16/6/25 MYSQL MCQ QUIZ

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

2. 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

3. 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

4. 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

5. 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

6. 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

7. 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

8. 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

9. 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

10. 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

11. 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

12. 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

13. 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

14. 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

15. 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

16. 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

17. 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

18. 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

19. 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

20. 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

21. 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

22. 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

23. 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

24. 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

25. 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

26. 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

27. 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

28. 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

29. 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

30. 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

31. 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

32. 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

33. 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

34. 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

35. 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

36. 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

37. 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

38. 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

39. 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

40. 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

41. 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

42. 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

43. 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

44. 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

45. 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

46. 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

47. 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

48. 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

16/6/25 Practise Question

Instructions:

- Answer all questions using **MySQL**.
- Use appropriate **subqueries**, **joins**, and **aggregate functions** wherever applicable.
- Make sure to use proper **aliasing**, **GROUP BY**, **HAVING**, **DISTINCT**, etc., as needed.
- 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.

```
SELECT Name
FROM Customers
WHERE NOT EXISTS (
SELECT 1
FROM (
SELECT DISTINCT MONTH(OrderDate) AS M
FROM Orders
WHERE YEAR(OrderDate) = YEAR(CURDATE())
) AS months
WHERE NOT EXISTS (
SELECT 1 FROM Orders
WHERE Orders.CustomerID = Customers.CustomerID
AND YEAR(OrderDate)=YEAR(CURDATE())
AND MONTH(OrderDate)=months.M ));
```

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

```
SELECT ProductName
FROM Products
WHERE (SELECT AVG(Quantity) FROM OrderDetails) < (
   SELECT SUM(Quantity)
   FROM OrderDetails
   WHERE ProductID = Products.ProductID);
```

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

```
SELECT Name
FROM Customers
WHERE NOT EXISTS (
   SELECT 1 FROM Orders
   JOIN OrderDetails USING (OrderID)
   JOIN Products USING (ProductID)
   WHERE Customers.CustomerID = Orders.CustomerID
     AND Products.Price > 1000);
```

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

```
SELECT Name
FROM Customers
WHERE NOT EXISTS (
   SELECT 1 FROM Orders
   JOIN OrderDetails USING (OrderID)
   JOIN Products USING (ProductID)
   WHERE Customers.CustomerID = Orders.CustomerID
     AND Products.Price > 1000 );
```

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

```
SELECT OrderID
FROM Orders o
WHERE (SELECT COUNT(*) FROM OrderDetails WHERE OrderID = o.OrderID)
= 1;
```

Part B – Correlated & Nested Subqueries (25 marks)

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

```
SELECT Name
FROM Customers
WHERE CustomerID IN (
   SELECT CustomerID FROM Orders
   WHERE OrderDate IN (
 SELECT OrderDate FROM Orders JOIN Customers USING (CustomerID) WHERE
Name='John' ));
```

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

```
SELECT Name

FROM Customers

WHERE CustomerID = (

   SELECT CustomerID FROM Orders ORDER BY OrderDate DESC LIMIT 1);
```

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

```
SELECT ProductName

FROM Products

WHERE Price = (

   SELECT MIN(Price) FROM Products WHERE Price > (SELECT MIN(Price)
FROM Products));
```

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

```
SELECT Name

FROM Customers

HAVING SUM((Quantity * Price)) > 2 * (SELECT AVG(total) FROM (

   SELECT SUM(Quantity * Price) AS total FROM Orders

   JOIN OrderDetails USING (OrderID)

   JOIN Products USING (ProductID)

   GROUP BY CustomerID) AS T);
```

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

```
SELECT Name

FROM Customers

HAVING SUM((Quantity * Price)) > ANY (

   SELECT SUM((Quantity * Price)) FROM Customers

   JOIN Orders USING (CustomerID)

   JOIN OrderDetails USING (OrderID)

   JOIN Products USING (ProductID)

   WHERE City='Delhi'

   GROUP BY CustomerID);
```

Part C – Join + Subquery Mix (30 marks)

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

```
SELECT Name
FROM Customers
HAVING COUNT(Orders.OrderID) > (SELECT AVG(cnt) FROM (SELECT CustomerID, COUNT(OrderID) AS cnt FROM Orders GROUP BY CustomerID) AS T);
```

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

```
SELECT ProductName
FROM Products
HAVING SUM(Quantity) > (SELECT AVG(sum_quantity) FROM (SELECT ProductID, SUM(Quantity) AS sum_quantity FROM OrderDetails GROUP BY ProductID) AS T);
```

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

```
SELECT Name
FROM Customers
WHERE EXISTS (
   SELECT 1 FROM Orders o
   JOIN OrderDetails od USING (OrderID)
   WHERE o.CustomerID = Customers.CustomerID
    AND od.ProductID NOT IN (
    SELECT ProductID FROM OrderDetails GROUP BY ProductID HAVING COUNT(DISTINCT OrderID) > 1 ));
```

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

```
SELECT o.*
FROM Orders o
WHERE o.Amount = (
   SELECT MAX(Amount) FROM Orders WHERE CustomerID = o.CustomerID);
```

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

```
SELECT Name
FROM Customers
WHERE NOT EXISTS (
   SELECT 1 FROM Orders o
   JOIN OrderDetails od USING (OrderID)
WHERE o.CustomerID = Customers.CustomerID
 AND od.Quantity > 5);
```

Part D – Joins & Set Operations (25 marks)

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

```
SELECT Name
FROM Customers
ORDER BY SUM((Quantity * Price)) DESC LIMIT 5;
```

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

```
SELECT Name
FROM Customers
HAVING COUNT(DISTINCT ProductID) = 1;
```

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

```
SELECT o.*
FROM Orders o
WHERE o.Amount NOT IN (SELECT Amount FROM Orders ORDER BY Amount DESC LIMIT 10);
```

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

```
SELECT Name
FROM Customers
WHERE EXISTS (
   SELECT 1 FROM Orders o
   WHERE o.CustomerID = Customers.CustomerID
    AND o.OrderDate >= CURDATE() - INTERVAL 7 DAY)
AND NOT EXISTS (
   SELECT 1 FROM Orders o
   WHERE o.CustomerID = Customers.CustomerID
    AND o.OrderDate < CURDATE()
    AND o.OrderDate >= CURDATE() - INTERVAL 30 DAY);
```

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

```
SELECT ProductName
FROM Products
HAVING COUNT(DISTINCT OrderID) = (
SELECT MAX(cnt) FROM (SELECT ProductID, COUNT(DISTINCT OrderID)
AS cnt FROM OrderDetails GROUP BY ProductID) AS T;
```

18/6/25 Python Training Assignment

SECTION A - TUPLES

1.CREATE TUPLE

```
my_tuple = ("Harcini", 20, "AI", "DS")
print("Created Tuple:", my_tuple)
```

