CBSE Class 12 Computer Science Question Paper 2026 with Solutions

Time Allowed :2 Hours	Maximum Marks :70	Total Questions :37

General Instructions

1. The paper is divided into Section A and Section B.
2. Section A includes objective-type questions.
3. All questions in Section A are compulsory.
4. Section B includes short answer, and long answer type questions.
5. Answers must be written legibly within the word limit.
6. Use of unfair means or electronic devices is prohibited.
7. Follow the correct format and instructions for each section.

1. State True or False :
In Python, data type of 74 is same as the data type of 74.0.

Correct Answer: False

Solution:
Step 1: Understanding the Concept:
Python categorizes numbers into different data types based on their specific format, specifically the presence or absence of a decimal point.
Step 2: Key Formula or Approach:
The approach is to identify the structure of the number. A whole number without a decimal is evaluated as an **int**, while a number containing a decimal point is evaluated as a **float**.
Step 3: Detailed Explanation:
The given value **74** is a whole number, therefore Python assigns it the data type **int**.
The value **74.0** includes a fractional part (even if it's zero) indicated by the decimal point, so its data type is **float**.
Consequently, the data types of **74** and **74.0** are distinct and not the same.
Step 4: Final Answer:
False

Quick Tip

You can always verify the data type of any variable or literal in Python using the built-in **type()** function, e.g., **type(74.0)**.

---

2. Identify the output of the following code snippet :
s = "the Truth"

print(s.capitalize())

(A) **The truth**
(B) **THE TRUTH**
(C) **The Truth**
(D) **the Truth**

Correct Answer: (A) **The truth**

Solution:
Step 1: Understanding the Concept:
The **capitalize()** method in Python is a built-in string method used to strictly format the casing of a string.
Step 2: Key Formula or Approach:
The specific rule for **capitalize()** is: The very first character of the entire string is converted to uppercase, and all subsequent characters are forcefully converted to lowercase.
Step 3: Detailed Explanation:
The original string is **"the Truth"**.
According to the rule, the first character **'t'** is converted to uppercase **'T'**.
The remaining portion of the string, which is **"he Truth"**, is converted entirely to lowercase, resulting in **"he truth"**.
Merging these two parts together yields the final output: **"The truth"**.
Step 4: Final Answer:
The truth

Quick Tip

Do not confuse **capitalize()** with **title()**. The **title()** method capitalizes the first letter of *every* word, which would output **"The Truth"**.

---

3. Which of the following expressions in Python evaluates to True ?

(A) **2>3 and 2<3**
(B) **3>1 and 2**
(C) **3>1 and 3>2**
(D) **3>1 and 3<2**

Correct Answer: (C) **3>1 and 3>2**

Solution:
Step 1: Understanding the Concept:
The **and** logical operator in Python connects two boolean expressions and evaluates the overall truth value.
Step 2: Key Formula or Approach:
The rule for the **and** operator states that the final result is **True** if, and only if, *both* the left

operand and the right operand evaluate to **True**. Otherwise, it evaluates to **False**.
Step 3: Detailed Explanation:
Let's evaluate each option systematically:
(A) **2>3** is **False**. Since the first operand is **False**, the entire expression is **False** (short-circuit evaluation).
(B) **3>1** is **True**. The expression becomes **True and 2**. In Python, this returns the second operand, which is **2**. While **2** is "truthy", the expression does not strictly return the boolean **True**.
(C) **3>1** is **True**, and **3>2** is **True**. Since both sides are **True**, the expression evaluates to **True**.
(D) **3>1** is **True**, but **3<2** is **False**. **True and False** evaluates to **False**.
Step 4: Final Answer:
3>1 and 3>2

Quick Tip

Python uses "short-circuiting" for logical operators. For **and**, if the first condition is **False**, Python immediately returns **False** without even checking the second condition.

---

4. What is the output of the following code snippet ?
s='War and Peace by Leo Tolstoy'
print(s.partition("by"))

(A) **('War and Peace ', 'by', ' Leo Tolstoy')**
(B) **['War and Peace ', 'by', ' Leo Tolstoy']**
(C) **('War and Peace ', ' Leo Tolstoy')**
(D) **['War and Peace ', ' Leo Tolstoy']**

Correct Answer: (A) **('War and Peace ', 'by', ' Leo Tolstoy')**

Solution:
Step 1: Understanding the Concept:
The **partition()** method is a built-in string function used to split a string into distinct parts based on a specified separator string.
Step 2: Key Formula or Approach:
The method searches for the first occurrence of the separator and strictly returns a *tuple* containing exactly three elements: the part before the separator, the separator itself, and the part after the separator.
Step 3: Detailed Explanation:
The given string is **"War and Peace by Leo Tolstoy"**.
The specified separator is **"by"**.
The part of the string located before the separator is **"War and Peace "**.
The separator itself is **"by"**.
The part of the string located after the separator is **" Leo Tolstoy"**.
Combining these into a tuple yields: **('War and Peace ', 'by', ' Leo Tolstoy')**.

Step 4: Final Answer:
('War and Peace ', 'by', ' Leo Tolstoy')

> Quick Tip
>
> Always remember that **partition()** returns a tuple with exactly 3 elements, whereas **split()** returns a list and completely removes the separator from the output.

---

5. What will be the output of the following statement ?
print("PythonProgram"[-1:2:-2])

Correct Answer: mroPo

Solution:
Step 1: Understanding the Concept:
This code demonstrates string slicing using negative step values to traverse a string backwards.
Step 2: Key Formula or Approach:
The syntax for slicing is **string[start : stop : step]**. A negative **step** reverses the traversal direction. The slicing stops immediately *before* reaching the **stop** index.
Step 3: Detailed Explanation:
The original string is **"PythonProgram"** (length 13).
The slice parameters are **start = -1**, **stop = 2**, and **step = -2**.
Index **-1** corresponds to the very last character **'m'** (which is also index 12).
With a step of **-2**, we move backwards by skipping one character at a time.
The indices generated will be: 12, 10, 8, 6, 4. (It stops before index 2).
Character at index 12: **'m'**
Character at index 10: **'r'**
Character at index 8: **'o'**
Character at index 6: **'P'**
Character at index 4: **'o'**
Extracting these characters in sequence yields **"mroPo"**.
Step 4: Final Answer:
mroPo

> Quick Tip
>
> When the step is negative, the slicing extracts right-to-left. Ensure that the **start** index is positioned to the right of the **stop** index; otherwise, the result will be an empty string.

