

**Name: Subhan Khalid**

**Intern ID: TN/IN02/PY/016**

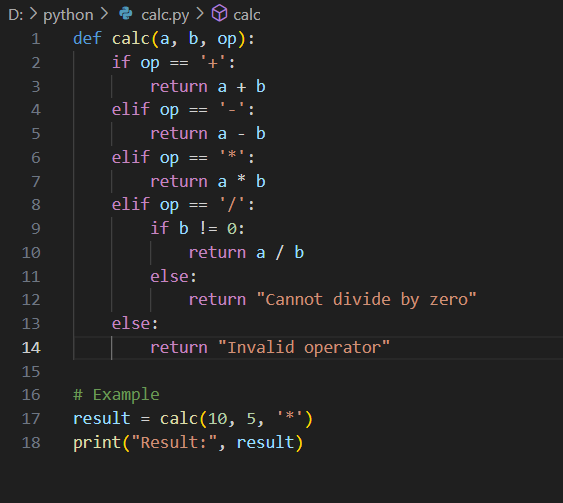
**Task no: Week 2**

**TASK NO:1**

**STEP BY STEP:**

1. Define a function calc(a, b, op) to perform operations.
2. Use if-elif to check which operator is given (+, -, \*, /).
3. Handle division safely by checking if b is not zero.
4. Return the calculated result or an error message if needed.
5. Call the function with example values and print the result.

**CODE:**

****

**OUTPUT:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to define and use a function in Python with parameters. I saw how to handle **basic arithmetic operations** using conditional statements (if, elif, else). I also understood how to avoid errors like **division by zero** using an if condition. Lastly, I saw how Python functions can return different data types like numbers or strings depending on the situation.

**TASK NO:2**

**STEP BY STEP:**

1. A tuple coordinates is created with three values.
2. Tuple unpacking is used to assign x = 10, y = 20, and z = 30.
3. Each variable is printed using print().

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**OUTPUT:**

**A black screen with white text

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to store multiple values in a **tuple**, and how to **unpack** them into separate variables in one line. This makes code cleaner and more readable

**TASK NO:3**

**STEP BY STEP:**

1. A sentence string is defined.
2. split() breaks the sentence into a list of words.
3. An empty dictionary word\_count is created.
4. Loop through each word:
5. If the word exists in the dictionary, increase its count.
6. If not, add it with count 1.
7. Print all words and their frequencies using a loop.

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**OUTPUT:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to split a sentence into **words**, and how to use a **dictionary** to count the frequency of each word. This is a basic intro to **data structures** and **loops** in Python. Also realized how dictionaries store data as key-value pairs, which is super handy for real-world text analysis stuff.

**TASK NO:4**

**STEP BY STEP:**

1. An empty dictionary student is created.
2. **Create:** Add name, age, and grade as key-value pairs.
3. **Read:** Access and print individual values using their keys.
4. **Update:** Change the value of the "grade" key.
5. **Delete:** Remove the "age" key using del.
6. Print the updated dictionary after deletion.

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**OUTPUT:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to perform **basic CRUD operations** using a Python dictionary. It shows how to add, read, update, and delete key-value pairs. This is super useful for storing structured data like records or objects

**TASK NO:5**

**STEP BY STEP:**

1. A list numbers is created with duplicate values.
2. The set() function is used to remove duplicates (since sets only store unique values).
3. The set is converted back to a list using list().
4. The cleaned list is printed.

**CODE:**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**OUTPUT:**

**A black screen with white text

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to quickly remove **duplicates** from a list using set(). Sets are great when you only want **unique values**. Also learned that you can convert between lists and sets easily in Python

**TASK NO:6**

**STEP BY STEP:**

1. A function factorial(n) is defined.
2. **Base case:** If n is 0 or 1, return 1.
3. **Recursive case:** Otherwise, return n \* factorial(n - 1).
4. The function keeps calling itself with smaller values until it hits the base case.
5. Finally, the factorial of 5 is printed.