2. ACCESS TUPLE

```
print("First item:", my_tuple[0])
print("Last item:", my_tuple[-1])
```

3. LOOP TUPLE

```
print("Looping through tuple:")
for item in my_tuple:
    print(item)
```

4. RANGE TUPLE

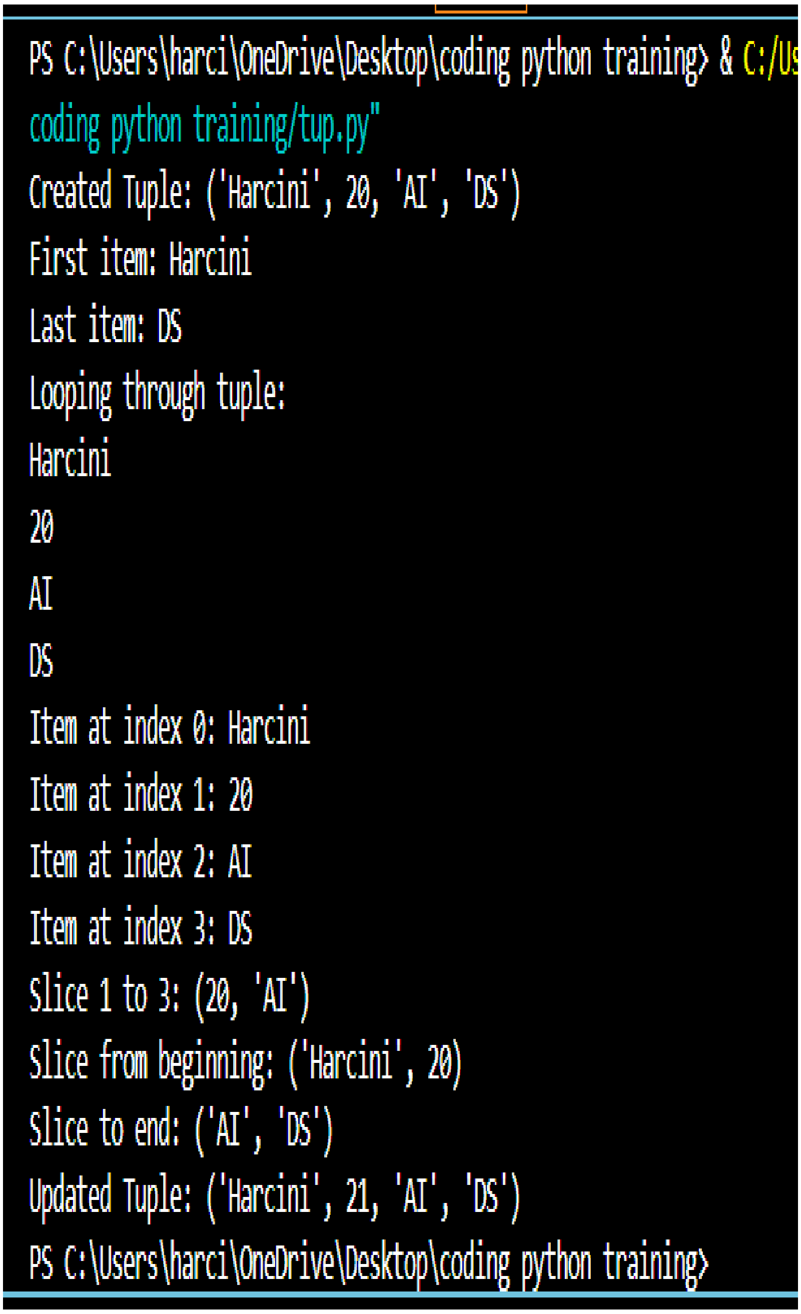
```
for i in range(len(my_tuple)):
 print(f"Item at index {i}:", my_tuple[i])
```

5. SLICE TUPLE

```
print("Slice 1 to 3:", my_tuple[1:3])
print("Slice from beginning:", my_tuple[:2])
print("Slice to end:", my_tuple[2:])
```

6. UPDATE ITEMS IN TUPLE

```
temp = list(my_tuple)
temp[1] = 21
my_tuple = tuple(temp)
print("Updated Tuple:", my_tuple)
```



```
PS C:\Users\harci\OneDrive\Desktop\coding python training> & C:/Us
coding python training/tup.py"
Created Tuple: ('Harcini', 20, 'AI', 'DS')
First item: Harcini
Last item: DS
Looping through tuple:
Harcini
20
AI
DS
Item at index 0: Harcini
Item at index 1: 20
Item at index 2: AI
Item at index 3: DS
Slice 1 to 3: (20, 'AI')
Slice from beginning: ('Harcini', 20)
Slice to end: ('AI', 'DS')
Updated Tuple: ('Harcini', 21, 'AI', 'DS')
PS C:\Users\harci\OneDrive\Desktop\coding python training>
```

SECTION B – LIST

Part -A

1. DECLARE LIST

```
fruits = ["apple", "banana", "cherry"]
print("Original List:", fruits)
```

2. SORT LIST

```
fruits.sort()
print("Sorted List:", fruits)
```

3. INSERT LIST

```
fruits.insert(1, "orange")
print("After Insert:", fruits)
```

4. REMOVE LIST

```
fruits.remove("banana")
print("After Remove:", fruits)
```

5. JOIN LIST

```
more_fruits = ["grape", "mango"]
combined = fruits + more_fruits
print("After Joining:", combined)
```

6. CHANGE LIST

```
combined[0] = "kiwi"
print("After Changing:", combined)
```

7. Access list

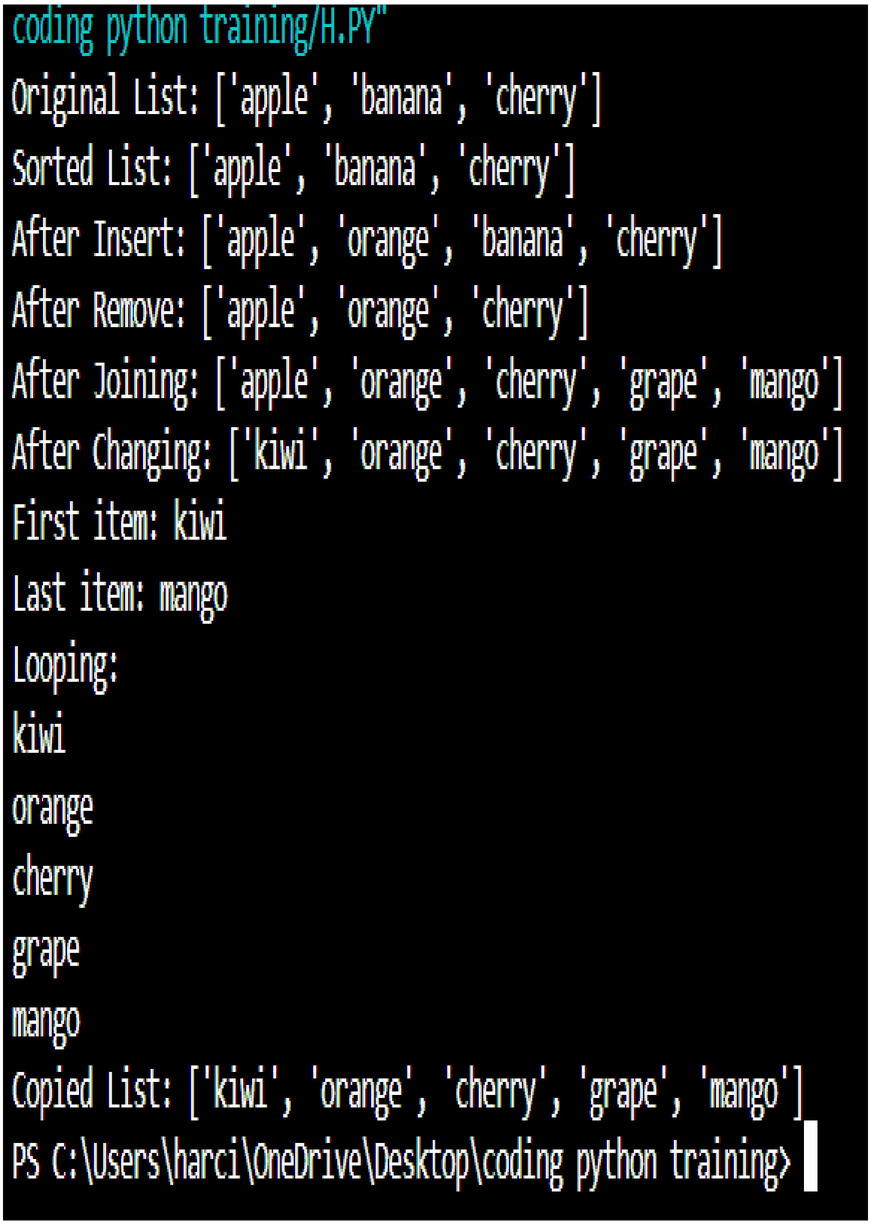
```
print("First item:", combined[0])
print("Last item:", combined[-1])
```

