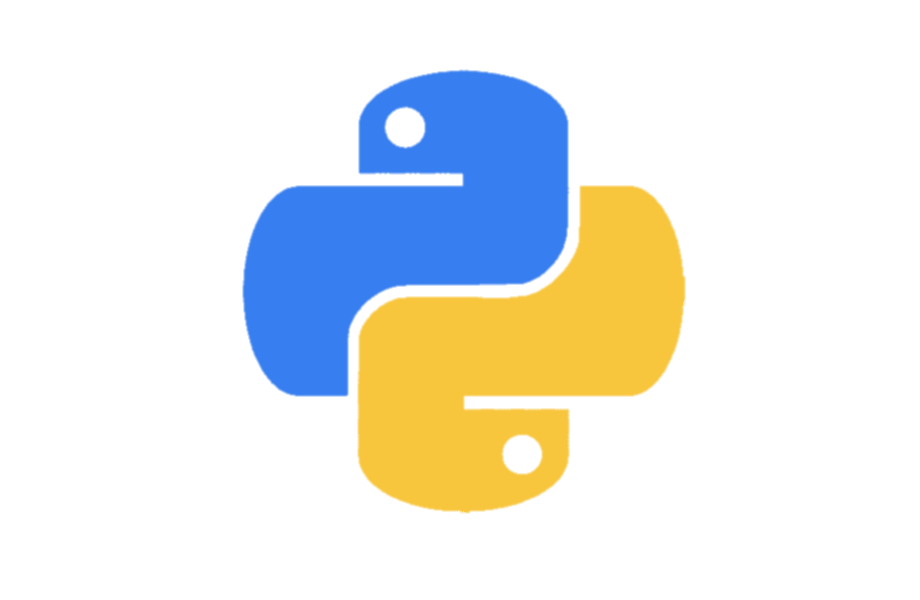
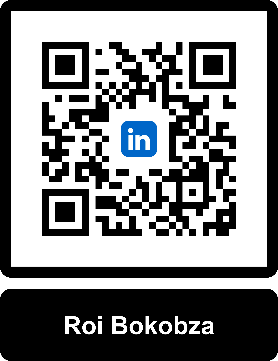
//

Python

Essential

  
  
  
  
  
  
  
  
  
  
  
  
  
  
[roibokobza017@gmail.com](mailto:roibokobza017@gmail.com)

By Roi Bokobza



**Table of Contents**Introduction

* Python Introduction…………………………………………………………………………………. 3
* Project Introduction……………………………………………………………………………..….. 6

Fundamentals

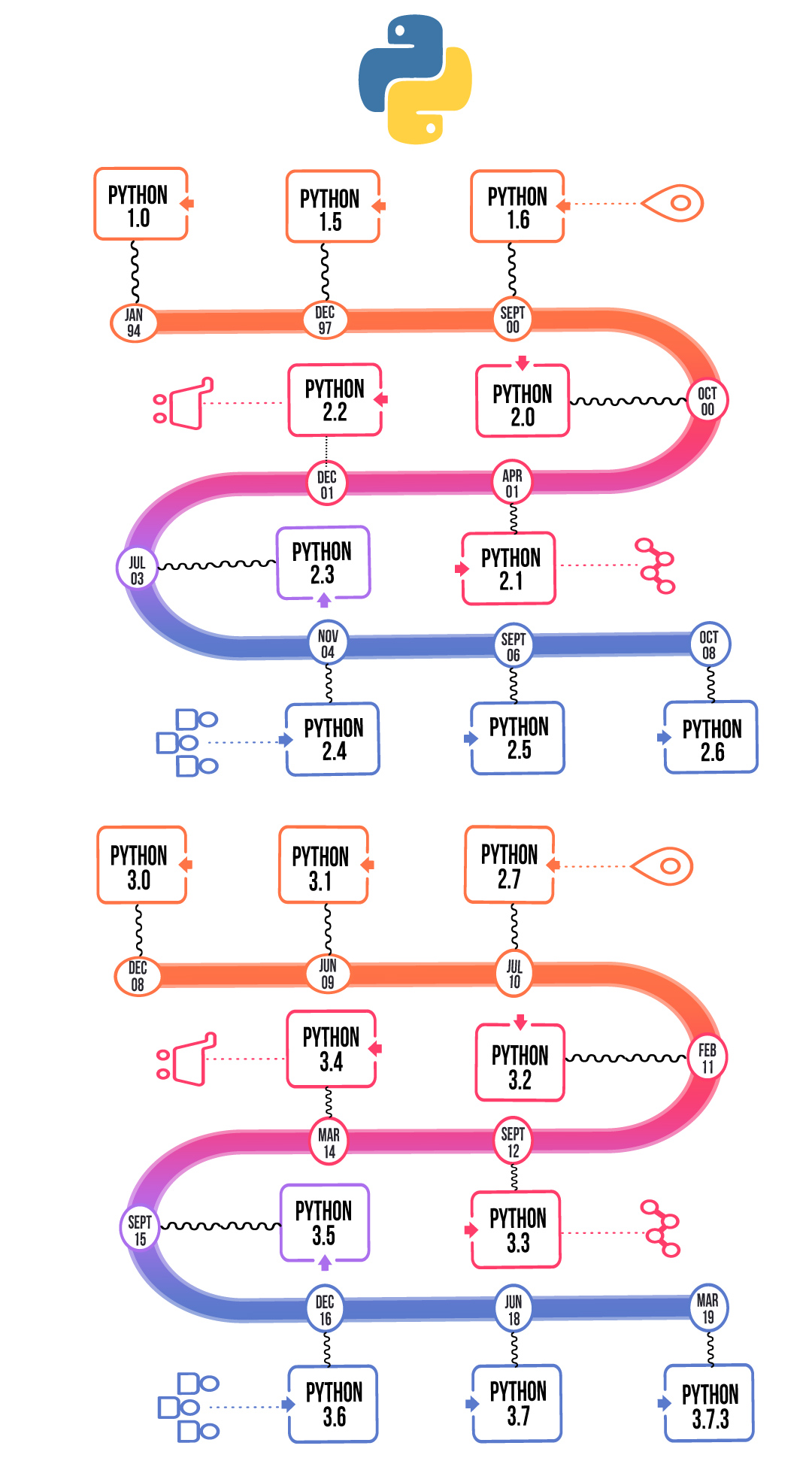
* Print and Operators…………………………………………………………………………...……. 8
* Variables and Data types………………………………………………………………………… 11
* User Input……………………………………………………………………………………………….. 15
* Pro Tips and Shortcuts……………………………………………………………………………..17

Conditions and Control Flow

* Boolean Values……………………………………………………………………………………….. 19
* Conditional statements…………………………………………………………………………… 20
* Logic Operators………………………………………………………………………………………. 25
* Loops………………………………………………………………………………………………………. 27

Advanced

* Libraries and functions..………………………………………………………………………..… 34
* Lists……………………………………………………………...…………………………………………. 38
* Guess the random number……………………………………………………………………... 42
* Bank Accounts Project…………………………………………………………………………….. 45
* Using SQL Injections to hack the accounts……………………………………………… 56



**Python Introduction**

**A Quick History of Python**

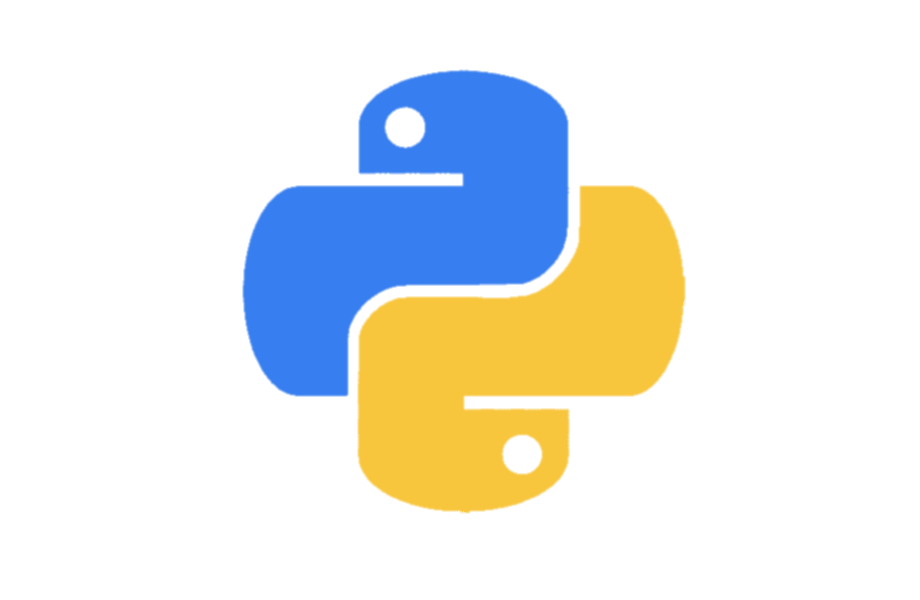
in 1980s a Dutch programmer **Guido van Rossum**, wanted to create a language that was easy to read, write, and actually made sense.

He officially released Python in **1991**. Unlike older programming languages that were a pain to use, Python was designed to be **beginner-friendly**.

Over time, it earned its success due to its **Flexibility and Power**.

Today, it’s one of the most popular languages in the world, used by companies like Google, Facebook, and Netflix.

**Where Python is Used Today**

* **Web Development** – Websites and web apps like Django, Flask.
* **Data Science & AI** – Machine learning, big data, AI models like TensorFlow, Pandas.
* **Automation & Scripting** – Automating long tasks, managing servers, handling files.
* **Game Development** – Used in some game engines and scripts (like Pygame).
* **Embedded Systems** – Even tiny computers (like Raspberry Pi) run Python.

**Python in Cybersecurity**

Python is a 'must-have' tool in Cybersecurity due to how fast and powerful it is for writing scripts.

Python makes building powerful tools easy and beginner-friendly, Tools for both attacking and defending systems. Hackers, ethical hackers, and security researchers rely on Python, making it an essential skill for anyone aspiring to become a cybersecurity expert.