---

6. What will be the output of the following code snippet ?
t = tuple('tuple')
t2 = t[2],
t += t2
print(t)

(A) **('tuple')**
(B) **('tuple','p')**
(C) **('t', 'u', 'p', 'l', 'e', 'p')**
(D) **('t', 'u', 'p', '1', 'e')**

Correct Answer: (C) **('t', 'u', 'p', 'l', 'e', 'p')**

Solution:
Step 1: Understanding the Concept:
The problem involves sequence conversion, indexing, singleton tuple creation, and tuple concatenation.
Step 2: Key Formula or Approach:
The **tuple()** function iterates through a string to create a tuple of characters. Adding a trailing comma to a single value creates a tuple. The + operator concatenates two tuples.
Step 3: Detailed Explanation:
Line 1: **t = tuple('tuple')** converts the string into a tuple of individual characters: **('t', 'u', 'p', 'l', 'e')**.
Line 2: **t[2]** extracts the character at index 2, which is **'p'**. The trailing comma in **t2 = t[2],** transforms this single character into a single-element tuple: **('p',)**.
Line 3: **t += t2** performs concatenation: **('t', 'u', 'p', 'l', 'e') + ('p',)**.
This operation yields a new combined tuple: **('t', 'u', 'p', 'l', 'e', 'p')**.
Line 4: **print(t)** outputs this final concatenated tuple.
Step 4: Final Answer:
('t', 'u', 'p', 'l', 'e', 'p')

Quick Tip

A trailing comma is strictly required to declare a single-element tuple in Python. Without the comma, **t2 = t[2]** would just assign the string **'p'** to **t2**.

---

7. Which of the following statements is true about dictionaries in Python ?

(A) **A dictionary is an example of sequence datatype.**
(B) **A dictionary cannot have two elements with same key.**
(C) **A dictionary cannot have two elements with same value.**
(D) **The key and value of an element cannot be the same.**

Correct Answer: (B) **A dictionary cannot have two elements with same key.**

Solution:
Step 1: Understanding the Concept:
Dictionaries in Python are mapping data types that store data in key-value pairs. They have specific rules regarding the properties of keys and values.
Step 2: Key Formula or Approach:
The fundamental rule of a dictionary is that all keys must be absolutely unique and immutable, whereas values can be duplicated and can be of any data type.
Step 3: Detailed Explanation:
(A) False. A dictionary is a *mapping* datatype, not a sequence datatype (like strings, lists, or tuples).
(B) True. Dictionary keys must be unique. If you attempt to assign a value to an already existing key, it simply overwrites the old value rather than creating a duplicate key.
(C) False. Multiple distinct keys can store the exact same value.
(D) False. A key and its corresponding value can be identical (e.g., **{1: 1, "A": "A"}** is perfectly valid).
Step 4: Final Answer:
A dictionary cannot have two elements with same key.

Quick Tip

When designing a dictionary, think of it like a real-world dictionary: you cannot have two identical words (keys) with separate entries, but two different words can have the same meaning (values).

---

8. If L is a list with 6 elements, then which of the following statements will raise an exception ?

(A) **L.pop(1)**
(B) **L.pop(6)**
(C) **L.insert(1,6)**
(D) **L.insert(6,1)**

Correct Answer: (B) **L.pop(6)**

Solution:
Step 1: Understanding the Concept:
The problem tests knowledge of list indexing boundaries and how different built-in list methods handle out-of-range indices.
Step 2: Key Formula or Approach:
A list of length N has valid positive indices from 0 to $N-1$. The **pop()** method strictly requires an index within this valid range. The **insert()** method is lenient and gracefully handles out-of-bounds indices by simply appending the element.
Step 3: Detailed Explanation:
The list **L** has 6 elements, meaning its valid indices are 0, 1, 2, 3, 4, and 5.
(A) **L.pop(1)**: Index 1 is valid. It successfully removes the second element.

(B) **L.pop(6)**: Index 6 is strictly outside the valid range (0-5). This forces Python to raise an **IndexError**.
(C) **L.insert(1,6)**: Inserts the value 6 at valid index 1.
(D) **L.insert(6,1)**: Index 6 is out of bounds, but **insert()** handles this by appending the value 1 to the very end of the list without raising any exception.
Step 4: Final Answer:
L.pop(6)

> Quick Tip
>
> Remember that **pop()** and direct indexing (like **L[6]**) will crash the program if the index is out of bounds, whereas **insert()** and slicing (like **L[6:8]**) are robust and will not raise an exception.

---

9. What will be the output of the following code ?

```
def f1(a,b=1):
print(a+b,end='-')
c=f1(1,2)
print(c,sep='*')
```

(A) **3-2**
(B) **3-2***
(C) **3-None**
(D) **3*None-**

Correct Answer: (C) **3-None**

Solution:
Step 1: Understanding the Concept:
The code tests function execution, parameter overriding, and the default return value of Python functions.
Step 2: Key Formula or Approach:
Trace the execution step-by-step. If a user-defined function executes completely without encountering a **return** statement, it implicitly returns the special value **None**.
Step 3: Detailed Explanation:
1. The function **f1** is called with arguments **1** and **2**. This assigns **a = 1** and **b = 2** (overriding the default value of **1**).
2. Inside the function, **print(a+b, end='-')** calculates **1+2 = 3** and prints **3-**. The **end='-'** ensures the cursor remains on the same line.
3. The function block ends without any **return** statement. Consequently, it returns **None**.
4. The variable **c** is assigned this returned value, so **c = None**.
5. The subsequent statement **print(c, sep='*')** prints the value of **c** (**None**) immediately after the previously printed hyphen. (The **sep** argument is irrelevant here because only one item is being printed).
Combining the printed components gives **3-None**.