8. LOOP LIST

```
print("Looping:")
for fruit in combined:
    print(fruit)
```

9. COPY LIST

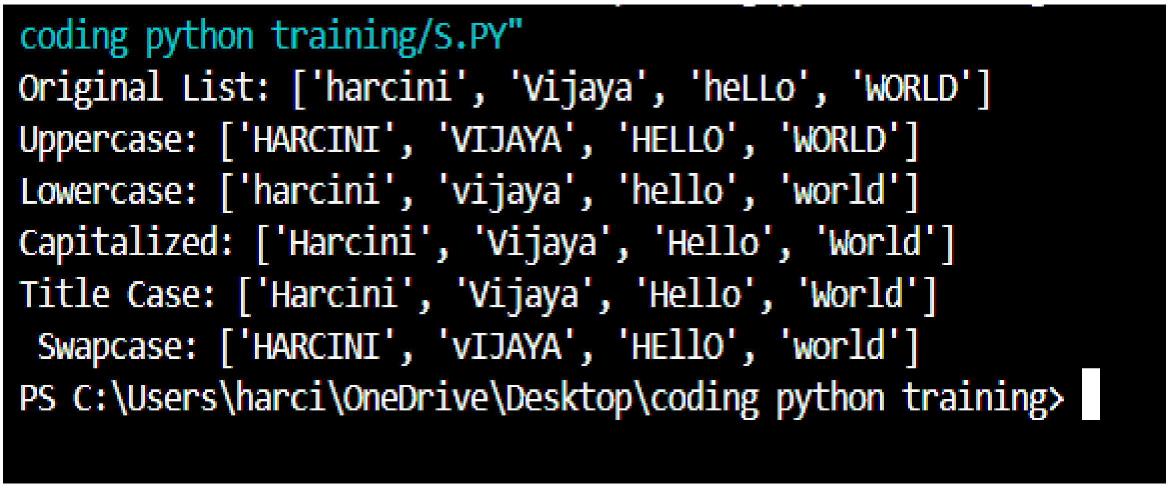
```
copy_list = combined.copy()
print("Copied List:", copy_list)
```



```
coding python training/H.PY"
Original List: ['apple', 'banana', 'cherry']
Sorted List: ['apple', 'banana', 'cherry']
After Insert: ['apple', 'orange', 'banana', 'cherry']
After Remove: ['apple', 'orange', 'cherry']
After Joining: ['apple', 'orange', 'cherry', 'grape', 'mango']
After Changing: ['kiwi', 'orange', 'cherry', 'grape', 'mango']
First item: kiwi
Last item: mango
Looping:
kiwi
orange
cherry
grape
mango
Copied List: ['kiwi', 'orange', 'cherry', 'grape', 'mango']
PS C:\Users\harci\OneDrive\Desktop\coding python training>
```

10. STRING METHODS IN LIST

```
# Original list of names
names = ["harcini", "Vijaya", "heLLo", "WORLD"]
print("Original List:", names)

#  Convert all to uppercase
upper_names = [name.upper() for name in names]
print("Uppercase:", upper_names)

# Convert all to lowercase
lower_names = [name.lower() for name in names]
print("Lowercase:", lower_names)

# Capitalize first letter only (rest lowercase)
capitalized_names = [name.capitalize() for name in names]
print("Capitalized:", capitalized_names)

# Title case (first letter of each word in uppercase)
title_names = [name.title() for name in names]
print("Title Case:", title_names)

# Swap case (uppercase becomes lowercase and vice versa)
swapped_names = [name.swapcase() for name in names]
print(" Swapcase:", swapped_names)
```



```
coding python training/S.PY"
Original List: ['harcini', 'Vijaya', 'heLLo', 'WORLD']
Uppercase: ['HARCINI', 'VIJAYA', 'HELLO', 'WORLD']
Lowercase: ['harcini', 'vijaya', 'hello', 'world']
Capitalized: ['Harcini', 'Vijaya', 'Hello', 'World']
Title Case: ['Harcini', 'Vijaya', 'Hello', 'World']
 Swapcase: ['HARCINI', 'vIJAYA', 'HEllO', 'world']
PS C:\Users\harci\OneDrive\Desktop\coding python training>
```

Part-B

1. **CREATES A LIST**

```
fruits = ["apple", "banana", "cherry", "mango"]
print("Original list:", fruits)
```

2. **PRINTS A SPECIFIC INDEX**

```
print("Item at index 2:", fruits[2])
```

3. **CHANGES AN ITEM**

```
fruits[1] = "orange"
print("After changing index 1:", fruits)
```

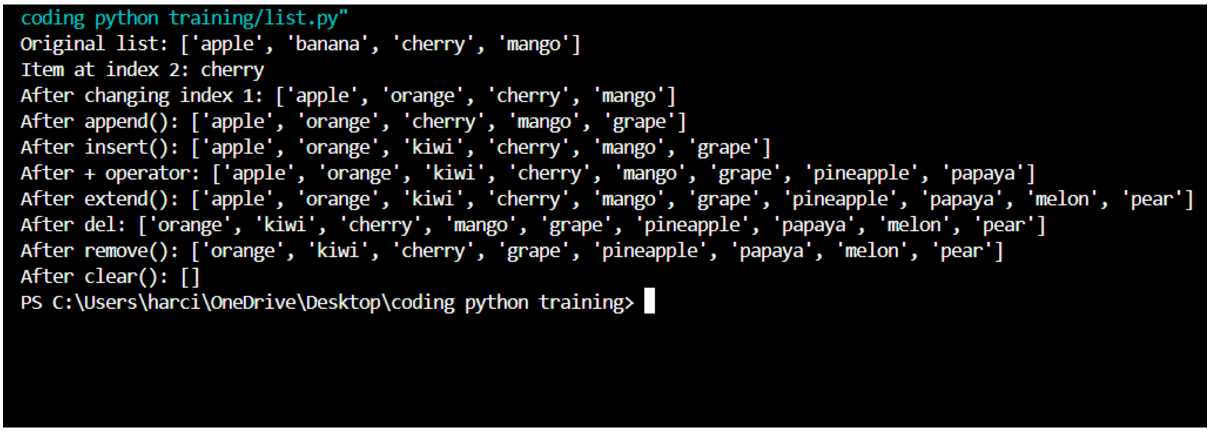
4. **APPENDS IN MULTIPLE WAYS**

```
# a. Using append()
fruits.append("grape")
print("After append():", fruits)

# b. Using insert(index, value)
fruits.insert(2, "kiwi")  # Inserts 'kiwi' at index 2
print("After insert():", fruits)

# c. Using + operator
more_fruits = ["pineapple", "papaya"]
fruits = fruits + more_fruits
print("After + operator:", fruits)

# d. Using extend()
fruits.extend(["melon", "pear"])
print("After extend():", fruits)
```

5. **Removes items using del, remove(), clear()**

```
5. Remove items in all 3 ways

# a. Using del to delete by index
del fruits[0]
print("After del:", fruits)
```

```
# b. Using remove() to remove by value
fruits.remove("mango")
print("After remove():", fruits)

# c. Using clear() to remove all items
fruits.clear()
print("After clear():", fruits)
```