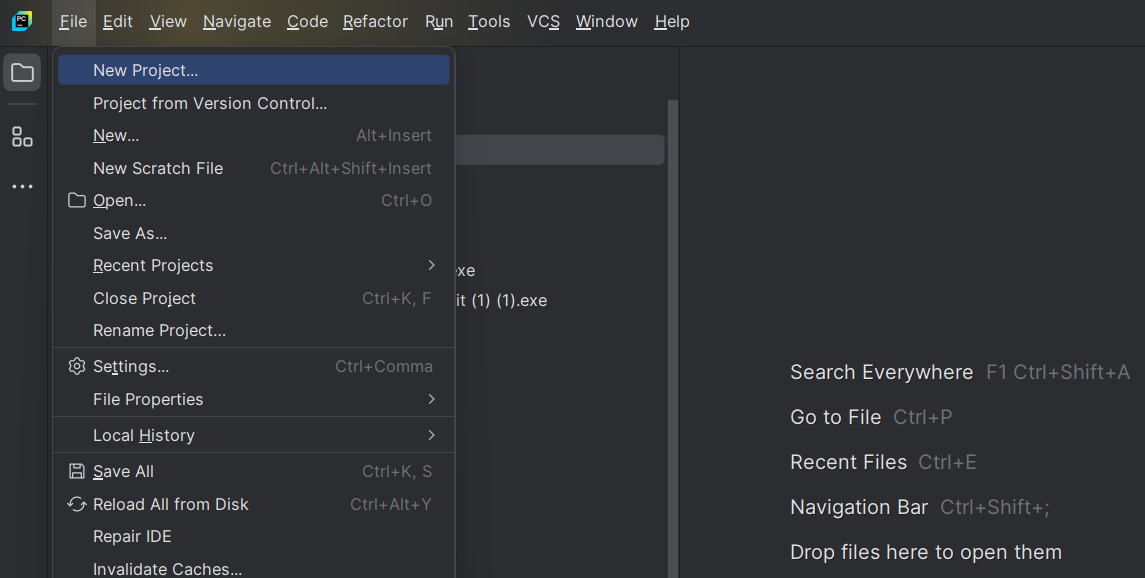
**Common use cases in CyberSecurity:**

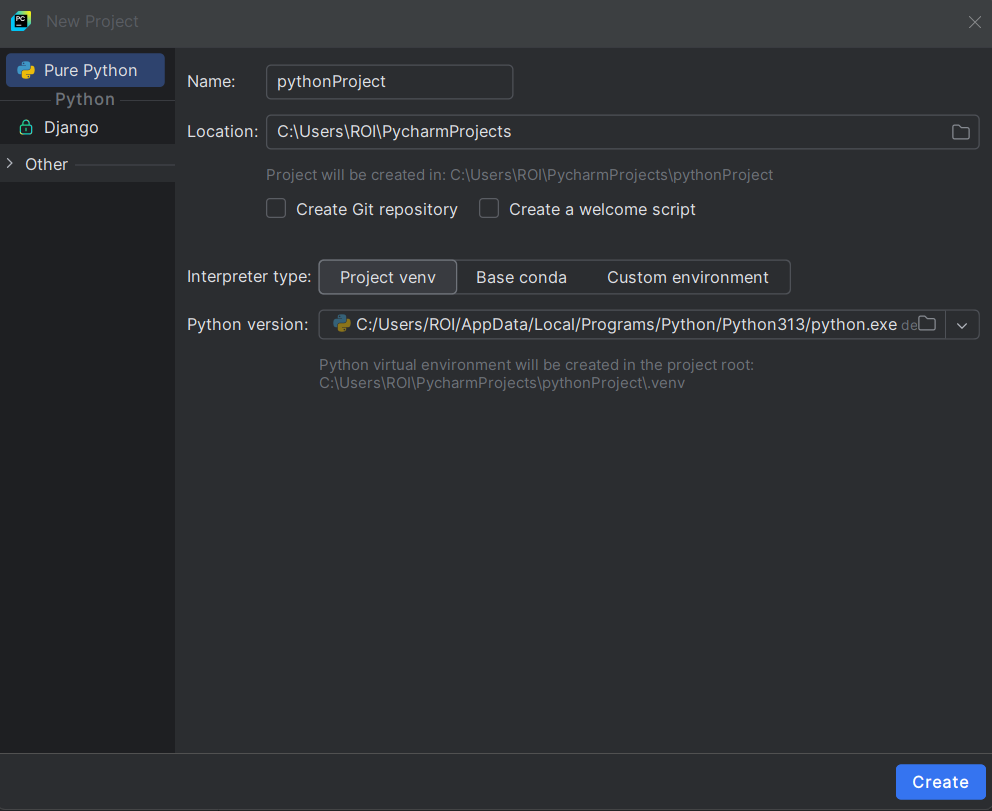
* **Penetration Testing** – Writing hacking tools, scanning networks, exploiting vulnerabilities (Scapy, Metasploit scripts).
* **Malware Analysis** – Reverse-engineering malicious code and analyzing threats.
* **Network Security** – Sniffing packets, monitoring traffic, automating security checks.
* **Automating Attacks & Defenses** – Brute force attacks, password cracking (with libraries like Hashlib).
* **Web Scraping & OSINT** – Gathering public data for intelligence (BeautifulSoup, Scrapy).

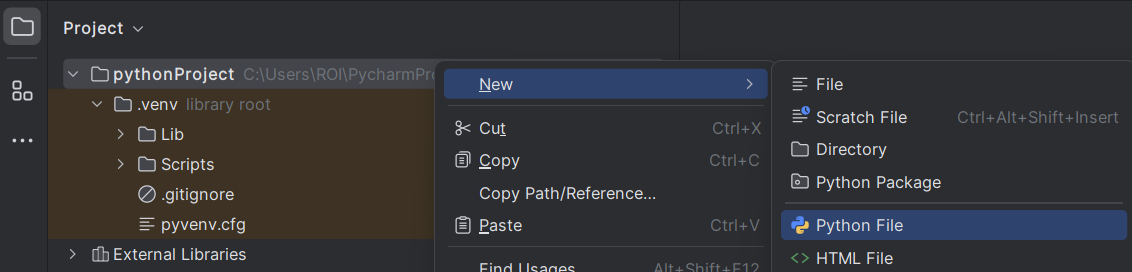
**Project Introduction**

For this Project, I will be using PyCharm, this is how you create a new project in pycharm:

Click File > New Project



After that a window will pop up, and you will give your project a name and the path it will be stored in. 



When the project is created right click on your project > new > python file.

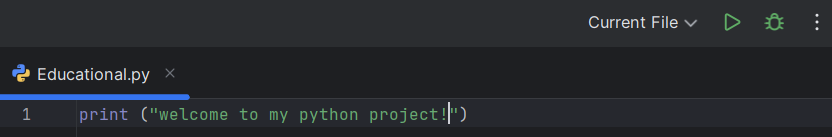
Hyper links (clickables) will appear in **This** color.

Examples of commands will look like this: *print("mycommand")*

**Print and Operators**

**1. Print:**

Print is an essential command for Python, it is used for everything, the command prints output to the screen.



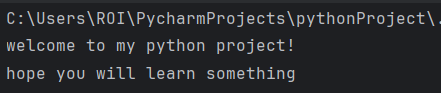
I asked Python to print a [**string**](#datatypes) input, the text within the brackets is what I want Python to print, and the quotation marks signify that I am using a string (plain text).

Click on this play button to run the command and you should get the output in the console.



To print the output on a new line, use **\n**

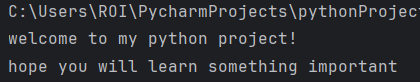




adding another print command within the same line will also print the output on a new line.

Notice that you type a comma if you want to type another command within the same line.

**2. end:**end= sets what comes after a print() statement.

****

**3. sep:**

sets the separator between arguments here is an example:

****

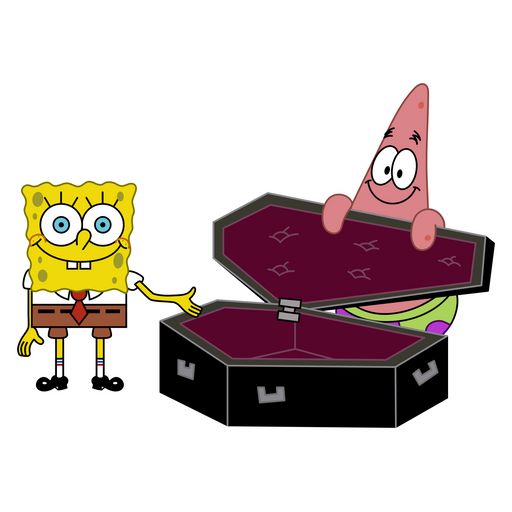
**4. Basic commands task:**Print: *'Python\*\*\*is\*\*\*fun…with miko'*

For this task, I will be using both 'sep' and 'end'.





Each word is in its own quotation marks, making it a separate argument. The sep parameter defines the **separator** between the arguments, and the value you provide is used to separate them.I finished my print command with end=, when I use end=, it changes what comes at the end of the print instead of a new line.

  
  
**What is a variable?**

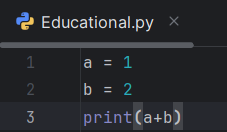
**Variables**

A Python variable can be considered a small storage unit meant to store data under a name of your choice.

MyVariable=

**Example:**

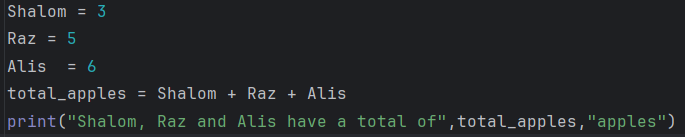
I created 2 variables and stored a number in each.

I then printed the value of a+b, now that a=1 and b=2, putting them together will result in 3.



I didn’t use quotes around the numbers because I want them to be treated as [**integers**](#datatypes). If I put the numbers in quotes, they become strings (text), and mathematical operations can't be performed on them.

**Variable task:**

create 3 variables and assign a number for each variable. calculate the total number of all variables combined. 



I assigned numbers to Shalom, Raz, and Alis to represent how many apples they have. Then, I added them together and stored the result in total\_apples.   
Finally, I printed a sentence showing the total number of apples.

**Math Operators in Python**

In Python, you’ll use these math operators to do basic calculations with your variables. these operators are essential for making your code work.

* **Basic Operators**: The fundamental operators include addition (+), subtraction (-), multiplication (\*), and division (/).
* **Modulus (%)**: Returns the remainder of a division operation; useful for checking if a number is divisible evenly

**Data types**

Data types are the building blocks of your data. They let Python know what type of information you're dealing with, whether it's numbers, text, true/false values, or more complex structures.

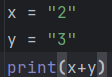
* Integers (22, 2025) for whole numbers.
* Floats (1.81) Decimal numbers, for precise values.
* String "Python is fun" for text.
* [**Booleans**](#boolean) ($true, $false) for yes/no.
* [**Arrays**](#lists) (= lists) to hold multiple values.