Step 4: Final Answer:
3-None

Quick Tip

Whenever a variable captures the result of a function call, always search the function definition for a **return** keyword. If absent, the captured variable will unequivocally hold **None**.

---

10. Consider the statement given below :
f1 = open("pqr.dat","___")
Which of the following is the correct file mode to open the file in read only mode ?

(A) **a**
(B) **rb**
(C) **r+**
(D) **rb+**

Correct Answer: (B) **rb**

Solution:
Step 1: Understanding the Concept:
Opening files in Python requires specifying the appropriate file access mode based on the file type (text vs binary) and the intended operation.
Step 2: Key Formula or Approach:
The file extension **.dat** universally indicates a binary data file. To interact with binary files, the mode string must include the character **'b'**. Reading requires the character **'r'**.
Step 3: Detailed Explanation:
(A) **'a'** is used for appending data to a text file.
(B) **'rb'** stands for read binary. It opens a binary file strictly for reading operations.
(C) **'r+'** is used for both reading and writing to a text file.
(D) **'rb+'** is used for both reading and writing to a binary file.
Since the prompt explicitly requests "read only mode" for a **.dat** file, **'rb'** is the correct choice.
Step 4: Final Answer:
rb

Quick Tip

Text files (like **.txt** or **.csv**) use standard modes (**'r', 'w', 'a'**), whereas binary files (like **.dat**, **.jpg**, **.bin**) mandate adding a **'b'** (**'rb', 'wb', 'ab'**).

---

11. State whether the following statement is True or False :
In Python, Logical errors can be handled using try...except...finally statement.

Correct Answer: False

Solution:
Step 1: Understanding the Concept:
Programming errors are generally categorized into syntax errors, runtime errors (exceptions), and logical errors. Exception handling structures target specific error types.
Step 2: Key Formula or Approach:
The **try...except** block is explicitly designed to catch and handle Runtime Errors (Exceptions) that crash the program during execution.
Step 3: Detailed Explanation:
Logical errors occur when the source code is syntactically perfect and executes without crashing, but contains flaws in the core logic or algorithm, thereby producing mathematically or operationally incorrect outputs.
Because logical errors do not raise built-in Python exceptions (like **ZeroDivisionError** or **ValueError**), the **try...except** block has absolutely no mechanism to detect or catch them.
Therefore, the statement is completely false.
Step 4: Final Answer:
False

Quick Tip

To fix logical errors, programmers must rely on debugging tools, dry runs, or strategically placed **print()** statements to trace variable values, not exception handling.

---

12. A table has two candidate keys, one of which is chosen as the primary key. How many alternate keys does this table have ?

(A) **0**
(B) **1**
(C) **2**
(D) **3**

Correct Answer: (B) **1**

Solution:
Step 1: Understanding the Concept:
Relational database design involves selecting appropriate keys to uniquely identify tuples within a table.
Step 2: Key Formula or Approach:
The relationship between keys is defined by the mathematical formula:

$$\text{Total Candidate Keys} = \text{Primary Key} + \text{Alternate Keys}$$

Step 3: Detailed Explanation:
A candidate key is an attribute or set of attributes capable of uniquely identifying a record. From the pool of available candidate keys, the Database Administrator selects exactly one to serve as the Primary Key.
Any candidate keys that remain unselected are officially classified as Alternate keys.
Given: Total Candidate Keys = 2.
Primary Key = 1.
Alternate Keys = $2 - 1 = 1$.
Step 4: Final Answer:
1

Quick Tip

An alternate key is simply any candidate key that "lost the election" to become the primary key.

---

13. Which of the following SQL command can change the degree of the existing relation ?

(A) **DROP TABLE**
(B) **ALTER TABLE**
(C) **UPDATE SET**
(D) **DELETE**

Correct Answer: (B) **ALTER TABLE**

Solution:
Step 1: Understanding the Concept:
In relational databases, the "degree" of a relation mathematically refers to the total number of attributes (columns) present in that table.
Step 2: Key Formula or Approach:
To change the degree, one must structurally add a new column or drop an existing column. This requires a Data Definition Language (DDL) command that modifies the table schema.
Step 3: Detailed Explanation:
(A) **DROP TABLE** destroys the entire table, removing both its structure and data.
(B) **ALTER TABLE** is utilized to modify the table's structural schema, specifically by adding (**ADD**) or removing (**DROP**) columns. This directly alters the degree.
(C) **UPDATE...SET** modifies the data values within existing rows (DML command).
(D) **DELETE** removes rows from the table, which changes the cardinality, not the degree.
Step 4: Final Answer:
ALTER TABLE

Quick Tip

Memorize the associations: Degree ↔ Columns ↔ **ALTER TABLE**. Cardinality ↔ Rows ↔ **INSERT/DELETE/UPDATE**.

---

14. What will be the output of the query ?
SELECT MACHINE_ID, MACHINE_NAME FROM INVENTORY WHERE QUANTITY ¡= 100;

(A) **All columns of INVENTORY table with quantity greater than 100**
(B) **ID and name of machines with quantity less than 100 from INVENTORY table**
(C) **All columns of INVENTORY table with quantity greater than or equal to 100**
(D) **ID and name of machines with quantity less than or equal to 100 from INVENTORY table.**

Correct Answer: (D) **ID and name of machines with quantity less than or equal to 100 from INVENTORY table.**

Solution:
Step 1: Understanding the Concept:
SQL queries translate directly into English descriptions outlining exactly what data is retrieved and under what constraints.
Step 2: Key Formula or Approach:
Deconstruct the query:
The **SELECT** clause specifies the precise columns to be retrieved.
The **WHERE** clause filters the rows using relational operators.
Step 3: Detailed Explanation:
1. **SELECT MACHINE_ID, MACHINE_NAME**: This indicates that only the ID and Name columns will be displayed, immediately eliminating options (A) and (C) which mention "All columns".
2. **WHERE QUANTITY ¡= 100**: The operator ¡= translates perfectly to "less than or equal to". This eliminates option (B), which only says "less than".
Combining these observations matches option (D) perfectly.
Step 4: Final Answer:
ID and name of machines with quantity less than or equal to 100 from INVENTORY table.