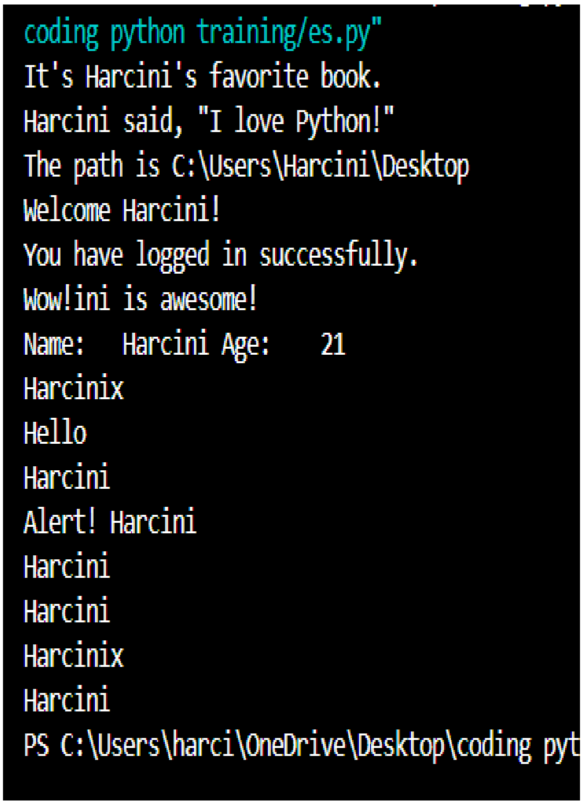


```
coding python training/list.py"
Original list: ['apple', 'banana', 'cherry', 'mango']
Item at index 2: cherry
After changing index 1: ['apple', 'orange', 'cherry', 'mango']
After append(): ['apple', 'orange', 'cherry', 'mango', 'grape']
After insert(): ['apple', 'orange', 'kiwi', 'cherry', 'mango', 'grape']
After + operator: ['apple', 'orange', 'kiwi', 'cherry', 'mango', 'grape', 'pineapple', 'papaya']
After extend(): ['apple', 'orange', 'kiwi', 'cherry', 'mango', 'grape', 'pineapple', 'papaya', 'melon', 'pear']
After del: ['orange', 'kiwi', 'cherry', 'mango', 'grape', 'pineapple', 'papaya', 'melon', 'pear']
After remove(): ['orange', 'kiwi', 'cherry', 'grape', 'pineapple', 'papaya', 'melon', 'pear']
After clear(): []
PS C:\Users\harci\OneDrive\Desktop\coding python training> 
```

SECTION-C STRING

1. ESCAPE CHARACTERS

```
# Escape characters demo using "Harcini"
print('It\'s Harcini\'s favorite book.')
print("Harcini said, \"I love Python!\"")
print("The path is C:\\Users\\Harcini\\Desktop

# Newline (\n)
print("Welcome Harcini!\nYou have logged in successfully.")

# Carriage return (\r) - replaces start of line with what's after \r
print("Harcini is awesome!\rWow!")
```



```
coding python training/es.py"
It's Harcini's favorite book.
Harcini said, "I love Python!"
The path is C:\Users\Harcini\Desktop
Welcome Harcini!
You have logged in successfully.
Wow!ini is awesome!
Name:   Harcini Age:    21
Harcinix
Hello
Harcini
Alert! Harcini
Harcini
Harcini
Harcinix
Harcini
PS C:\Users\harci\OneDrive\Desktop\coding pyt
```

# Tab (\t)

print("Name:\tHarcini\tAge:\t21")

# Backspace (\b) - deletes the last character

print("Harcinix\b")

# Octal representation of "Harcini"

print("\110\141\162\143\151\156\151")

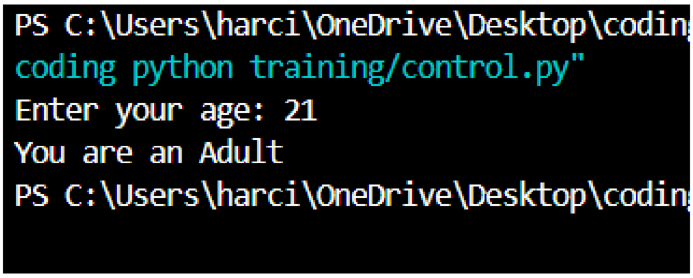
# Hexadecimal representation of "Harcini"

print("\x48\x61\x72\x63\x69\x6e\x69")

SECTION-D CONTROL STRUCTURE

1. **TASK:- KID, TEEN, ADULT, OLD KID <12 TEEN :< KID TO 18 ADULT 18 TO 60**

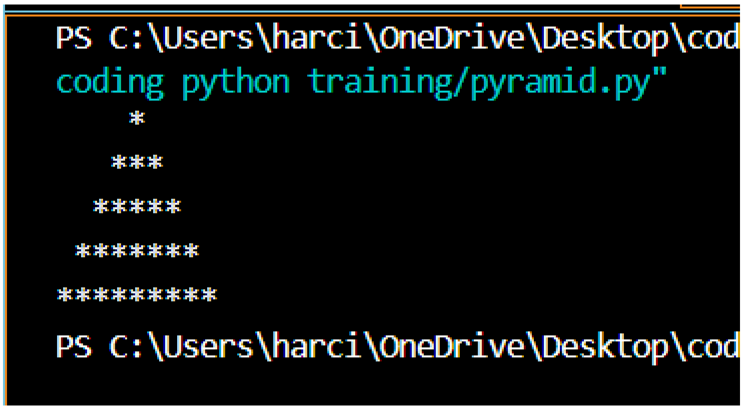
```
if age < 12:
   print("You are a Kid ")
elif age < 18:
   print("You are a Teen ")
elif age <= 60:
   print("You are an Adult ")
print("You are Old ")
```



```
PS C:\Users\harci\OneDrive\Desktop\codin
coding python training/control.py"
Enter your age: 21
You are an Adult
PS C:\Users\harci\OneDrive\Desktop\codin
```

2.STAR PYRAMID

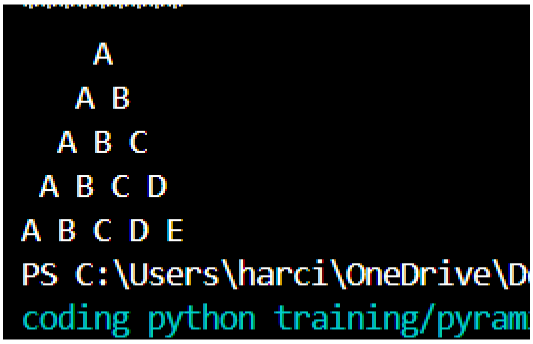
```
rows = 5
for i in range(rows):
   spaces = ' ' * (rows - i - 1)
   stars = '*' * (2 * i + 1)
print(spaces + stars)
```



```
PS C:\Users\harci\OneDrive\Desktop\cod
coding python training/pyramid.py"
    *
   ***
  *****
 *******
*********
PS C:\Users\harci\OneDrive\Desktop\cod
```

3.ALPHABET PYRAMID

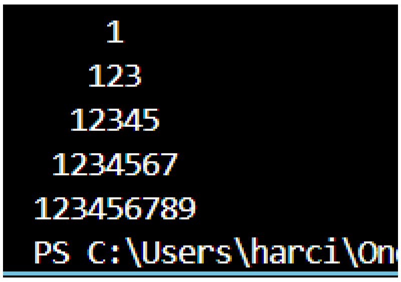
```
rows = 5
for i in range(rows):
    chars = ' '.join(chr(65 + j) for j in range(i + 1))
    print(chars.center(2 * rows - 1))
```



```
    A
   A B
  A B C
 A B C D
A B C D E
PS C:\Users\harci\OneDrive\D
coding python training/pyram
```

4.NUMBER PYRAMID

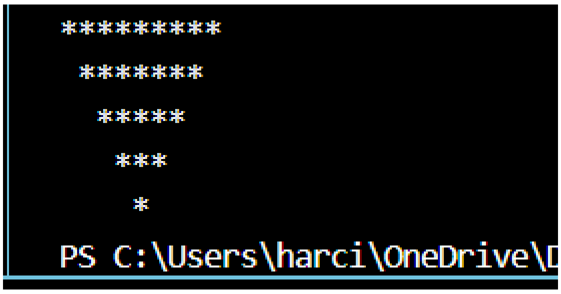
```
rows = 5
for i in range(rows):
    numbers = ''.join(str(j) for j in range(1, 2 * i + 2, 1))
    print(numbers.center(2 * rows - 1))
```



```
    1
   123
  12345
 1234567
123456789
PS C:\Users\harci\On
```