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**OUTPUT:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**LEARNED:**

I learned how **recursion** works in Python — a function calling itself with a smaller problem until it reaches a stopping point. It’s a super elegant way to solve problems like factorials, but you gotta be careful with the base case or you’ll fall into the infinite loop abyss

**TASK NO:7**

**STEP BY STEP:**

1. Two functions are defined: add(a, b) and subtract(a, b) for basic math operations.
2. Variables x and y are assigned values 10 and 5.
3. add(x, y) is called to add the numbers, and the result is printed.
4. subtract(x, y) is called to subtract the numbers, and the result is printed.

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**OUTPUT:**

**A black screen with white text and yellow text

AI-generated content may be incorrect.**

**LEARNED:**

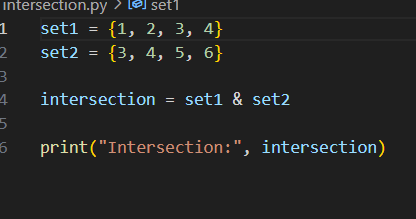
I learned how to define **multiple functions** in Python to separate different tasks. It shows how functions make code **modular, reusable, and clean**. Instead of repeating logic, we just call the function with new values — like having your own custom-built calculator

**TASK NO:8**

**STEP BY STEP:**

1. Two sets, set1 and set2, are created with some overlapping numbers.
2. The & operator is used to find the **intersection** — values common to both sets.
3. The result is stored in intersection.
4. The common elements are printed.

**CODE:**

****

**OUTPUT:**

**A computer screen shot of a code

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to find the **intersection of two sets** in Python using the & operator. Sets automatically remove duplicates and are great for comparing data. Perfect for checking shared items

**TASK NO:9**

**STEP BY STEP:**

1. **Try block:** Attempts to open the file data.txt in read mode ("r").
2. If the file is found:
3. It reads the entire content using read().
4. Prints the content.
5. Closes the file using file.close().
6. **Except block:** If the file doesn’t exist, it catches the FileNotFoundError.
7. Prints a custom error message: "File not found."

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**OUTPUT:**

**A black background with white text

AI-generated content may be incorrect.**

**LEARNED:**

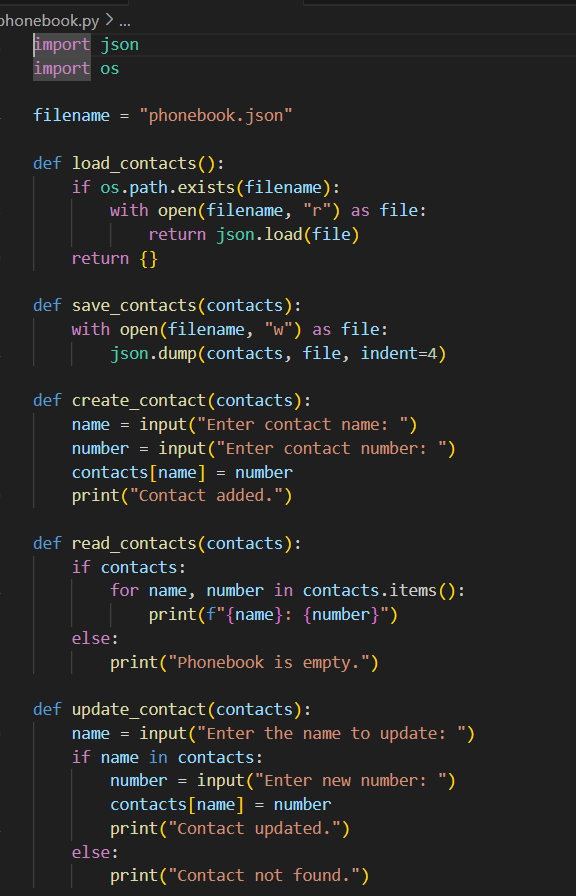
I learned how to use **try-except** in Python to handle errors gracefully. Instead of crashing the program when a file is missing, Python lets us **catch the error** and respond with a helpful message.

**TASK NO:10**

**STEP BY STEP:**

1. **Imports:** json for saving data in JSON format and os to check if the file exists.
2. **File Handling:**
   * load\_contacts() reads existing contacts from phonebook.json if it exists.
   * save\_contacts() writes updated contacts back to the file.
3. **CRUD Functions:**
   * create\_contact(): Adds a new contact with user input.
   * read\_contacts(): Displays all saved contacts.
   * update\_contact(): Changes the number of an existing contact.
   * delete\_contact(): Removes a contact by name.
4. **Main Menu Loop:**
   * Displays options in a loop.
   * Based on user input, calls the relevant function.
   * On exit (choice == 5), saves contacts to file.