Quick Tip

Always read relational operators carefully. ¡= is "less than or equal to", whereas ¡ is strictly "less than". Missing the "or equal to" is a common trap.

---

15. A relation in MySQL database consists of 2 tuples and 3 attributes. If 2 attributes are deleted and 4 tuples are added, what will be the cardinality of the relation ?

(A) **4**
(B) **5**
(C) **6**
(D) **7**

Correct Answer: (C) **6**

Solution:
Step 1: Understanding the Concept:
Database terminology relies heavily on specific words representing structural properties of a table.
Step 2: Key Formula or Approach:
Cardinality = Total Number of Tuples (Rows).
Degree = Total Number of Attributes (Columns).
Step 3: Detailed Explanation:
The initial state of the table is:
Tuples (Cardinality) = 2
Attributes (Degree) = 3
The problem states that 2 attributes are deleted. This action exclusively affects the Degree, reducing it from 3 to 1. It has absolutely no impact on the Cardinality.
Next, 4 new tuples are added to the table.
New Cardinality = Initial Tuples + Added Tuples
New Cardinality = $2 + 4 = 6$.
Step 4: Final Answer:
6

Quick Tip

When a question asks for final cardinality, safely ignore any given information regarding adding or deleting attributes (columns), as they are irrelevant distractions.

---

16. Which aggregate function in SQL returns the smallest value from a column in a table ?

(A) **MIN()**
(B) **MAX()**
(C) **SMALL()**
(D) **LOWER()**

Correct Answer: (A) **MIN()**

Solution:
Step 1: Understanding the Concept:
SQL provides built-in aggregate functions to perform calculations across a designated set of values, returning a single scalar result.
Step 2: Key Formula or Approach:
Recall the core standard aggregate functions: **SUM()**, **AVG()**, **COUNT()**, **MIN()**, and **MAX()**.
Step 3: Detailed Explanation:
The **MIN()** function evaluates all non-null values in a specified column and strictly returns the minimum (smallest) numerical or alphabetical value.
The **MAX()** function does the exact opposite, returning the largest value.
SMALL() does not exist as an aggregate function in SQL.
LOWER() is a single-row scalar function used to convert text string characters to lowercase, not to identify the smallest value in a column.
Step 4: Final Answer:
MIN()

Quick Tip

Aggregate functions can be applied to numerical data, date data (earliest/latest), and even string data (alphabetical order).

---

17. With respect to computer networks, which of the following is the correct expanded form of RJ 45 ?

(A) **Radio Jockey 45**
(B) **Registered Jockey 45**
(C) **Radio Jack 45**
(D) **Registered Jack 45**

Correct Answer: (D) **Registered Jack 45**

Solution:
Step 1: Understanding the Concept:
Networking hardware relies on standardized physical connectors to bridge devices and cables.
Step 2: Key Formula or Approach:
Recognize standard acronyms. RJ represents a standardized telecommunication network interface.
Step 3: Detailed Explanation:
The acronym RJ stands specifically for "Registered Jack".
An RJ45 is an 8-pin/8-position connector used extensively worldwide to terminate twisted-pair cables for Ethernet local area networks (LANs).
Consequently, the correct expansion is Registered Jack 45.
Step 4: Final Answer:

Registered Jack 45

Quick Tip

RJ45 is the standard connector for Ethernet network cables, while the smaller RJ11 connector is typically used for standard telephone lines.

---

18. Which network device serves as the entry and exit point of a network, as all data coming in or going out of a network must first pass through it in order to use routing paths ?

(A) **Modem**
(B) **Gateway**
(C) **Switch**
(D) **Repeater**

Correct Answer: (B) **Gateway**

Solution:
Step 1: Understanding the Concept:
Different networking hardware components perform highly specialized roles to govern data traffic within and between networks.
Step 2: Key Formula or Approach:
Match the functional definition "entry and exit point connecting different networks" to the correct device.
Step 3: Detailed Explanation:
A **Gateway** is a network node that operates as a distinct "gate" between two dissimilar networks, often running entirely different protocols. It governs the flow of traffic moving in and out of the local network to the internet or other external networks, making it the definitive entry and exit point.
A **Modem** strictly modulates/demodulates analog and digital signals.
A **Switch** filters and forwards data packets strictly within a local area network (LAN).
A **Repeater** merely regenerates and boosts degraded physical signals.
Step 4: Final Answer:
Gateway

Quick Tip

Anytime a question mentions a device resolving different protocols or serving as an entry/exit checkpoint, the answer is undoubtedly a Gateway.

---

19. Expand the term XML.

Correct Answer: eXtensible Markup Language

Solution:
Step 1: Understanding the Concept:
Web technologies utilize various markup languages to structure, display, and transport data.
Step 2: Key Formula or Approach:
Recall standard web acronyms used for data exchange files.
Step 3: Detailed Explanation:
XML stands for eXtensible Markup Language.
Unlike HTML, which uses predefined tags strictly to display data, XML allows programmers to define their own custom tags to store and transport data systematically in a format that is both human-readable and machine-readable.
Step 4: Final Answer:
eXtensible Markup Language

> Quick Tip
>
> It is a common convention to capitalize the 'X' inside "eXtensible" when writing out the full form of XML.

---

20. Assertion (A) : [1,2,3]+'123' is an invalid expression in Python.
Reason (R) : In Python, a list cannot be concatenated with a string.

