**DAY 3:- TASK 1:-**

1) create database HexaAirlines

create database hexaairlines; use hexaairlines;

2) create tables flight, piolets ,airhostess , foodTeam, customers

create table flight(flight\_id int primary key,flight\_name varchar(50),destination varchar(50),price int);

create table piolets(piolet\_id int primary key,piolet\_name varchar(50),age int,flight\_id int,foreign key(flight\_id) references flight(flight\_id));

create table airhostess(hostess\_id int primary key,hostess\_name varchar(50),age int,experience int,flight\_id int,foreign key(flight\_id) references flight(flight\_id));

create table foodteam(food\_id int primary key,food\_type varchar(20),cost int);

create table customers(cust\_id int primary key,cust\_name varchar(50),destination varchar(50),flight\_id int,food\_id int,foreign key(flight\_id) references flight(flight\_id),foreign key(food\_id) references foodteam(food\_id));

3) fill 6 to 7 rows in these and have primary and foreign keys

insert into flight values(1,'indigo','bangalore',4500),(2,'jetairways','mumbai',5500),(3,'airindia','goa',6000),(4,'vistara','delhi',5000);

insert into piolets values(1,'ravi',45,1),(2,'arun',52,2),(3,'naveen',38,3),(4,'kumar',61,4);

insert into airhostess values(1,'meena',30,4,1),(2,'rekha',40,6,2),(3,'anu',36,7,3),(4,'seema',50,10,4);

insert into foodteam values(1,'veg',200),(2,'non-veg',300);

insert into customers values(1,'ram','bangalore',1,1),(2,'raj','mumbai',2,2),(3,'rani','goa',3,1),(4,'nina','goa',3,2),(5,'john','delhi',4,1),(6,'kavi','mumbai',2,1),(7,'lavanya','bangalore',1,2);

4) select customers going to bangalore in indigo

select cust\_name from customers where destination='bangalore' and flight\_id=1;

5) select customers going to mumbai in jetairways

select cust\_name from customers where destination='mumbai' and flight\_id=2;

6) select customers going to goa in airindia and having veg food

select cust\_name from customers where destination='goa' and flight\_id=3 and food\_id=1;

7)add new column in piolets table, covid vaccination date and fill values in column

alter table piolets add covid\_vaccine\_date date;

update piolets set covid\_vaccine\_date='2022-01-15' where piolet\_id=1;

update piolets set covid\_vaccine\_date='2022-02-10' where piolet\_id=2;

update piolets set covid\_vaccine\_date='2022-03-05' where piolet\_id=3;

update piolets set covid\_vaccine\_date='2022-04-20' where piolet\_id=4;

8) change the age of airhostess who are elder then 35 to 34

update airhostess set age=34 where age>35;

9) update the price for the customer food or flight charge and make changes

update flight set price=6500 where flight\_name='airindia';

update foodteam set cost=250 where food\_type='veg';

10) delete piolets whose age is greater then 60

delete from piolets where age>60;

11) delete airhostess who are working from last 6 years or elder then 50

delete from airhostess where experience>=6 or age>50;

12) list out the name of piolets

select piolet\_name from piolets;

13) list out the name of airhostess

select hostess\_name from airhostess;

14) use where clause to retrieve data from above tables

select \* from customers where destination='mumbai';

select \* from flight where price>5000;

select \* from foodteam where food\_type='veg';

select \* from airhostess where age<40;

**DAY3: TASK 2:-**

**Section 1: Managing Databases**

1. **Which of the following is NOT a system database in SQL Server?**  
   a) master  
   b) model  
   c) tempdb  
   d) userdb
2. **Which system database stores all login accounts and configuration settings?**  
   a) tempdb  
   b) model  
   c) master  
   d) msdb
3. **What is the purpose of the model database in SQL Server?**  
   a) Backup  
   b) Log storage  
   c) Template for new databases  
   d) System configuration
4. **What are the two main types of database files in SQL Server?**  
   a) MDF and NDF  
   b) LDF and MDF  
   c) NDF and BAK  
   d) BAK and TRN
5. **Which SQL command is used to create a new database?**  
   a) MAKE DATABASE  
   b) NEW DATABASE  
   c) CREATE DATABASE  
   d) INIT DATABASE
6. **What happens when you execute DROP DATABASE SalesDB?**  
   a) SalesDB is backed up  
   b) SalesDB is renamed  
   c) SalesDB is deleted permanently  
   d) SalesDB is restored
7. **Which command renames a database in SQL Server?**  
   a) RENAME DATABASE old\_name TO new\_name  
   b) ALTER DATABASE old\_name MODIFY NAME = new\_name  
   c) UPDATE DATABASE NAME  
   d) SET DATABASE NAME

**Section 2: Managing Tables**

1. **Which data type should be used to store a date of birth?**  
   a) VARCHAR  
   b) DATE  
   c) INT  
   d) TEXT
2. **What command is used to create a table?**  
   a) MAKE TABLE  
   b) INSERT TABLE  
   c) CREATE TABLE  
   d) DEFINE TABLE
3. **How do you add a new column to an existing table?**  
   a) ALTER TABLE table\_name ADD column\_name datatype  
   b) MODIFY TABLE table\_name ADD column\_name  
   c) UPDATE TABLE table\_name ADD column\_name  
   d) APPEND column\_name TO table\_name
4. **Which command is used to rename a table?**  
   a) RENAME TABLE old\_name TO new\_name  
   b) ALTER TABLE old\_name RENAME TO new\_name  
   c) EXEC sp\_rename 'old\_name', 'new\_name'  
   d) MODIFY TABLE RENAME
5. **What is the command to delete a table permanently?**  
   a) DELETE TABLE table\_name  
   b) ERASE TABLE table\_name  
   c) DROP TABLE table\_name  
   d) REMOVE TABLE table\_name

**Section 3: DML - Manipulating Data**

1. **Which command adds data into a table?**  
   a) INSERT INTO  
   b) ADD ROW  
   c) CREATE DATA  
   d) APPEND TO
2. **Which clause is used to update data in a table?**  
   a) MODIFY  
   b) UPDATE  
   c) CHANGE  
   d) SET TABLE
3. **What does the DELETE statement do?**  
   a) Removes a column  
   b) Removes all data from a table  
   c) Removes specific rows  
   d) Deletes the table schema
4. **Which clause is used to filter rows in a SELECT statement?**  
   a) HAVING  
   b) SELECT  
   c) WHERE  
   d) ORDER BY
5. **Which keyword ensures no duplicate records are returned?**  
   a) UNIQUE  
   b) NO\_REPEAT  
   c) DISTINCT  
   d) ONLY
6. **What does the LIKE keyword do in SQL?**  
   a) Finds exact matches  
   b) Finds pattern-based matches  
   c) Sorts records  
   d) Deletes matches
7. **Which operator is used to combine multiple conditions in a WHERE clause?**  
   a) TO  
   b) WITH  
   c) AND / OR  
   d) IF / ELSE
8. **What does the BETWEEN operator do?**  
   a) Compares text fields  
   b) Finds rows outside a range  
   c) Filters values within a range  
   d) Joins tables

**DAY 3:- TASK: 4:-**

**Managing Databases**

1)List all system databases in SQL Server

select name from sys.databases where database\_id <= 4;

2. List physical file paths for all databases

select name,physical\_name from sys.master\_files;

3. Create a new user-defined database named TeamDB

create database TeamDB;

4. Rename the database TeamDB to ProjectDB

alter database TeamDB modify name=ProjectDB;

5. Drop the ProjectDB database

drop database ProjectDB;

**Managing Tables**

1. Create a table Employees with the following columns

create table Employees(EmpID int primary key,Name varchar(50),Department varchar(30),JoiningDate date,IsActive bit,Salary decimal(10,2));

2. Add a column Salary (DECIMAL) to the table

alter table Employees add Salary decimal(10,2);

3. Rename table Employees to TeamMembers

exec sp\_rename 'Employees','TeamMembers';

4. Drop the table TeamMembers.

drop table TeamMembers;

**DML Operations**

1. Insert three rows into Employees

insert into Employees values(1,'Amit','HR','2022-01-01',1,50000),(2,'Sneha','IT','2021-06-15',1,75000),(3,'John','Finance','2020-10-10',0,65000);

2. Update salary of 'Sneha' to 80000

update Employees set Salary=80000 where Name='Sneha';

3. Delete employee with IsActive = 0

delete from Employees where IsActive=0;

4. Retrieve names and departments of all employees.

select Name,Department from Employees;

5. Fetch employees from 'IT' department with salary above 70000

select \* from Employees where Department='IT' and Salary>70000;

6. Apply filtering using LIKE, BETWEEN, and IN

select \* from Employees where Name like 'S%';

select \* from Employees where Salary between 60000 and 80000;

select \* from Employees where Department in('IT','Finance');

**DAY 4:- ASSIGNMENT**

1. Insert and Update with Integrity: Create a 'students' table with constraints (NOT NULL, UNIQUE). Insert 5 records. Then, update a student's marks ensuring data integrity is maintained.

QUERY:-

create table students(student\_id int primary key,name varchar(100) not null,email varchar(100) unique not null,marks int check(marks>=0 and marks<=100));

insert into students(student\_id,name,email,marks) values(1,'john','john@gmail.com',85),(2,'alice','alice@gmail.com',92),(3,'bob','bob@gmail.com',76),(4,'neha','neha@gmail.com',88),(5,'rahul','rahul@gmail.com',67);

update students set marks=95 where student\_id=3;

select \* from students;

2. String Function Challenge: Given a 'customers' table with a 'full\_name' column, write a query to display: - First name - Last name - Length of each name

QUERY:-

create table customers(customer\_id int primary key,full\_name varchar(100));

insert into customers(customer\_id,full\_name) values(1,'kavya dharshni'),(2,'theshna shree');

select substring\_index(full\_name,' ',1) as first\_name,

substring\_index(full\_name,' ',-1) as last\_name,

length(substring\_index(full\_name,' ',1)) as first\_name\_length,

length(substring\_index(full\_name,' ',-1)) as last\_name\_length

from customers;

3. Date Function Usage: From a 'sales' table with a 'sale\_date' column, write a query to: - Extract the month name and year - Display how many days ago the sale happened

QUERY:-

create table sales(sale\_id int primary key,sale\_date date);

insert into sales(sale\_id,sale\_date) values(1,'2024-06-01'),(2,'2025-06-10');

select sale\_date, monthname(sale\_date) as month\_name, year(sale\_date) as year,

datediff(curdate(),sale\_date) as days\_ago from sales;

4. Mathematical Functions on Salary: In an 'employees' table, calculate: - Salary after a 10% hike - Round the salary to the nearest hundred

QUERY:-

create table employees(emp\_id int primary key,name varchar(50),salary decimal(10,2));

insert into employees(emp\_id,name,salary) values(1,'YUVA',43250),(2,'THESHI',51890);

select emp\_id,name,salary,salary\*1.10 as hiked\_salary,round(salary,-2) as rounded\_salary

from employees;

5. System Function Check: Retrieve: - Current date and time - Database name and logged-in user

QUERY:-

Select now() as currentdatetime,current\_date() as currentdate,current\_time() ascurrenttime,

database() as currentdatabase,user() as loggedinuser;

6. Demo: Custom Result Set: From the 'products' table, write a query that: - Returns product name in uppercase - Replaces any NULL prices with 'Not Available'

QUERY:-

create table products(productid int primary key,productname varchar(50),price decimal(10,2));

insert into products(productid,productname,price) values(1,'pen',25.50),(2,'notebook',null);

select upper(productname) as productname, ifnull(cast(price as char),'Not Available') as price from products;

7. Aggregate Functions Practice: From a 'transactions' table, get: - Total sales - Average sale value - Maximum and minimum sale on a single transaction

QUERY:-

Select sum(amount) as totalsales, avg(amount) as averagesale,max(amount) as maxsale,

min(amount) as minsale from transactions;

8. Grouping with Aggregation: From a 'sales' table: - Group by product category - Show total sales and number of transactions in each category

QUERY:-

Select category,sum(amount) as totalsales,count(\*) as transactioncount from salesgroup by category;

9. Inner Join for Orders and Customers: Join 'orders' and 'customers' to show: - Customer name - Order amount - Only for customers who made orders

create table customers(id int primary key,name varchar(50));

create table orders(id int primary key,customerid int,amount decimal(10,2),foreign key(customerid) references customers(id));

insert into customers(id,name) values(1,'kavya'),(2,'theshna'),(3,'aarav');

insert into orders(id,customerid,amount) values(1,1,250.00),(2,2,500.00);

select c.name as customername,o.amount as orderamount from customers c inner join orders o on c.id=o.customerid;

10. Left Join for Products with or without Orders: Show all products with: - Their order details (if available) - Use LEFT JOIN

QUERY:-

Select p.productname, o.orderid,o.quantity,o.orderdate from products pleft join orders o on p.productid=o.productid;

11. Right Join for Customer Contacts: Use a RIGHT JOIN between 'contacts' and 'customers' to display: - All customers, even if they don''t have contact info

QUERY:-

Select c.name as customername,ct.phone as contactphone,ct.email as contactemail

from contacts ct right join customers c on ct.customerid=c.id;

12. Full Outer Join for Suppliers and Products: Use a FULL OUTER JOIN to list: - All suppliers and products - Match supplier to product, or show NULLs where not available

QUERY:-

Select s.suppliername, p.productname from suppliers s left join products p on s.id = p.supplierid union

Select s.suppliername,p.productname from suppliers s right join products p on s.id = p.supplierid;

13. Cross Join for Offers: Suppose you have tables 'products' and 'offers'. Write a CROSS JOIN to show: - All possible combinations of products and offers

QUERY:-

Select p.productname,o.offername from products p cross join offers o;

14. Join with Aggregation: Join 'orders' and 'products', then group by product category and: - Show total quantity sold and average price per

QUERY:-

Select p.category,sum(o.quantity) as totalquantity,avg(p.price) as averageprice

from orders o join products p on o.productid = p.id group by p.category;

15. . Demo: Join with Grouping and Filter: Join 'students' and 'marks' tables. Display: - Student name - Average marks - Filter to show only students with average marks > 75

QUERY:-

Select s.name as studentname,avg(m.mark) as averagemark from students s

join marks m on s.id = m.studentid group by s.name having avg(m.mark) > 75;

**Day:-5**

1) Querying Data by Using Subqueries  
select name from studentt where marks=(select max(marks) from studentt);

2) Querying Data by Using Subqueries Using the EXISTS,

select name from studentt s where exists(select 1 from studentt where grade=s.grade and marks>85);

3) select name from studentt where marks>any(select marks from studentt where grade='b');

select name from studentt where marks>any(select marks from studentt where grade='b');

4) Querying Data by Using Subqueries using ALL Keywords

select name from studentt where marks>all(select marks from studentt where grade='c');

5) Querying Data by Using Subqueries using Using Nested Subqueries

select name from studentt where marks=(select max(marks) from studentt where marks<(select max(marks) from studentt));

6) Querying Data by Using Subqueries Using Correlated Subqueries

select name from studentt s where marks>(select avg(marks) from studentt where grade=s.grade);

7) Querying Data by Using Subqueries Using UNION,

select name from studentt where marks>85 union select name from studentt where grade='c';

8) Querying Data by Using Subqueries using INTERSECT

select name from studentt where marks>70 intersect select name from studentt where grade='b';

9) Querying Data by Using Subqueries using EXCEPT,

select name from studentt where marks>70 except select name from studentt where grade='b';

10) Querying Data by Using Subqueries using MERGE

merge into studentt as t using(select 6 as student\_id,'gokul' as name,78 as marks,'b' as grade) as s on(t.student\_id=s.student\_id) when matched then update set t.marks=s.marks when not matched then insert(student\_id,name,marks,grade) values(s.student\_id,s.name,s.marks,s.grade);

# **DAY 5:- TASK 2:-**

# Section A: Basics & Data Definition (10 Marks)

Q1. (3 marks)

Differentiate between SQL and NoSQL. Provide two advantages and two disadvantages of each with real-world examples.

|  |  |
| --- | --- |
| SQL | NOSQL |
| It has data in the rows and column format | It doesn’t have any structure to organize the data |
| The schema is in fixed type | The schema is flexible |
| It uses sql languages | It uses MongoDB which is applicable for large data |

Realtime example for SQL: banking system, College management

Realtime example for NoSQL: social media platforms like facebook,instagarm.

Q2. (2 marks)

Given the below unnormalized data, convert it to 1NF, 2NF, and 3NF:

Student (StudentID,Name,CourseID,CourseName,InstructorName,InstructorPhone)

1Nf- all the data should be atomic

Student(StudentID,Name,CourseID,CourseName,InstructorName,InstructorPhone)

2NF- should be one nF but doesn’t have partial dependency

Student(StudentID,Name)) Course(CourseID,CourseName,InstructorName,InstructorPhone) Enrollment(StudentID, CourseID)

3NF- must be 2NF but does not hava transitive dependencies

Student(StudentID,Name) Course(CourseID,CourseName,InstructorName) Instructor(InstructorName,InstructorPhone) Enrollment(StudentID,CourseID)

Q3. (5 marks)

1. Create a database named StudentDB. create database studentdb;

create database studentdb;

1. Create a table Students with fields: StudentID, Name, DOB, Email.

create table students(studentid int,name varchar(50),dob date,email varchar(100));

1. Rename the table to Student\_Info.

rename table students to student\_info;

1. Add a column PhoneNumber.

alter table student\_info add phonenumber varchar(15);

1. Drop the table.

drop table student\_info;

# Section B: DML & Filtering Data (15 Marks)

Q4. (5 marks)

1. Insert 3 student records into Student\_Info.

insert into student\_info(studentid,name,dob,email,phonenumber) values

(1,'yuva','2003-05-14','yuva@gmail.com','9876543210'),

(2,'theshi','2002-11-22','theshi@gmail.com','8765432109'),

(3,'kavya','2004-03-30','kavya@gmail.com','7654321098');

1. Update one student's phone number.

update student\_info set phonenumber='9123456780' where studentid=1;

1. Delete one student whose email ends with @gmail.com.

delete from student\_info where email like '%@gmail.com' limit 1;

1. Retrieve only names and emails of students born after the year 2000.

select name,email from student\_info where year(dob)>2000;

1. Retrieve distinct domain names from the email column.

select distinct substring\_index(email,'@',-1) as domain from student\_info;

Q5. (5 marks)

1. Retrieve students with names starting with 'A'.

select \* from student\_info where name like 'a%';

1. Retrieve students with phone number between 9000000000 and 9999999999.

select \* from student\_info where phonenumber between “9000000000” and “9999999999”;

1. Retrieve students using IN operator on city names.

select \* from student\_info where city in('chennai','Namakkal','coimbatore');

1. Use AND, OR to filter students based on age and email provider.

select \* from student\_info where (year(dob)>2000 and email like '%@gmail.com') or (year(dob)<2000 and email like '%@yahoo.com');

1. Use table and column aliasing in a query to get all student names and DOBs.

select s.name as student\_name,s.dob as birth\_date from student\_info as s;

Q6. (5 marks)

a)Create a new table Marks(StudentID, Subject, Marks). Insert at least 3 rows.

create table marks(studentid int,subject varchar(50),marks int);

insert into marks(studentid,subject,marks) values

(1,”maths”,85),

(2,”science”,90),

(3,”english”,78);

b)Display student IDs and their subjects where marks > 70.

select studentid,subject from marks where marks>70;

c)Display subjects with average marks.

select subject,avg(marks) as average\_marks from marks group by subject;

d)Filter subjects with average marks between 60 and 90.

select subject,avg(marks) as average\_marks from marks group by subject having avg(marks) between 60 and 90;

# Section C: Functions & Grouping (10 Marks)

Q7. (5 marks)

1. Get the current date and format it as "YYYY-MM-DD".

select date\_format(curdate(),'%Y-%m-%d') as todaydate;

1. Extract month and year from a DOB column.

select month(dob) as birth\_month,year(dob) as birth\_year from student\_info;

1. Convert a student's name to uppercase.

select upper(name) as uppercase\_name from student\_info;