5.INVERTED STAR PYRAMID

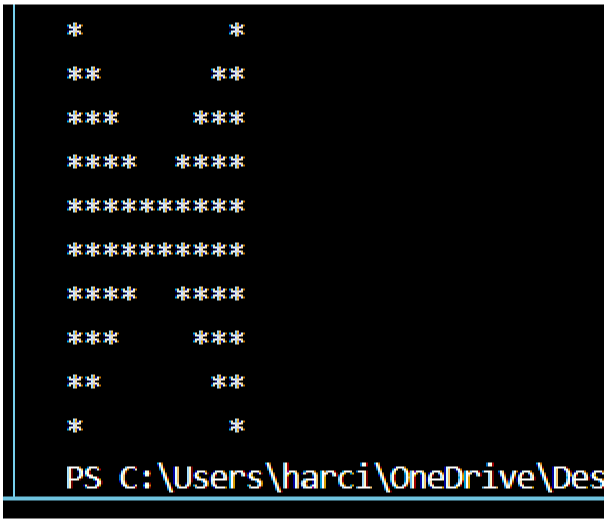
```
rows = 5
for i in range(rows):
    spaces = ' ' * i
    stars = '*' * (2 * (rows - i) - 1)
    print(spaces + stars)
```



```
*********
 *******
  *****
   ***
    *

PS C:\Users\harci\OneDrive\D
```

6.BUTTERFLY PATTERN

```
rows = 5
for i in range(1, rows + 1):
    stars = '*' * i
    spaces = ' ' * (2 * (rows - i))
    print(stars + spaces + stars)
for i in range(rows, 0, -1):
    stars = '*' * i
   spaces = ' ' * (2 * (rows - i))
    print(stars + spaces + stars)
```

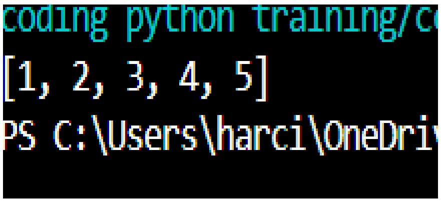


```
*        *
**      **
***    ***
****  ****
**********
**********
****  ****
***    ***
**      **
*        *

PS C:\Users\harci\OneDrive\Des
```

Python Coding Challenge Topic: List, Tuple, Dictionary, Set

Q1. Write a Python program to remove all duplicates from a list without using the set() function.
Input Example: [1, 2, 2, 3, 4, 4, 5] Output: [1, 2, 3, 4, 5]

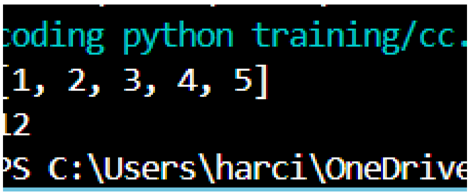
```
lst = [1, 2, 2, 3, 4, 4, 5]
unique = []
for item in lst:
    if item not in unique:
        unique.append(item)
print(unique)
```



```
coding python training/c
[1, 2, 3, 4, 5]
PS C:\Users\harci\OneDri
```

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

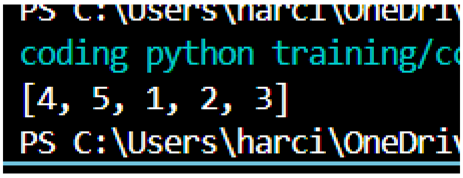
```
lst = [12, 5, 9, 21, 21, 3]
unique = list(set(lst))
unique.sort(reverse=True)
print(unique[1])
```



```
coding python training/cc.
[1, 2, 3, 4, 5]
12
PS C:\Users\harci\OneDrive
```

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]

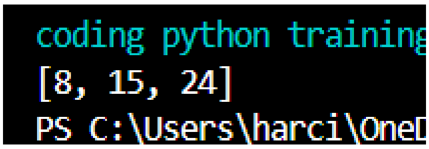
```
lst = [1, 2, 3, 4, 5]
k = 2
k = k % len(lst)  # In case k > len
rotated = lst[-k:] + lst[:-k]
print(rotated)
```



```
PS C:\Users\harci\OneDri
coding python training/c
[4, 5, 1, 2, 3]
PS C:\Users\harci\OneDri
```

Q4. Write a Python program to multiply the elements of each tuple in a list of tuples and return a new list
Input: [(2, 4), (3, 5), (4, 6)] Output: [8, 15, 24]

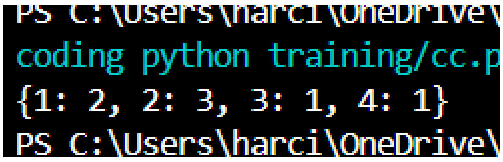
```
tpl_list = [(2, 4), (3, 5), (4, 6)]
result = [a * b for a, b in tpl_list]
print(result)
```



```
coding python training
[8, 15, 24]
PS C:\Users\harci\OneD
```

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}

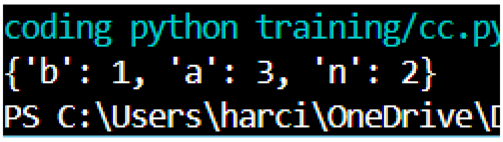
```
tpl = (1, 2, 2, 3, 1, 4, 2)
freq = {}
for item in tpl:
    freq[item] = freq.get(item, 0) + 1
print(freq)
```



```
PS C:\Users\harci\OneDrive\
coding python training/cc.p
{1: 2, 2: 3, 3: 1, 4: 1}
PS C:\Users\harci\OneDrive\
```

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}

```
text = 'banana'
freq = {}
for char in text:
freq[char] = freq.get(char, 0) + 1
print(freq)
```

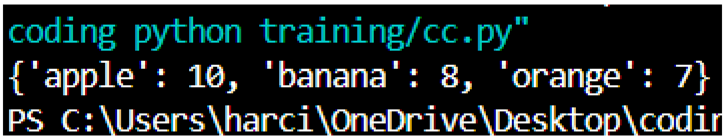


```
coding python training/cc.py
{'b': 1, 'a': 3, 'n': 2}
PS C:\Users\harci\OneDrive\D
```

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}

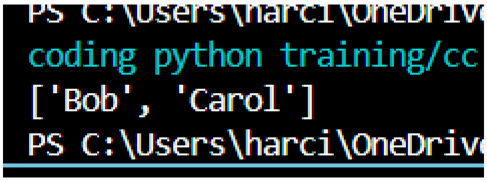
```
d1 = {'apple': 10, 'banana': 5}
d2 = {'banana': 3, 'orange': 7}
merged = d1.copy()
for key, value in d2.items():
    merged[key] = merged.get(key, 0) + value
print(merged)
```



```
coding python training/cc.py"
{'apple': 10, 'banana': 8, 'orange': 7}
PS C:\Users\harci\OneDrive\Desktop\codin
```

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']

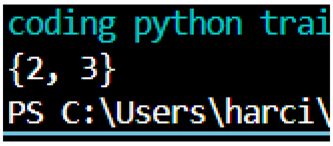
```
marks = {'Alice': 85, 'Bob': 92, 'Carol': 92}
max_score = max(marks.values())
top_students = [name for name, score in marks.items() if score == max_score]
print(top_students)
```



```
PS C:\Users\harci\OneDriv
coding python training/cc
['Bob', 'Carol']
PS C:\Users\harci\OneDriv
```

