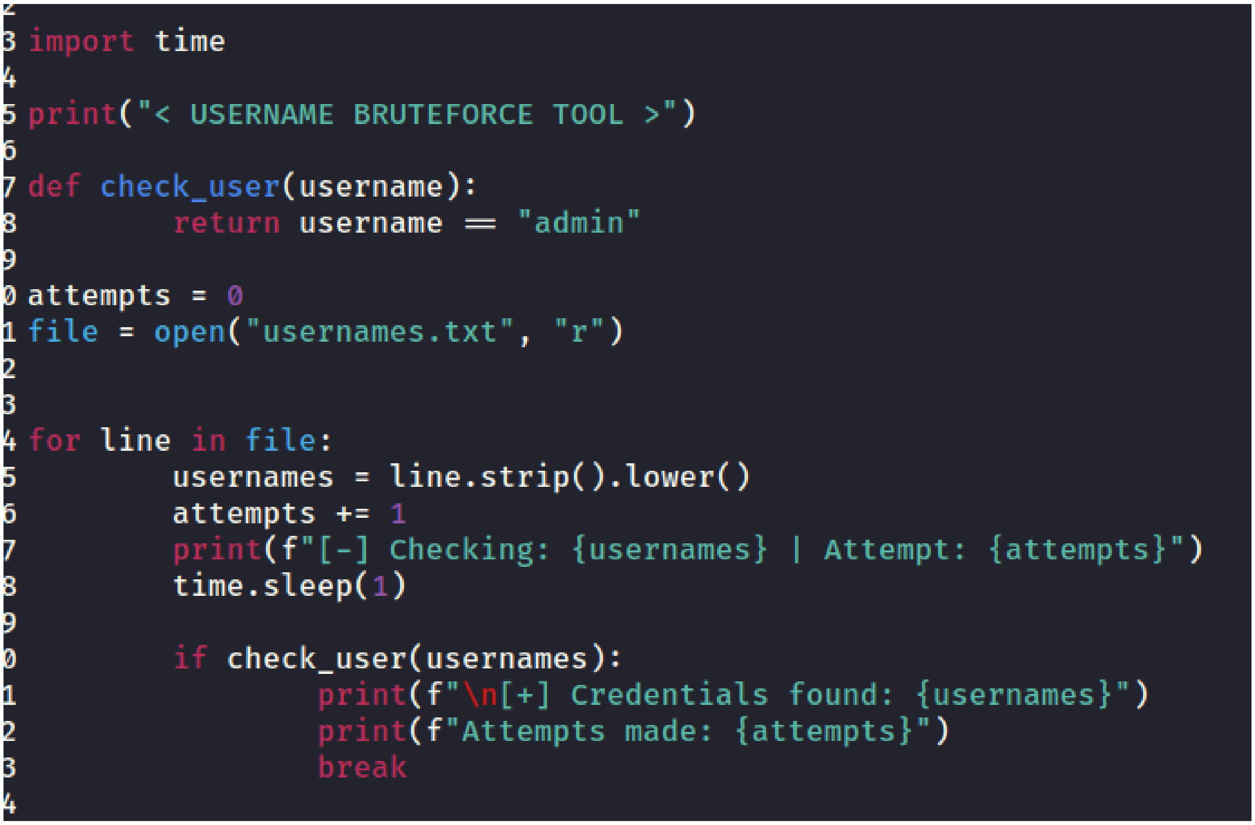
```
file = open("usernames.txt", "r")
```

Line by Line reading:

```
for line in file:
    username = line.strip().lower()
    print(username)
```

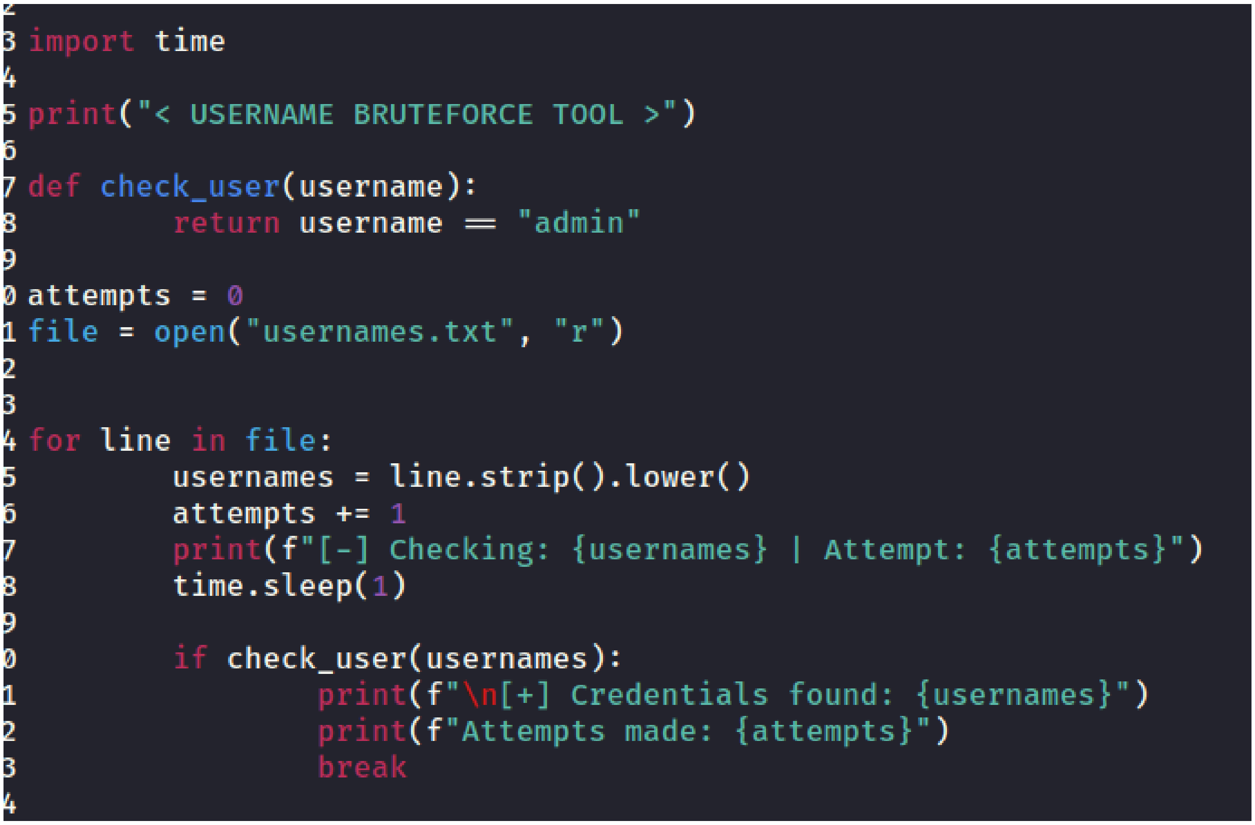
Example #1:

If we take the code from the previous lesson we can upgrade it by using a wordlist for usernames we have created.



```python
import time

print("< USERNAME BRUTEFORCE TOOL >")

def check_user(username):
        return username == "admin"

attempts = 0
file = open("usernames.txt", "r")


for line in file:
        usernames = line.strip().lower()
        attempts += 1
        print(f"[-] Checking: {usernames} | Attempt: {attempts}")
        time.sleep(1)

        if check_user(usernames):
                print(f"\n[+] Credentials found: {usernames}")
                print(f"Attempts made: {attempts}")
                break

```

I added **time.sleep()** to make it more realistic plus when an actual tool is used it is better to have some delay to avoid suspicion.



```
import time

print("< USERNAME BRUTEFORCE TOOL >")

def check_user(username):
        return username == "admin"

attempts = 0
file = open("usernames.txt", "r")


for line in file:
        usernames = line.strip().lower()
        attempts += 1
        print(f"[-] Checking: {usernames} | Attempt: {attempts}")
        time.sleep(1)

        if check_user(usernames):
                print(f"\n[+] Credentials found: {usernames}")
                print(f"Attempts made: {attempts}")
                break

```

Code Analysis:

`def …` -> we create a function

`Attempts = 0` -> we set the attempts counter

`file = …` -> **file** is a variable. And we set that this variable will open the usernames.txt and read it (“r”).

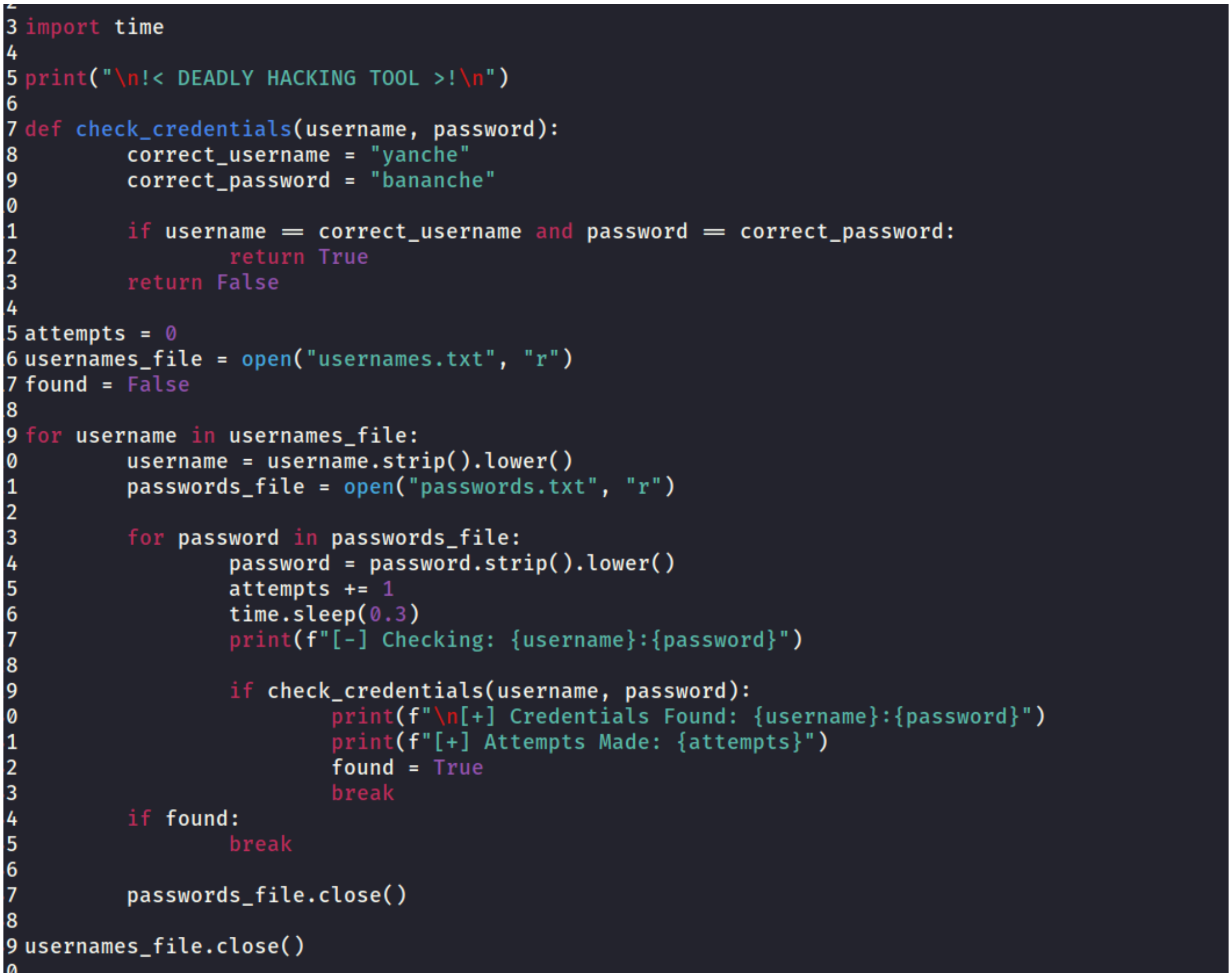
`for line in file:` -> **line** is a variable in the **file** variable. So for each line in the file usernames.txt we begin a loop.