(A) **Both Assertion (A) and Reason (R) are true and Reason (R) is the correct explanation for Assertion (A).**
(B) **Both Assertion (A) and Reason (R) are true and Reason (R) is not the correct explanation for Assertion (A).**
(C) **Assertion (A) is true, but Reason (R) is false.**
(D) **Assertion (A) is false, but Reason (R) is true.**

Correct Answer: (A) **Both Assertion (A) and Reason (R) are true and Reason (R) is the correct explanation for Assertion (A).**

Solution:
Step 1: Understanding the Concept:
Python operators exhibit different behaviors depending on the data types of their operands. The + operator performs concatenation on sequence types.
Step 2: Key Formula or Approach:
The strict rule for sequence concatenation using + dictates that both operands must be of the exact same sequence type (e.g., list + list, or string + string).
Step 3: Detailed Explanation:
The given expression attempts to execute **[1, 2, 3] + '123'**. The left operand is a **list**, and

the right operand is a **string**.
Because Python prohibits cross-type concatenation, this expression directly raises a **TypeError**, making Assertion (A) fundamentally true.
The Reason (R) explicitly states this rule: "a list cannot be concatenated with a string", which is a true statement and serves as the exact technical justification for why Assertion (A) is true.
Step 4: Final Answer:
Both Assertion (A) and Reason (R) are true and Reason (R) is the correct explanation for Assertion (A).

> Quick Tip
>
> To safely combine them, you must explicitly typecast one operand. For example, **[1,2,3] + list('123')** will result in **[1, 2, 3, '1', '2', '3']**.

---

21. Assertion (A) : The PRIMARY KEY constraint in SQL ensures that each value in the column(s) is unique and cannot be NULL.
Reason (R) : Candidate keys are not eligible to become a primary key.

(A) **Both Assertion (A) and Reason (R) are true and Reason (R) is the correct explanation for Assertion (A).**
(B) **Both Assertion (A) and Reason (R) are true and Reason (R) is not the correct explanation for Assertion (A).**
(C) **Assertion (A) is true, but Reason (R) is false.**
(D) **Assertion (A) is false, but Reason (R) is true.**

Correct Answer: (C) **Assertion (A) is true, but Reason (R) is false.**

Solution:
Step 1: Understanding the Concept:
Database keys and constraints dictate the integrity and unique identification of records within a table.
Step 2: Key Formula or Approach:
Define Primary Key: Enforces strictly unique and non-null values.
Define Candidate Key: The set of keys that possess the potential eligibility to be selected as the primary key.
Step 3: Detailed Explanation:
Assertion (A) defines the Primary Key correctly. A primary key combines the **UNIQUE** and **NOT NULL** constraints perfectly to ensure uncompromised record identification. Thus, (A) is true.
Reason (R) states that candidate keys are *not* eligible to become primary keys. This statement fundamentally contradicts database theory. Candidate keys are specifically the very pool of keys from which the primary key is chosen. Thus, (R) is entirely false.
Step 4: Final Answer:
Assertion (A) is true, but Reason (R) is false.

Quick Tip

Candidate Key = Potential Primary Key. Any key that qualifies as a candidate key is automatically eligible to be the primary key.

---

22. What is the difference between default parameters and positional parameters in Python ? Also give an example of a function header which uses both.

Correct Answer: Refer to the solution for a detailed theoretical explanation and code example.

Solution:
Step 1: Understanding the Concept:
Function arguments in Python dictate how data is passed into a user-defined function. Parameters can act dynamically based on how they are defined in the header.
Step 2: Key Formula or Approach:
Differentiate parameters based on mandatory provision versus fallback values, and note the strict syntax ordering required in function headers.
Step 3: Detailed Explanation:
Positional Parameters:
These are mandatory parameters. During a function call, arguments must be passed to these parameters in the exact sequential order defined in the function header. If omitted, Python immediately throws a **TypeError**.
Default Parameters:
These are optional parameters. They are defined in the function header utilizing an assignment operator to provide a fallback (default) value. If the user omits providing an argument during the call, the function seamlessly utilizes this pre-assigned default value instead of throwing an error.
Syntax Rule: In a function header, default parameters must strictly follow positional parameters.
Example of a function header using both:
def calculate_interest(principal, rate=5.0):
In this example, **principal** is a positional parameter, while **rate** is a default parameter with a fallback value of **5.0**.
Step 4: Final Answer:
Positional parameters require mandatory ordered arguments. Default parameters provide fallback values if arguments are omitted. Example: **def greet(name, msg="Hello"):**

Quick Tip

Defining a header like **def info(age=18, name):** will crash with a **SyntaxError**. Always place parameters without defaults first.

23(i). Write a Python statement to perform the following tasks : (USE BUILT_IN FUNCTIONS / METHODS ONLY)
To create a new list L1 containing the elements of list L arranged in ascending order, without modifying list L.

Correct Answer: L1 = sorted(L)

Solution:
Step 1: Understanding the Concept:
Python offers two primary ways to sort lists: an in-place list method and a built-in returning function.
Step 2: Key Formula or Approach:
The list method **L.sort()** sorts the list in place, irreversibly modifying the original list. The built-in function **sorted(L)** returns an entirely new sorted list, preserving the original data.
Step 3: Detailed Explanation:
The prompt strictly specifies creating a *new* list **L1** while ensuring the original list **L** is *not modified*.
Applying **L1 = L.sort()** is incorrect because **sort()** returns **None** and modifies **L**.
Therefore, utilizing the built-in function **sorted()** is mandatory to achieve the specified result.
Step 4: Final Answer:
L1 = sorted(L)

> Quick Tip
>
> Use **sorted(L, reverse=True)** if descending order is required.

23(ii). Write a Python statement to perform the following tasks : (USE BUILT_IN FUNCTIONS / METHODS ONLY)
A statement to check whether the given character, ch is an alphabet or a number.

Correct Answer: ch.isalnum()

Solution:
Step 1: Understanding the Concept:
Python string objects possess built-in validation methods to inspect the categorical nature of characters.
Step 2: Key Formula or Approach:
To check for alphabets, use **isalpha()**. To check for numbers, use **isdigit()**. To check for either (alphanumeric), use the combined method **isalnum()**.
Step 3: Detailed Explanation:
The question demands a single statement utilizing a built-in method to evaluate if a character is an alphabet OR a number.
The string method **isalnum()** evaluates to **True** if all characters within the string fall into the

alphanumeric category, which perfectly satisfies the condition.
Step 4: Final Answer:
ch.isalnum()

> Quick Tip
>
> Special characters, symbols, and spaces will cause **isalnum()** to return **False**.