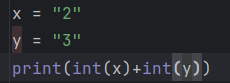
I will use the *'type'* command to demonstrate data types:



The type command before each bracket tells the console to display the data type inside the brackets. When I run it, the console shows the type of data contained within each bracket.

**Converting data types:**

When I print the 2 variables here and add them together, they result in the number 23, the reason being both numbers are strings thus the numbers 2 and 3 are added together, however, I can change their data type by typing the following:

typing "int" before a variable will change its data type to an integer, now the console will treat my variables as integers and the result would be 5.

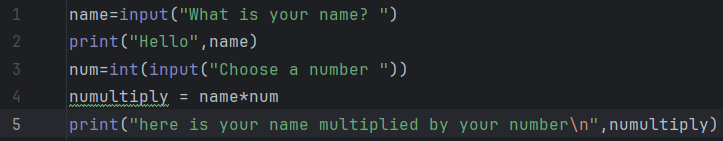
The same can be done with different data types:

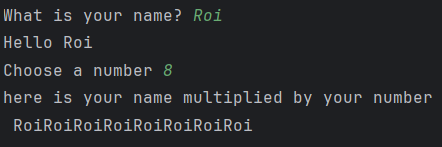
* str() converts to a string.
* int() converts to integer.
* float() converts to a decimal number.
* bool() converts to True or False.

  
  
In the following command, I prompted the user to type his name, the input will be stored under the variable "name".  
to do that simply type *input()* before a text you'd want to print.  
  
  
  
**Ask for a number:**

**User Input**

In line 3, the console prompts for a number. The int before input ensures that the input is treated as an integer, meaning the console expects a numerical entry.





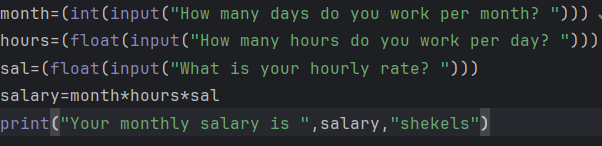
After the user enters a name and a number, I’ll create a new variable that multiplies them. Then, the print command will show the result.

**User Input task:**

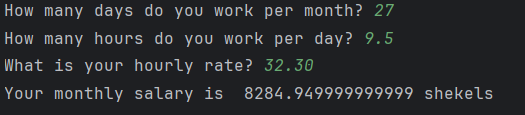
I created 3 variables that prompt the user to input a numeral entry for the number of days they work per month, the number of hours they work per day, and the amount they get paid per hour.

I then created the variable "salary" that multiplies them all together. The print command will show the result.

I used float instead of int so that my script could read decimal numbers. here is an example of me calculating the exact minimum wage in Israel.

****

Here is the result:

****

**Pro tips and Shortcuts**

**Store Data in 2 Variables in 1 line**

Instead of: *x1 = 10*

*x2 = 20*

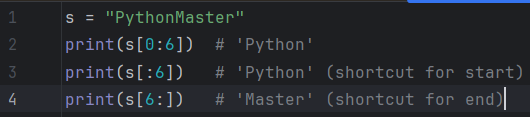
Type: *x1 , x2 = 10 , 20*

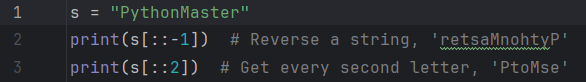
**Notes**

You can leave notes (comments) to make your code more readable and easily understood. Use # to leave quick notes explaining your code:



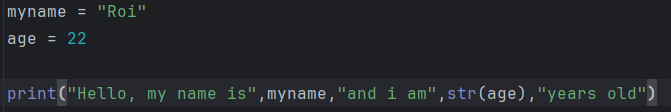
**String Slicing**



****

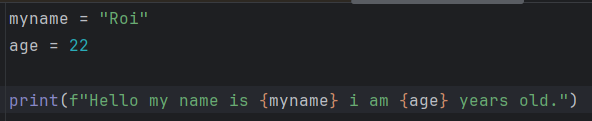
**F Strings**

Makes strings look better, easier to write, and straightforward. instead of typing:



In this format we have to break in and out of our string.

In the new format, simply insert the letter f before the string to turn it into an f-string, and place arguments inside curly brackets {} to include variables dynamically.

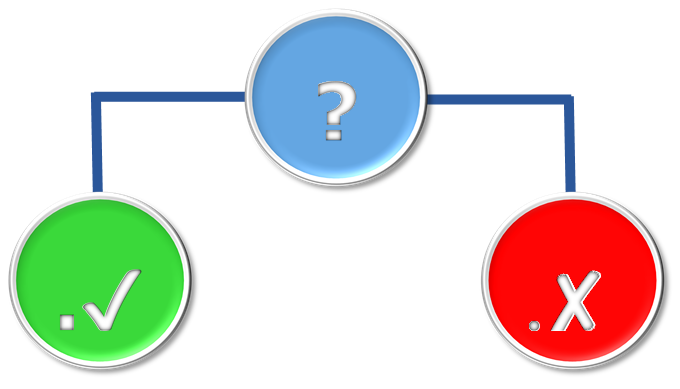
****

**Floor Division**

The // operator in Python is used for floor division, which means it divides two numbers and rounds the result down to the nearest whole number.



**Boolean Value**

**What is Boolean?**

A Boolean in Python is a data type that can have one of two values:

✅ True  
❌ False

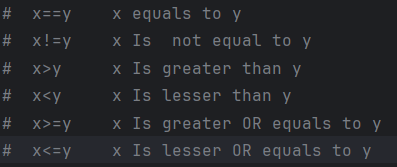
**What do we use Boolean for?**

Booleans are used for decision-making, controlling loops, checking conditions, and turning features on and off.

**Comparison Operators**

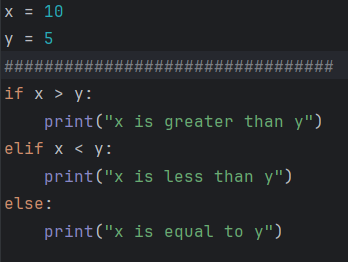
Comparison operators are used to compare values, they return a Boolean value (true or false), based on the comparison.

Here is a list of comparison operators:

x and y are just an example of a variable, this can be done with other variables.

**Conditional Statements**

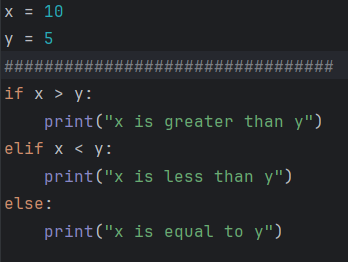
You can use conditions in control flow statements, such as if, elif, and else, to execute different blocks of code based on whether the condition is met.

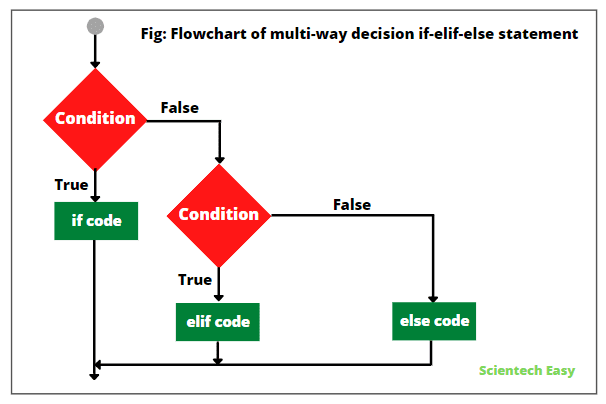
In this example I check if x is greater than y using the "if" condition and ">" operator.

* if Statement – Checks if the condition (x > y) is true. If true, the code below executes; if false, it exits (or moves on depends on the code.
* Condition (x > y) – Evaluates if x is greater than y.
* print Function – If true, *prints ("x is greater than y")*

**if, elif, else**

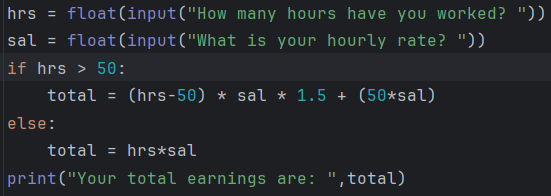
* elif Statement – Short for "else if," this checks an additional condition after an 'if'. It allows you to specify multiple alternative conditions to evaluate.
* else Statement – Acts as a fallback when all previous conditions (if and elif) are false. It runs the code block when none of the specified conditions apply.

In this example, I set values for two variables, x, and y, and used conditional statements to check which comparison conditions they meet.



**Boolean and Conditions Exercise**

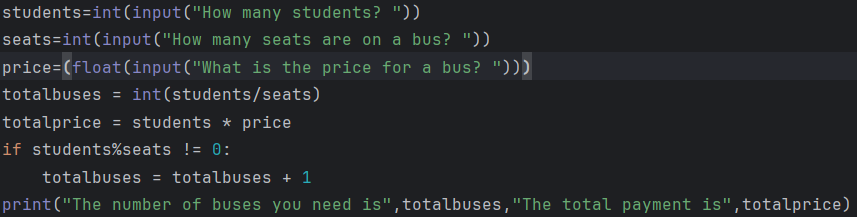
Write a program that prompts the user to input how many hours they worked and their hourly rate. If they worked more than 50 hours, their rate should be multiplied by 1.5. Finally, print the total earnings.



* *float* – reads a decimal number inputted by the user.
* *if hrs > 50* – if the hours exceed **50** do this next command.
* *(hours-50) \* sal \* 1.5* – I'm subtracting **50** from my variable **hrs** to determine the overtime hours and multiplying that amount by the hourly rate **(sal)** and then by **1.5** to account for the increased pay rate.
* *+ (50\*sal)* - calculates the regular pay for the first **50** hours by multiplying **50** by the hourly rate **(sal)**
* *else* – if the hours don't exceed **50** the variable total will calculate the regular pay **(hrs\*sal)**.

**Boolean and Conditions Exercise 2**

Make a program that asks for a number of; students, bus seats and price.  
Calculate the price for each bus seat and how many buses are needed.



* In lines one and two, I ask the user to input a number for the students and seats.
* In line three, I ask the user for a decimal number for the price using **float**.
* totalbuses is a variable that stores the integer value of students divided by seats (the reason I used **int** is that I want a full number rather than a decimal).
* totalprice calculates how much each student should pay.
* The condition if *students % seats != 0:* checks if students can't be evenly divided by the number of seats. If true, it adds **+1** to the variable totalbuses, meaning no student is left behind.