`usernames = line.strip().lower()` -> we create another variable which is for every line in the file to remove the space and to lower case it.

Exercise

Create a tool that goes through both usernames and passwords and checks if there is a match.

Solution

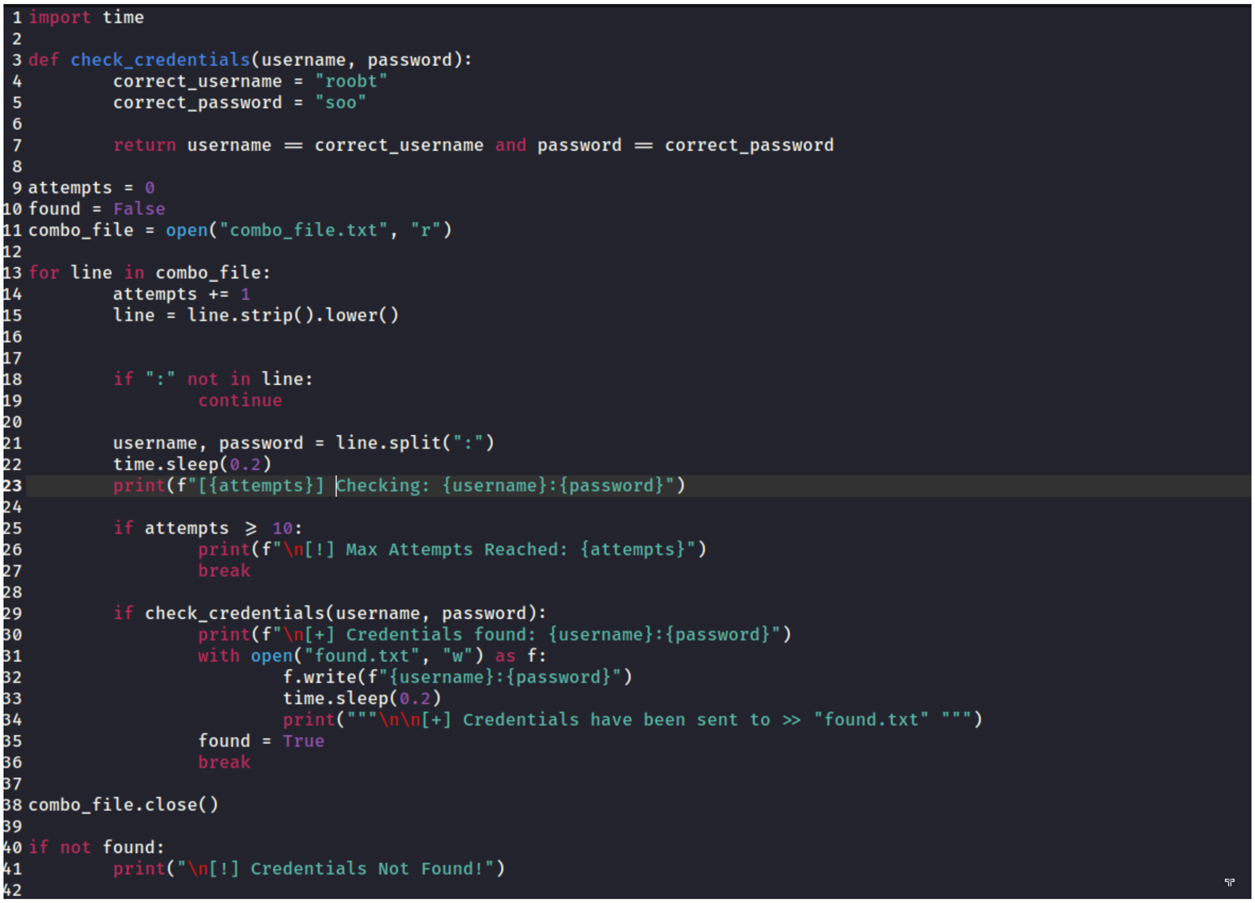


```
import time

print("\n!< DEADLY HACKING TOOL >!\n")

def check_credentials(username, password):
        correct_username = "yanche"
        correct_password = "bananche"

        if username == correct_username and password == correct_password:
                return True
        return False

attempts = 0
usernames_file = open("usernames.txt", "r")
found = False

for username in usernames_file:
        username = username.strip().lower()
        passwords_file = open("passwords.txt", "r")

        for password in passwords_file:
                password = password.strip().lower()
                attempts += 1
                time.sleep(0.3)
                print(f"[-] Checking: {username}:{password}")

                if check_credentials(username, password):
                        print(f"\n[+] Credentials Found: {username}:{password}")
                        print(f"[+] Attempts Made: {attempts}")
                        found = True
                        break
        if found:
                break

        passwords_file.close()

usernames_file.close()
```

We create the function where we set the correct_username;correct_password and if both are met it returns True.

Found -> this is the flag that will be used to break out of the loop after credentials are found.