24(i)(a). Assuming that D1 is a dictionary in Python,
Write a Python expression to check if the key, 'RNo' is present in D1.

Correct Answer: 'RNo' in D1

Solution:
Step 1: Understanding the Concept:
Testing for membership within data structures is achieved using specific operators.
Step 2: Key Formula or Approach:
The **in** membership operator is utilized to verify if a specified element exists within an iterable. For dictionaries, **in** strictly checks the dictionary's keys by default.
Step 3: Detailed Explanation:
To construct an expression that checks if the string **'RNo'** exists specifically as a key in the dictionary **D1**, we apply the membership operator directly to the dictionary object.
The resulting expression evaluates to a Boolean **True** or **False**.
Step 4: Final Answer:
'RNo' in D1

> Quick Tip
>
> While **'RNo' in D1.keys()** is technically correct, **'RNo' in D1** is the universally preferred and most efficient Pythonic approach.

OR

24(i)(b) Assuming that D1 is a dictionary in Python,
Write a Python expression to check if any key in D1 has a value 12.

Correct Answer: 12 in D1.values()

Solution:
Step 1: Understanding the Concept:

Dictionaries partition data into keys and values. Extracting values requires utilizing a specific dictionary method.
Step 2: Key Formula or Approach:
The method **D1.values()** returns a dynamic view object comprising all values present inside the dictionary. Apply the **in** operator to this view.
Step 3: Detailed Explanation:
By default, the **in** operator only searches through keys.
To search for the specific integer **12** among the assigned values, we must explicitly direct the operator to search the values collection.
This is accomplished by utilizing **D1.values()**.
Step 4: Final Answer:
12 in D1.values()

> Quick Tip
>
> Searching through values operates significantly slower than searching through keys because it requires a linear scan rather than a direct hash lookup.

24(ii)(a). Assuming that D1 is a dictionary in Python,
Write a single statement using a BUILT_IN function to add the key : value pair 'RNo' : 12, if the key 'RNo' is not present in D1. However, if the key 'RNo' is present, the function should return its value.

Correct Answer: D1.setdefault('RNo', 12)

Solution:
Step 1: Understanding the Concept:
Dictionaries feature advanced built-in methods to conditionally manipulate key-value pairs without requiring explicit **if...else** blocks.
Step 2: Key Formula or Approach:
The method **setdefault(key, default_value)** serves this exact dual purpose.
Step 3: Detailed Explanation:
The prompt strictly demands a *single statement* performing conditional logic.
When **D1.setdefault('RNo', 12)** is executed:
1. It queries the dictionary for the key **'RNo'**.
2. If the key exists, it safely returns the associated current value, ignoring the number 12.
3. If the key does not exist, it automatically inserts the new key-value pair **'RNo': 12** into the dictionary and returns 12.
Step 4: Final Answer:
D1.setdefault('RNo', 12)

Quick Tip

Do not confuse this with the **get()** method. While **get('RNo', 12)** also returns a fallback value if the key is missing, it explicitly *does not* insert the key-value pair into the dictionary.

OR

24(ii)(b) Assuming that D1 is a dictionary in Python,
Write a single statement to delete all the elements from D1.

Correct Answer: D1.clear()

Solution:
Step 1: Understanding the Concept:
Python provides specific methods to safely empty container data structures while preserving the actual object variable in memory.
Step 2: Key Formula or Approach:
The **clear()** method removes all items from a dictionary entirely.
Step 3: Detailed Explanation:
To erase every key-value pair stored inside **D1**, making it an empty dictionary **{}**, the most efficient syntax is utilizing the **clear()** method.
Step 4: Final Answer:
D1.clear()

Quick Tip

Executing **del D1** deletes the variable from memory entirely, causing subsequent references to crash. Executing **D1.clear()** leaves the variable active but empty.

---

25. What possible output(s) from the given options will NOT be displayed when the following code is executed ? Also, mention, for how many iterations the for loop in the given code will run ?

```
import random
a = [1,2,3,4,5,6]
for i in range(4):
j = random.randrange(i,5)
print(a[j],end='-')
print()
```

(A) **3-4-5-4-**
(B) **2-2-4-5-**

(C) **4-3-3-5-**
(D) **5-1-2-4-**

Correct Answer: (D) **5-1-2-4-** will NOT be displayed. The loop runs for 4 iterations.

Solution:
Step 1: Understanding the Concept:
The **random.randrange(start, stop)** function generates a random integer ranging from **start** (inclusive) to **stop** (exclusive). The limits change dynamically during loop execution.
Step 2: Key Formula or Approach:
Trace the possible values of **j** for each successive value of **i**, and subsequently map those valid index values to the corresponding elements stored in list **a**.
Step 3: Detailed Explanation:
The loop condition **for i in range(4)** dictates that **i** assumes values 0, 1, 2, and 3. Thus, the loop runs exactly 4 times.
Let's analyze the limits for **j** and the resulting output **a[j]**:
Iter 1 (**i=0**): **j** is randomly chosen from range(0, 5) $\rightarrow$ {0,1,2,3,4}. Corresponding values in **a[j]**: {1,2,3,4,5}.
Iter 2 (**i=1**): **j** is chosen from range(1, 5) $\rightarrow$ {1,2,3,4}. Corresponding values in **a[j]**: {2,3,4,5}.
Iter 3 (**i=2**): **j** is chosen from range(2, 5) $\rightarrow$ {2,3,4}. Corresponding values in **a[j]**: {3,4,5}.
Iter 4 (**i=3**): **j** is chosen from range(3, 5) $\rightarrow$ {3,4}. Corresponding values in **a[j]**: {4,5}.
Now evaluate Option (D): **5-1-2-4-**.
The second value printed is **1**. However, during Iteration 2 (**i=1**), the generated index **j** must be ≥ 1. Therefore, **a[j]** must be ≥ 2. Because the value **1** is physically impossible to generate in the second position, Option (D) represents an impossible outcome.
Step 4: Final Answer:
(D) **5-1-2-4-** will NOT be displayed. The loop executes for exactly 4 iterations.