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}

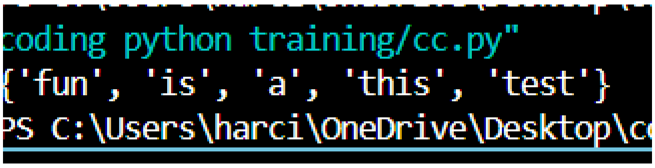
```
a = [1, 2, 3]
b = [2, 3, 4]
c = [3, 2, 5]
common = set(a) & set(b) & set(c)
print(common)
```



```
coding python trai
{2, 3}
PS C:\Users\harci\
```

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'}

```
sentence = 'this is a test this is fun'
words = set(sentence.split())
print(words)
```



```
coding python training/cc.py"
{'fun', 'is', 'a', 'this', 'test'}
PS C:\Users\harci\OneDrive\Desktop\c
```

20.6.25

Python Coding Task

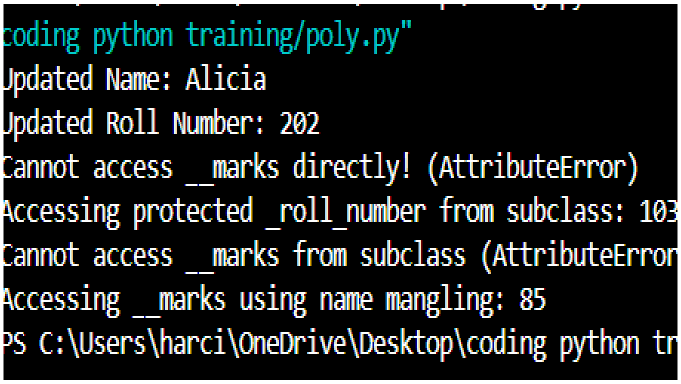
Q1. Understanding Access Specifiers

Create a class `Student` with the following properties:

Class Requirements:
1. `name` → Public attribute
2. `_roll_number` → Protected attribute
3. `__marks` → Private attribute

Implement the following methods:
- Constructor to initialize all attributes.
- `display_details()` → Public method to display all attribute values.
- `_update_roll_number(new_roll)` → Protected method to update roll number.
- `__update_marks(new_marks)` → Private method to update marks.
- `access_private_method(new_marks)` → Public method that uses the private method `__update_marks`.

```
class Student:

    def __init__(self, name, roll_number, marks):

        self.name = name

        self._roll_number = roll_number

        self.__marks = marks

    def display_details(self):

        print(f"Name: {self.name}")

        print(f"Roll Number: {self._roll_number}")

        print(f"Marks: {self.__marks}")

    def _update_roll_number(self, new_roll):

        self._roll_number = new_roll
```

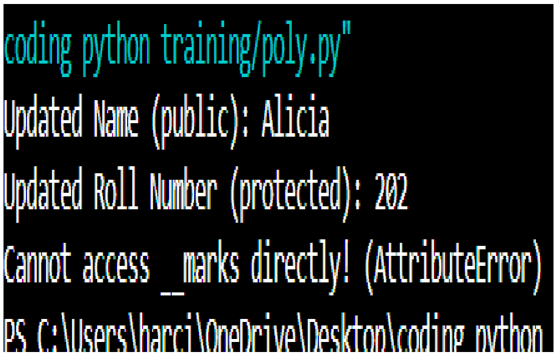


```
coding python training/poly.py"
Updated Name: Alicia
Updated Roll Number: 202
Cannot access __marks directly! (AttributeError)
Accessing protected _roll_number from subclass: 103
Cannot access __marks from subclass (AttributeError
Accessing __marks using name mangling: 85
PS C:\Users\harci\OneDrive\Desktop\coding python tr
```

```
    def __update_marks(self, new_marks):
        self.__marks = new_marks
    def access_private_method(self, new_marks):
        self.__update_marks(new_marks)
class Topper(Student):
    def try_access(self):
        print("Accessing protected _roll_number from subclass:", self._roll_number)
        try:
            print("Accessing private __marks from subclass:", self.__marks)
        except AttributeError:
            print("Cannot access __marks from subclass (AttributeError)")
s1 = Student("Alice", 101, 85)
s1.name = "Alicia"
print("Updated Name:", s1.name)
s1._roll_number = 202
print("Updated Roll Number:", s1._roll_number)
try:
    print("Marks:", s1.__marks)
except AttributeError:
    print("Cannot access __marks directly! (AttributeError)")
t1 = Topper("Bob", 103, 95)
t1.try_access()
print("Accessing __marks using name mangling:", s1._Student__marks)
```

Q2. Demonstrate Access

In the main section:
- Create an object of the `Student` class.
- Modify and print the `name` directly.
- Modify and print the `_roll_number` directly.
- Try accessing `__marks` directly and observe the result.



```
coding python training/poly.py"
Updated Name (public): Alicia
Updated Roll Number (protected): 202
Cannot access __marks directly! (AttributeError)
PS C:\Users\harci\OneDrive\Desktop\coding python
```

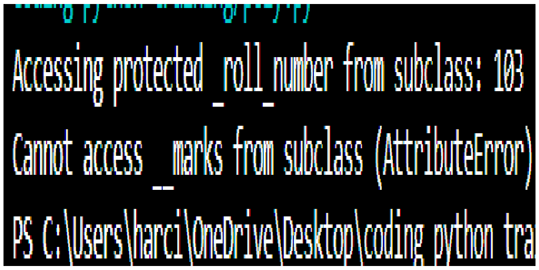
```
class Student:

    def __init__(self, name, roll_number, marks):

        self.name = name

        self._roll_number = roll_number

        self.__marks = marks

    def display_details(self):

        print(f"Name: {self.name}")

        print(f"Roll Number: {self._roll_number}")

        print(f"Marks: {self.__marks}")

s1 = Student("Alice", 101, 85)

s1.name = "Alicia"

print("Updated Name (public):", s1.name)

s1._roll_number = 202

print("Updated Roll Number (protected):", s1._roll_number)

try:

    print("Marks (private):", s1.__marks)

except AttributeError:

    print("Cannot access __marks directly! (AttributeError)")
```

Q3. Inheritance and Access Control

Create a subclass `Topper` that inherits from `Student` and includes:

- A method `try_access()` that attempts to access `_roll_number` and `__marks` from the subclass.
- Show what works and what doesn't.

```
class Student:
  def __init__(self, name, roll_number, marks):
    self.name = name               # Public
    self._roll_number = roll_number  # Protected
    self.__marks = marks           # Private
  def display_details(self):
    print(f"Name: {self.name}")
    print(f"Roll Number: {self._roll_number}")
    print(f"Marks: {self.__marks}")
class Topper(Student):
  def try_access(self):
    # Accessing protected attribute
    print(" Accessing protected _roll_number from subclass:", self._roll_number)
    try:
      print("Trying to access private __marks from subclass:", self.__marks)
    except AttributeError:
      print("Cannot access __marks from subclass (AttributeError)")
```

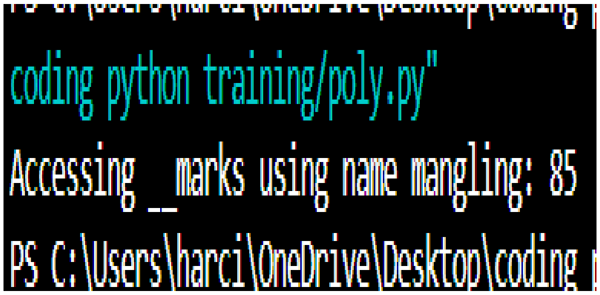


```
Accessing protected _roll_number from subclass: 103
Cannot access __marks from subclass (AttributeError)
PS C:\Users\harci\OneDrive\Desktop\coding python tra
```

Q4. Use of Name Mangling

Demonstrate how to access the private attribute `__marks` using name mangling technique from outside the class.

```
class Student:

    def __init__(self, name, roll_number, marks):

        self.name = name

        self._roll_number = roll_number

        self.__marks = marks

    def display_details(self):

        print(f"Name: {self.name}")

        print(f"Roll Number: {self._roll_number}")

        print(f"Marks: {self.__marks}")

s1 = Student("Alice", 101, 85)

print("Accessing __marks using name mangling:", s1._Student__marks)
```



```
coding python training/poly.py"
Accessing __marks using name mangling: 85
PS C:\Users\harci\OneDrive\Desktop\coding
```

Q5. Reflection

Answer the following short questions:
1. Why can't private members be accessed directly?