**Boolean and Conditions Exercise 3**

asks the user to enter their age and then evaluates it using conditional statements. Based on the input, it will print an appropriate message.

****I first created the variable "Age" and prompted the user to enter their age which will be converted to integer.

I used comparison operators to evaluate the age.

* if age is negative, you get an error message.
* if age is less than or equal to 3 *(<=3)* it classifies you as a baby.
* if age is less than or equal to 120 (<=120) it classifies you as 'old'.
* any number over 121 (> 121) will print the same message.

**Logical Operators**

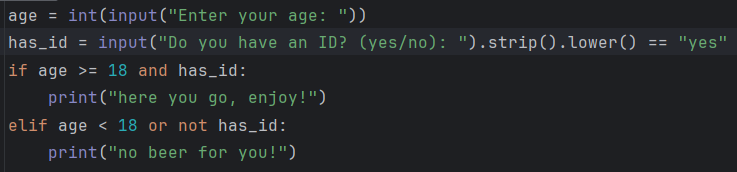
Logical operators help you make complex decisions in your code. They are used for combining multiple conditions and returning True or False based on the evaluation.

**Meet the Operators:**

* **and** – Returns True if both conditions are true.
* **or** – Returns True if at least one condition is true.
* **not** – Reverses the result, turning True into False and vice versa.

**Practice:**

A program that asks the user for an age and if they have an ID, the user is required to satisfy both conditions (ID and age) to earn his beer.

****

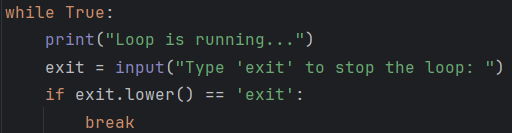
* *strip()* removes extra spaces.
* *lower()* makes everything lowercase.  
  >>>
* If they type " YES ", it becomes "yes", which equals "yes" (==), so has\_id = True.
* *and:* Combines boolean values to determine the truth value. Both values must be True for the condition to be satisfied.
* *or not:* if you're either below 18 or do not possess an id the boolean value = false.

**What are loops?**

**Loops**

Loops repeat a block of code multiple times until a specific condition is met.

**Example:**

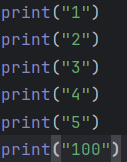


* *while True* – Starts an infinite loop, the condition True means the loop will keep running until it's broken.
* *print("Loop is running")* – Indicates that the loop is currently active.
* *exit = input("Type 'exit' to stop the loop: ")*- This line prompts the user to type 'exit' if they want to stop the loop, the input is stored in the variable "exit".
* *if exit.lower() == 'exit':* – checks if the user's input equals to "exit", and converts the user's input to lowercase allowing the user to type "exit," "EXIT," or any other variation.
* *break* - If the condition in the if statement is met (meaning the user typed "exit"), this line breaks out of the loop.

**Why do we use loops?**

Loops are used to:

* **Avoid Repetition:** Loops let you run the same code multiple times without copying it.



* **Handle Dynamic Data:** They work well with [**lists or arrays**](#lists).
* **Save Time and Effort:** Loops automate tasks like data iteration and calculations, making coding faster and easier.

**Ranges:**

range() in Python is an object that generates a sequence of numbers.

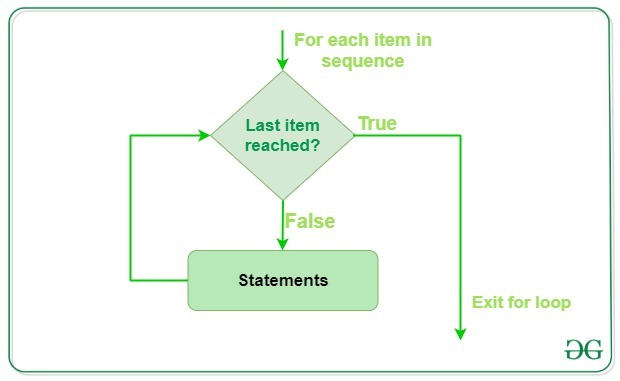
In Python: *for i in range(5)* generates numbers 0 to 4.

We can play with how we count numbers within a range.

For example, range(0, 10, 2) tells the program to start counting at 0, end at 10, and increment by jumps of 2.

**How does a loop work?**

A loop starts by setting a variable to the first number in a range. It checks if there are more numbers to go through.   
If there are, it runs the code inside the loop for that number. After that, it moves to the next number.   
This keeps happening until all the numbers in the range have been used.   
Once it’s done, the loop stops, and the program continues with whatever comes next.



Last item reached?

Statements

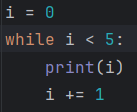
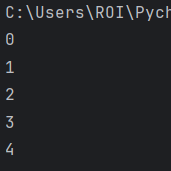
**Types of Loops**

Loops come in different types to handle different repetitive tasks.

**For loop** – This loop is ideal when you know how many times you want to repeat a command block or when working with collections.

The role of the for loop is to iterate through a sequence of numbers generated by range(5).

**While loop** – This loop is used when the number of iterations is unknown and depends on a condition.



Output

While tells the program to continue the loop as long as i is below five. (As long as that condition is true).

Print the value of i every time the loop runs (*print(i)*), and increment its value by 1 (*i += 1*).

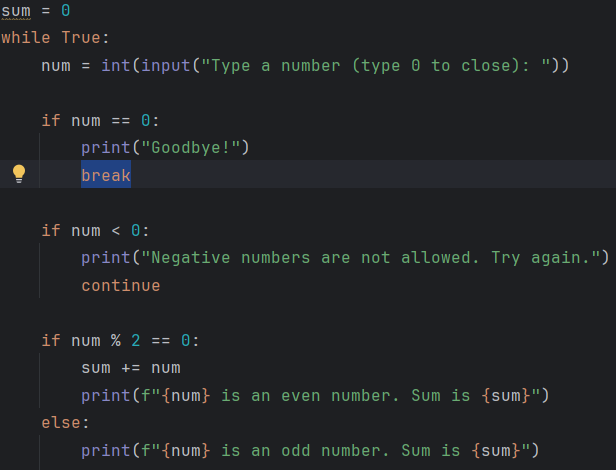
In this instance, i is a variable that holds the current loop iteration value. It's common to use i in loops to distinguish loop counters from other variables

**Continue and Break**

in Python, there are two ways to exit or control a loop:

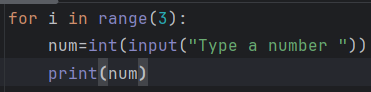
**break** – Stops the loop, but the program keeps running.

**continue** – Skips the current iteration and moves to the next one.

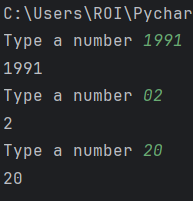


**Practice**

Create a program that asks for three numbers. Each number will be printed on the screen immediately using a for loop.



This simple program asks the user for a number three times (*range (3)*), when typing a number and hitting enter the number will be printed on screen (*print(num)*).



**Practicality of Loops**

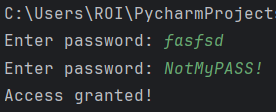
Here are commonly used cases for loops:



**Password Authentication with While loops:**

The loop keeps asking until the correct password is entered.  
As long as password is not equal *(!=)* to "NotMyPASS!" the loop will keep running.

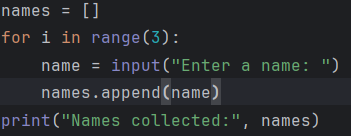
**Output:**



I attempted to enter a password the first time, the password did not match the correct one, so the command ran again.

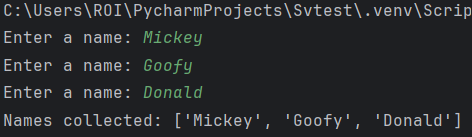
When I entered the correct password the console printed the message "Access granted".

**Name Collecting using For loops:**



* *names = []* – Creates an empty **list** called names to store user input.
* *for i in range(3):* – Starts a loop that repeats 3 times. The variable i is a 'throwaway' variable since we don’t need it.
* *names.append(name)* – Adds the entered name to the names list.

**Output:**



The program asked me to enter a name three times, the program then appended those names into the **list** called "names", it finally displayed the list with my names on it with a print command.

More on Lists later.

**Libraries**

Python libraries are pre-written code collections that save you time and effort.   
Instead of writing everything from scratch, you can import a library and use its built-in functions.

**Types of Libraries**

**Built in libraries**

* **math** – Advanced math functions (sqrt, sin, log, etc.).
* **random** – Generates random numbers.
* **datetime** – Works with dates and times.
* **sys** – System-related functions (sys.exit(), command-line arguments, etc…)

**Standard External Libraries** **(Require installation)**

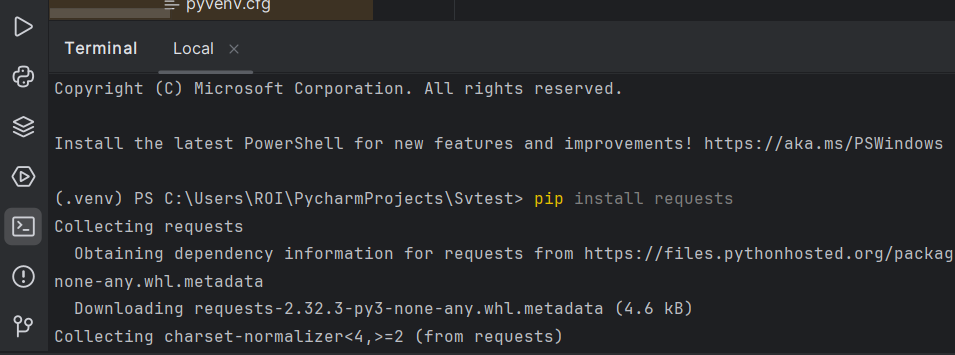
* **numpy** – fast math and large arrays.
* **pandas** – Data analysis and tables.
* **matplotlib** – Graphs and charts.
* **requests** – Web scraping and APIs.

There are plenty of libraries, some are used in cybersecurity, but for now let's focus on the basics.

**Install Libraries:**

Libraries are installed from the Terminal, not the Console, access the terminal by pressing **this** icon and type:  
*pip install your\_library*

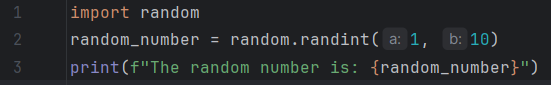
The installation will then start, once the installation has been complete, the terminal will print "Successfully installed"



**Import Libraries**

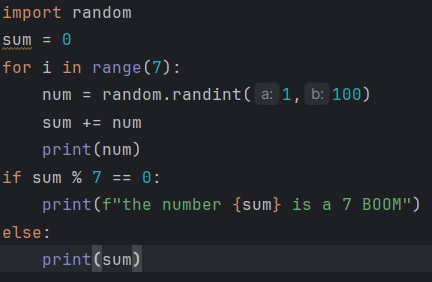
Here's an example of how to import the 'random' library and use the 'randint' function in Python.