4. Combo Wordlists and Real Tool Structure



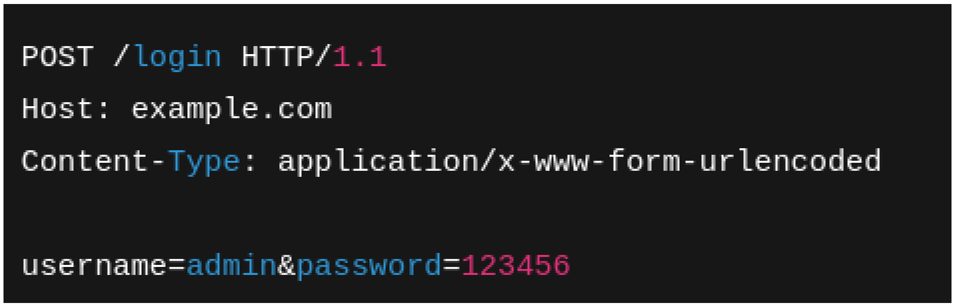
```python
import time

def check_credentials(username, password):
        correct_username = "roobt"
        correct_password = "soo"

        return username == correct_username and password == correct_password

attempts = 0
found = False
combo_file = open("combo_file.txt", "r")

for line in combo_file:
        attempts += 1
        line = line.strip().lower()


        if ":" not in line:
                continue

        username, password = line.split(":")
        time.sleep(0.2)
        print(f"[{attempts}] Checking: {username}:{password}")

        if attempts >= 10:
                print(f"\n[!] Max Attempts Reached: {attempts}")
                break

        if check_credentials(username, password):
                print(f"\n[+] Credentials found: {username}:{password}")
                with open("found.txt", "w") as f:
                        f.write(f"{username}:{password}")
                        time.sleep(0.2)
                        print("""\n\n[+] Credentials have been sent to >> "found.txt" """)
                found = True
                break

combo_file.close()

if not found:
        print("\n[!] Credentials Not Found!")

```

As you can see we have added an attempt limit to simulate real detection. I also printed out the output to a file called found.txt. I think it would be better to **append “a”** instead of **“w”** just overwrite it.

5. Web Login Attacks (HTTP & REQUESTS)

When you login into a site your browser sends an **HTTP request**.

This is how it looks from behind the scenes.



```
POST /login HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded

username=admin&password=123456
```

5.1. **HTTP Request Structure**

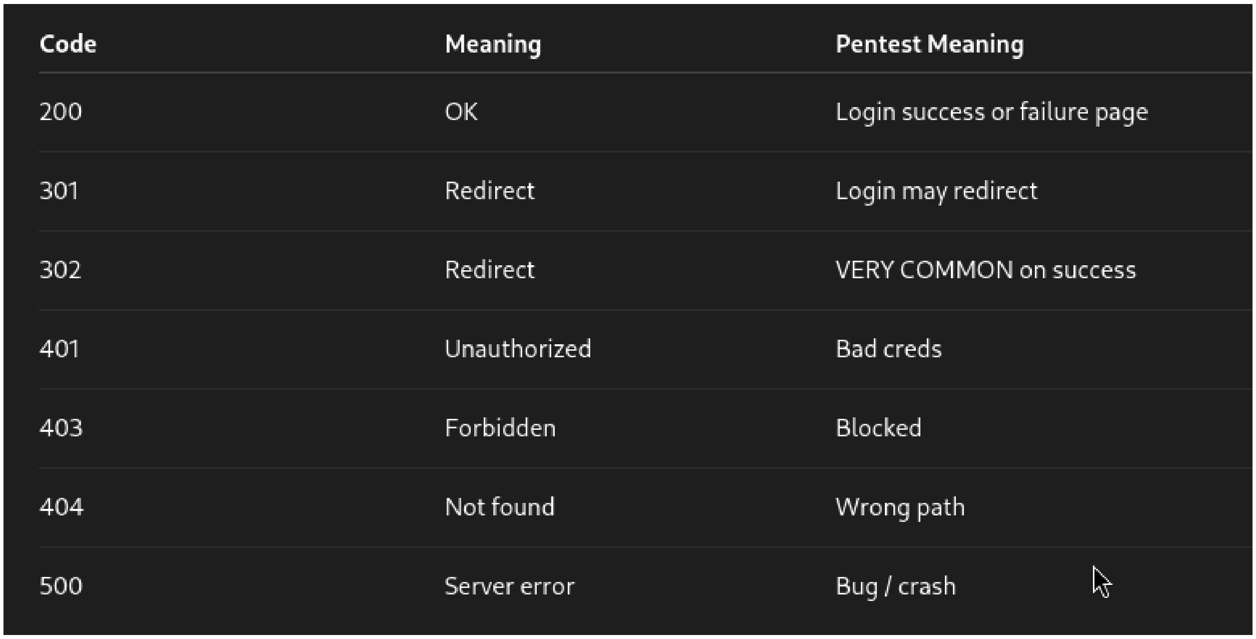
1. **Method** → GET, POST
2. **Path** → /login
3. **Headers** → metadata
4. **Body** → credentials (for POST)

GET vs POST (very important)

Method	**Where data is**	**Used for**
GET	URL	Search, links
POST	Request body	Logins, forms

Login forms almost always use POST

HTTP RESPONSE STATUS CODES



Code	Meaning	Pentest Meaning
200	OK	Login success or failure page
301	Redirect	Login may redirect
302	Redirect	VERY COMMON on success
401	Unauthorized	Bad creds
403	Forbidden	Blocked
404	Not found	Wrong path
500	Server error	Bug / crash

Many logins work like this:

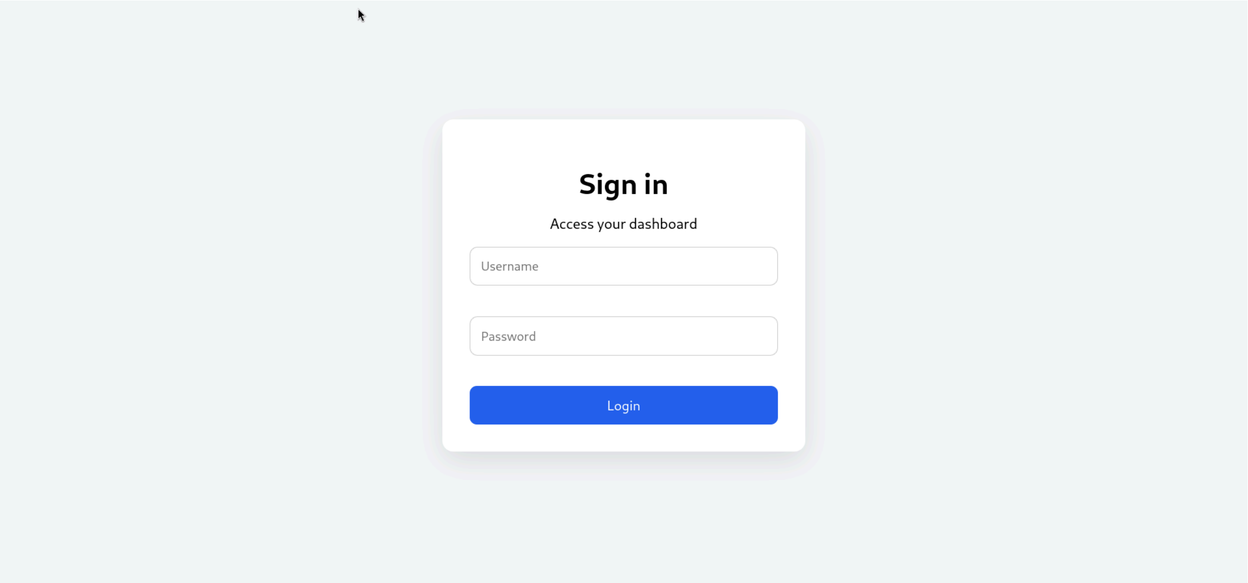
- ❌ Wrong password → 200 OK + "Invalid credentials"
- ✅ Correct password → 302 Redirect to /dashboard

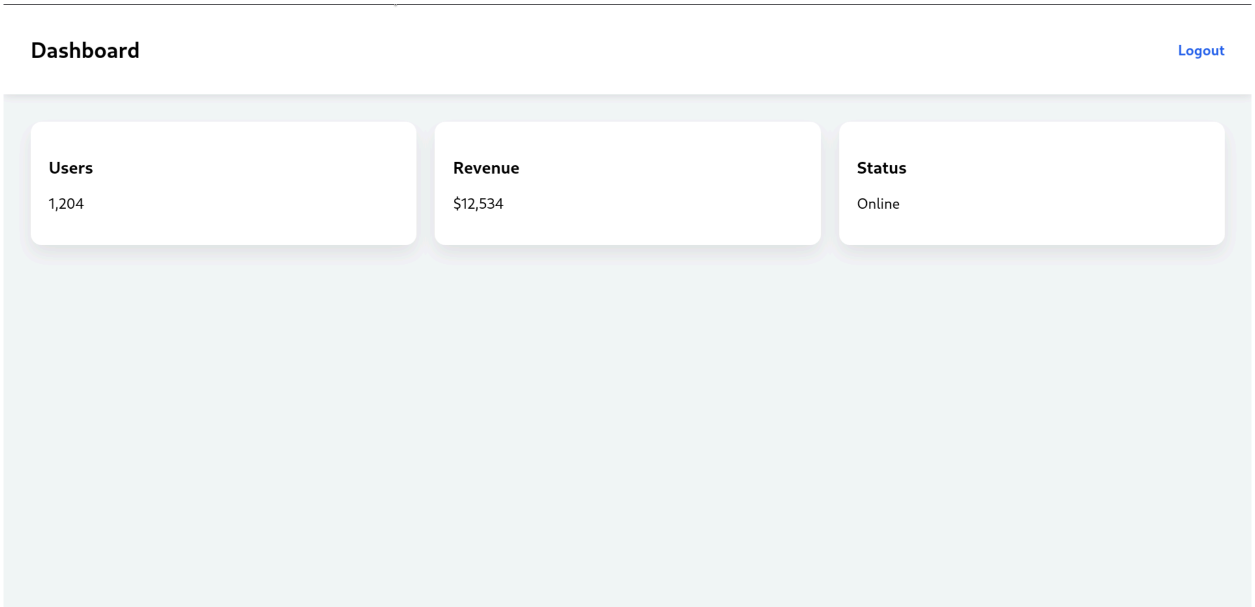
6. Server Update

I wasn’t able to document my progress from the last two days so I am going to do a summary on what I have improved in terms of code etc.

First I created a server.py which runs on http://127.0.0.1:5000/login.

It is a login page where I test my web_bruteforce.py tool. At first the server was very very basic but right now I am developing it and also designing it to look cool. It has a home page. And a login page. It simulates a real website.:





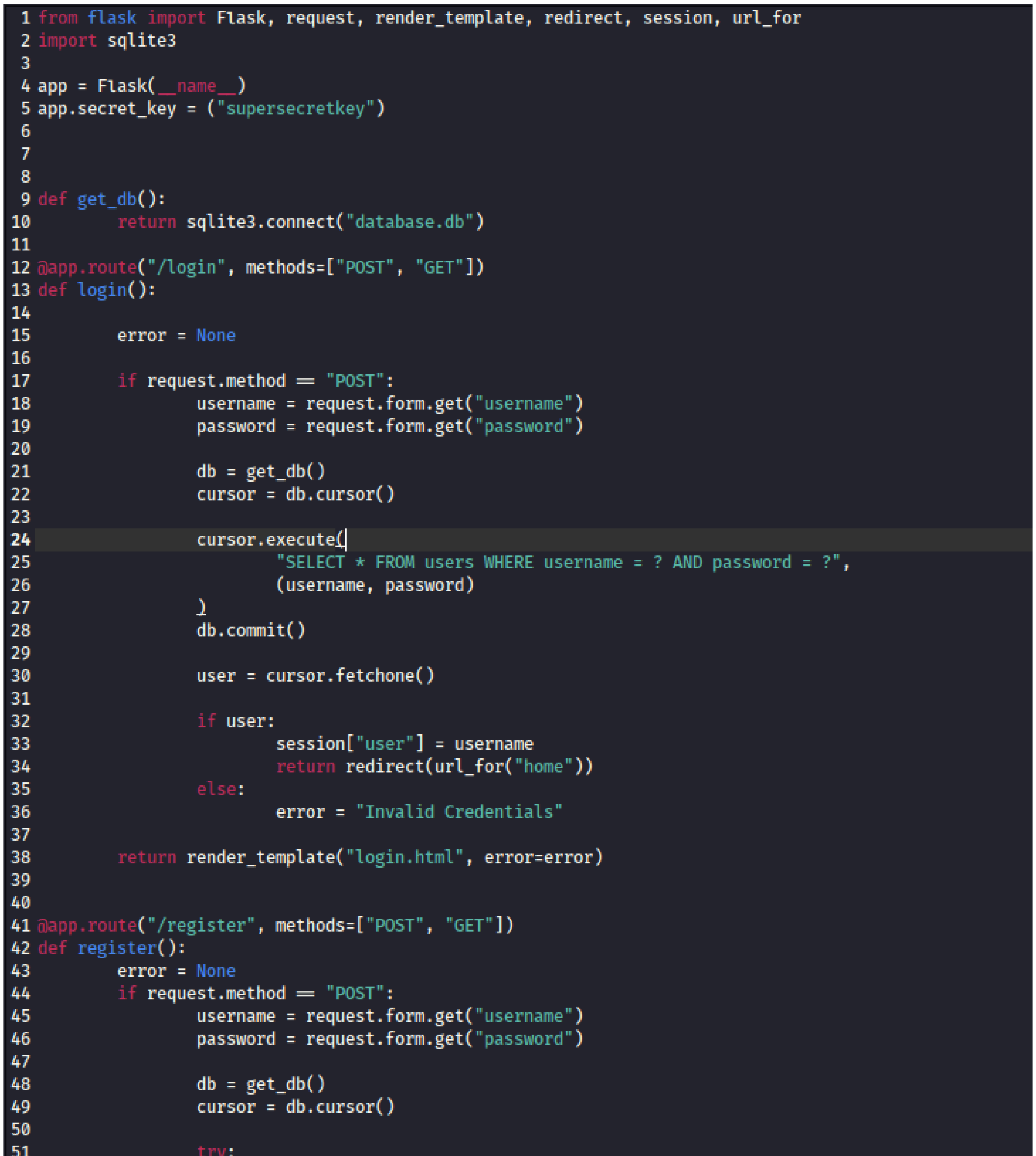
It is simple but I learned A LOT. A lot about requests; about how to code a server; how to html and css; how login auth works etc. The problem is that the credentials are still hard-coded in the code so I am going to create a database that checks if it matches the correct username and password provided by the user.

This is my structure:

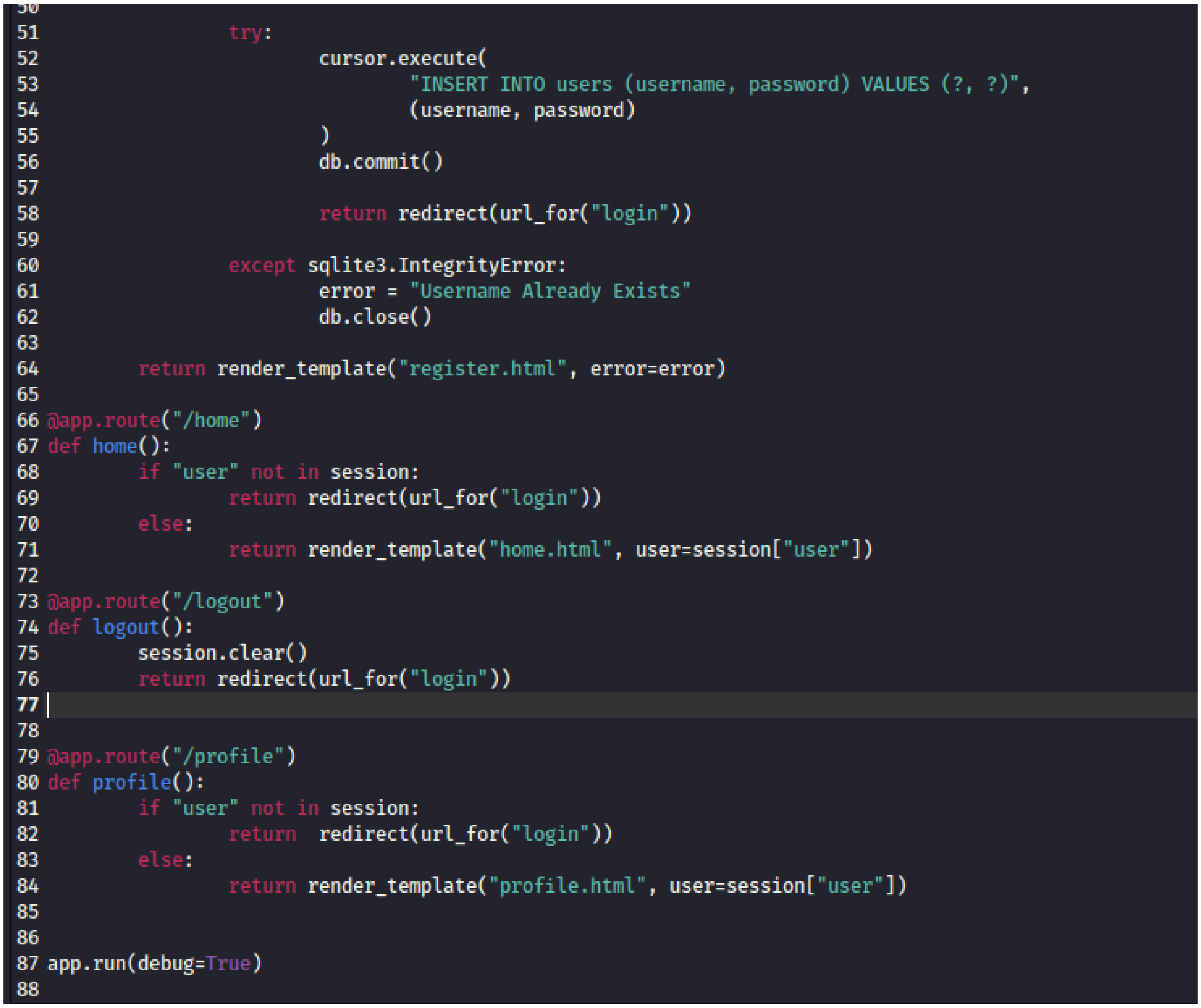
/login_lab

- server.py
- templates/ -> home.html; login.html; register.html; profile.html
- static/ -> login.css; register.css; profile.css; home.css