1. Round off marks to 2 decimal places.

select round(marks,2) as rounded\_marks from marks;

1. Use system function to return user name or current database.

select user() as current\_user,database() as currentdatabase;

Q8. (5 marks)

1. Display total marks of each student.

select studentid,sum(marks) as totalmarks from marks group by studentid;

1. Display subject-wise highest mark.

select subject,max(marks) as highest\_mark from marks group by subject;

1. Use GROUP BY and HAVING to display subjects with average marks > 75.

select subject,avg(marks) as average\_marks from marks group by subject having avg(marks)>75;

# Section D: Joins and Subqueries (25 Marks)

Q9. (5 marks)

create table courses(courseid int,course\_name varchar(50));

create table enrollments(studentid int,courseid int);

insert into courses(courseid,course\_name) values(101,'maths'),(102,'science'),(103,'english');

insert into enrollments(studentid,courseid) values(1,101),(2,102);

1. Inner Join to retrieve students and their courses.

select s.studentid,s.name,c.course\_name from student\_info s inner join enrollments e on s.studentid=e.studentid

inner join courses c on e.courseid=c.courseid;

1. Left Join to get all students even if not enrolled.

select s.studentid,s.name,c.course\_name from student\_info s left join enrollments e on s.studentid=e.studentid

left join courses c on e.courseid=c.courseid;

1. Right Join to get all courses even if no students.

select s.studentid,s.name,c.course\_name from student\_info s right join enrollments e on s.studentid=e.studentid

right join courses c on e.courseid=c.courseid;

1. Full Outer Join equivalent using UNION.

select s.studentid,s.name,c.course\_name from student\_info s left join enrollments e on s.studentid=e.studentid

left join courses c on e.courseid=c.courseid union select s.studentid,s.name,c.course\_name

from student\_info s right join enrollments e on s.studentid=e.studentid

right join courses c on e.courseid=c.courseid;

1. Cross Join to show all combinations.

select s.studentid,s.name,c.course\_name from student\_info s cross join courses c;

Q10. (5 marks)

1. Students who scored more than average in 'Maths'.

select studentid,marks from marks where subject='maths' and marks > (select avg(marks) from marks where subject='maths');

1. Students not in the Marks table.

select \* from student\_info where studentid not in (select distinct studentid from marks);

1. Use EXISTS to get students with at least one subject.

select \* from student\_info s where exists ( select 1 from marks m where m.studentid=s.studentid);

1. Use ALL to find those scoring more than all in 'Science'.

select \* from marks where marks > all (select marks from marks where subject='science');

1. Use ANY for students scoring better than some in 'English'.

select \* from marks where marks > any (select marks from marks where subject='english');

Q11. (5 marks)

create table student2(studentid int,name varchar(50));

insert into student2(studentid,name) values (5,”arjun”),(6,”divya”),(3,”kavya”);

1. UNION of student names from two tables.

select name from student\_info union select name from student2;

1. INTERSECT to find common students.

select name from student\_info where name in(select name from student2);

1. EXCEPT to list students in Students but not in Marks.

select \* from student\_info where studentid not in(select studentid from marks);

1. MERGE concept or simulate with UPDATE and INSERT.

update student\_info set email='yuvanewmail@gmail.com' where studentid=1;

insert into student\_info(studentid,name,dob,email,phonenumber)

select 4,'newstudent','2005-01-01','new@gmail.com','9123456789';

where not exists(select 1 from student\_info where studentid=4);

1. Correlated subquery to list students with above average per subject

select \* from marks m1 where marks>(select avg(marks) from marks m2 where m2.subject=m1.subject);

# Section A: Advanced Concepts & Schema Design (10 Marks)

Q1. (4 marks)

Explain with examples the scenarios where NoSQL is preferred over SQL. Discuss types of NoSQL databases and suggest a real-time application for each.

NOSQL always stores unstructured data while sql has fixed schema to store the data . NOSQL can handle large amount of data than sql. NOSQL need not to be predefined it can be changed at any time.

Types of Nosql : Document-oriented,keyvalue stores,column-oriented, graph databases

Realtime application:-sql:- ERP(oracle) NOSQL:-social media plateforms

Q2. (6 marks)

A retail store keeps the following unnormalized record:

Customer (CustomerID, Name, Orders (OrderID, ProductID, Quantity, ProductName)) Normalize the data up to BCNF with appropriate table structures.

1nf:-

Customer(CustomerID, Name) OrderDetails(OrderID, CustomerID, ProductID, Quantity, ProductName)

2nf:-

Customer(CustomerID, Name) Orders(OrderID, CustomerID) OrderItems(OrderID, ProductID, Quantity) Product(ProductID, ProductName)

BCNF:-

Customer(CustomerID, Name) Orders(OrderID, CustomerID) OrderItems(OrderID, ProductID, Quantity) Product(ProductID, ProductName)

# Section B: Complex DDL and DML (15 Marks)

Q3. (5 marks)

1. Create a database RetailDB and design a schema for Customers, Orders, and Products with primary and foreign keys.

create database retaildb;

use retaildb;

create table customers(customerid int primary key,name varchar(100),email varchar(100)); create table products(productid int primary key,productname varchar(100),price decimal(10,2));

create table orders(orderid int primary key,customerid int,productid int,quantity int,orderdate date,foreign key(customerid) references customers(customerid),foreign key(productid) references products(productid));

1. Implement a check constraint on Quantity (>0) in Orders.

alter table orders add constraint chk\_quantity check(quantity > 0);

1. Alter the Products table to add 'Discount' column and update some values.

alter table products add discount decimal(5,2); update products set discount=10.00 where productid=1; update products set discount=5.00 where productid=2;

Q4. (5 marks)

Using the above schema:

1. Insert 3 sample orders per customer.

insert into orders(orderid,customerid,productid,quantity,orderdate) values(1,1,1,2,'2025-06-10'),(2,1,2,4,'2025-06-11'),(3,1,3,1,'2025-06-12'),(4,2,1,6,'2025-06-10'),(5,2,2,3,'2025-06-11'),(6,2,3,8,'2025-06-12');

1. Update prices with 10% increase where quantity sold > 5.

update products set price=price\*1.10 where productid in(select productid from orders group by productid having sum(quantity)>5);

1. Delete orders where the product has never been sold.

delete from orders where productid not in(select distinct productid from orders);

Q5. (5 marks) Retrieve the following:

1. Customers who ordered more than 3 different products.

select customerid from orders group by customerid having count(distinct productid)>3;

1. Products not ordered by any customer.

select \* from products where productid not in(select distinct productid from orders);

1. Count of orders placed by each customer in the last 30 days.

select customerid,count(\*) as order\_count from orders where orderdate >= curdate() - interval 30 day group by customerid;

# Section C: Advanced Functions and Aggregations (10 Marks)

Q6. (5 marks)

1. Use string functions to standardize and extract parts from customer email IDs.

select customerid,lower(email) as standardized\_email,substring\_index(email,'@',1) as username,substring\_index(email,'@',-1) as domain from customers;

1. Use date functions to compute days between order date and today.

select orderid,datediff(curdate(),orderdate) as days\_since\_order from orders;

1. Use system functions to return current user and host.

select user(),current\_user(),version();

1. Use nested functions to format a customer greeting string.

select concat('hello ',upper(left(name,1)),lower(substring(name,2))) as greeting from customers;

Q7. (5 marks)

alter table products add category varchar(50);

1. Aggregate total revenue by product category.

select p.category,sum(p.price\*o.quantity) as total\_revenue from products p join orders o on p.productid=o.productid group by p.category;

1. Use GROUP BY with ROLLUP to compute subtotal and grand total sales.

select p.category,sum(p.price\*o.quantity) as total\_revenue from products p join orders o on p.productid=o.productid group by p.category with rollup;

1. Use HAVING clause to filter categories with revenue > 100000.

select p.category,sum(p.price\*o.quantity) as total\_revenue from products p join orders o on p.productid=o.productid group by p.category having total\_revenue>100000;

# Section D: Complex Joins, Subqueries, and Set Ops (25 Marks)

Q8. (5 marks)

1. Self join to list customers referred by other customers.

select c1.customerid as referred\_customer,c1.name as customer\_name,c2.name as referrer\_name from customers c1 join customers c2 on c1.referred\_by=c2.customerid;

1. Equi join across Orders and Products.

select o.orderid,o.customerid,p.productname,o.quantity from orders o join products p on o.productid=p.productid;

1. Join Customers and Orders to display top 3 spenders using window function.

select customerid,name,total\_spent from (select c.customerid,c.name,sum(p.price\*o.quantity) as total\_spent,row\_number() over(order by sum(p.price\*o.quantity) desc) as rank from customers c join orders o on c.customerid=o.customerid join products p on o.productid=p.productid group by c.customerid,c.name) as ranked where rank<=3;

1. LEFT OUTER JOIN with WHERE NULL to identify inactive customers.

select c.customerid,c.name from customers c left join orders o on c.customerid=o.customerid where o.customerid is null;

1. Cross join for all product combinations in a bundle offer.

select p1.productname as product1,p2.productname as product2 from products p1 cross join products p2 where p1.productid<p2.productid;

Q9. (5 marks)

1. Correlated subquery to get customers whose order amount exceeds their average.

select o.customerid,o.orderid,sum(p.price\*o.quantity) as order\_total from orders o join products p on o.productid=p.productid group by o.orderid,o.customerid having order\_total > (select avg(p2.price\*o2.quantity) from orders o2 join products p2 on o2.productid=p2.productid where o2.customerid=o.customerid);

1. Subquery using EXISTS to find customers with at least 2 different products.

select \* from customers c where exists(select 1 from orders o where o.customerid=c.customerid group by o.customerid having count(distinct o.productid)>=2);

1. Use ALL to find customers who ordered more than every other customer.

select customerid from (select customerid,count(\*) as order\_count from orders group by customerid) as t where order\_count > all(select count(\*) from orders group by customerid);

1. Use ANY to find products costlier than some in category 'Electronics'.

select \* from products where price > any(select price from products where category='electronics');

1. Nested subquery to list top 3 best-selling products.

select productid,productname,total\_sold from (select p.productid,p.productname,sum(o.quantity) as total\_sold,row\_number() over(order by sum(o.quantity) desc) as rk from products p join orders o on p.productid=o.productid group by p.productid,p.productname) as ranked where rk<=3;

Q10. (5 marks)

1. Simulate INTERSECT using INNER JOIN on two customer segments.

select cn.customerid,cn.name from customer\_north cn inner join customer\_south cs on cn.customerid=cs.customerid;

1. Use EXCEPT to find products in inventory not yet ordered.

select \* from products where productid not in(select distinct productid from orders);

1. Simulate MERGE: If customer exists, update; else insert.

update customers set email='newemail@gmail.com' where customerid=4; insert into customers(customerid,name,email) select 4,'newcustomer','new@gmail.com' where not exists(select 1 from customers where customerid=4);

1. Use UNION to combine two regional customer tables.

Write a WITH CTE that ranks customers by total spend and filters

select customerid,name,email from customer\_north union select customerid,name,email from customer\_south;

Day 6:

# MySQL MCQ Quiz

Duration: 45 Minutes

1. Q1. What is a key characteristic of SQL vs NoSQL?

* A. SQL vs NoSQL ensures data duplication
* B. SQL vs NoSQL is used only in NoSQL databases
* C. SQL vs NoSQL improves data integrity
* D. SQL vs NoSQL is not related to database design

1. Q2. What is a key characteristic of Advantages of SQL?

* A. Advantages of SQL ensures data duplication
* B. Advantages of SQL is used only in NoSQL databases
* C. Advantages of SQL improves data integrity
* D. Advantages of SQL is not related to database design

1. Q3. What is a key characteristic of Disadvantages of SQL?

* A. Disadvantages of SQL ensures data duplication
* B. Disadvantages of SQL is used only in NoSQL databases
* C. Disadvantages of SQL improves data integrity
* D. Disadvantages of SQL is not related to database design

1. Q4. What is a key characteristic of System Databases in SQL Server?

* A. System Databases in SQL Server ensures data duplication
* B. System Databases in SQL Server is used only in NoSQL databases
* C. System Databases in SQL Server improves data integrity
* D. System Databases in SQL Server is not related to database design

1. Q5. What is a key characteristic of Managing Databases?

* A. Managing Databases ensures data duplication
* B. Managing Databases is used only in NoSQL databases
* C. Managing Databases improves data integrity
* D. Managing Databases is not related to database design

1. Q6. What is a key characteristic of 1NF?

* A. 1NF ensures data duplication
* B. 1NF is used only in NoSQL databases
* C. 1NF improves data integrity
* D. 1NF is not related to database design

1. Q7. What is a key characteristic of 2NF?

* A. 2NF ensures data duplication
* B. 2NF is used only in NoSQL databases
* C. 2NF improves data integrity
* D. 2NF is not related to database design

1. Q8. What is a key characteristic of 3NF?

* A. 3NF ensures data duplication
* B. 3NF is used only in NoSQL databases
* C. 3NF improves data integrity
* D. 3NF is not related to database design

1. Q9. What is a key characteristic of BCNF?

* A. BCNF ensures data duplication
* B. BCNF is used only in NoSQL databases
* C. BCNF improves data integrity
* D. BCNF is not related to database design

1. Q10. What is a key characteristic of Identifying System Databases?

* A. Identifying System Databases ensures data duplication
* B. Identifying System Databases is used only in NoSQL databases
* C. Identifying System Databases improves data integrity
* D. Identifying System Databases is not related to database design

1. Q11. What is a key characteristic of Database Files?

* A. Database Files ensures data duplication
* B. Database Files is used only in NoSQL databases
* C. Database Files improves data integrity
* D. Database Files is not related to database design

1. Q12. What is a key characteristic of Creating Databases?

* A. Creating Databases ensures data duplication
* B. Creating Databases is used only in NoSQL databases
* C. Creating Databases improves data integrity
* D. Creating Databases is not related to database design

1. Q13. What is a key characteristic of Renaming Databases?

* A. Renaming Databases ensures data duplication
* B. Renaming Databases is used only in NoSQL databases
* C. Renaming Databases improves data integrity
* D. Renaming Databases is not related to database design

1. Q14. What is a key characteristic of Dropping Databases?

* A. Dropping Databases ensures data duplication
* B. Dropping Databases is used only in NoSQL databases
* C. Dropping Databases improves data integrity
* D. Dropping Databases is not related to database design

1. Q15. What is a key characteristic of Data Types?

* A. Data Types ensures data duplication
* B. Data Types is used only in NoSQL databases
* C. Data Types improves data integrity
* D. Data Types is not related to database design

1. Q16. What is a key characteristic of Creating Tables?

* A. Creating Tables ensures data duplication
* B. Creating Tables is used only in NoSQL databases
* C. Creating Tables improves data integrity
* D. Creating Tables is not related to database design

1. Q17. What is a key characteristic of Modifying Tables?

* A. Modifying Tables ensures data duplication
* B. Modifying Tables is used only in NoSQL databases
* C. Modifying Tables improves data integrity
* D. Modifying Tables is not related to database design

1. Q18. What is a key characteristic of Renaming Tables?

* A. Renaming Tables ensures data duplication
* B. Renaming Tables is used only in NoSQL databases
* C. Renaming Tables improves data integrity
* D. Renaming Tables is not related to database design

1. Q19. What is a key characteristic of Dropping Tables?

* A. Dropping Tables ensures data duplication
* B. Dropping Tables is used only in NoSQL databases
* C. Dropping Tables improves data integrity
* D. Dropping Tables is not related to database design

1. Q20. What is a key characteristic of Insert/Update/Delete?

* A. Insert/Update/Delete ensures data duplication
* B. Insert/Update/Delete is used only in NoSQL databases
* C. Insert/Update/Delete improves data integrity
* D. Insert/Update/Delete is not related to database design

1. Q21. What is a key characteristic of Retrieving Data?

* A. Retrieving Data ensures data duplication
* B. Retrieving Data is used only in NoSQL databases
* C. Retrieving Data improves data integrity
* D. Retrieving Data is not related to database design

1. Q22. What is a key characteristic of Filtering: WHERE, IN, AND, OR, LIKE?

* A. Filtering: WHERE, IN, AND, OR, LIKE ensures data duplication
* B. Filtering: WHERE, IN, AND, OR, LIKE is used only in NoSQL databases
* C. Filtering: WHERE, IN, AND, OR, LIKE improves data integrity
* D. Filtering: WHERE, IN, AND, OR, LIKE is not related to database design

1. Q23. What is a key characteristic of Aliases?

* A. Aliases ensures data duplication
* B. Aliases is used only in NoSQL databases
* C. Aliases improves data integrity
* D. Aliases is not related to database design

1. Q24. What is a key characteristic of DISTINCT?

* A. DISTINCT ensures data duplication
* B. DISTINCT is used only in NoSQL databases
* C. DISTINCT improves data integrity
* D. DISTINCT is not related to database design

1. Q25. What is a key characteristic of BETWEEN?

* A. BETWEEN ensures data duplication
* B. BETWEEN is used only in NoSQL databases
* C. BETWEEN improves data integrity
* D. BETWEEN is not related to database design

1. Q26. What is a key characteristic of Data Integrity?

* A. Data Integrity ensures data duplication
* B. Data Integrity is used only in NoSQL databases
* C. Data Integrity improves data integrity
* D. Data Integrity is not related to database design

1. Q27. What is a key characteristic of String Functions?

* A. String Functions ensures data duplication
* B. String Functions is used only in NoSQL databases
* C. String Functions improves data integrity
* D. String Functions is not related to database design

1. Q28. What is a key characteristic of Date Functions?

* A. Date Functions ensures data duplication
* B. Date Functions is used only in NoSQL databases
* C. Date Functions improves data integrity
* D. Date Functions is not related to database design

1. Q29. What is a key characteristic of Math Functions?

* A. Math Functions ensures data duplication
* B. Math Functions is used only in NoSQL databases
* C. Math Functions improves data integrity
* D. Math Functions is not related to database design

1. Q30. What is a key characteristic of System Functions?

* A. System Functions ensures data duplication
* B. System Functions is used only in NoSQL databases
* C. System Functions improves data integrity
* D. System Functions is not related to database design

1. Q31. What is a key characteristic of Aggregate Functions?

* A. Aggregate Functions ensures data duplication
* B. Aggregate Functions is used only in NoSQL databases
* C. Aggregate Functions improves data integrity
* D. Aggregate Functions is not related to database design

1. Q32. What is a key characteristic of GROUP BY?

* A. GROUP BY ensures data duplication
* B. GROUP BY is used only in NoSQL databases
* C. GROUP BY improves data integrity
* D. GROUP BY is not related to database design

1. Q33. What is a key characteristic of Customizing Result Sets?

* A. Customizing Result Sets ensures data duplication
* B. Customizing Result Sets is used only in NoSQL databases
* C. Customizing Result Sets improves data integrity
* D. Customizing Result Sets is not related to database design

1. Q34. What is a key characteristic of Inner Join?

* A. Inner Join ensures data duplication
* B. Inner Join is used only in NoSQL databases
* C. Inner Join improves data integrity
* D. Inner Join is not related to database design

1. Q35. What is a key characteristic of Left Join?

* A. Left Join ensures data duplication
* B. Left Join is used only in NoSQL databases
* C. Left Join improves data integrity
* D. Left Join is not related to database design

1. Q36. What is a key characteristic of Right Join?

* A. Right Join ensures data duplication
* B. Right Join is used only in NoSQL databases
* C. Right Join improves data integrity
* D. Right Join is not related to database design

1. Q37. What is a key characteristic of Full Outer Join?

* A. Full Outer Join ensures data duplication
* B. Full Outer Join is used only in NoSQL databases
* C. Full Outer Join improves data integrity
* D. Full Outer Join is not related to database design

1. Q38. What is a key characteristic of Cross Join?

* A. Cross Join ensures data duplication
* B. Cross Join is used only in NoSQL databases
* C. Cross Join improves data integrity
* D. Cross Join is not related to database design

1. Q39. What is a key characteristic of GROUP BY with Joins?

* A. GROUP BY with Joins ensures data duplication
* B. GROUP BY with Joins is used only in NoSQL databases
* C. GROUP BY with Joins improves data integrity
* D. GROUP BY with Joins is not related to database design

1. Q40. What is a key characteristic of Aggregate Functions with Joins?

* A. Aggregate Functions with Joins ensures data duplication
* B. Aggregate Functions with Joins is used only in NoSQL databases
* C. Aggregate Functions with Joins improves data integrity
* D. Aggregate Functions with Joins is not related to database design

1. Q41. What is a key characteristic of Equi Join?

* A. Equi Join ensures data duplication
* B. Equi Join is used only in NoSQL databases
* C. Equi Join improves data integrity
* D. Equi Join is not related to database design

1. Q42. What is a key characteristic of Self Join?

* A. Self Join ensures data duplication
* B. Self Join is used only in NoSQL databases
* C. Self Join improves data integrity
* D. Self Join is not related to database design

1. Q43. What is a key characteristic of HAVING, GROUPING SETS?

* A. HAVING, GROUPING SETS ensures data duplication
* B. HAVING, GROUPING SETS is used only in NoSQL databases
* C. HAVING, GROUPING SETS improves data integrity
* D. HAVING, GROUPING SETS is not related to database design

1. Q44. What is a key characteristic of Subqueries?

* A. Subqueries ensures data duplication
* B. Subqueries is used only in NoSQL databases
* C. Subqueries improves data integrity
* D. Subqueries is not related to database design

1. Q45. What is a key characteristic of EXISTS, ANY, ALL?

* A. EXISTS, ANY, ALL ensures data duplication
* B. EXISTS, ANY, ALL is used only in NoSQL databases
* C. EXISTS, ANY, ALL improves data integrity
* D. EXISTS, ANY, ALL is not related to database design

1. Q46. What is a key characteristic of Nested Subqueries?

* A. Nested Subqueries ensures data duplication
* B. Nested Subqueries is used only in NoSQL databases
* C. Nested Subqueries improves data integrity
* D. Nested Subqueries is not related to database design

1. Q47. What is a key characteristic of Correlated Subqueries?

* A. Correlated Subqueries ensures data duplication
* B. Correlated Subqueries is used only in NoSQL databases
* C. Correlated Subqueries improves data integrity
* D. Correlated Subqueries is not related to database design

1. Q48. What is a key characteristic of UNION, INTERSECT, EXCEPT, MERGE?

* A. UNION, INTERSECT, EXCEPT, MERGE ensures data duplication
* B. UNION, INTERSECT, EXCEPT, MERGE is used only in NoSQL databases
* C. UNION, INTERSECT, EXCEPT, MERGE improves data integrity
* D. UNION, INTERSECT, EXCEPT, MERGE is not related to database design.