The randint() function from the random module generates a random integer between two specified numbers.



This program prints a random number between 1 and 10 using *random.randint*. Specify the range by typing the starting number and a closing number to set a range.

*random.randint(1,10)* a: and b: are added automatically.



I made the game 7 Boom using randomly picked numbers. The program adds all 7 randomly generated numbers from 1 to 100 to the variable sum and checks if the total is divisible by 7.

****

**Lists**

A list in Python is an ordered collection of items that can be of different data types.   
a list is a container holding multiple values, similar to an array in other programming languages.

You create a list by using square brackets [], and separating the items by commas;

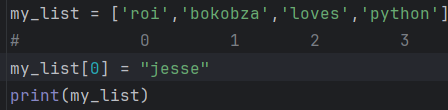


Items within a list are accessed by their index, which starts counting from 0.

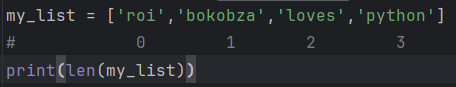


Print an item out of a list by referring to its index number as seen in this example:

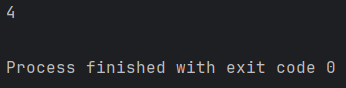


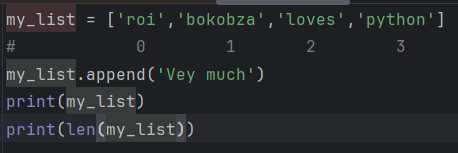
****

Replace an item in the list by specifying the item you want to replace, in this case, I replaced the word 'roi' with 'Jesse'. An item can also be replaced with an integer, float, and Boolean data types.

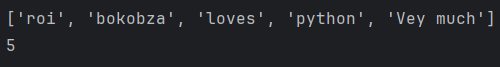


*len()* returns the number of items in a list, it counts how many things are in the container.

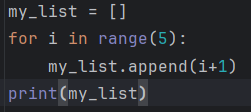




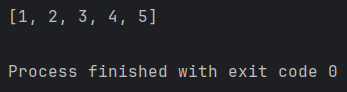
Add names to a list by typing *your\_list.append*



We can use a [**For**](#forloop) loop to add items to a list.

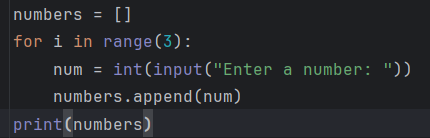


This code initializes (sets up the variable) an empty list and runs a loop five times, appending i + 1 to the list. Since i ranges from 0 to 4, the final list is [1, 2, 3, 4, 5].



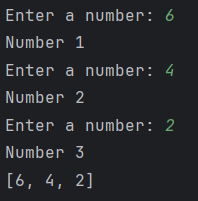
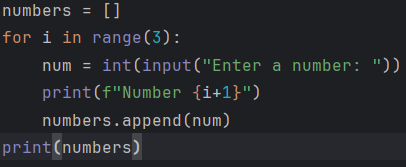
**Practice:**

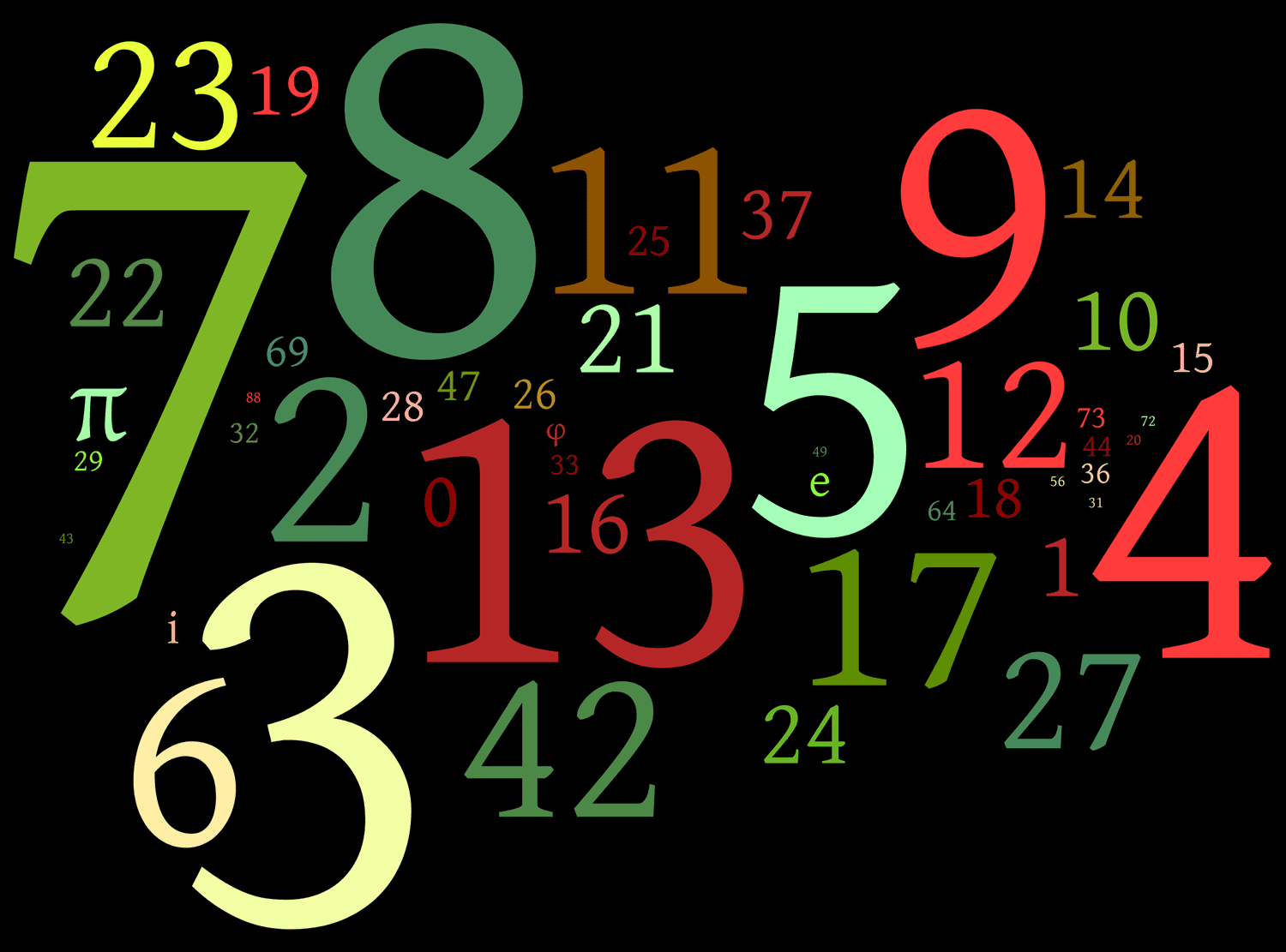
Write a program that asks for three numbers from the user, each number will be added to a list, finally print the list.



I initialized the list **numbers**, then repeated the input command three times using the For loop, finally I added the values of the variable num to the list called **numbers,** then print.

We can upgrade this using the curly brackets, the program now tells you how many numbers you inserted by using *{i+1}.*





**Guess the Random Number**

Before diving into the more advanced commands and features of Python, I'd like to make a project with all the knowledge we have accumulated thus far.

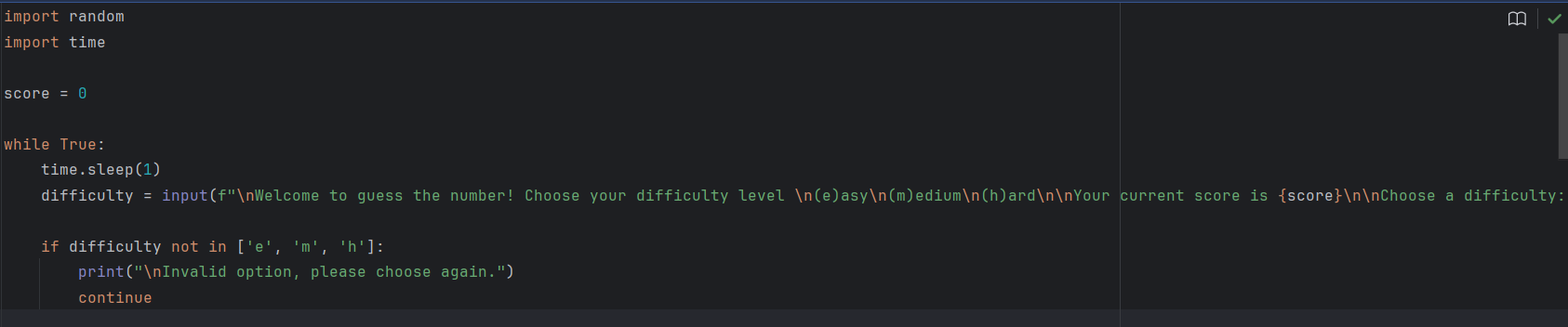
This includes Variables, Loops, conditions, Libraries and lists.

**Concept:**

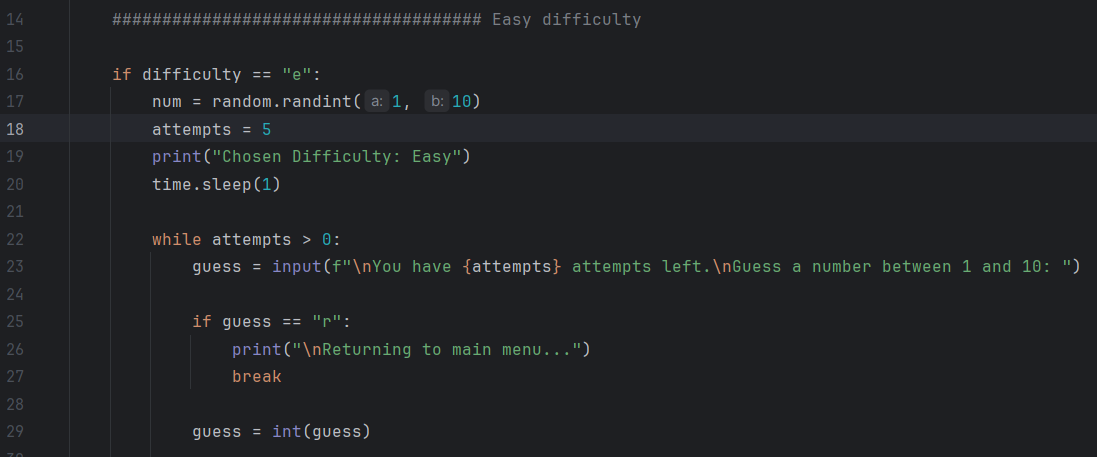
The goal is to create a game where the user tries to guess a randomly generated number. The program will inform the user if their guess is too low or too high. The user has five attempts to guess correctly.