Server.py code:

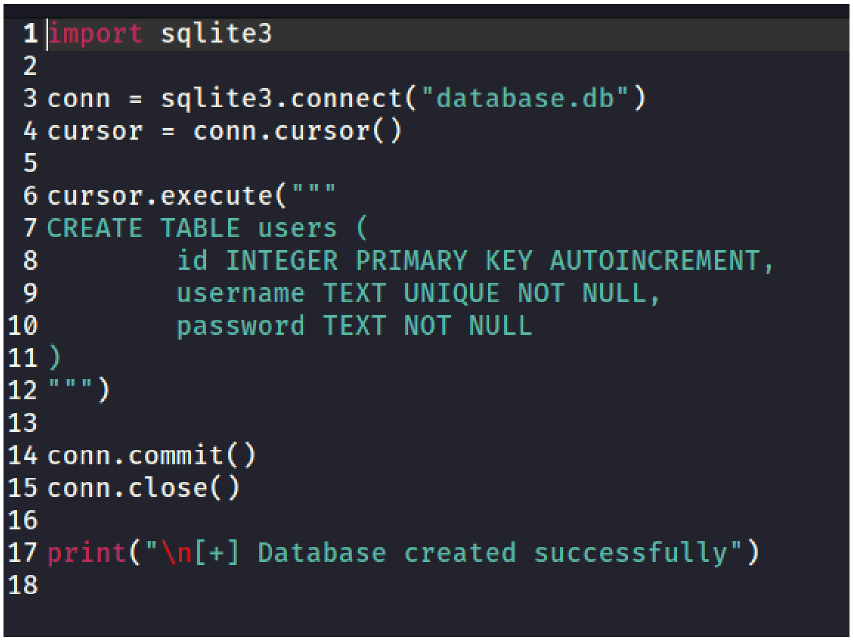


```
from flask import Flask, request, render_template, redirect, session, url_for
import sqlite3

app = Flask(__name__)
app.secret_key = ("supersecretkey")



def get_db():
        return sqlite3.connect("database.db")

@app.route("/login", methods=["POST", "GET"])
def login():

        error = None

        if request.method == "POST":
                username = request.form.get("username")
                password = request.form.get("password")

                db = get_db()
                cursor = db.cursor()

                cursor.execute(
                        "SELECT * FROM users WHERE username = ? AND password = ?",
                        (username, password)
                )
                db.commit()

                user = cursor.fetchone()

                if user:
                        session["user"] = username
                        return redirect(url_for("home"))
                else:
                        error = "Invalid Credentials"

        return render_template("login.html", error=error)


@app.route("/register", methods=["POST", "GET"])
def register():
        error = None
        if request.method == "POST":
                username = request.form.get("username")
                password = request.form.get("password")

                db = get_db()
                cursor = db.cursor()

                try:
```



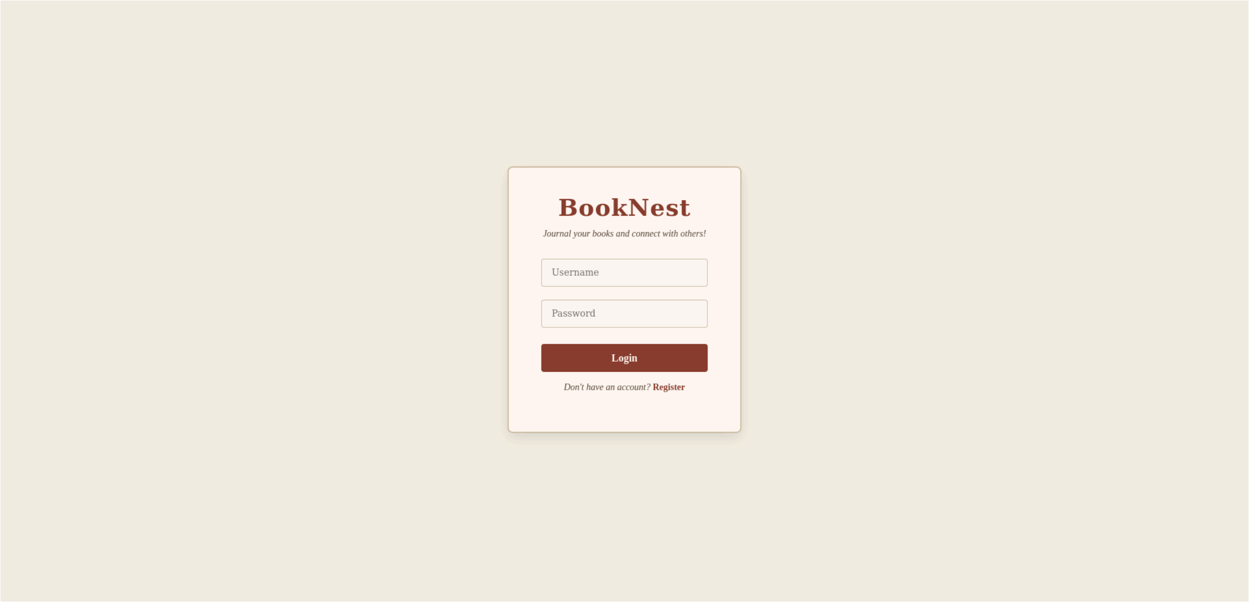
```
        try:
            cursor.execute(
                "INSERT INTO users (username, password) VALUES (?, ?)",
                (username, password)
            )
            db.commit()

            return redirect(url_for("login"))

        except sqlite3.IntegrityError:
            error = "Username Already Exists"
            db.close()

    return render_template("register.html", error=error)

@app.route("/home")
def home():
    if "user" not in session:
        return redirect(url_for("login"))
    else:
        return render_template("home.html", user=session["user"])

@app.route("/logout")
def logout():
    session.clear()
    return redirect(url_for("login"))


@app.route("/profile")
def profile():
    if "user" not in session:
        return  redirect(url_for("login"))
    else:
        return render_template("profile.html", user=session["user"])


app.run(debug=True)