Data

-- Customers Table

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

City VARCHAR(100)

);

-- Orders Table

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

Amount DECIMAL(10,2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Products Table

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Price DECIMAL(10,2)

);

-- OrderDetails Table

CREATE TABLE OrderDetails (

OrderDetailID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

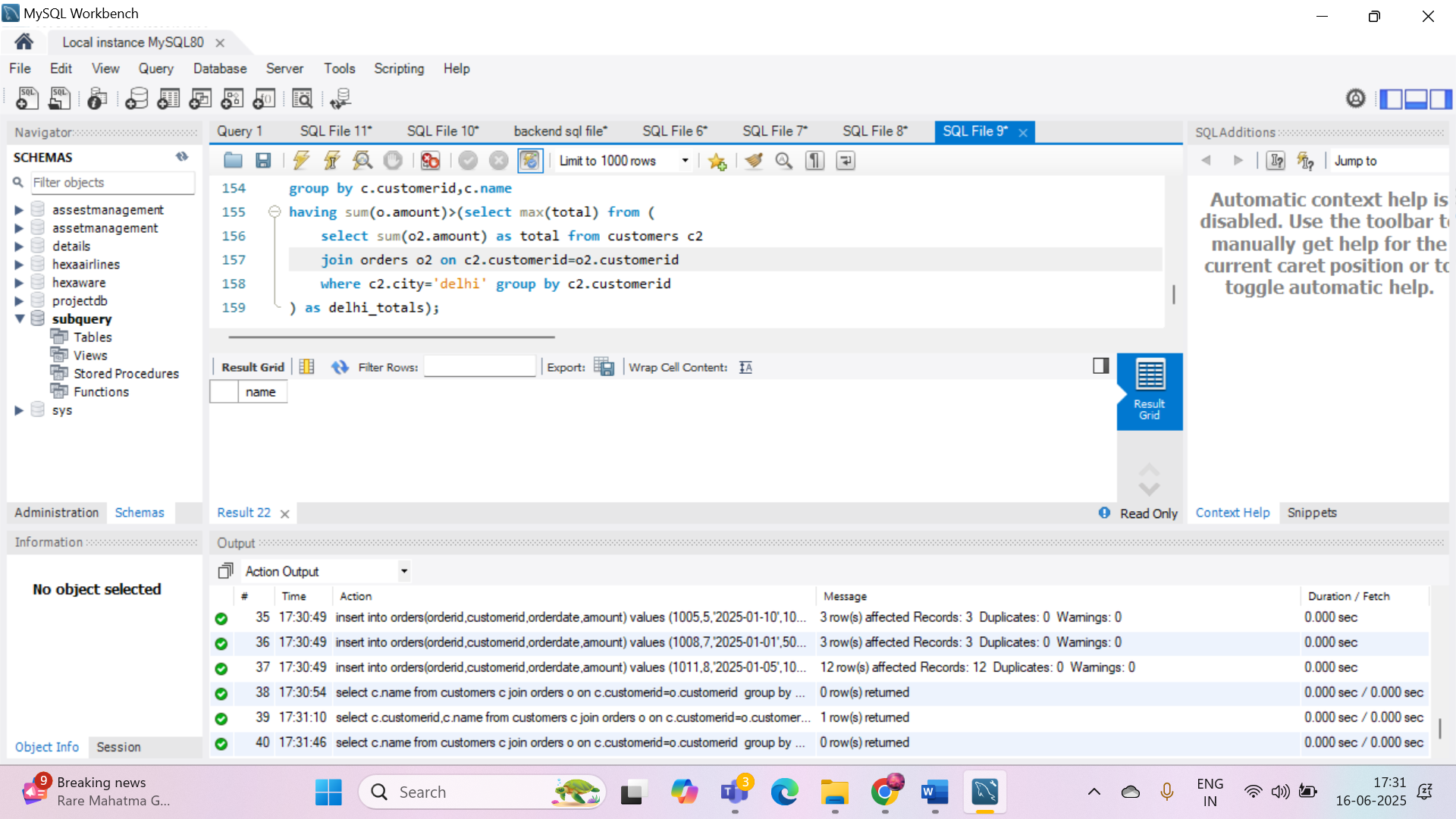
**Part A – Subqueries (20 marks)**

1. Write a query to find customers who have placed orders in **every month** of the current year.

QUERY: select c.customerid,c.name from customers c join orders o on c.customerid=o.customerid

where year(o.orderdate)=year(curdate()) group by c.customerid,c.name

having count(distinct month(o.orderdate))=12;

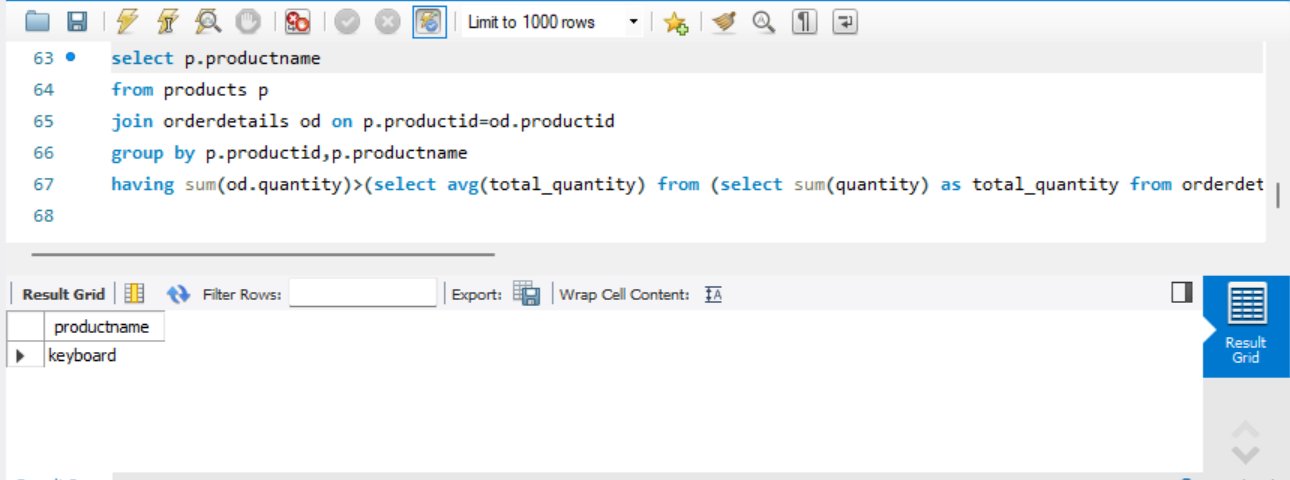


1. Retrieve the names of products that have been ordered **more than the average quantity** across all products.

QUERY: select p.productname from products p join orderdetails od on p.productid=od.productid

group by p.productid,p.productname having sum(od.quantity)>(select avg(total\_quantity) from

(select sum(quantity) as total\_quantity from orderdetails group by productid) as avg\_table);

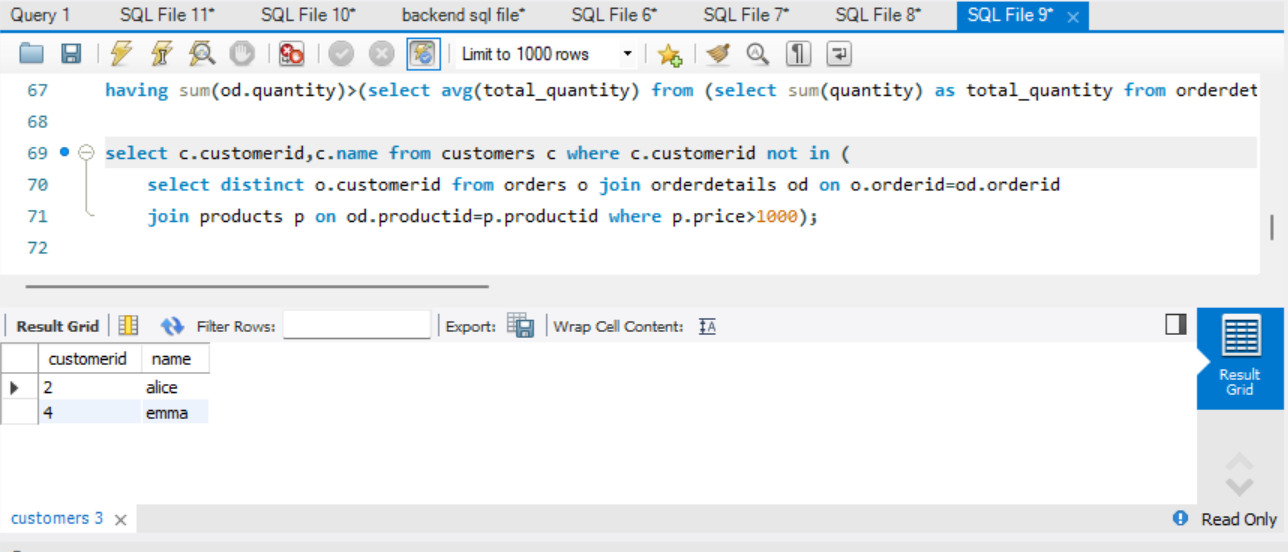


1. Find customers who have **never ordered a product** priced above ₹1000.

QUERY: select c.customerid,c.name from customers c where c.customerid not in (

select distinct o.customerid from orders o join orderdetails od on o.orderid=od.orderid

join products p on od.productid=p.productid where p.price>1000);



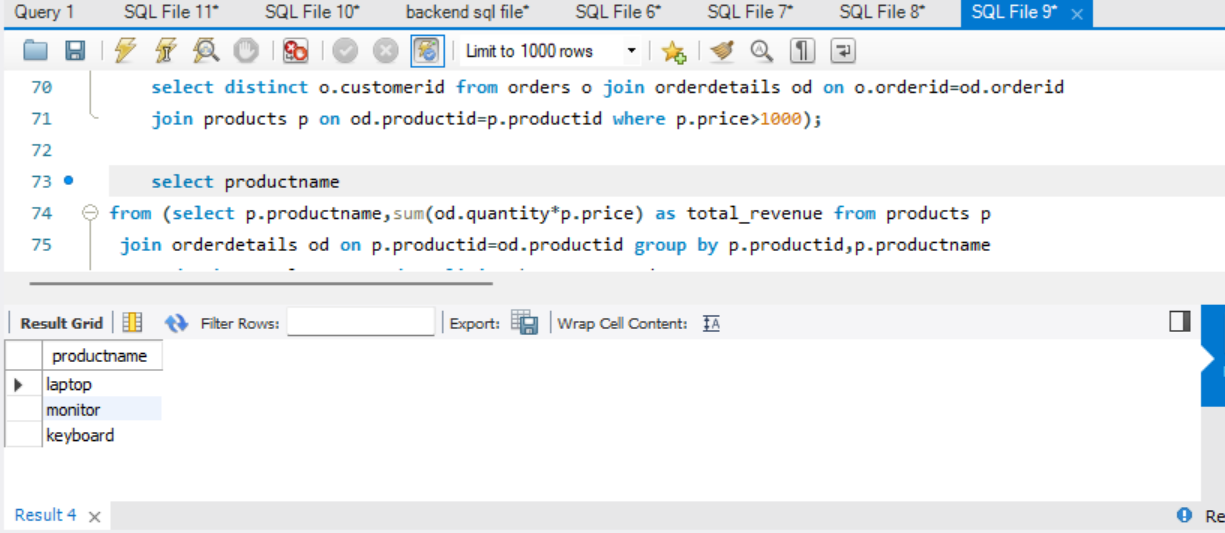
1. List the **top 3 products by total revenue** using a subquery.

QUERY: select productname

from (select p.productname,sum(od.quantity\*p.price) as total\_revenue from products p

join orderdetails od on p.productid=od.productid group by p.productid,p.productname

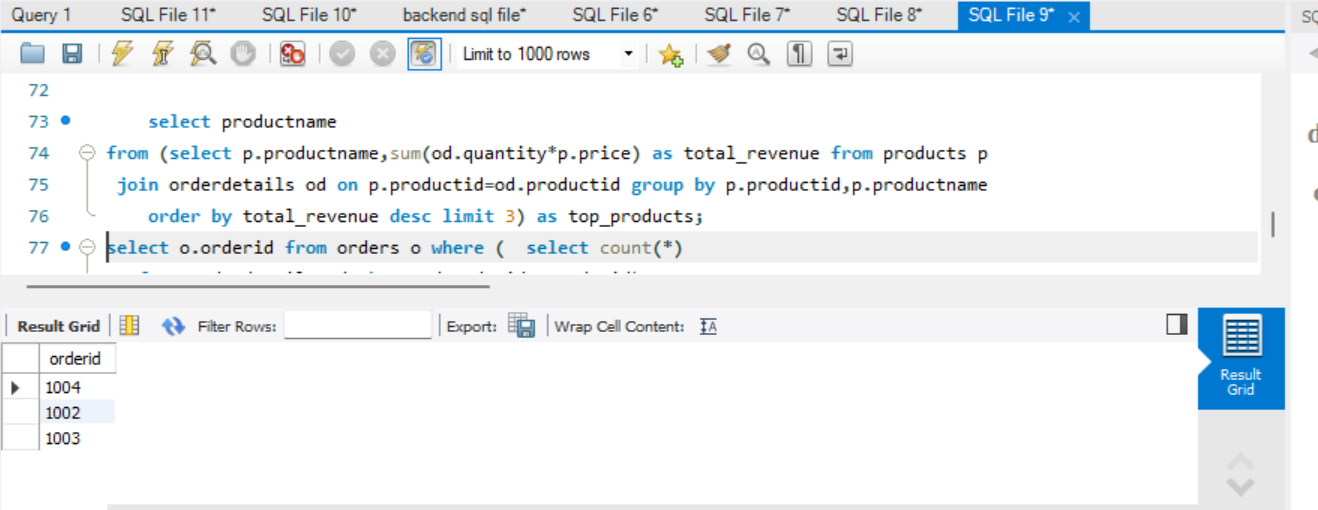
order by total\_revenue desc limit 3) as top\_products;



1. Find orders that contain **only one product** using a **correlated subquery**.

QUERY: select o.orderid from orders o where ( select count(\*)

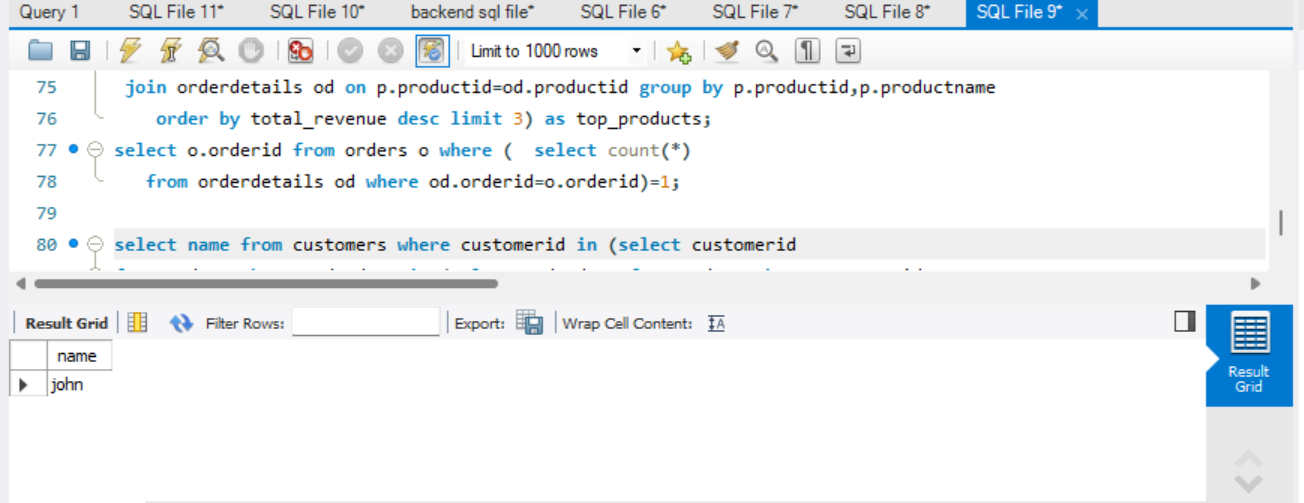
from orderdetails od where od.orderid=o.orderid)=1;



**Part B – Correlated & Nested Subqueries (25 marks)**

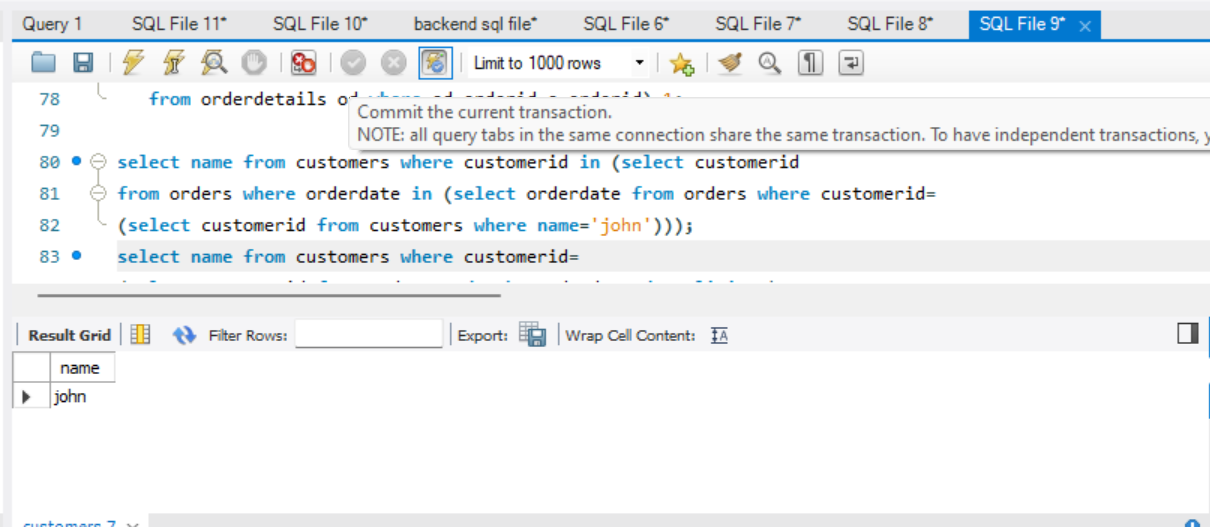
1. Retrieve the names of customers who placed an order on the **same date as 'John'**.