> Quick Tip
>
> To quickly solve random module output questions, always trace the *minimum* and *maximum* possible values for every single position in the sequence and eliminate options that violate these boundaries.

---

26. The function given below is written to accept a string s as a parameter and return the number of vowels appearing in the string. The code has certain errors. Observe the code carefully and rewrite it after removing all the logical and syntax errors. Underline all the corrections made.

```
def CountVowels(s):
c=0
for ch in range(s):
if 'aeiouAEIOU' in ch:
c=+1
return(ch)
```

Correct Answer: Refer to the code block below for the corrected version.

Solution:
Step 1: Understanding the Concept:
Code debugging involves identifying syntax errors (code that won't run) and logical errors (code that yields incorrect results) and replacing them with correct Pythonic logic.
Step 2: Key Formula or Approach:
Iterate correctly over a string, apply the **in** membership operator correctly (item **in** container), accumulate values using standard assignment operators, and return the proper counter variable.
Step 3: Detailed Explanation:
Error 1: **for ch in range(s):** is illegal because **range()** expects integers, not a string.
Correction: Iterate natively over the string using **for ch in s:**.
Error 2: **if 'aeiouAEIOU' in ch:** logically checks if the massive 10-character string exists inside a single character **ch**, which is impossible.
Correction: Reverse the operands to check if the single character exists within the vowel string: **if ch in 'aeiouAEIOU':**.
Error 3: **c=+1** acts merely as a reassignment of the positive integer 1 to **c**, repeatedly overriding the counter instead of incrementing it.
Correction: Use the addition assignment operator **c+=1** (or **c=c+1**).
Error 4: **return(ch)** bizarrely returns the last processed character from the loop instead of the accumulated count.
Correction: Return the integer counter variable via **return c**.
Step 4: Final Answer:

```
def CountVowels(s):
    c=0
    for ch in s:                # Correction 1
        if ch in 'aeiouAEIOU': # Correction 2
            c+=1                # Correction 3
    return c                    # Correction 4
```

Quick Tip

Always ensure the syntax for the **in** operator places the smaller element to be searched on the left side, and the sequence being searched on the right side.

27(i)(a). Ms. Zoya is a Production Manager in a factory which packages mineral water. She decides to create a table in a database to keep track of the stock present in the factory. Each record of the table will have the following fields :
W_Code - Code of the item (type - CHAR(5))
W_Description - Description of the item (type - VARCHAR(20))
B_Qty - Balance quantity of the item (type - INTEGER)
U_Price - Unit Price of the item (type - FLOAT)
The name of the table is W_STOCK.

Write an SQL command to create the above table (W_Code should be the primary key).

Correct Answer: CREATE TABLE W_STOCK (W_Code CHAR(5) PRIMARY KEY, W_Description VARCHAR(20), B_Qty INTEGER, U_Price FLOAT);

Solution:
Step 1: Understanding the Concept:
The Data Definition Language (DDL) command **CREATE TABLE** builds the structural framework of a new relational table, defining columns, datatypes, and integrity constraints.
Step 2: Key Formula or Approach:
The required standard syntax is: **CREATE TABLE TableName (Column1 DataType Constraint, Column2 DataType, ...);**
Step 3: Detailed Explanation:
TableName is explicitly given as **W_STOCK**.
Field 1: Name is **W_Code**, type is **CHAR(5)**, constraint is **PRIMARY KEY**.
Field 2: Name is **W_Description**, type is **VARCHAR(20)**.
Field 3: Name is **B_Qty**, type is **INTEGER**.
Field 4: Name is **U_Price**, type is **FLOAT**.
Assembling these pieces separated by commas yields the correct statement.
Step 4: Final Answer:
CREATE TABLE W_STOCK (W_Code CHAR(5) PRIMARY KEY, W_Description VARCHAR(20), B_Qty INTEGER, U_Price FLOAT);

> Quick Tip
>
> Ensure that no comma is placed after the final column definition just before the closing parenthesis.

27(i)(b). Can U_Price be the primary key of the above table ? Justify your answer.

Correct Answer: No, because unit prices are generally not unique across different items.

Solution:
Step 1: Understanding the Concept:
The role of a primary key in a relational database.
Step 2: Key Formula or Approach:
A primary key demands absolute uniqueness; no two tuples (rows) can contain the same identical value in that specific column.
Step 3: Detailed Explanation:
The attribute **U_Price** signifies the selling Unit Price of an inventory item.
In standard commercial environments, it is incredibly common for multiple distinct items to possess the exact same selling price (e.g., two different water bottle sizes might both cost 50.0).

Because **U_Price** naturally accommodates duplicate values, it fails the uniqueness constraint entirely, rendering it ineligible to act as a primary key.
Step 4: Final Answer:
No, **U_Price** cannot act as a primary key because multiple diverse stock items can logically possess the exact same unit price, violating the uniqueness rule of primary keys.

> Quick Tip
>
> When justifying why an attribute cannot be a primary key, cite real-world scenarios where data duplication would inevitably occur.

27(ii)(a). Assuming that the table W_STOCK is already created, write an SQL command to add an attribute E_Date (of DATE type) to the table.

Correct Answer: ALTER TABLE W_STOCK ADD E_Date DATE;

Solution:
Step 1: Understanding the Concept:
Modifying the structural schematic (adding columns) of a pre-existing table.
Step 2: Key Formula or Approach:
The structural modification utilizes the DDL command: **ALTER TABLE TableName ADD ColumnName DataType;**
Step 3: Detailed Explanation:
The table to be modified is **W_STOCK**.
The objective is to append a new column. This demands the **ADD** keyword.
The new column name is designated as **E_Date**.
The datatype is explicitly defined as **DATE**.
Combining these clauses produces the final SQL syntax.
Step 4: Final Answer:
ALTER TABLE W_STOCK ADD E_Date DATE;

> Quick Tip
>
> Never confuse **ALTER TABLE** with the **UPDATE** command. **ALTER** fundamentally changes the table's skeleton (columns), whereas **UPDATE** edits the actual data values (rows).