**CODE:**

****

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A black background with white text and colorful symbols

AI-generated content may be incorrect.**

**OUTPUT:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**LEARNED:**

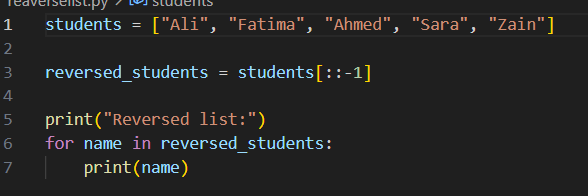
1. **JSON** to store and load data persistently.
2. **Functions** to organize tasks cleanly (modularity ftw).
3. **Loops and conditionals** to handle user interaction.
4. **Error checking** using os.path.exists() for safer file access.

**TASK NO:11**

**STEP BY STEP:**

1. A list students is created with 5 names.
2. The list is reversed using slicing: [::-1].
3. The reversed list is stored in reversed\_students.
4. A loop prints each name from the reversed list.

**CODE:**

****

**OUTPUT:**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to **reverse a list** using Python’s slicing trick [::-1]. It’s a super clean one-liner — no loops, no fuss. Also practiced how to loop through a list and print items one by one.

**TASK NO:12**

**STEP BY STEP:**

1. A while True loop keeps running until valid input is received.
2. Inside the loop, the program tries to convert the input to an integer using int().
3. If the conversion succeeds, it **breaks** out of the loop.
4. If a ValueError occurs (e.g., user enters text), an error message is printed.
5. The loop repeats until the user finally enters a valid integer.

**CODE:**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**OUTPUT:**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**LEARNED:**

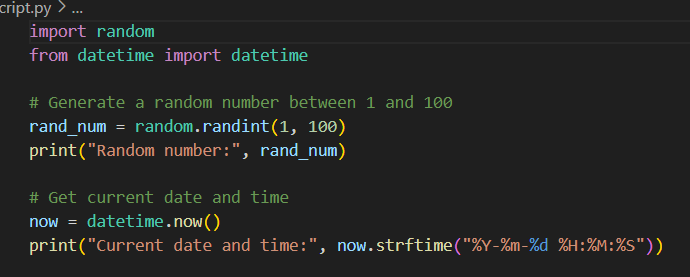
I learned how to make user input **safe and reliable** using try-except inside a loop. This is how real programs avoid crashing when users type nonsense. It’s a solid intro to **input validation**, and now I see how while True loops are great for keeping control until the job is *done right*

**TASK NO:13**

**STEP BY STEP:**

1. random and datetime modules are imported for randomness and time handling.
2. random.randint(1, 100) generates a random number between 1 and 100.
3. The random number is printed.
4. datetime.now() fetches the current date and time.
5. strftime() formats it nicely into YYYY-MM-DD HH:MM:SS.
6. The formatted date and time are printed.

**CODE:**

****

**OUTPUT:**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to generate **random numbers** in Python for dynamic results, and how to fetch and format the **current date and time** using the datetime module. This is super useful for everything from **games to logs to timestamps**

**TASK NO:14**

**STEP BY STEP:**

1. A list students is created with five names.
2. A for loop goes through each name in the list one by one.
3. Inside the loop, each student’s name is printed.

**CODE:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**OUTPUT:**

**A black screen with white text

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to use a **for loop** to iterate through a list in Python. It’s the simplest way to go through a collection and perform an action — like printing, calculating, or checking. This kind of loop is the **foundation of automation** in code.

**TASK NO:15**

**STEP BY STEP:**

1. A tuple coordinates is created with three values.
2. The values are **unpacked** into variables x, y, and z in a single line.
3. Each variable is printed individually with a label.

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**OUTPUT:**

**A black screen with white text

AI-generated content may be incorrect.**

**LEARNED:**

I learned how to unpack a **tuple** directly into multiple variables. It saves time, makes code cleaner, and avoids writing separate assignments. Python’s syntax is for working with grouped data like coordinates, RGB values, or anything in sets of 2s and 3s.