QUERY: select name from customers where customerid in (select customerid from orders where orderdate in (select orderdate from orders where customerid=(select customerid from customers where name='john')));



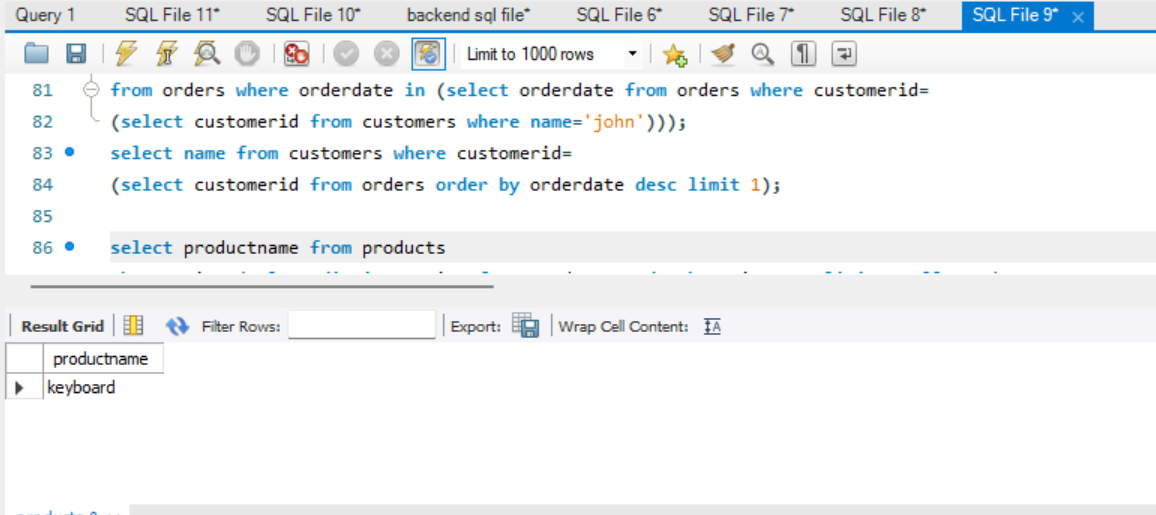
1. Find the name of the customer who placed the **most recent order**.

QUERY: select name from customers where customerid=(select customerid from orders order by orderdate desc limit 1);



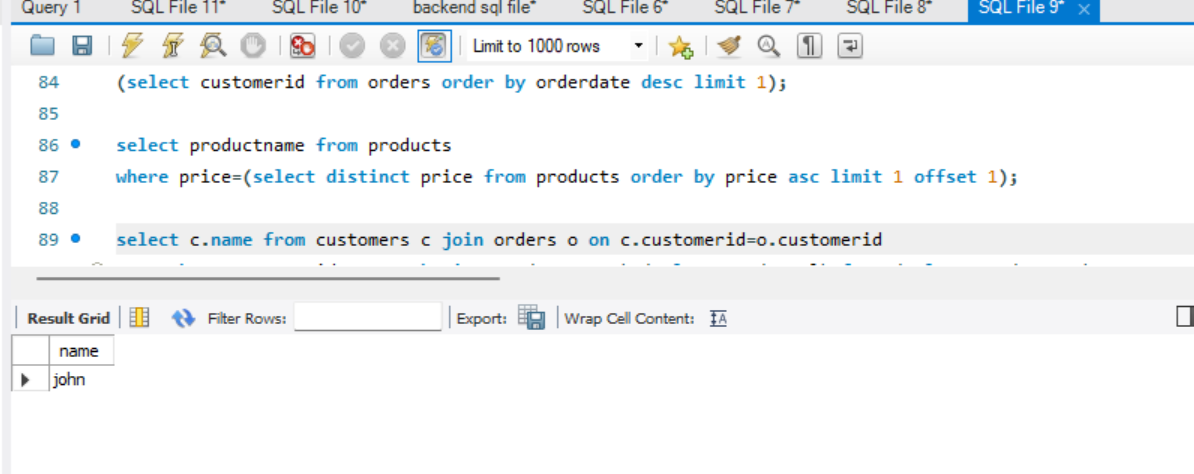
1. Write a query to find the product that has the **second lowest price** using a subquery.

QUERY: select productname from products where price=(select distinct price from products order by price asc limit 1 offset 1);



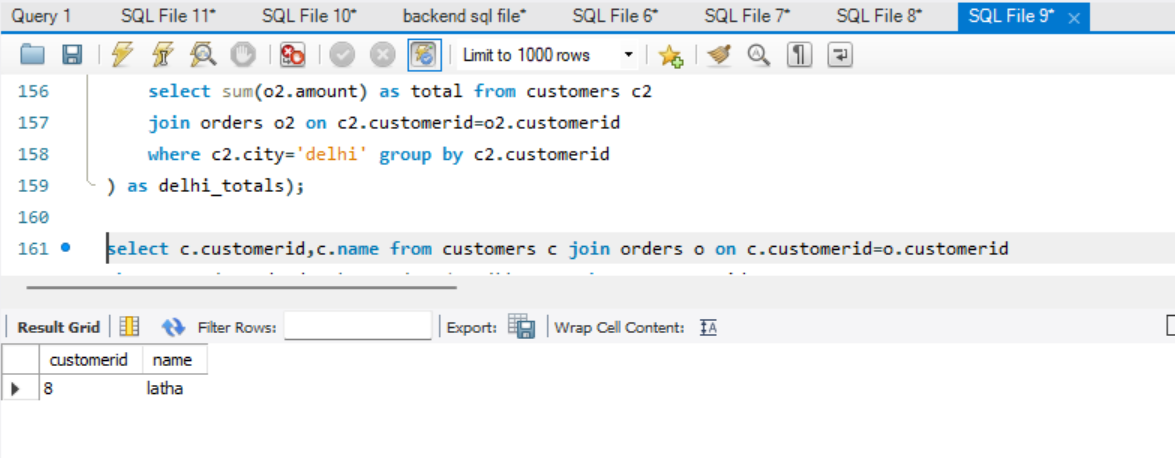
1. Display customer names who have spent **more than double the average spending**.

QUERY: select c.name from customers c join orders o on c.customerid=o.customerid group by c.customerid,c.name having sum(o.amount)>(select avg(total) from (select sum(amount) as total from orders group by customerid) as avg\_spend)\*2;



1. List customers whose **total order amount is more than the total order amount of any customer from 'Delhi'**.

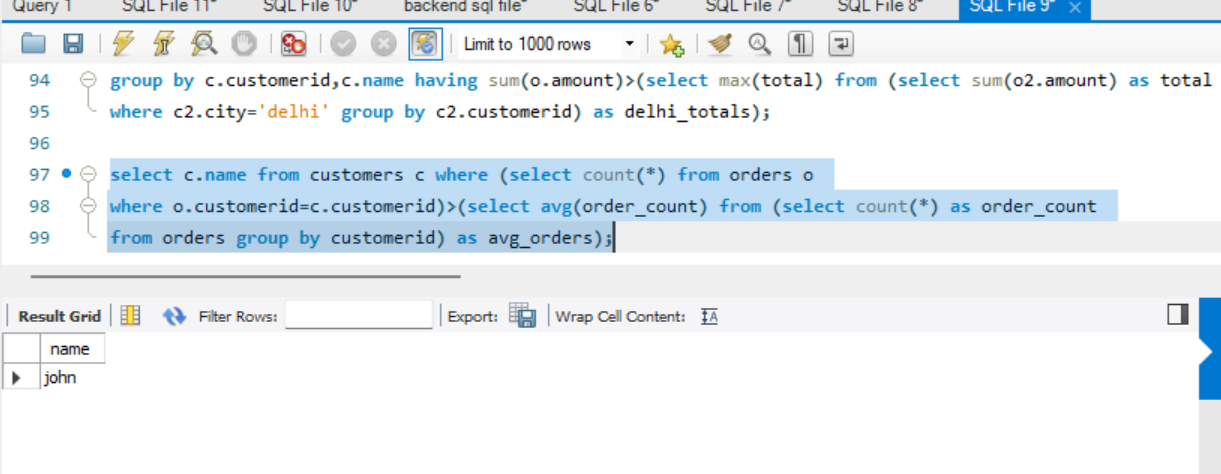
QUERY: select c.name from customers c join orders o on c.customerid=o.customerid group by c.customerid,c.name having sum(o.amount)>(select max(total) from (select sum(o2.amount) as total from customers c2 join orders o2 on c2.customerid=o2.customerid where c2.city='delhi' group by c2.customerid) as delhi\_totals);



**Part C – Join + Subquery Mix (30 marks)**

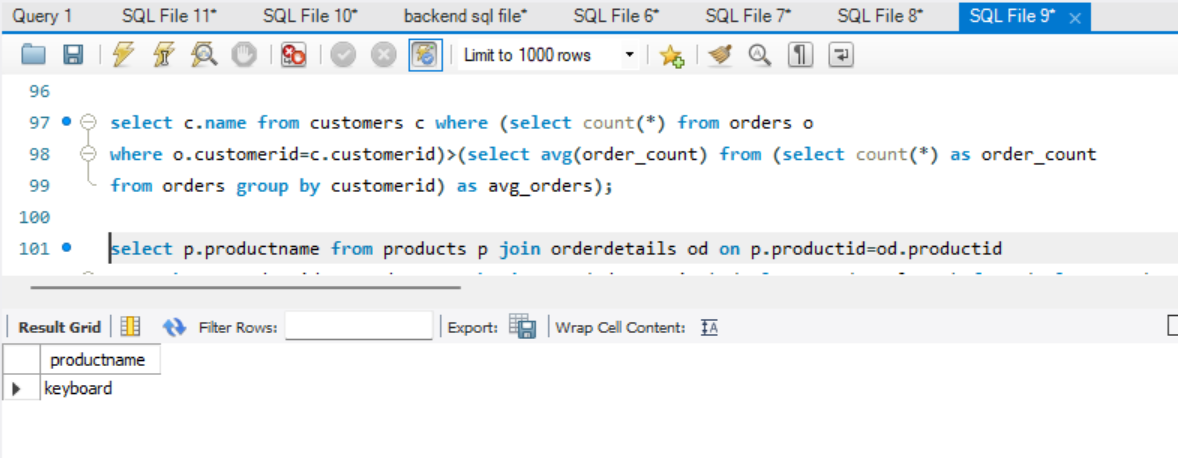
1. Use a correlated subquery to find customers who have placed **more orders than the average** number of orders placed by all customers.

QUERY: select c.name from customers c where (select count(\*) from orders o where o.customerid=c.customerid)>(select avg(order\_count) from (select count(\*) as order\_count from orders group by customerid) as avg\_orders);



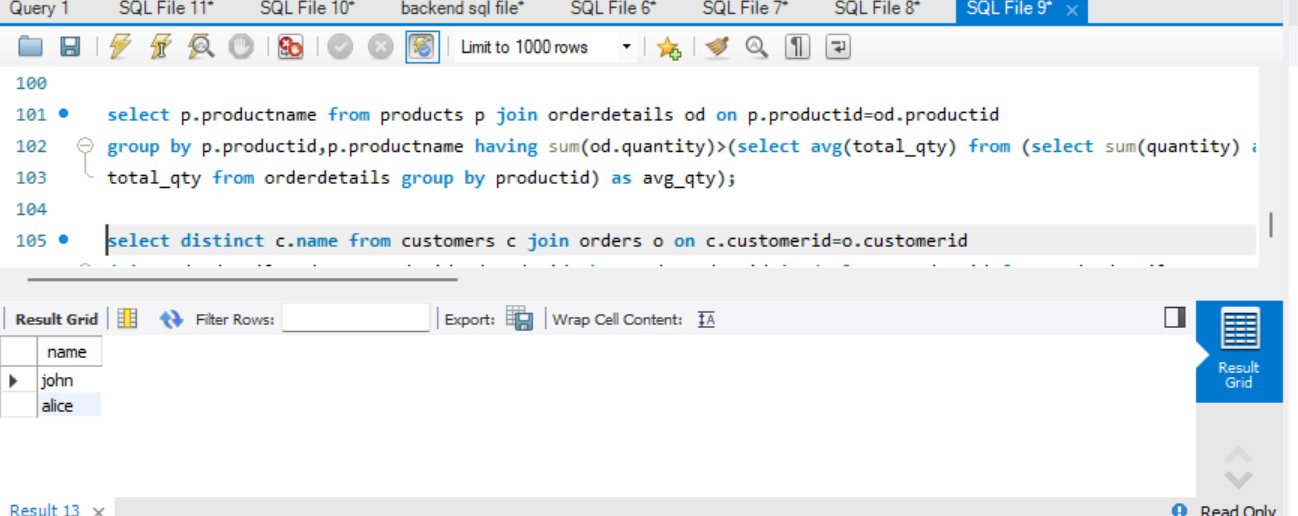
1. Find all products whose **total sales quantity** is higher than the average total quantity sold per product.

QUERY: select p.productname from products p join orderdetails od on p.productid=od.productid group by p.productid,p.productname having sum(od.quantity)>(select avg(total\_qty) from (select sum(quantity) as total\_qty from orderdetails group by productid) as avg\_qty);



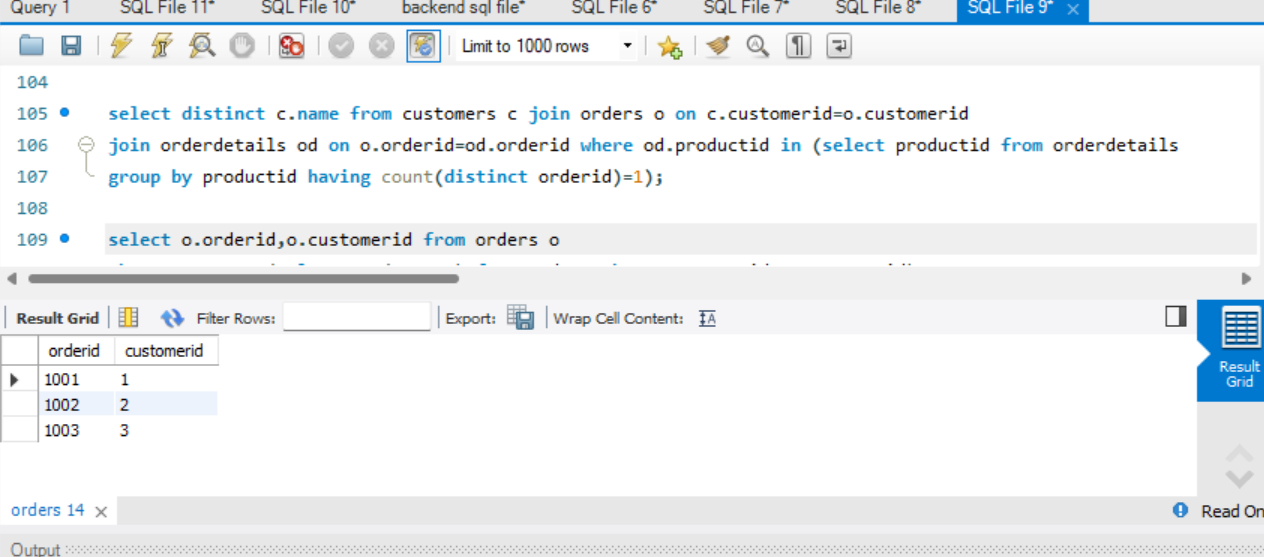
1. Get customers who have ordered at least **one product that no one else has ordered**.

QUERY: select distinct c.name from customers c join orders o on c.customerid=o.customerid join orderdetails od on o.orderid=od.orderid where od.productid in (select productid from orderdetails group by productid having count(distinct orderid)=1);



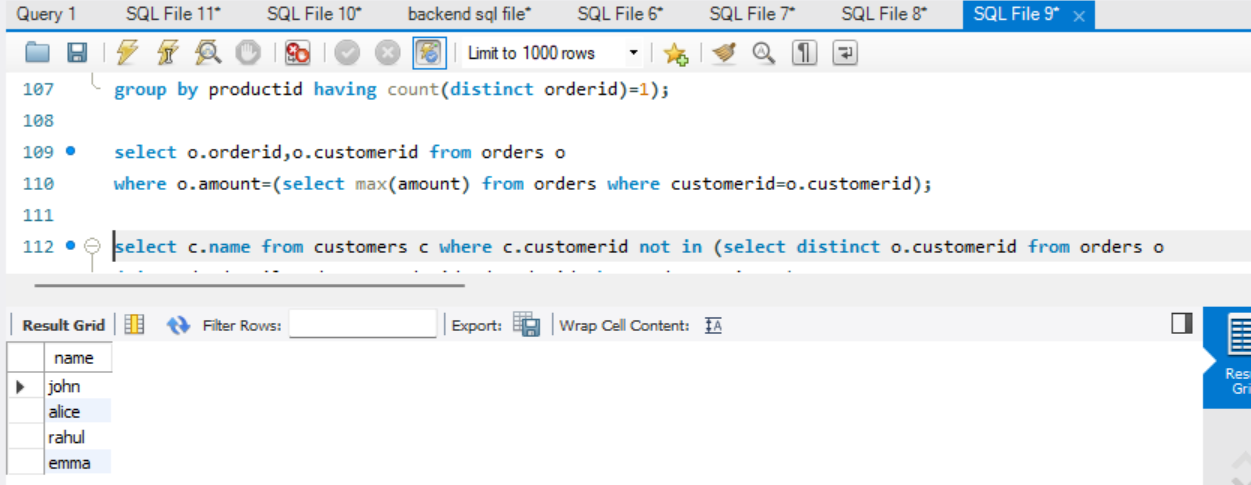
1. Retrieve all orders where the total order amount is equal to the **maximum order amount for that customer**.