```

Now we have a database made with a simple database_generator.py:



```
import sqlite3

conn = sqlite3.connect("database.db")
cursor = conn.cursor()

cursor.execute("""
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL
)
""")

conn.commit()
conn.close()

print("\n[+] Database created successfully")

```

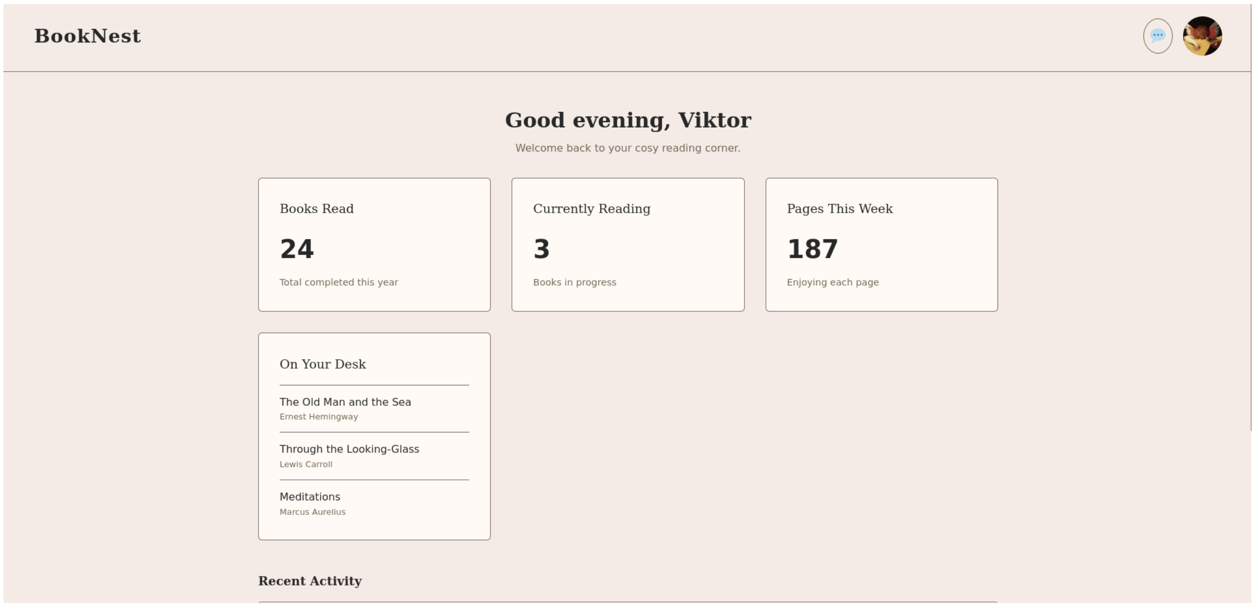
I am currently working on a different project. A project about a book tracker website. I am making it look more believable as well as i am practicing html and css (not the most important things but i want to not feel so lost). Now the server looks like this.



BookNest

Journal your books and connect with others!

Don't have an account? **Register**



BookNest

Good evening, Viktor

Welcome back to your cosy reading corner.

Books Read

24

Total completed this year

Currently Reading

3

Books in progress

Pages This Week

187

Enjoying each page

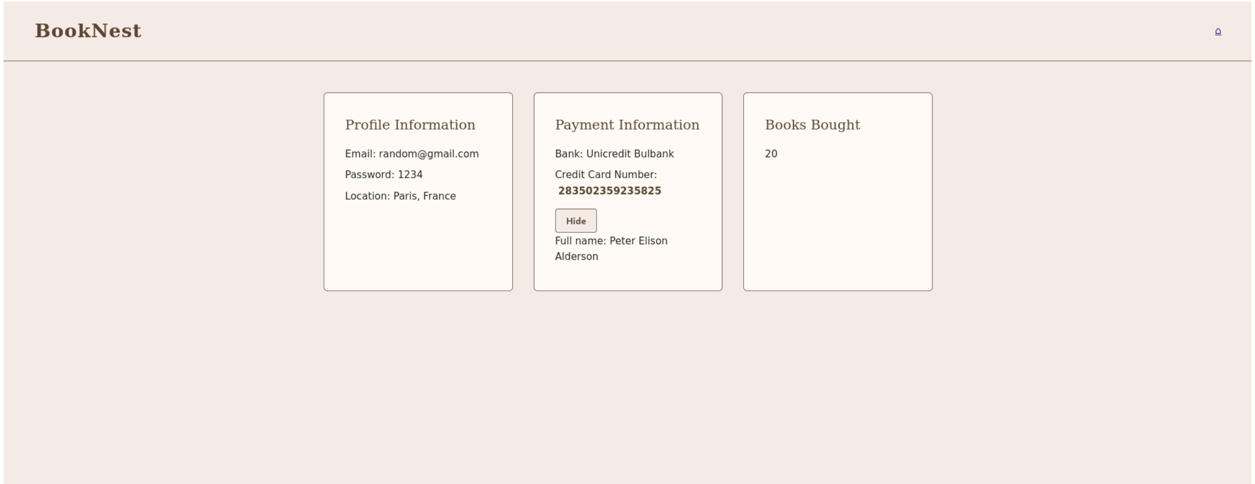
On Your Desk

The Old Man and the Sea
Ernest Hemingway

Through the Looking-Glass
Lewis Carroll

Meditations
Marcus Aurelius

Recent Activity



BookNest

Profile Information

Email: random@gmail.com

Password: 1234

Location: Paris, France

Payment Information

Bank: Unicredit Bulbank

Credit Card Number:
283502359235825

Full name: Peter Elison Alderson

Books Bought

20

The htmls are made by me but the css was made with the help of ai.