Private members are meant to hide internal details of a class to protect data and prevent accidental modification from outside.
In Python, private members (those with names starting with __) are name-mangled to make them **inaccessible from outside the class**, enforcing **encapsulation** and safe data handling.

2. What is the purpose of using protected members in class design?

Protected members (with a single underscore _) allow access within the class and its subclasses, but discourage direct access from outside the class.
They are used when a variable or method is meant for internal use and **might be needed by subclasses**, but should **not be modified directly** by external code.

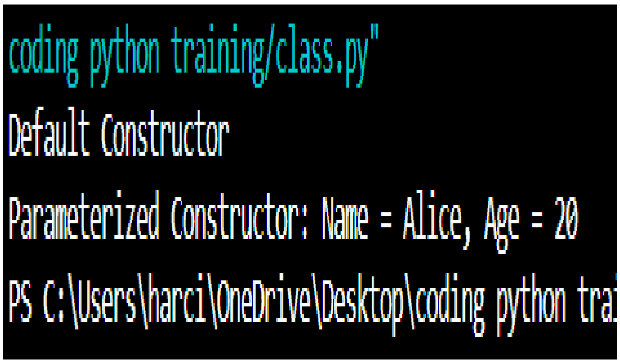
3. How does name mangling help with private members in Python?

Name mangling automatically changes the name of private attributes (e.g., __marks becomes _ClassName__marks) to prevent direct access.
This allows the class to **hide internal variables**, but still **provides a way to access them intentionally** (e.g., for debugging or testing), without exposing them publicly.

20.6.25 CLASSS ASSIGNMENTS

1. how to default n parameterized constructor together in single class

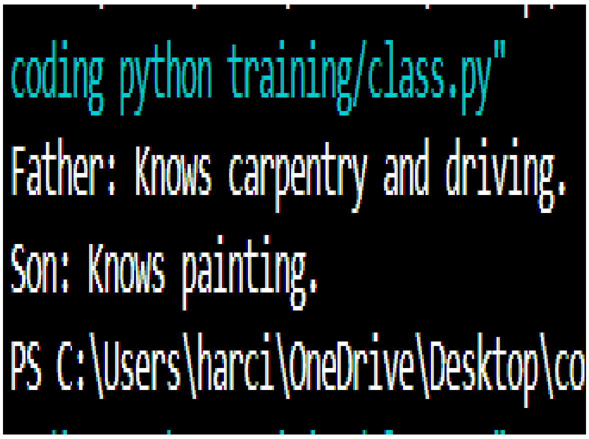
```
class Student:
   def __init__(self, name=None, age=None):
      if name and age:
    print(f"Parameterized Constructor: Name =
   {name},
         Age = {age}")
      else:
         print("Default Constructor")
# Default constructor
s1 = Student()
# Parameterized constructor
s2 = Student("Alice", 20)
```



```
coding python training/class.py"
Default Constructor
Parameterized Constructor: Name = Alice, Age = 20
PS C:\Users\harci\OneDrive\Desktop\coding python trai
```

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

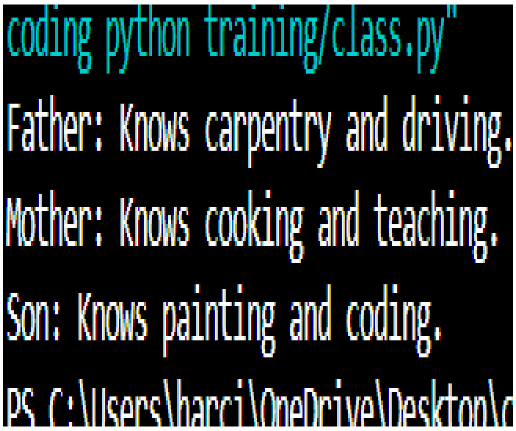
```
 class Father:
   def skills(self):
      print("Father: Knows carpentry and
driving.")
class Son(Father):
   def own_skills(self):
      print("Son: Knows painting.")
# Create object
s = Son()
s.skills()
s.own_skills()
```



```
coding python training/class.py"
Father: Knows carpentry and driving.
Son: Knows painting.
PS C:\Users\harci\OneDrive\Desktop\co
```

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

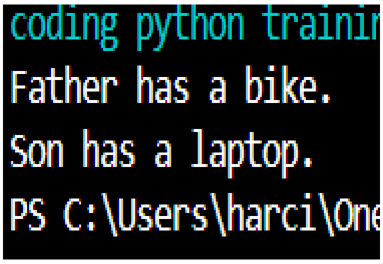
```
class Father:
    def father_skills(self):
        print("Father: Knows carpentry and driving.")
class Mother:
    def mother_skills(self):
        print("Mother: Knows cooking and teaching.")
class Son(Father, Mother):
    def own_skills(self):
        print("Son: Knows painting and coding.")
# Create object
s = Son()
s.father_skills()     # Inherited from Father
s.mother_skills()
s.own_skills()
```



```
coding python training/class.py"
Father: Knows carpentry and driving.
Mother: Knows cooking and teaching.
Son: Knows painting and coding.
PS C:\Users\harci\OneDrive\Desktop\c
```

4. Single Inheritance

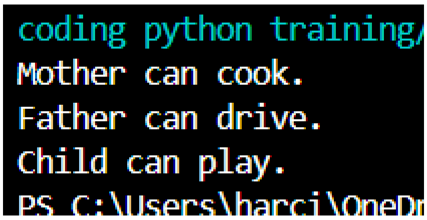
```
class Father:
    def bike(self):
        print("Father has a bike.")
class Son(Father):
    def laptop(self):
        print("Son has a laptop.")
s = Son()
s.bike()
s.laptop()
```



```
coding python trainir
Father has a bike.
Son has a laptop.
PS C:\Users\harci\One
```

5. Multiple Inheritance

```
class Mother:
    def cook(self):
        print("Mother can cook.")
class Father:
    def drive(self):
        print("Father can drive.")
```

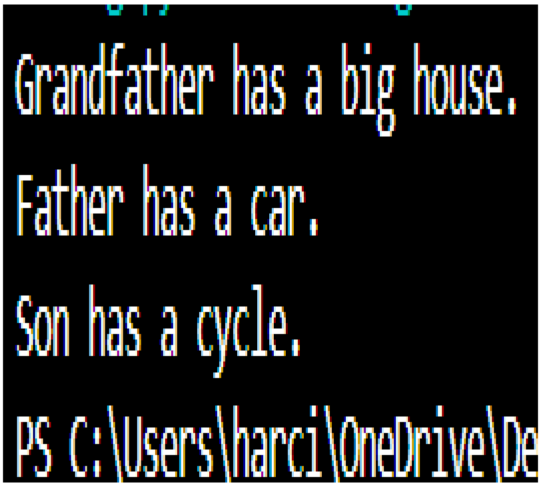


```
coding python training/
Mother can cook.
Father can drive.
Child can play.
PS C:\Users\harci\OneDr
```

```
class Child(Mother, Father):
    def play(self):
        print("Child can play.")
c = Child()
c.cook()
c.drive()
c.play()
```