**Features:**

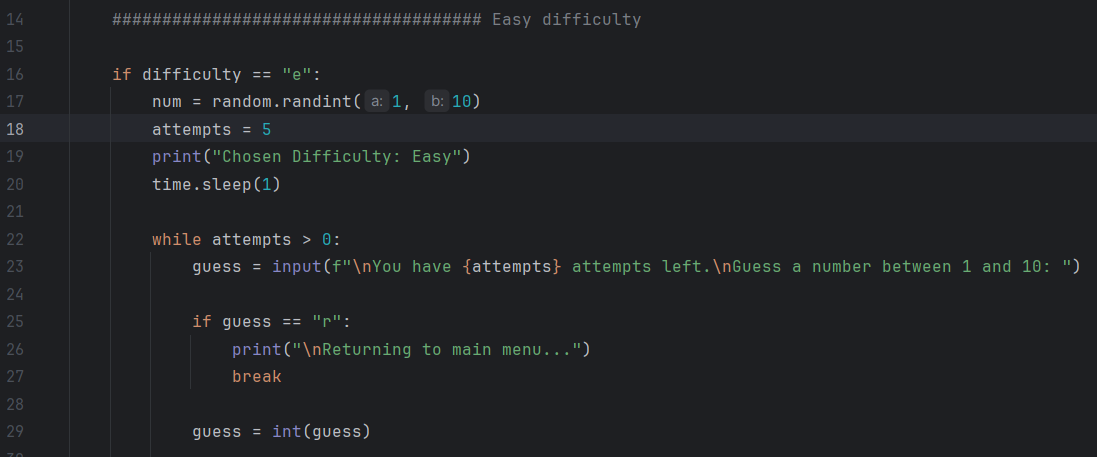
* Difficulty level selection
* Option to restart the game
* Scoring system

First, I imported two modules: one for adding delays between prints (*time.sleep*) and another for generating random numbers (*random.randint*). I initialized a score variable to track points, starting at 0.

The game runs inside a while loop to allow restarting. A menu prompts the user to choose a difficulty level: (e)asy, (m)edium, (h)ard. If the input is invalid, the program prints an error and restarts using continue.

****

Each difficulty sets a different number range and attempts. The game generates a random number (num), and the player has limited guesses (attempts). The user enters a guess, which is first treated as a string to allow a restart option (r). If valid, it converts to an integer (*guess = int(guess)*)

****

If the user's guess is **out of range** (less than 1 or more than 100), the loop restarts (continue) without deducting an attempt.

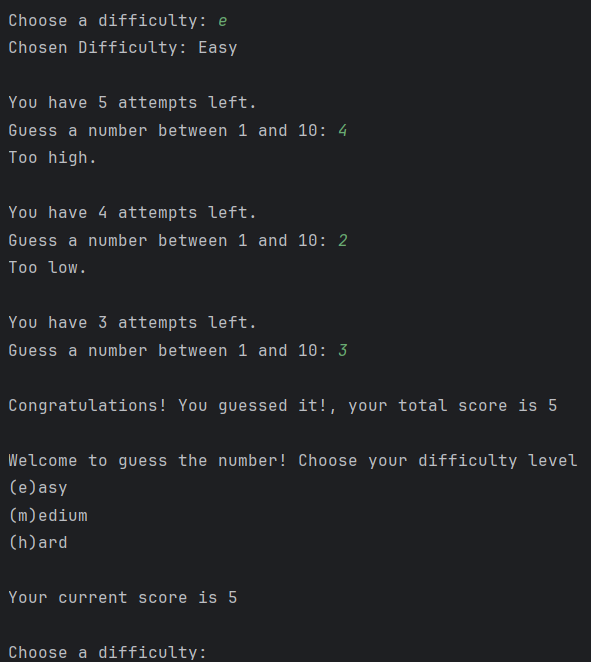
If the guess is **too low**, the program prints *"Too low."* If it's **too high**, an elif handles that case. >>>

If the guess is **correct**, the player earns 5 points, the loop breaks, and the program returns to the menu, now showing the updated score.

Every wrong guess reduces *attempts -= 1*. If attempts reach 0, the program prints the correct number and restarts.

Then do the same for the rest of the difficulty levels, in my program increasing difficulty levels reward with more score, but the range of the numbers is increased.

**Console:**

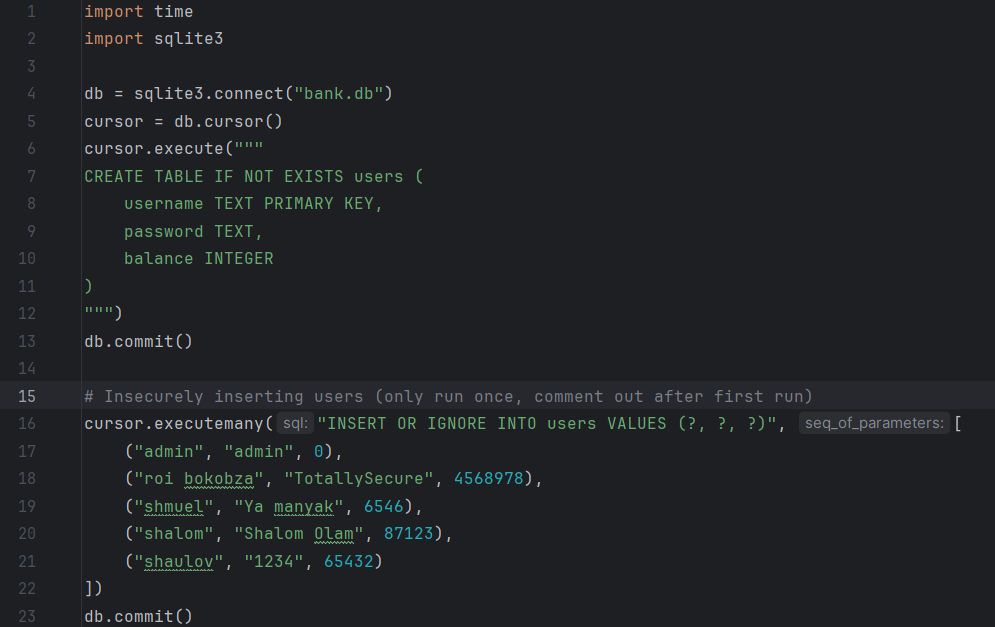
****

Here is an example of me winning the easy difficulty level.

**Banking System**

To conclude this Python project, I'd like to show you a project I made using the tools I learned from this project, and hopefully get you excited to start developing bigger and more sophisticated software and apps on your Python journey.

It’s a basic banking system with:  
✅ **User authentication** (username + password)  
✅ **Admin privileges** (view all accounts, reset passwords)  
✅ **User actions** (view balance, reset password)  
✅ **Logout system**



I will be using SQLite for this project to store credentials and bank account details in a database. [**MySQL Project.**](https://www.linkedin.com/posts/roi-bokobza-1163b0314_sql-management-and-injections-activity-7303760985389948930-kSgg?utm_source=share&utm_medium=member_desktop&rcm=ACoAAE_K36IBGi7l0HZWYzeU7_6Tv_r2LgPjX-o)

time.sleep(1) is used later to pause execution for 1 second, making it feel more natural.

*"db = sqlite3.connect("bank.db")"*

This line connects to an SQLite database file called bank.db. The variable db now holds the connection to that database.

*"cursor = db.cursor()"*

This line creates the **cursor**, **which is then used to run SQL commands**. Cursors are used to execute SQL queries and fetch results from the database.

*cursor.execute("""  
create table if not exists users (  
 username text primary key,  
 password text,  
 balance integer  
)  
""")*

This creates a table called users, it has 3 columns, username, password, and balance.  
usernames and password require a text data type and the bank account balance requires an integer (number).

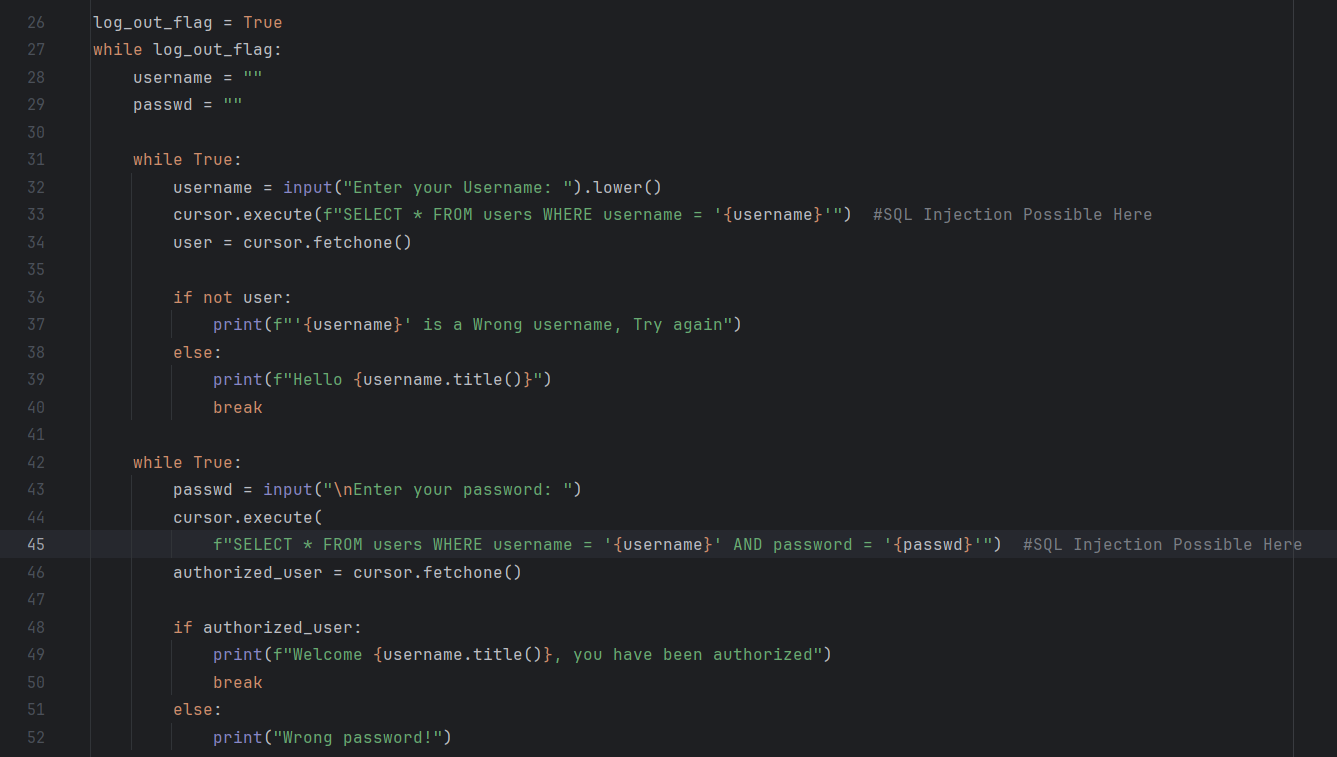
*db.commit()*

used to save (or "commit") any changes you've made to the database, like inserting or updating records. (used for resetting passwords)