QUERY: select o.orderid,o.customerid from orders o where o.amount=(select max(amount) from orders where customerid=o.customerid);



1. Write a query to list customers who have **never placed an order with a quantity greater than 5**.

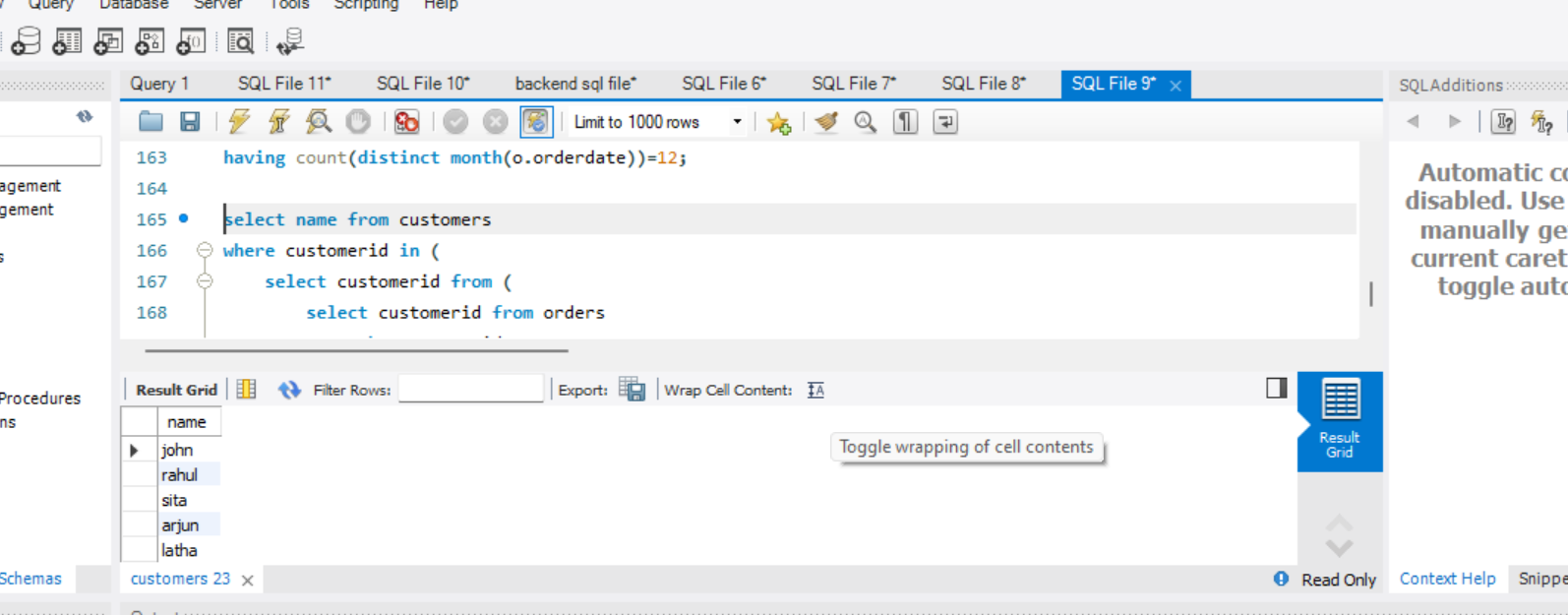
QUERY: select c.name from customers c where c.customerid not in (select distinct o.customerid from orders o join orderdetails od on o.orderid=od.orderid where od.quantity>5);



**Part D – Joins & Set Operations (25 marks)**

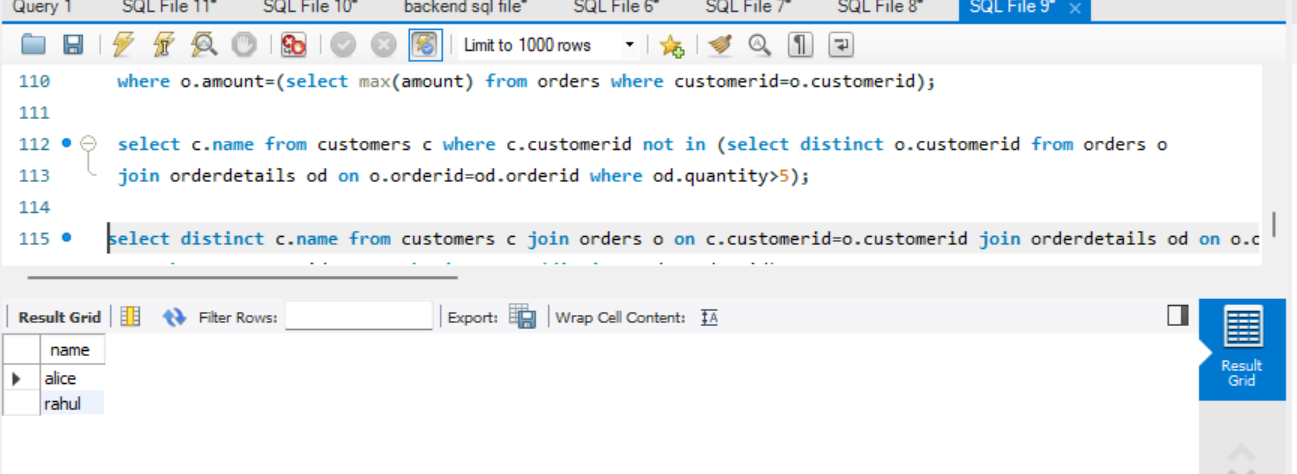
1. Use a subquery to list the **top 5 customers by total spending**.

select name from customers where customerid in (select customerid from (select customerid from orders group by customerid order by sum(amount) desc limit 5 ) as top\_customers

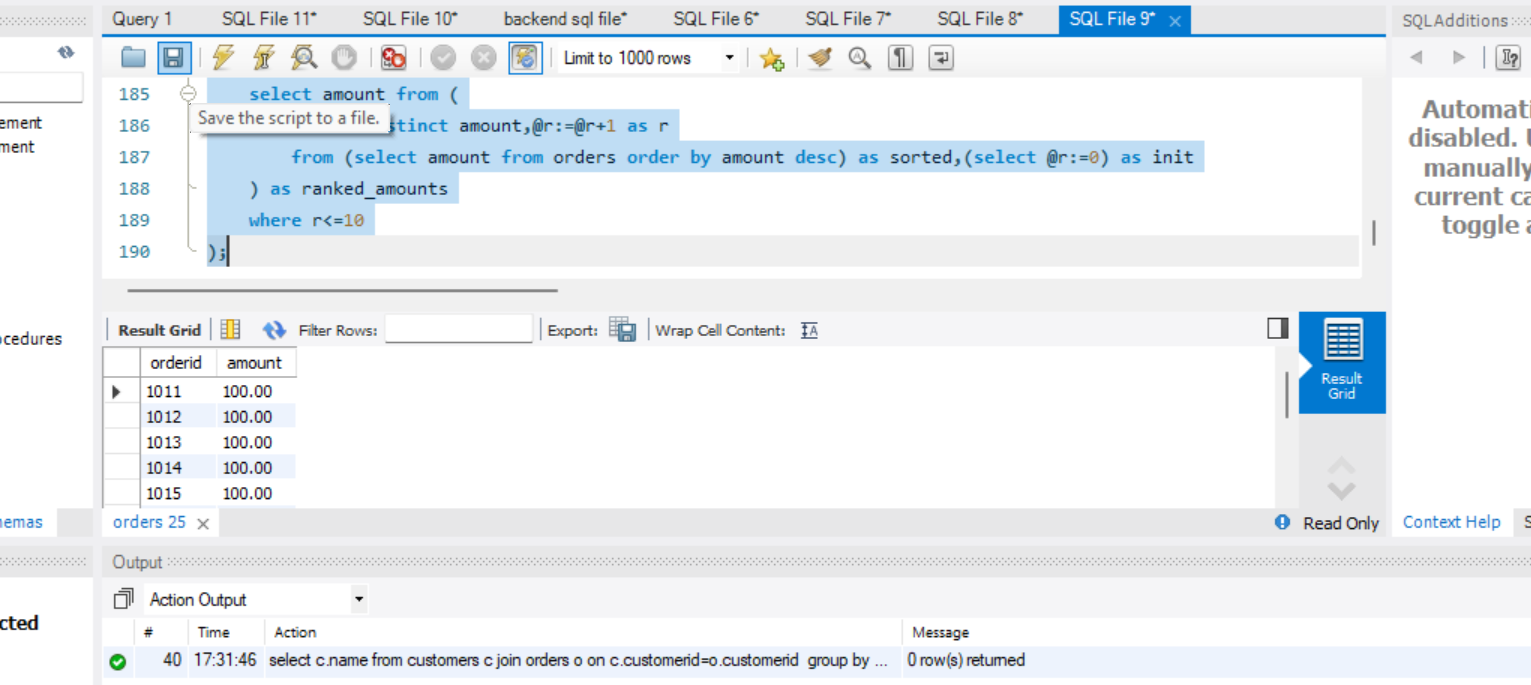
);

1. Find all customers who have only ordered **one unique product** using subqueries.

QUERY: select distinct c.name from customers c join orders o on c.customerid=o.customerid join orderdetails od on o.orderid=od.orderid group by c.customerid,c.name having count(distinct od.productid)=1;

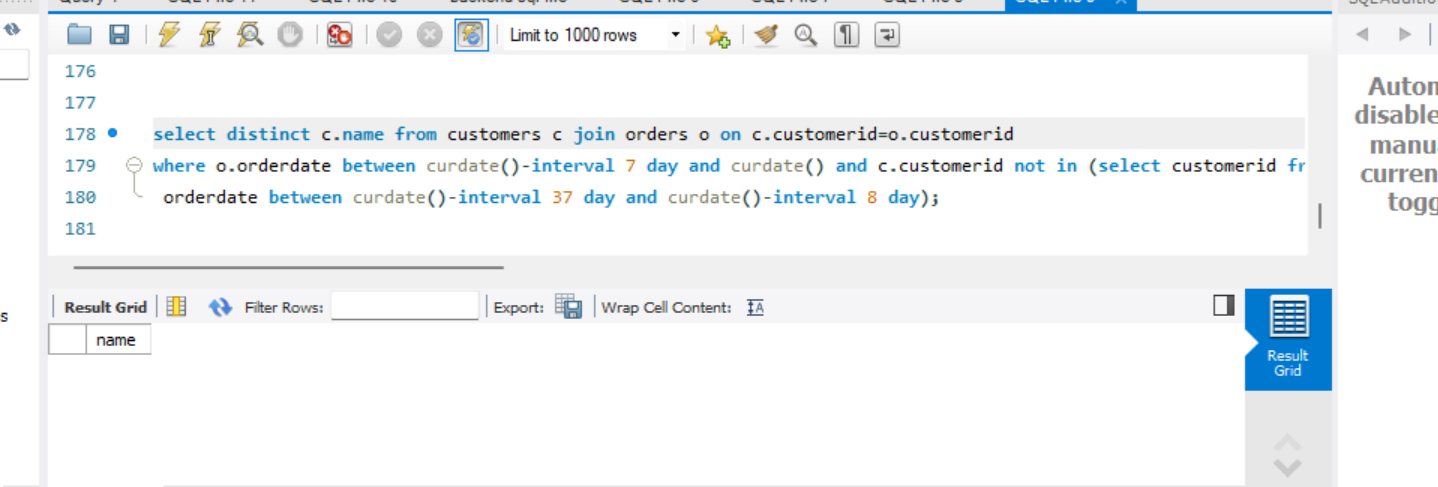


1. List all orders where the amount is **not in the top 10 highest order amounts**.

select \* from Orders where Amount not in(select Amount from(select Amount from Orders order by Amount desc limit 10) as Top10);

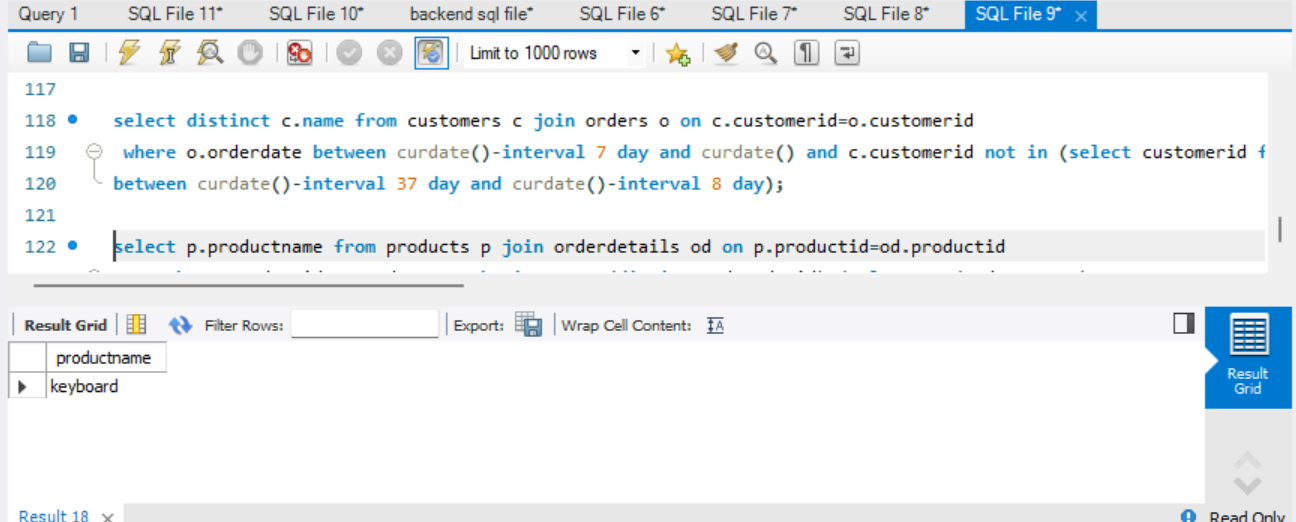
1. Retrieve customer names who placed an order in the **last 7 days** but **not** in the **previous 30 days** before that.

QUERY: select distinct c.name from customers c join orders o on c.customerid=o.customerid where o.orderdate between curdate()-interval 7 day and curdate() and c.customerid not in (select customerid from orders where orderdate between curdate()-interval 37 day and curdate()-interval 8 day);



1. Write a query to list all products ordered in the **highest number of distinct orders**.

QUERY: select p.productname from products p join orderdetails od on p.productid=od.productid group by p.productid,p.productname having count(distinct od.orderid)=(select max(order\_count) from (select count(distinct orderid) as order\_count from orderdetails group by productid) as order\_counts);



**DAY 8:**

task1:- write example of string formatting and operations on that

name=" i am YUVASRI from namakkaL"

print(name.upper())

print(name.lower())

print(name.strip())

print(name.replace("f","u"))

print(name.split(","))

print(name.capitalize())

print(name.casefold())

print(name.count('a'))

print(name.encode())

value=21

string =f"i am yuvasri and my age is {value}"

next=" and my sister name is theshitha"

final=string+next

print(final)

year=2024

name="yuvasri"

print(f"my name is {name}and i completed my college in {year}")

print("my name is {} and i completed my college in {}".format(name,year))

print("\101")

print("\x43")

print("hello\fworld")

print("I am\rstudying")

print("it\'s yuvasri")

print("\"hello\"")

print("backslash:\\")

print("good\tmorning")

print("i am yuavsri\n i completed BE")

task:2) create list, print specified index item, change item and append list by all ways, remove item using del,clear,remove all

list1=["hi","i","am","yuvasri"]

print (list1)

print(list1[3])

list1[3]="learning"

print(list1)

list1.append("python")

list1.insert(1,"hello")

print(list1)

list1.pop()

print(list1)

list1.extend(["awesome", "in", "Python"])

print(list1)

list1.extend("good")

print(list1)

list1.remove("Python")

print(list1)

del list1[1]

print(list1)

list1.clear()

print(list1)

task 3:-create tup, access tup, loop, range, slice, update items

print(mytuple)

newtup=tuple((7,6,2))

print(newtup)

print(mytuple[0])

print(mytuple[4])

for i in mytuple:

print(i)

for i in range(len(mytuple)):

print(f"index {i}={mytuple[i]}")

print(mytuple[:4])

mylist=list(mytuple)

mylist[6]="hiii"

mytuple=tuple(mylist)

print(mytuple)

task 4:- kid, teen, adult, old

age=40

if(age<12):

print("kid")

elif(age>12 and age<18):

print("teen")

elif(age>18 and age<60):

print("adult")

else:

print("old")

CODING CHALLENGE

Q1. Write a Python program to remove all duplicates from a list without using the set() function.

def mylistduplicates(mylist):

    newlist=[]

    for i in mylist:

        if i not in newlist:

            newlist.append(i)

    return newlist

mylist=[1, 2, 2, 3, 4, 4, 5]

result=mylistduplicates(mylist)

print(mylist)

print("duplicate removed:",result)



Input Example: [1, 2, 2, 3, 4, 4, 5]

Output: [1, 2, 3, 4, 5]

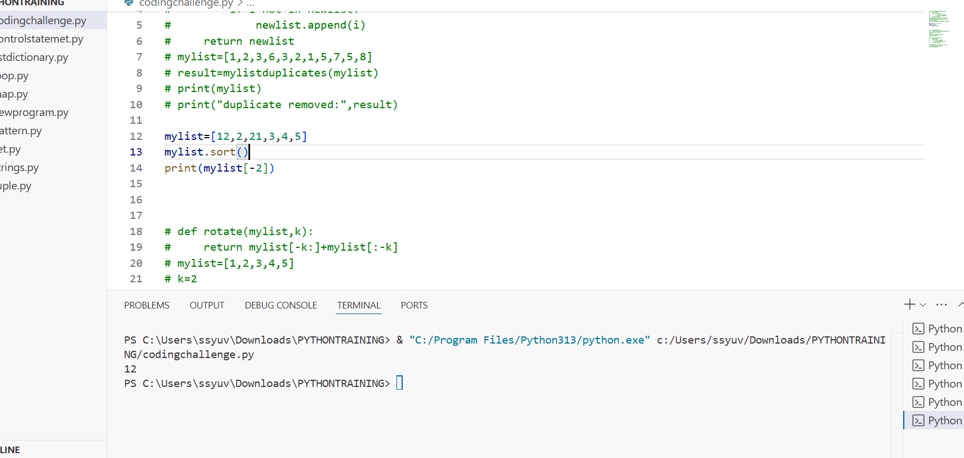
Q2. Given a list of integers, write a program to find the second highest unique number. Input Example: [12, 5, 9, 21, 21, 3]

Output: 12

mylist=[12, 5, 9, 21, 21, 3]

mylist.sort()

print(mylist[-2])



Q3. Rotate a list to the right by k positions. Input: List = [1, 2, 3, 4, 5], k = 2

Output: [4, 5, 1, 2, 3]

def rotate(mylist,k):

    return mylist[-k:]+mylist[:-k]

mylist=[1,2,3,4,5]

k=2

result=rotate(mylist,k)

print(result)



Q4. Write a Python program to multiply the elements of each tuple in a list of tuples and return a new list.

def multiply(mylist):

    newlist=[]

    for i,j in mylist:

        newlist.append(i\*j)

    return newlist

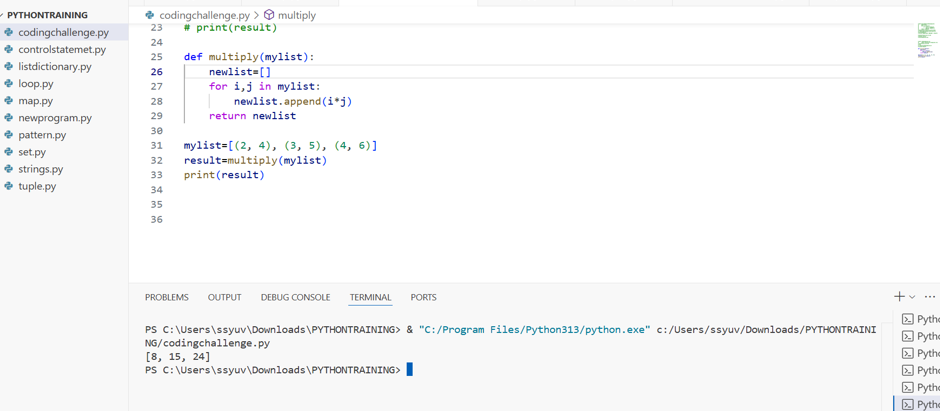
mylist=[(2, 4), (3, 5), (4, 6)]

result=multiply(mylist)

print(result)

Input: [(2, 4), (3, 5), (4, 6)]

Output: [8, 15, 24]



Q5. Given a tuple of integers, write a program to count how many times each element occurs. Input: (1, 2, 2, 3, 1, 4, 2)

Output: {1: 2, 2: 3, 3: 1, 4: 1}

def count\_elements(tup):

    freq = {}

    for num in tup:

        if num in freq:

            freq[num] += 1

        else:

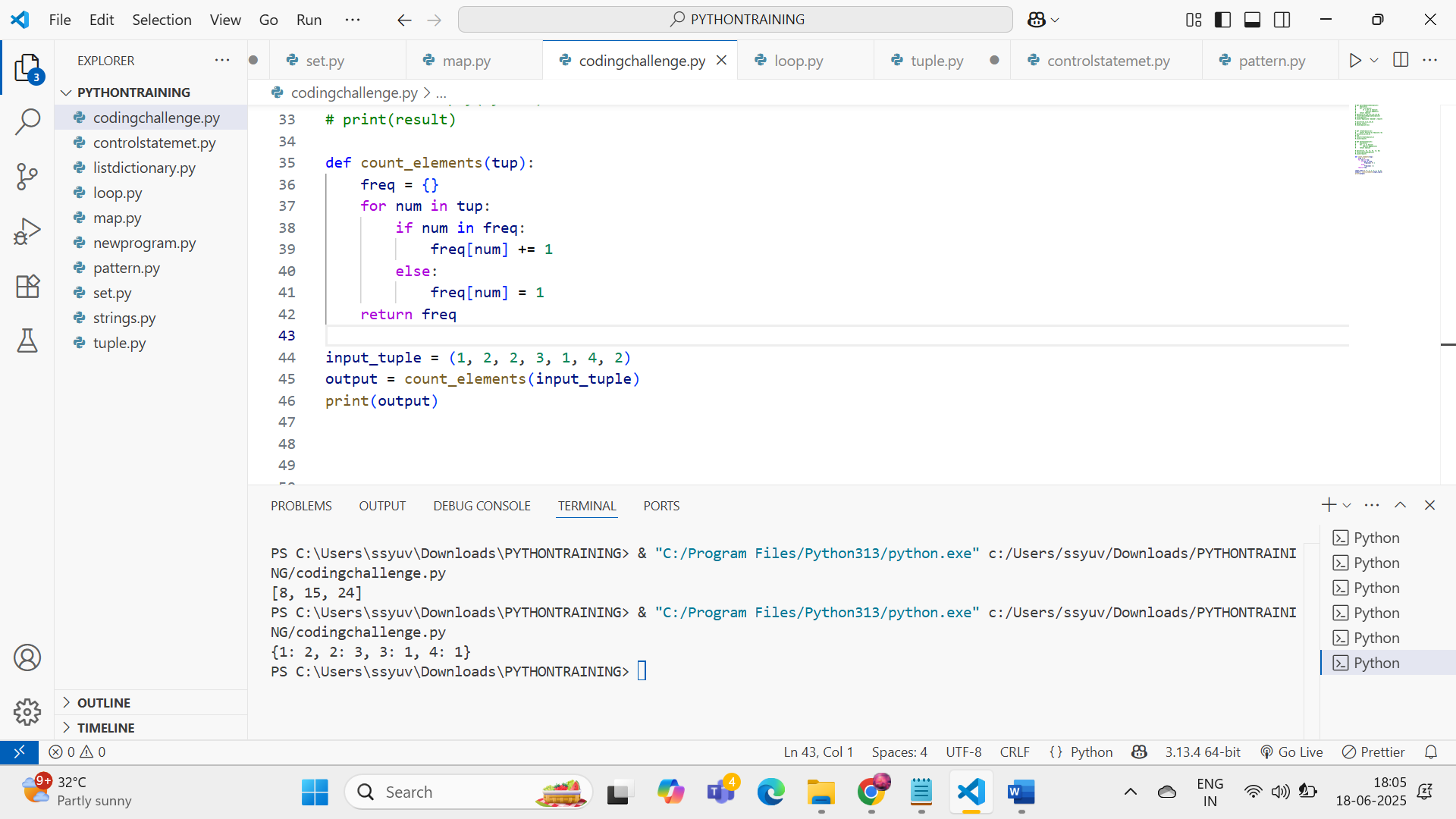
            freq[num] = 1

    return freq

input\_tuple = (1, 2, 2, 3, 1, 4, 2)

output = count\_elements(input\_tuple)

print(output)



Q6. Write a Python program to count the frequency of each character in a string using a dictionary. Input: 'banana'

Output: {'b': 1, 'a': 3, 'n': 2}

def count\_characters(s):

    freq = {}

    for char in s:

        if char in freq:

            freq[char] += 1

        else:

            freq[char] = 1

    return freq

input\_str = 'banana'

output = count\_characters(input\_str)

print(output)



Q7. Merge two dictionaries such that common keys have their values summed. Input: {'apple': 10, 'banana': 5}, {'banana': 3, 'orange': 7}

Output: {'apple': 10, 'banana': 8, 'orange': 7}

def merge\_dicts\_sum(d1, d2):

result = d1.copy()

for key, value in d2.items():

if key in result:

result[key] += value

else:

result[key] = value

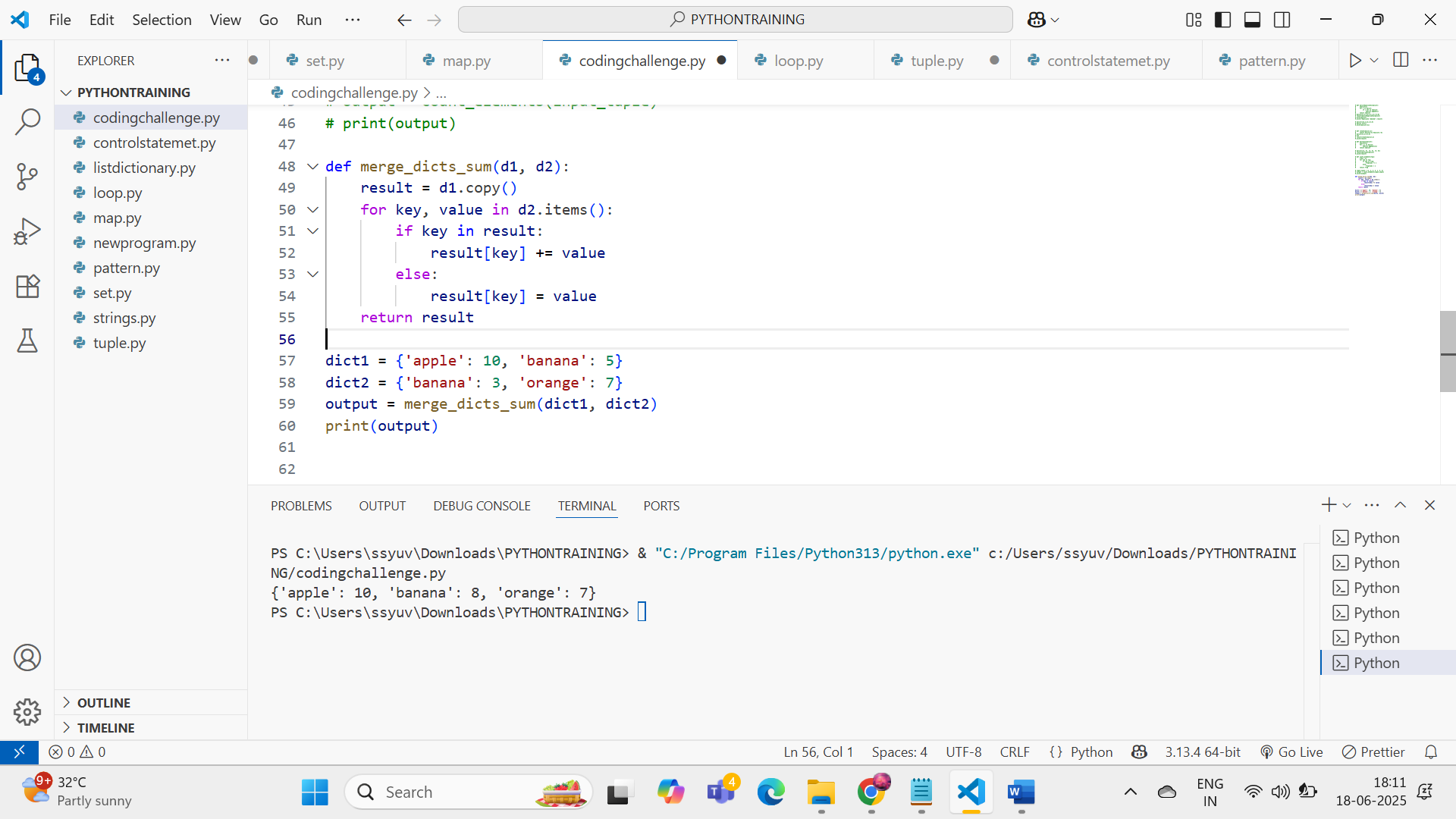
return result

dict1 = {'apple': 10, 'banana': 5}

dict2 = {'banana': 3, 'orange': 7}

output = merge\_dicts\_sum(dict1, dict2)

print(output)



Q8. Given a dictionary of student names and their marks, print the name(s) of the student(s) with the highest marks.

Input: {'Alice': 85, 'Bob': 92, 'Carol': 92} Output: ['Bob', 'Carol']

def top\_students(marks\_dict):

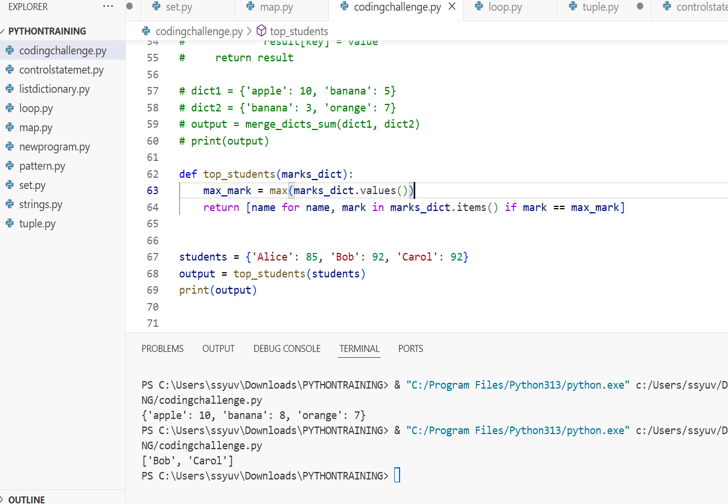
max\_mark = max(marks\_dict.values())

return [name for name, mark in marks\_dict.items() if mark == max\_mark]

students = {'Alice': 85, 'Bob': 92, 'Carol': 92}

output = top\_students(students)

print(output)



Q9. Write a Python program to find all common elements among three lists using set operations. Input: [1, 2, 3], [2, 3, 4], [3, 2, 5]

Output: {2, 3}

def find\_common\_elements(list1, list2, list3):

    return set(list1) & set(list2) & set(list3)

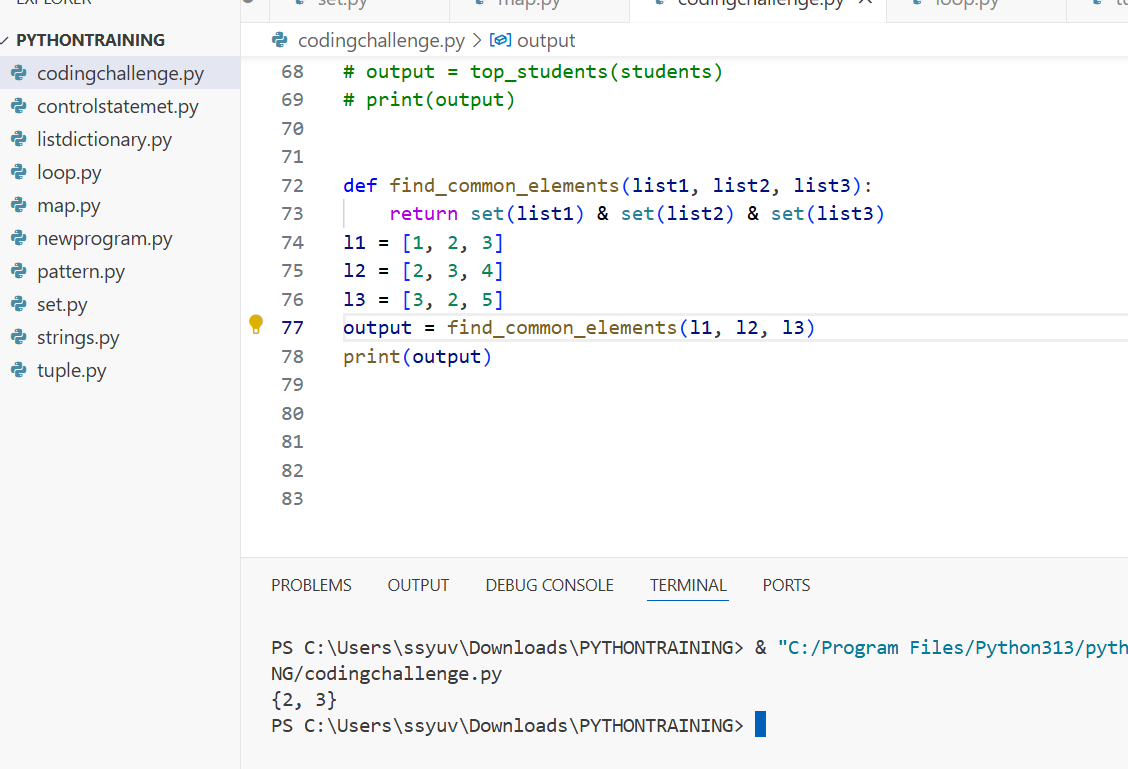
l1 = [1, 2, 3]

l2 = [2, 3, 4]

l3 = [3, 2, 5]

output = find\_common\_elements(l1, l2, l3)

print(output)



Q10. From a sentence entered by the user, extract and display all unique words using a set. Input: 'this is a test this is fun'

Output: {'this', 'is', 'a', 'test', 'fun'}

def extract\_unique\_words(sentence):

    words = sentence.split()

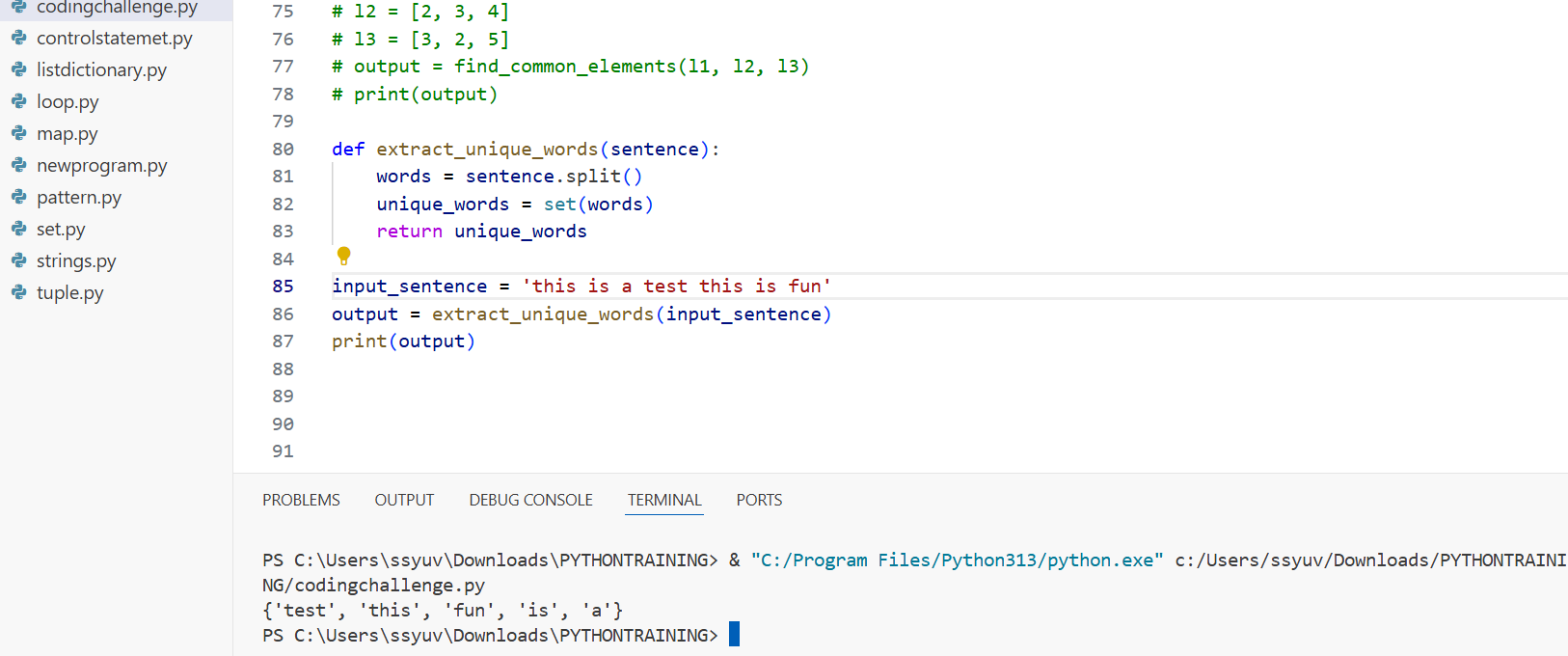
    unique\_words = set(words)

    return unique\_words

input\_sentence = 'this is a test this is fun'

output = extract\_unique\_words(input\_sentence)

print(output)



**DAY 9 :**

task1:- write example of string formatting and operations on that

name=" i am YUVASRI from namakkaL"

print(name.upper())

print(name.lower())

print(name.strip())

print(name.replace("f","u"))

print(name.split(","))

print(name.capitalize())

print(name.casefold())

print(name.count('a'))

print(name.encode())

value=21

string =f"i am yuvasri and my age is {value}"

next=" and my sister name is theshitha"

final=string+next

print(final)

year=2024

name="yuvasri"

print(f"my name is {name}and i completed my college in {year}")

print("my name is {} and i completed my college in {}".format(name,year))

print("\101")

print("\x43")

print("hello\fworld")

print("I am\rstudying")

print("it\'s yuvasri")

print("\"hello\"")

print("backslash:\\")

print("good\tmorning")

print("i am yuavsri\n i completed BE")

task:2) create list, print specified index item, change item and append list by all ways, remove item using del,clear,remove all

list1=["hi","i","am","yuvasri"]

print (list1)

print(list1[3])

list1[3]="learning"

print(list1)

list1.append("python")

list1.insert(1,"hello")

print(list1)

list1.pop()

print(list1)

list1.extend(["awesome", "in", "Python"])

print(list1)

list1.extend("good")

print(list1)

list1.remove("Python")

print(list1)

del list1[1]

print(list1)

list1.clear()

print(list1)

task 3:-create tup, access tup, loop, range, slice, update items

print(mytuple)

newtup=tuple((7,6,2))

print(newtup)

print(mytuple[0])

print(mytuple[4])

for i in mytuple:

print(i)

for i in range(len(mytuple)):

print(f"index {i}={mytuple[i]}")

print(mytuple[:4])

mylist=list(mytuple)

mylist[6]="hiii"

mytuple=tuple(mylist)

print(mytuple)

task 4:- kid, teen, adult, old

age=40

if(age<12):

print("kid")

elif(age>12 and age<18):

print("teen")

elif(age>18 and age<60):

print("adult")

else:

print("old")

**DAY 10:**

day 10:

write example like Father as a parent class son as child class to show single inheritance

class Father:

def property(self,property):

self.property=property

print(self.property)

class Child(Father):

def newproperty(self):

print(self.property)

p=Father()

c=Child()

c.property("car")

c.newproperty()

write example like Father, mother as a parent class son as child class to show multiple inheritance

class Father1:

def land(self,acres):

self.acres=acres

print("father property",self.acres)

class Mother:

def jewls(self,number):

self.number=number

def amount(self,money):

self.money=money

print("mothers",self.number,self.money)

class child1(Father1,Mother):

def Allpro(self):

print("childs",self.acres,self.number,self.money)

f=Father1()

m=Mother()

c=child1()

c.land("10 acres")

c.jewls("50 savaran")

c.amount("1 lakh")

c.Allpro()

TASK 2:-

1) single:-

parent to child

example :- Father to son

class Father:

def skills(self):

print("Gardening")

class Son(Father):

def sports(self):

print("Cricket")

obj = Son()

obj.skills()

obj.sports()

2) multiple:-

two parent or multiple parents to single child

example:- father,mother to son

example:- grandfather,father,mother to son

class Father:

def work(self):

print("Engineer")

class Mother:

def cook(self):

print("Chef")

class Son(Father, Mother):

def hobby(self):

print("Guitarist")

obj = Son()

obj.work()

obj.cook()

obj.hobby()

3) multilevel:-

class inherit from a class who is already inheriting from another class

human to male, male to Arun

grandfather to father, father to son

vehicle to fourwheeler, fourwheeler to car, car to maruthi

class Human:

def breath(self):

print("Breathing")

class Male(Human):

def gender(self):

print("Male")

class Arun(Male):

def name(self):

print("Name is Arun")

obj = Arun()

obj.breath()

obj.gender()

obj.name()

4) hierarchical

same parent multiple child

father

son daughter youngerdaughter

class Father:

def work(self):

print("Hardworking")

class Son(Father):

def play(self):

print("Football")

class Daughter(Father):

def dance(self):

print("Classical Dance")

obj1 = Son()

obj2 = Daughter()

obj1.work()

obj1.play()

obj2.work()

obj2.dance()

5) hybrid

combination of any two types mentioned above

combination of more then one type

class Person:

def speak(self):

print("Can speak")

class Father(Person):

def drive(self):

print("Can drive")

class Mother(Person):

def cook(self):

print("Can cook")

class Son(Father, Mother):

def play(self):

print("Can play games")

obj = Son()

obj.speak()

obj.drive()

obj.cook()

obj.play()

**DAY 11:**

Section A: Basic Understanding (Short Answer Questions)  
1) What is a Python package? How is it different from a module?