6. Multilevel Inheritance

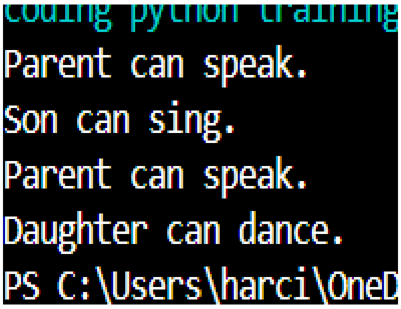
```
class Grandfather:
    def house(self):
        print("Grandfather has a big house.")
class Father(Grandfather):
    def car(self):
        print("Father has a car.")
class Son(Father):
    def cycle(self):
        print("Son has a cycle.")
s = Son()
s.house()
s.car()
s.cycle()
```



```
Grandfather has a big house.
Father has a car.
Son has a cycle.
PS C:\Users\harci\OneDrive\De
```

7. Hierarchical Inheritance

```
class Parent:
    def speak(self):
        print("Parent can speak.")
class Son(Parent):
    def sing(self):
        print("Son can sing.")
class Daughter(Parent):
    def dance(self):
```

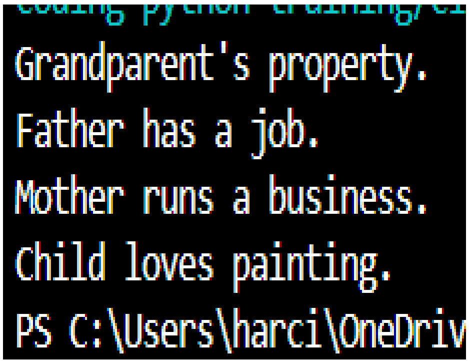


```
coding python training
Parent can speak.
Son can sing.
Parent can speak.
Daughter can dance.
PS C:\Users\harci\OneD
```

```
        print("Daughter can dance.")
s = Son()
d = Daughter()
s.speak()
s.sing()
d.speak()
d.dance()
```

8. Hybrid Inheritance

```
class Grandparent:
    def property(self):
        print("Grandparent's property.")
class Father(Grandparent):
    def job(self):
        print("Father has a job.")
class Mother:
    def business(self):
        print("Mother runs a business.")
class Child(Father, Mother):
    def hobby(self):
        print("Child loves painting.")
c = Child()
c.property()
c.job()
c.business()
c.hobby()
```



```
Grandparent's property.
Father has a job.
Mother runs a business.
Child loves painting.
PS C:\Users\harci\OneDriv
```

Python Question Paper

Subject: Python Programming

Topic: File Handling

Total Questions: 10

Instructions:
- Write Python programs to solve the following problems.
- Use appropriate file handling modes and exception handling where necessary.

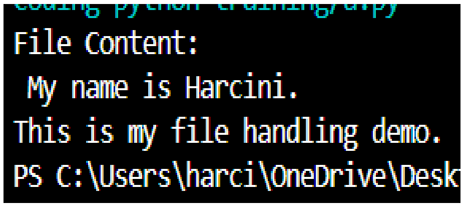
Section A: Basic File Operations (Q1 - Q3)

Q1. Write a Python program to create a text file named `sample.txt`, write your name and a message into it, and then close the file.

```
file = open("sample.txt", "w")
file.write("My name is Harcini.\n This is my file handling demo.")
file.close()
```

Q2. Write a program to read and display the contents of `sample.txt`.

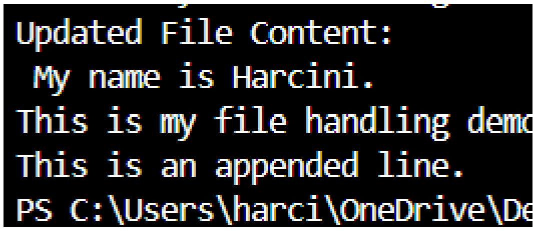
```
file = open("sample.txt", "r")
content = file.read()
print("File Content:\n", content)
file.close()
```



```
File Content:
 My name is Harcini.
This is my file handling demo.
PS C:\Users\harci\OneDrive\Desk
```

Q3. Write a Python script to append a new line `"This is an appended line"` to `sample.txt` and display the updated content.

```
file = open("sample.txt", "a")
file.write("\n This is an appended line.")
file.close()
file = open("sample.txt", "r")
print("Updated File Content:\n", file.read())
file.close()
```

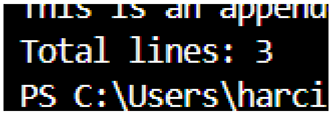


```
Updated File Content:
 My name is Harcini.
This is my file handling demo
This is an appended line.
PS C:\Users\harci\OneDrive\De
```

Section B: File Processing and Analysis (Q4 - Q7)

Q4. Write a Python program to count the total number of lines in a given file `sample.txt`.

```
with open("sample.txt", "r") as file:

    lines = file.readlines()

    print("Total lines:", len(lines))
```



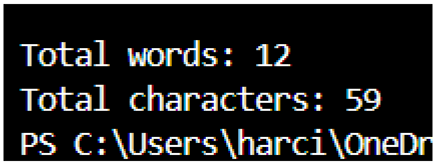
```
This is an append
Total lines: 3
PS C:\Users\harci
```

Q5. Write a Python program that reads a file and prints only those lines that contain the word “Python” (case-sensitive).

```
with open("sample.txt", "r") as file:

    for line in file:

        if "Python" in line:

            print(line.strip())
```

Q6. Write a Python program to count the number of words and characters in the file `sample.txt`.

```
with open("sample.txt", "r") as file:

    content = file.read()

    words = content.split()

    print("Total words:", len(words))

    print("Total characters:", len(content))
```



```
Total words: 12
Total characters: 59
PS C:\Users\harci\OneDr
```

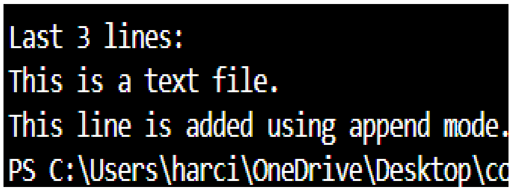
Q7. Write a program to copy the contents of `sample.txt` to another file `copy_sample.txt`.

```
with open("sample.txt", "r") as src:

    with open("copy_sample.txt", "w") as dst:

        dst.write(src.read())
```

Section C: Advanced File Handling (Q8 - Q10)

Q8. Write a Python program to display the last 3 lines of a text file.

```
with open("sample.txt", "r") as file:

    lines = file.readlines()
```



```
Last 3 lines:
This is a text file.
This line is added using append mode.
PS C:\Users\harci\OneDrive\Desktop\co
```

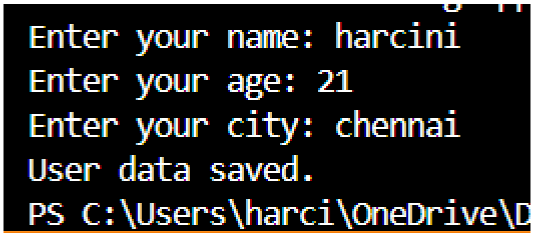
```
    print("Last 3 lines:")

    for line in lines[-3:]:

        print(line.strip())
```

Q9. Write a Python program that reads numbers from a file `numbers.txt`, one per line, and writes only the even numbers to a new file `even_numbers.txt`.

```
with open("numbers.txt", "r") as file:

    numbers = file.readlines()

with open("even_numbers.txt", "w") as even_file:

    for num in numbers:

        if num.strip().isdigit() and int(num) % 2 == 0:

            even_file.write(num)
```

Q10. Create a program that accepts user input (name, age, city) and stores it in a CSV file `users.csv`. Ensure that every new entry is stored on a new line.

```
import csv

name = input("Enter your name: ")

age = input("Enter your age: ")

city = input("Enter your city: ")

with open("users.csv", "a", newline='') as file:

    writer = csv.writer(file)

    writer.writerow([name, age, city])

print("User data saved.")
```



```
Enter your name: harcini
Enter your age: 21
Enter your city: chennai
User data saved.
PS C:\Users\harci\OneDrive\D
```