*cursor.executemany("insert or ignore into users values (?, ?, ?)", [  
 ("admin", "admin", 0),  
 ("roi bokobza", "totallysecure", 4568978),  
 ("shmuel", "ya manyak", 6546),  
 ("shalom", "shalom olam", 87123),  
 ("shaulov", "1234", 65432)  
])  
db.commit()*

This inserts multiple rows of user data into the users table. Each line represents a user with username, password and balance.

The *insert or ignore* statement ensures that if any row already exists with the same username, it will be ignored and not inserted again.



**Username Login:**

*log\_out\_flag = True:*

The log\_out\_flag is set to True, keeping the login loop active. It turns False when the user logs out, exiting the loop and allowing for re-login.

*cursor.execute(f"select \* from users where username = '{username}'")*

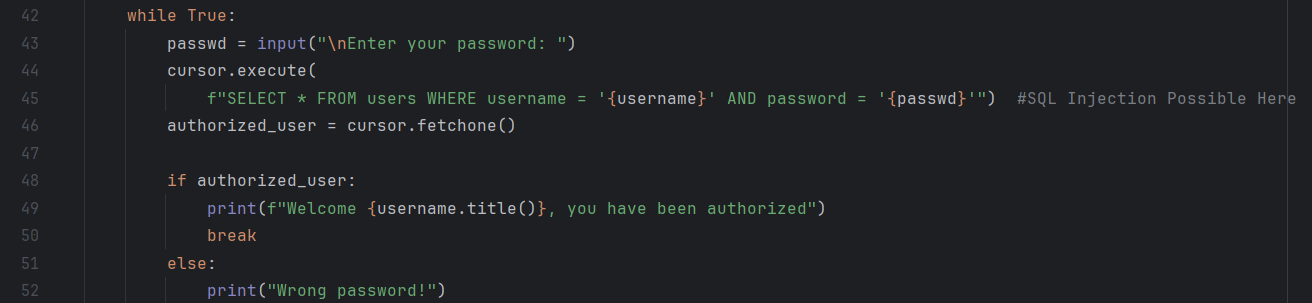
*user = cursor.fetchone()*

It fetches the first matching row using cursor.fetchone().

If no user is found (if not user:), it prompts the user to try again.

If the user is found, it greets them and breaks out of the loop.

**Password Login:**



This loop handles the password input

*cursor.execute(f"SELECT \* FROM users WHERE username = '{username}' AND password = '{passwd}'")*

This cursor runs an SQL query to check if there is a user with the given username and password.

*authorized\_user = cursor.fetchone()*

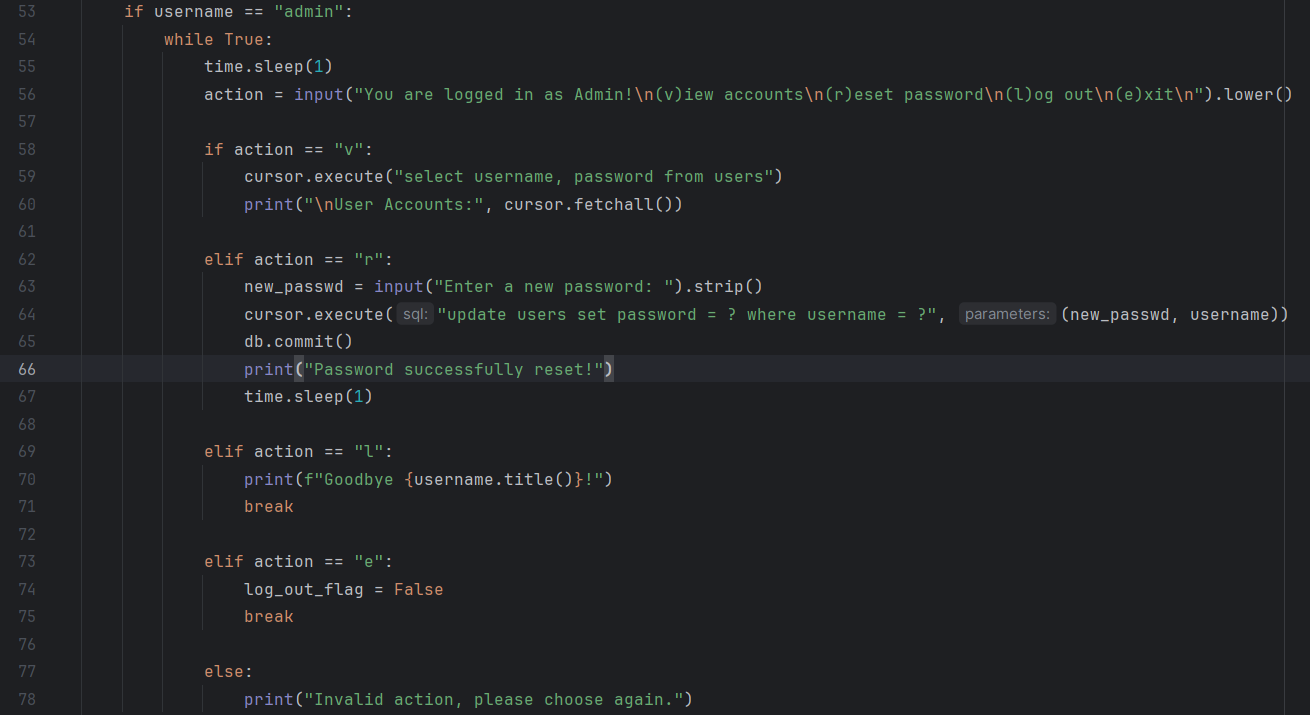
*if authorized\_user:*

*print(f"Welcome {username.title()}, you have been authorized")*

*break*

If the credentials are correct (if authorized\_user:), it prints a welcome message (.title makes the first letter a capital letter) and breaks out of the loop.

If the password is wrong, it prompts the user to try again.



This part of the program is only dedicated to the admin user since he has Admin Privileges.

*if username == "admin"* it runs a loop that requires the user to choose 1 of 3 options.

**View Accounts:**

if the variable action == v then all accounts are displayed (cursor.fetchall)

*cursor.execute("select username, password from users")*

This line runs an SQL query to select (retrieve) the username and password fields from the users table in the database.

*print("\nUser Accounts:", cursor.fetchall())*

Prints all the fields fetched from the users table.

**Reset Password:**

If the user types r it initiates a loop that requires a new password to be inserted.

*cursor.execute(f"update users set password = '{new\_passwd}' where username = '{username}'")  
db.commit()*

Updates the password in the users table in the corresponding column of the user we are logged in with.

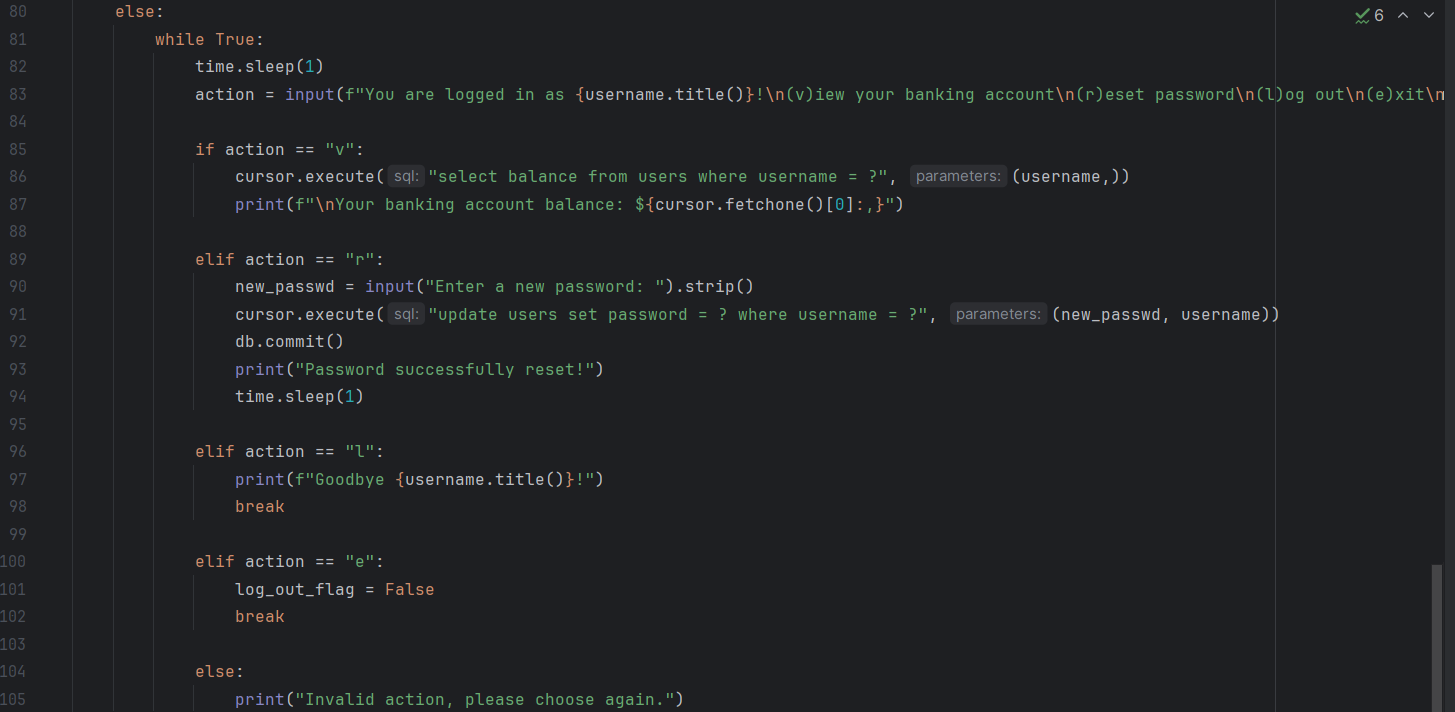
**Logout:**

If the user types L it breaks the loop, and returns you to the log\_out\_flag which prompts the user to enter a username.