PYTHON MODULE:

A **module** is just a **single .py file** that contains **Python code** — like functions, classes, or variables — that you can **import** and reuse in other programs.

PYTHON PACKAGE:

A **package** is a **folder** (directory) that contains an \_\_init\_\_.py file and one or more **modules**. It is used to **organize related modules together**.

2)What is the purpose of \_\_init\_\_.py in a package directory?

The \_\_init\_\_.py file is used to **mark a folder as a Python package** so it can be imported. It Signals to Python that the directory should be treated as a package and Can include initialization code for the package and also define \_\_all\_\_ for controlling import \* behavior.

3)What happens when you use from package import \* in Python?

It **imports all public modules or symbols** listed in the package's \_\_init\_\_.py file using \_\_all\_\_.

4)What is the effect of defining \_\_all\_\_ in a package’s \_\_init\_\_.py file?

\_\_all\_\_ is a **list of module names** (as strings) that defines what will be imported when you use from package import \*.

5)How can you create and use a subpackage in Python?

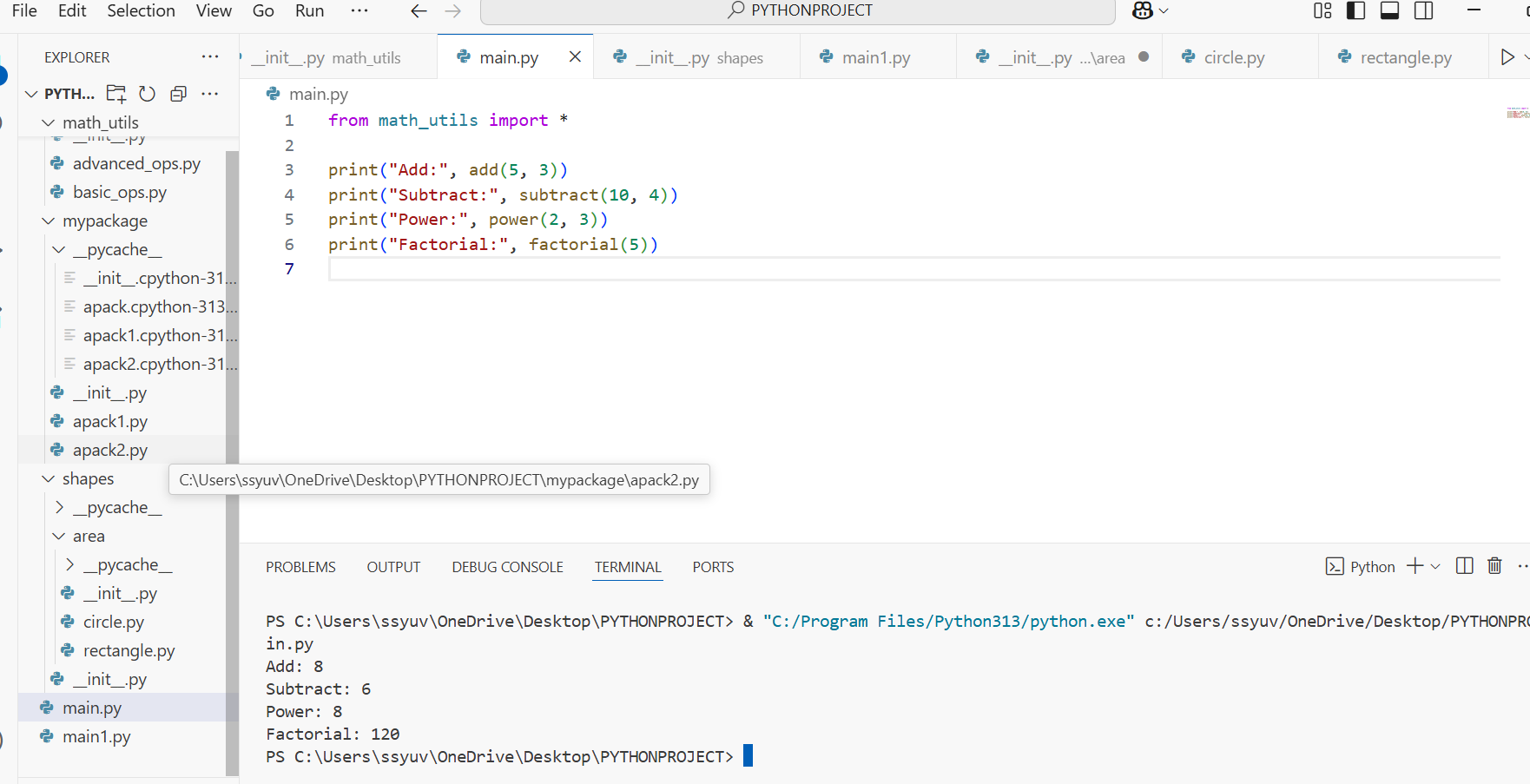
A **subpackage** is a **package inside another package**. For example : there is a folder mypackage that contains \_\_init.py\_\_ file and another folder subpackage that contains \_\_init\_\_.py and submodule.py files. Here mypackage/ contains another package subpackage /which is known as subpackage

Section B: Coding-Based Questions  
6) Create a package called math\_utils with the following modules:

basic\_ops.py – containing functions for add, subtract

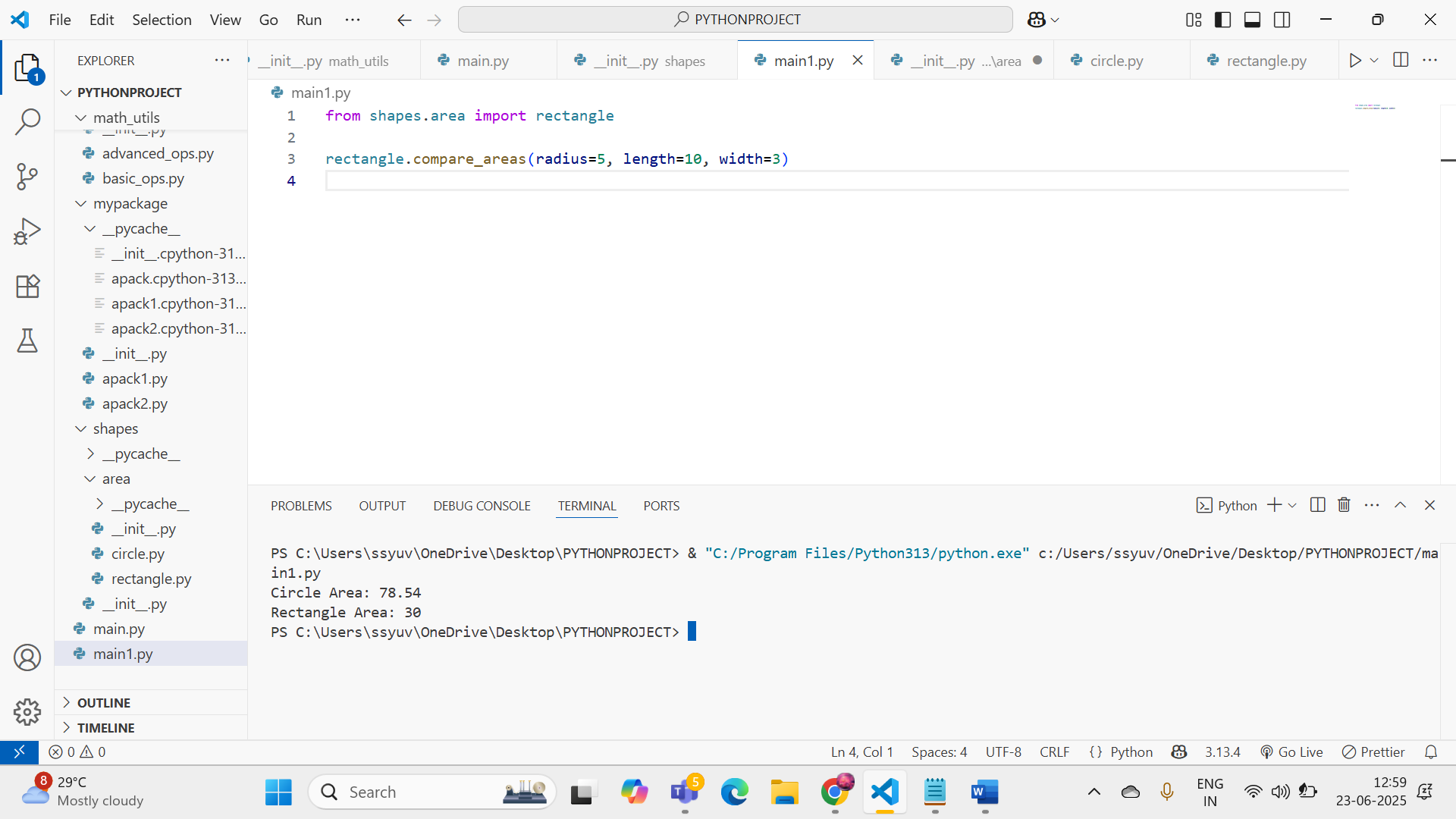
advanced\_ops.py – containing functions for power and factorial

Demonstrate how to import and use all functions using from math\_utils import \*.



7) Write Python code to show how intra-package references work when module\_a.py imports a function from module\_b.py inside the same package.

8) Create a package shapes with a subpackage area. In area, create modules circle.py and rectangle.py. Show how to import area\_of\_circle from circle.py in rectangle.py using relative import.



9)Modify your package math\_utils to include \_\_all\_\_ = ['basic\_ops'] in the \_\_init\_\_.py. What will happen if you run from math\_utils import \* after that?

When you include \_\_all\_\_ = ['basic\_ops'] in the math\_utils/\_\_init\_\_.py file, only the basic\_ops module will be imported when you use from math\_utils import \*. The advanced\_ops module will **not** be imported automatically.

So, attempting to access advanced\_ops without explicitly importing it will result in a NameError.

# Exception Handling – Question Paper

## Section A: Basic Try-Except (2 marks each)

1. Write a program to divide two numbers entered by the user. Handle ZeroDivisionError using try-except.

try:

    num1 = float(input("Enter the numerator: "))

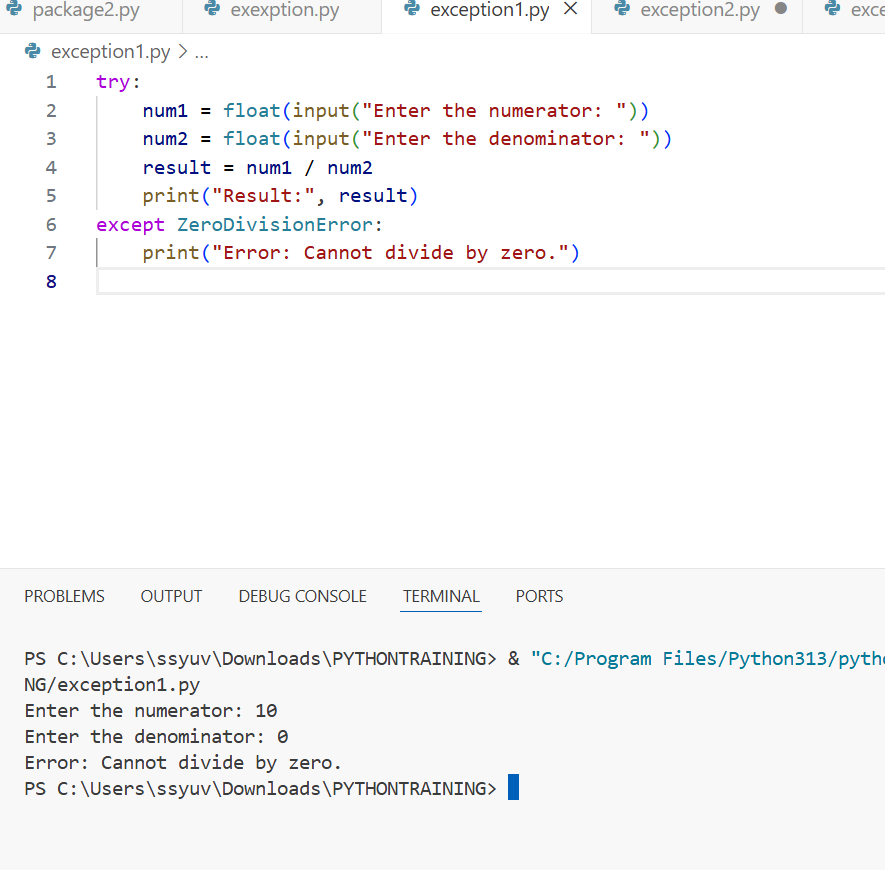
num2 = float(input("Enter the denominator: "))

result = num1 / num2

    print("Result:", result)

except ZeroDivisionError:

  print("Error: Cannot divide by zero.")



1. Write a program to convert a string to an integer. Handle ValueError if the input is not a valid number.

try:

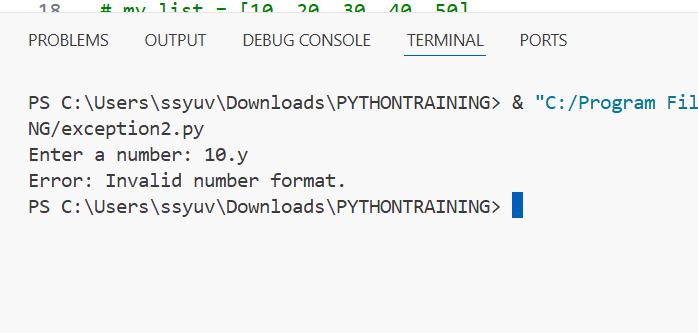
    user\_input = input("Enter a number: ")

    number = int(user\_input)

    print("Integer:", number)

except ValueError:

    print("Error: Invalid number format.")



1. Accept two numbers from the user and perform addition. Use try-except to handle invalid input types.

try:

    num1 = float(input("Enter first number: "))

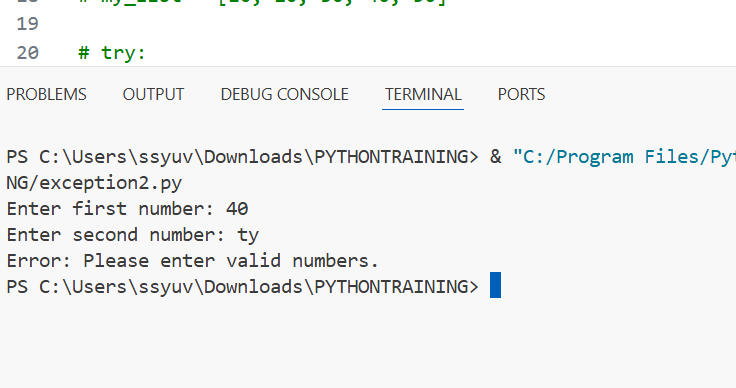
  num2 = float(input("Enter second number: "))

  total = num1 + num2

    print("Sum:", total)

except ValueError:

    print("Error: Please enter valid numbers.")



1. Write a program to read an element from a list using an index entered by the user. Handle IndexError. my\_list = [10, 20, 30, 40, 50]

try:

    index = int(input("Enter the index (0-4): "))

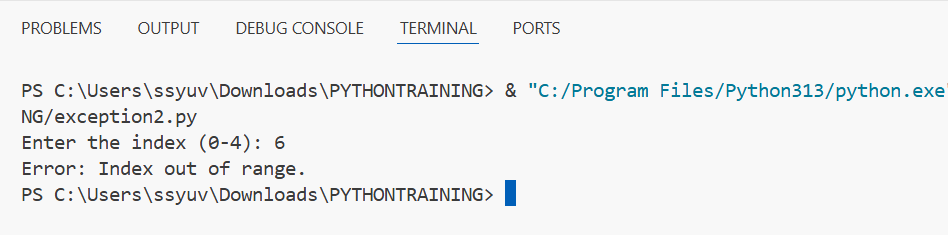
    print("Element at index", index, "is", my\_list[index])

except IndexError:

    print("Error: Index out of range.")

except ValueError:

    print("Error: Please enter a valid integer index.")



## Section B: Try-Except-Else (4 marks each)

1. Create a program that accepts a number from the user and prints its square. Use try-except-else to handle ValueError and ensure successful computation is shown only if there's no error.

try:

    num = float(input("Enter a number: "))

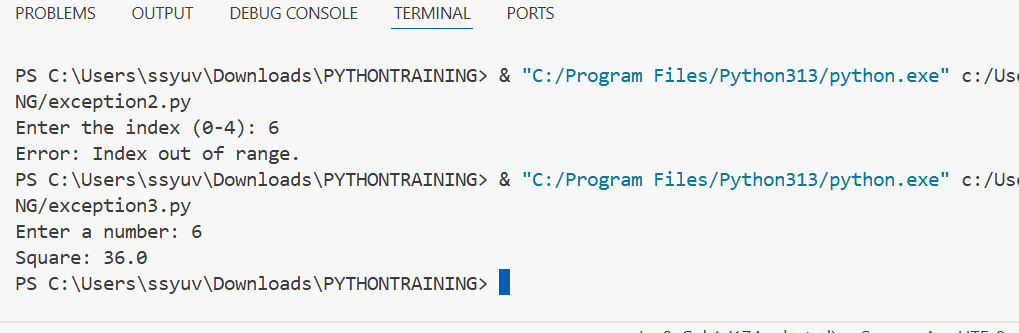
except ValueError:

    print("Error: Invalid number format.")

else:

    square = num \*\* 2

    print("Square:", square)



1. Write a program to open a file and read contents. Use try-except-else to handle FileNotFoundError. try:

    file\_name = input("Enter the filename: ")

    f = open(file\_name, 'r')

except FileNotFoundError:

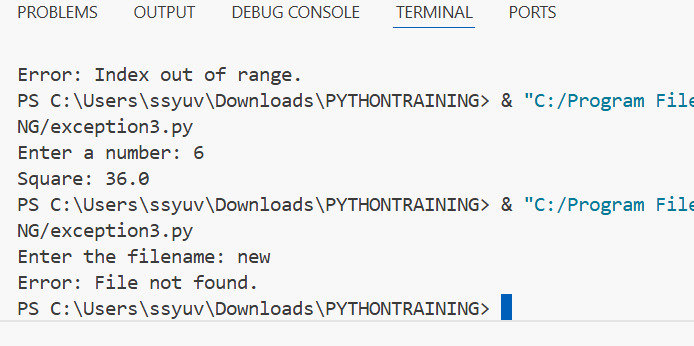
    print("Error: File not found.")

else:

    content = f.read()

    print("File content:\n", content)

  f.close()



1. Write a Python program to convert a number to its binary format. Use try-except-else to handle any invalid input.

try:

    number = int(input("Enter an integer: "))

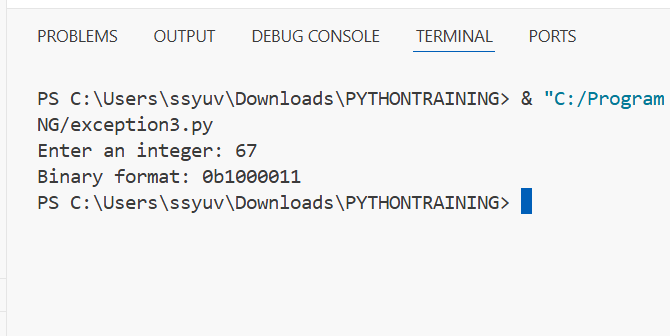
except ValueError:

    print("Error: Please enter a valid integer.")

else:

    binary = bin(number)

    print("Binary format:", binary)



## Section C: Try-Finally (5 marks each)

1. Write a program that opens a file and ensures it gets closed, whether or not an exception occurs. Use try-finally.

try:

    file = open("example.txt", "r")

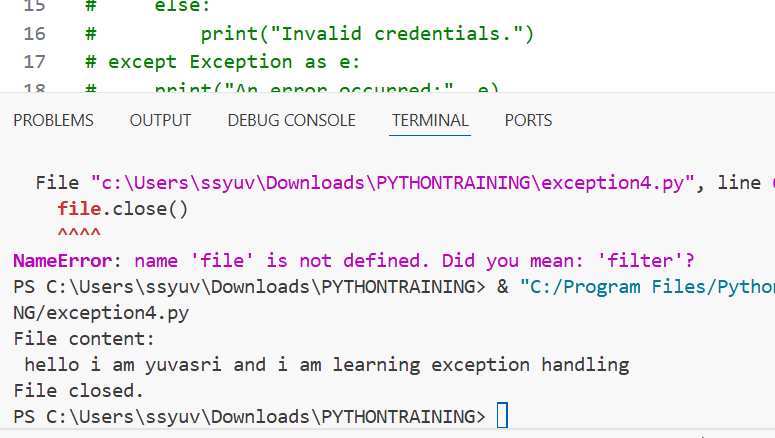
    content = file.read()

    print("File content:\n", content)

finally:

    file.close()

    print("File closed.")