**Exit:**

Breaks out of the current loop and also **sets log\_out\_flag to FALSE**, as you remember, **the program runs as long as log\_out\_flag = False!**

and 'else' that prints "invalid input" when an input does not match.

****

Because Admin had administrative privileges it had to be separated from the rest, in the previous screenshot, the entirety of the admin functions were under an If condition, the rest are under an else condition.

the details will be display and updated in the table based on the username you inserted.

**View Balance:**

*if action == "v":*

*cursor.execute("select balance from users where username = ?", (username,))*

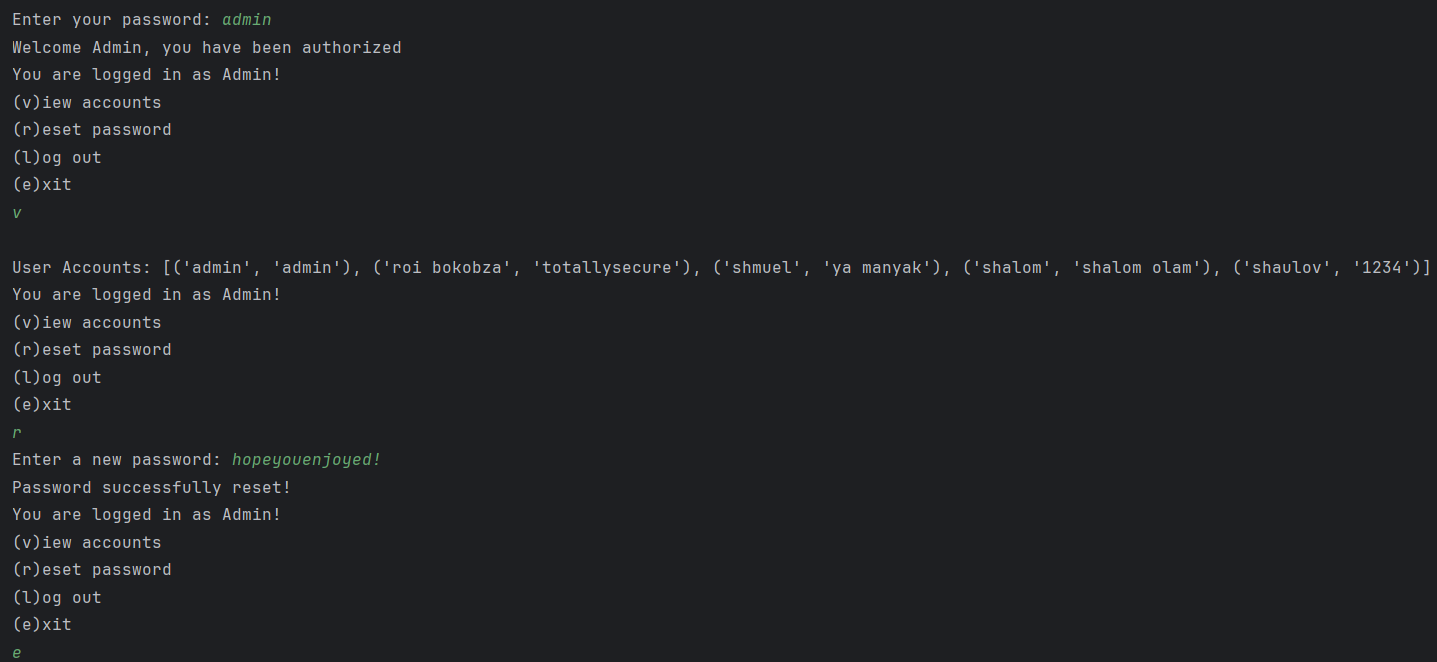
*print(f"\nYour banking account balance: ${cursor.fetchone()[0]:,}")*

This line executes a SQL query to fetch the balance of the user with the given username. It then prints the balance, formatted with commas.

It knows to fetch the data based on the WHERE username = ? condition, which filters the table for the row where the username matches the provided value. **The ? is a placeholder for the username variable** passed in the second argument.

R, L and E have the same functions as admin, but this time the functions apply to the username you entered.

Here is the program running, showing login, wrong password, balance and logouts



Here is a demonstration of the Admin Privileges of the user Admin

**The best part of it all is this is SQL Injection Vulnerable!**

Since all of our data is stored in an SQL database we can insert some commands that will reveal sensitive information of our banking accounts.

**Let's make it Injection Vulnerable:**

**String Concatenation**:

Instead of using parameterized queries (?), user input is now a part SQL query using f-strings or string concatenation, meaning our input can manipulate the table by being a part of it.

*query = f"select \* from users where username = '{username}'"*

*cursor.execute(query)*

**Vulnerable Points**: The username and password fields are now vulnerable to SQL injection because user input is treated as part of the SQL query.

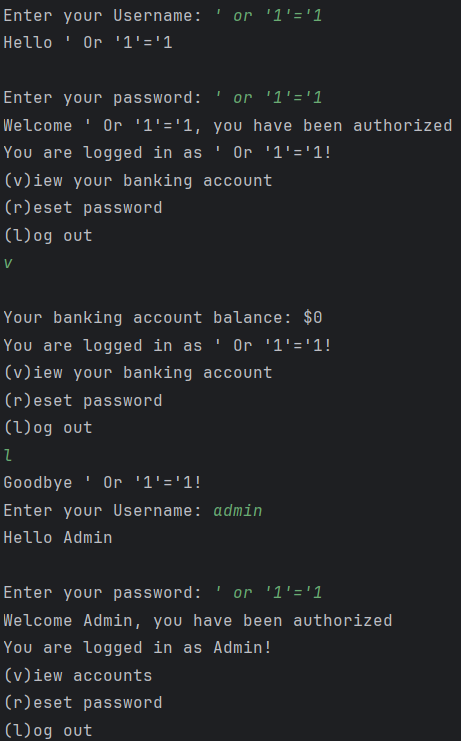
Now we can start injecting >>

**Here different examples of SQL Injections:**

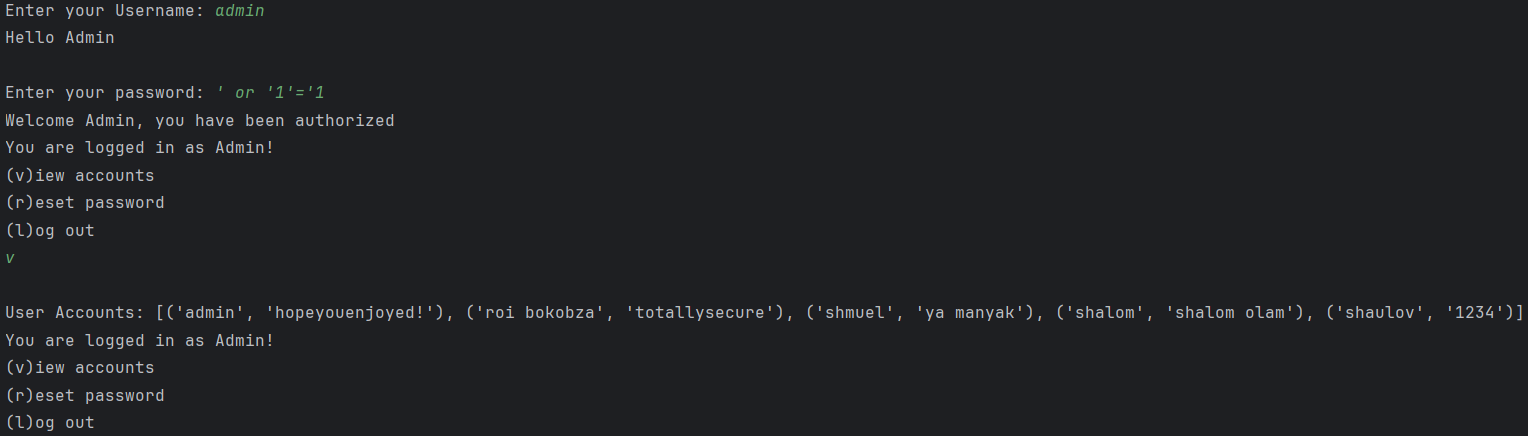
**Basic Authentication Bypass**: Injecting *' or '1'='1* would cause the condition to always be true (in SQL 1=1 is a Boolean data type that means True in SQL Language).

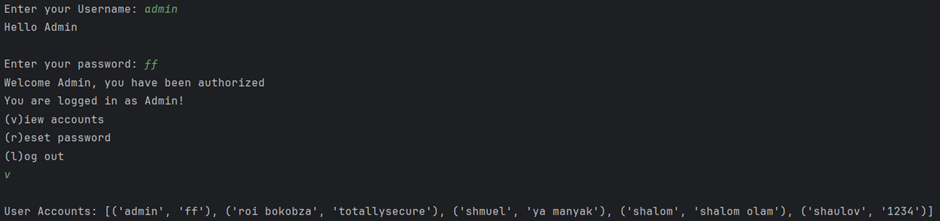
On the backend it would look like this:

**select** \* **from** users **where** username = **''** **or** **'1'**='1' **and** password = **''** **or** **'1'**='1';

We could use *' or '1'='1';* to bypass both username and password but it would not grant us access to the actual accounts stored in the table. but its enough to know a username, let's say I know the admin's username and perform this SQL Injection:

>>

I have used *' or '1'='1* to bypass the password query, now I have full access to the admin and I can view the rest of the accounts on my banking program.

**SQL Injection via Comment Out**

 the attacker manipulates the query by injecting malicious input and using SQL comments (--) to ignore the rest of the query.

**Backend:**

**select** \* **from** users **where** username = **'admin'** **--' and password = '1234'**