1. Simulate a login process where the user input is handled in a try block and a log message is printed in finally regardless of success or failure.

try:

    username = input("Enter username: ")

    password = input("Enter password: ")

    if username == "admin" and password == "1234":

        print("Login successful!")

    else:

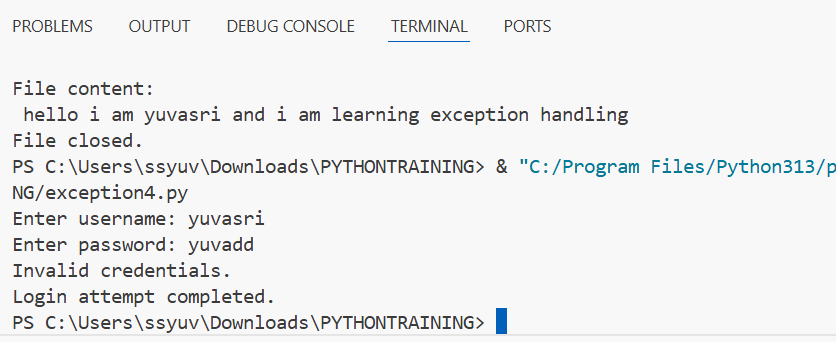
        print("Invalid credentials.")

except Exception as e:

    print("An error occurred:", e)

finally:

    print("Login attempt completed.")



1. Write a program that divides two numbers, catching errors with try-except, and printing a clean-up message using finally.

try:

    num1 = float(input("Enter numerator: "))

    num2 = float(input("Enter denominator: "))

    result = num1 / num2

    print("Result:", result)

except ZeroDivisionError:

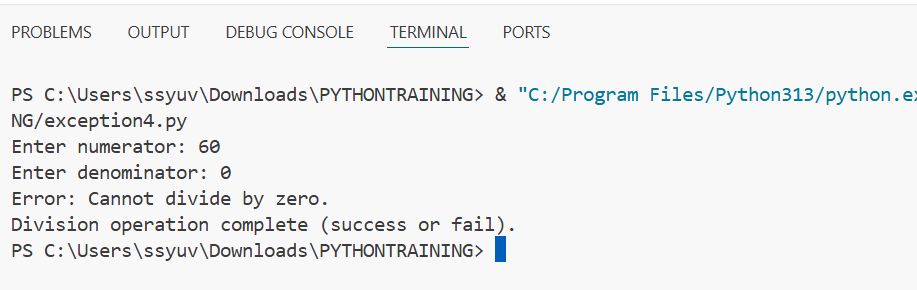
    print("Error: Cannot divide by zero.")

except ValueError:

    print("Error: Please enter valid numbers.")

finally:

    print("Division operation complete (success or fail).")



## Section D: Combined Exception Handling (6 marks each)

1. Create a program that handles multiple exceptions: ZeroDivisionError, ValueError, and always prints "Execution complete" using finally.

try:

  num1 = float(input("Enter numerator: "))

  num2 = float(input("Enter denominator: "))

    result = num1 / num2

    print("Result:", result)

except ZeroDivisionError:

    print("Error: Cannot divide by zero.")

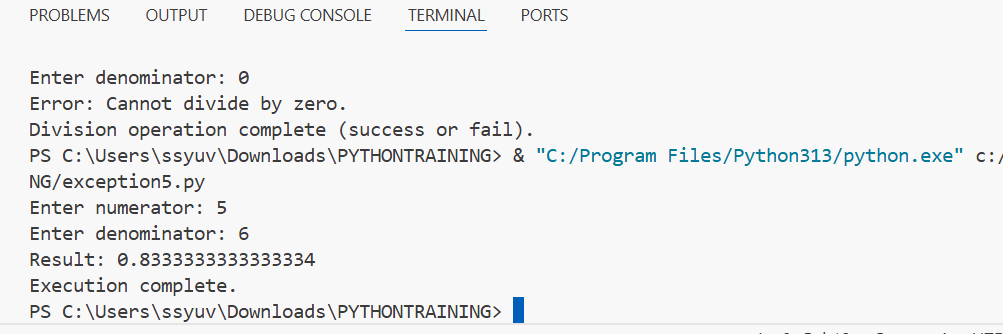
except ValueError:

    print("Error: Please enter valid numbers.")

finally:

    print("Execution complete.")

balance = 1000.0



1. Write a program to simulate bank withdrawal. Use try-except-else-finally to handle incorrect amount input, and always print a message whether the transaction succeeded or failed.

balance = 1000.0

try:

    amount = float(input("Enter withdrawal amount: "))

    if amount <= 0:

        raise ValueError("Amount must be positive.")

    if amount > balance:

        raise ValueError("Insufficient balance.")

except ValueError as ve:

    print("Transaction failed:", ve)

else:

    balance -= amount

    print(f"Withdrawal successful! Remaining balance: ₹{balance}")

finally:

    print("Transaction process completed.")



**DAY-12:**

**SQL + Python Quiz**

**Duration:- 60 min**

**🔹 Section A: Algorithm Basics**

1. **Which of the following problems is best suited for the Greedy approach?**  
   a) 0/1 Knapsack  
   b) Matrix Chain Multiplication  
   c) Activity Selection Problem  
   d) Longest Common Subsequence
2. **Which is NOT true about the Divide and Conquer technique?**  
   a) Breaks problems into sub-problems  
   b) Merges sorted arrays  
   c) Suitable only for linear time solutions  
   d) Used in Quick Sort
3. **Dynamic Programming is preferred over recursion when:**  
   a) The problem is large  
   b) Overlapping sub-problems exist  
   c) Greedy fails  
   d) Time is not a constraint
4. **Brute force technique is best characterized by:**  
   a) Optimal substructure  
   b) Recursive backtracking  
   c) Exhaustive search  
   d) Memoization

**🔹 Section B: Data Structures Basics**

1. **Which data structure provides constant time access to elements by index?**  
   a) Array  
   b) Linked List  
   c) Stack  
   d) Queue
2. **Which of the following is not a LIFO structure?**  
   a) Call Stack  
   b) Queue  
   c) Stack  
   d) Recursion
3. **Which data structure allows insertion from one end and deletion from the other?**  
   a) Stack  
   b) Queue  
   c) Deque  
   d) Array

**🔹 Section C: Sorting Techniques**

1. **Which of the following sorting algorithms has the best average-case performance?**  
   a) Bubble Sort  
   b) Insertion Sort  
   c) Merge Sort  
   d) Selection Sort
2. **Quick sort fails to provide O(n log n) performance when:**  
   a) Array is already sorted  
   b) Median is chosen as pivot  
   c) Pivot is random  
   d) Array has duplicate values
3. **Which sorting technique is based on the concept of "divide and merge"?**  
   a) Selection Sort  
   b) Merge Sort  
   c) Insertion Sort  
   d) Quick Sort

**🔹 Section D: Searching Techniques**

1. **Binary search is applicable only when:**  
   a) Data is unsorted  
   b) Data is sorted  
   c) Data is in a stack  
   d) Data contains strings
2. **The time complexity of linear search is:**  
   a) O(log n)  
   b) O(1)  
   c) O(n)  
   d) O(n²)

**🔹 Section E: Tree Structures**

1. **Which of the following trees maintains a balanced height after every operation?**  
   a) Binary Tree  
   b) AVL Tree  
   c) BST  
   d) N-ary Tree
2. **In-order traversal of a binary search tree results in:**  
   a) Random order  
   b) Sorted order  
   c) Post-order sequence  
   d) Descending order
3. **What is the maximum number of children a node can have in a Binary Tree?**  
   a) 1  
   b) 2  
   c) 3  
   d) Unlimited

**🔹 Section F: Agile and Scrum**

1. **Which of the following best describes Agile?**  
   a) Waterfall methodology  
   b) A set of tools  
   c) Iterative and incremental development  
   d) Traditional project management
2. **The Scrum Master is primarily responsible for:**  
   a) Managing the team’s budget  
   b) Coaching the team and removing impediments  
   c) Writing code  
   d) Assigning tasks to developers
3. **TRELLO is mostly used for:**  
   a) Coding IDE  
   b) Database Management  
   c) Task Tracking and Collaboration  
   d) Software Testing
4. **Which SDLC model emphasizes detailed documentation and minimal client interaction?**  
   a) Spiral Model  
   b) Agile Model  
   c) Waterfall Model  
   d) V-Model

**🔹 Section G: Bonus Mixed Concept**

1. **Which combination is best for solving the shortest path in a weighted graph with non-negative weights?**  
   a) Brute Force + DFS  
   b) BFS + Memoization  
   c) Greedy + Dijkstra’s Algorithm  
   d) Dynamic Programming + Quick Sort

**🔹 Section 1: Algorithm Basics (Q1–10)**

1. Greedy algorithms work best when a problem exhibits:  
   a) Overlapping subproblems  
   b) Optimal substructure and greedy choice property  
   c) Backtracking  
   d) Recursion
2. Which technique is most effective for problems like Fibonacci using memoization?  
   a) Divide and Conquer  
   b) Brute Force  
   c) Greedy  
   d) Dynamic Programming
3. The divide and conquer approach is used in:  
   a) Bubble Sort  
   b) Selection Sort  
   c) Merge Sort  
   d) Insertion Sort
4. Brute-force algorithm for pattern matching in text searches:  
   a) KMP Algorithm  
   b) Rabin-Karp  
   c) Naive Approach  
   d) Boyer-Moore
5. Dynamic programming approach stores:  
   a) Entire input  
   b) Solutions to subproblems  
   c) Output in trees  
   d) Search indexes
6. Which of the following problems is not suitable for greedy?  
   a) Huffman Coding  
   b) Fractional Knapsack  
   c) 0/1 Knapsack  
   d) Prim’s Algorithm
7. Divide and conquer recursively splits the problem into:  
   a) Single input  
   b) Sub-problems  
   c) Non-recursive loops  
   d) Memory blocks
8. Dynamic Programming avoids:  
   a) Repeating subproblems  
   b) Loop iterations  
   c) Base case solutions  
   d) Recursive steps
9. Which strategy tries all possibilities?  
   a) Dynamic  
   b) Greedy  
   c) Divide and Conquer  
   d) Brute Force
10. Time complexity of Fibonacci using DP (bottom-up)?  
    a) O(n²)  
    b) O(log n)  
    c) O(n)  
    d) O(2^n)

**🔹 Section 2: Data Structures (Q11–20)**

1. Which structure uses LIFO?  
   a) Queue  
   b) Array  
   c) Stack  
   d) Linked List
2. Linked lists are preferred over arrays when:  
   a) Random access is needed  
   b) Memory is limited  
   c) Frequent insertions/deletions occur  
   d) Fixed size is needed
3. Which data structure allows insertion from both ends?  
   a) Queue  
   b) Stack  
   c) Deque  
   d) List
4. Which is best for recursion call tracking?  
   a) Queue  
   b) Stack  
   c) Tree  
   d) Heap
5. Arrays have time complexity O(1) for:  
   a) Insertion  
   b) Deletion  
   c) Indexing  
   d) Searching
6. Queue follows:  
   a) LIFO  
   b) FILO  
   c) FIFO  
   d) Circular logic
7. To implement undo in an app, use:  
   a) Array  
   b) Queue  
   c) Stack  
   d) Graph
8. Linked list traversal time complexity:  
   a) O(1)  
   b) O(n)  
   c) O(log n)  
   d) O(n log n)
9. Which structure is linear?  
   a) Tree  
   b) Graph  
   c) Array  
   d) Heap
10. Which supports priority element retrieval?  
    a) Queue  
    b) Stack  
    c) Priority Queue  
    d) Linked List

**🔹 Section 3: Sorting & Searching (Q21–30)**

1. Best case for Bubble Sort?  
   a) O(n)  
   b) O(n log n)  
   c) O(n²)  
   d) O(log n)
2. Which sort is non-comparative?  
   a) Selection  
   b) Radix  
   c) Merge  
   d) Quick
3. Binary search requires:  
   a) Hash table  
   b) Sorted array  
   c) Unsorted array  
   d) Tree
4. Which search method is linear?  
   a) Binary  
   b) Hash  
   c) Linear  
   d) B-tree
5. Quick sort worst case occurs when:  
   a) Elements are random  
   b) All elements are same  
   c) Already sorted  
   d) All options
6. Merge Sort space complexity:  
   a) O(1)  
   b) O(n)  
   c) O(log n)  
   d) O(n log n)
7. Selection sort compares elements to:  
   a) First  
   b) Middle  
   c) Minimum  
   d) Last
8. Quick sort is:  
   a) Stable  
   b) In-place  
   c) Iterative only  
   d) Heap-based
9. Which has worst case O(n²)?  
   a) Merge  
   b) Quick  
   c) Bubble  
   d) Radix
10. Which is not a comparison-based sort?  
    a) Merge  
    b) Radix  
    c) Selection  
    d) Heap

**🔹 Section 4: Trees (Q31–40)**

1. Full binary tree has:  
   a) All nodes with two children  
   b) All nodes with one child  
   c) Root only  
   d) Leaves only
2. AVL Tree ensures:  
   a) Sorted data  
   b) Duplicate entries  
   c) Balanced height  
   d) No children
3. BST right child always:  
   a) Smaller  
   b) Greater  
   c) Equal  
   d) Random
4. Pre-order traversal visits in order:  
   a) Left, Root, Right  
   b) Root, Left, Right  
   c) Left, Right, Root  
   d) Root, Right, Left
5. Which tree allows self-balancing?  
   a) BST  
   b) AVL  
   c) Binary  
   d) Threaded
6. In a tree, level order traversal uses:  
   a) Stack  
   b) Queue  
   c) Recursion  
   d) Array
7. Height of a tree with one node:  
   a) 0  
   b) 1  
   c) -1  
   d) Undefined
8. Inorder traversal of BST gives:  
   a) Pre-order  
   b) Reverse  
   c) Sorted list  
   d) None
9. A complete binary tree is:  
   a) All nodes filled  
   b) Height-balanced  
   c) All leaves equal level  
   d) Any binary tree
10. Number of null links in a binary tree with n nodes:  
    a) n  
    b) n-1  
    c) n+1  
    d) 2n

**🔹 Section 5: Agile & Scrum (Q41–50)**

1. Agile delivers:  
   a) Final product only  
   b) Early and continuous delivery  
   c) Strict documentation  
   d) Delayed updates
2. SDLC stands for:  
   a) System Defined Life Cycle  
   b) Software Design Life Cycle  
   c) Software Development Life Cycle  
   d) Software Debug Life Cycle
3. Agile promotes:  
   a) Heavy documentation  
   b) End delivery  
   c) Iterative delivery  
   d) Zero customer interaction
4. Scrum roles include:  
   a) Product Owner, Tester  
   b) Developer, Scrum Master, Product Owner  
   c) DBA, UX  
   d) CEO
5. TRELLO is used for:  
   a) Writing Python code  
   b) Project collaboration  
   c) Database query  
   d) Test automation
6. Traditional model SDLC is:  
   a) Agile  
   b) Waterfall  
   c) Spiral  
   d) Scrum
7. Stand-up meetings in Scrum are:  
   a) Daily  
   b) Weekly  
   c) Monthly  
   d) Annually
8. Which is an Agile methodology?  
   a) Waterfall  
   b) V-Model  
   c) Scrum  
   d) Spiral
9. Sprint duration is usually:  
   a) 1 day  
   b) 1-4 weeks  
   c) 1 month  
   d) 2 months
10. Product backlog is maintained by:  
    a) Scrum Master  
    b) Product Owner  
    c) Developer  
    d) Tester

**🔹 Section 6: SQL + Python + OOP (Q51–70)**

1. SQL JOIN that returns only matching rows:  
   a) LEFT JOIN  
   b) FULL JOIN  
   c) INNER JOIN  
   d) CROSS JOIN
2. 2NF removes:  
   a) Transitive dependency  
   b) Partial dependency  
   c) Multivalued dependency  
   d) Redundancy
3. DROP TABLE removes:  
   a) Rows only  
   b) Structure only  
   c) Rows + Structure  
   d) Indexes only
4. Which clause is used for filtering?  
   a) GROUP BY  
   b) SELECT  
   c) WHERE  
   d) HAVING
5. Python default function arguments must:  
   a) Come first  
   b) Be last  
   c) Be global  
   d) Be required
6. lambda in Python is used for:  
   a) Looping  
   b) Recursion  
   c) Anonymous function  
   d) Decorators
7. Which keyword raises exceptions in Python?  
   a) throw  
   b) raise  
   c) error  
   d) except
8. Which is a Python set method?  
   a) pop()  
   b) get()  
   c) discard()  
   d) append()
9. OOP access specifier for private variable:  
   a) \_var  
   b) \_\_var  
   c) public  
   d) global
10. Polymorphism allows:  
    a) Multiple classes  
    b) Same function name, different behavior  
    c) No inheritance  
    d) One object per class
11. Self in Python represents:  
    a) A class  
    b) A method  
    c) The current object  
    d) Global variable
12. Which file mode in Python opens a file for reading only?  
    a) w  
    b) a  
    c) r  
    d) x
13. Which SQL keyword ensures non-null values?  
    a) CHECK  
    b) NOT NULL  
    c) UNIQUE  
    d) DEFAULT
14. Which SQL clause is used to group rows?  
    a) HAVING  
    b) GROUP BY  
    c) ORDER BY  
    d) SELECT
15. Function to fetch all rows in Python SQL query:  
    a) fetchall()  
    b) fetchone()  
    c) selectall()  
    d) get()
16. What is the output of type([]) in Python?  
    a) tuple  
    b) list  
    c) dict  
    d) set
17. import \* is used to:  
    a) Import selected items  
    b) Import all public names  
    c) Import nothing  
    d) Import private functions
18. Python file object’s read() method returns:  
    a) dict  
    b) list  
    c) string  
    d) int
19. SQL function to return current date:  
    a) GETDATE()  
    b) SYSDATE()  
    c) CURDATE()  
    d) NOW()
20. Python exception for invalid index:  
    a) TypeError  
    b) IndexError  
    c) NameError  
    d) ValueError