27(ii)(b). Assuming that the table W_STOCK is already created, write an SQL command to remove the column B_Qty from the table.

Correct Answer: ALTER TABLE W_STOCK DROP B_Qty;

Solution:
Step 1: Understanding the Concept:
Deleting an existing column and its associated data entirely from a table schema.
Step 2: Key Formula or Approach:
The structural modification utilizes the DDL command: **ALTER TABLE TableName DROP ColumnName;**
Step 3: Detailed Explanation:
The table intended for modification is **W_STOCK**.
The objective is permanent removal of a column. This demands the **DROP** keyword.
The target column designated for removal is **B_Qty**.
Structuring the query systematically yields the correct command.
Step 4: Final Answer:
ALTER TABLE W_STOCK DROP B_Qty;

Quick Tip

While MySQL optionally accepts **DROP COLUMN ColumnName**, omitting the word **COLUMN** (i.e., just **DROP ColumnName**) is perfectly valid and shorter to write.

28(a). List one advantage and one disadvantage of Bus topology.

Correct Answer: Advantage: Simple to install and requires minimal cable. Disadvantage: Entire network crashes if the central backbone cable fails.

Solution:
Step 1: Understanding the Concept:
Bus topology is a specific network layout architecture where all active nodes are connected sequentially to a single centralized communication cable, known as a backbone.
Step 2: Key Formula or Approach:
Evaluate the physical structure to determine installation costs (advantage) and reliance on single points of failure (disadvantage).
Step 3: Detailed Explanation:
Advantage: The structural design is highly simplistic. Because every computer directly taps into a single straight cable, it utilizes significantly less physical cabling compared to alternative layouts like Star or Mesh, rendering it exceptionally cost-effective and easy to deploy for small-scale networks.
Disadvantage: The single centralized backbone cable acts as an extreme bottleneck. If this main cable suffers physical damage or is severed, the entire communication network collapses instantly, severely impacting reliability.
Step 4: Final Answer:
Advantage: It is highly cost-effective and extremely easy to physically install due to reduced

cable requirements.
Disadvantage: The network is vulnerable to a single point of failure; if the main backbone cable breaks, the entire network shuts down.

Quick Tip

When answering topology questions, associate "Bus" immediately with "single backbone cable" to easily derive its pros and cons.

28(b). What is protocol in the context of computer networks ? Which protocol is used to transmit hypertext across the web ?

Correct Answer: A protocol is a formalized set of communication rules. The protocol for hypertext is HTTP.

Solution:
Step 1: Understanding the Concept:
Computers manufactured by vastly different global companies require a universal standardized language to exchange data without misinterpretation.
Step 2: Key Formula or Approach:
Define the term "protocol" accurately, and associate the task of transmitting web pages with the correct acronym.
Step 3: Detailed Explanation:
In networking contexts, a **protocol** is defined as a formalized set of standardized rules and procedures that govern how data is formatted, routed, transmitted, and successfully received over a network infrastructure. It essentially acts as a universal communication language between independent devices.
The specific protocol dedicated exclusively to transmitting hypertext documents (web pages formatted in HTML) across the World Wide Web is the **Hypertext Transfer Protocol (HTTP)**.
Step 4: Final Answer:
A protocol is a standardized set of rules governing data communication across a network. The protocol used to transmit hypertext over the web is HTTP (or HTTPS for secure transmission).

Quick Tip

Always correlate data types with specific protocols: HTTP for web pages, FTP for standard files, and SMTP for email transmission.

29(a). Write a Python function that counts and returns the number of digits appearing in the text file "Space.txt". For example, if the file contains :

Space exploration has unlocked incredible advancements in technology and science. Since the first moon landing in 1969, space agencies have sent probes to Mars, Jupiter and beyond. The ISS, orbiting Earth at about 400 km, serves as a hub for research. With missions planned for 2030, humanity's cosmic journey continues!
Then the function should return 11.

Correct Answer: Refer to the code block below.

Solution:
Step 1: Understanding the Concept:
Text file manipulation involves sequentially accessing the physical file, reading its raw character contents into memory, iterating through them, and applying conditional logic to identify numerical digits.
Step 2: Key Formula or Approach:
Use **open(filename, 'r')** to access the file. Extract string data using **read()**. Employ the string method **isdigit()** inside a **for** loop to validate characters. Accumulate matches.
Step 3: Detailed Explanation:
We open **"Space.txt"** safely in read mode.
By assigning the result of **file.read()** to a variable, we acquire the entire text content as one massive, continuous string.
We initialize a numeric counter variable identically to 0.
We deploy a **for** loop to inspect every individual character sequentially.
The condition **if char.isdigit():** acts as a filter, resolving to **True** solely for numeric characters (0-9).
When validated, the counter safely increments. After complete iteration, the function returns this accumulated total.
Step 4: Final Answer:

```
def count_digits():
    with open("Space.txt", "r") as file:
        data = file.read()
        count = 0
        for char in data:
            if char.isdigit():
                count += 1
        return count
```

Quick Tip

Utilizing the **with open()** structure serves as a context manager, automatically and safely closing the file upon completion, preventing silent resource leaks.

29(b). Write a Python function that displays the words in which the lowercase letter 'e' appears at least twice in the text file 'Space.txt'. For example, if the file contains :
Space exploration has unlocked incredible advancements in technology and science. Since the first moon landing in 1969, space agencies have sent probes to Mars, Jupiter and beyond. The ISS, orbiting Earth at about 400 km, serves as a hub for research. With missions planned for 2030, humanity's cosmic journey continues!
Then the function should display :
incredible advancements science. agencies serves research.

Correct Answer: Refer to the code block below.

Solution:
Step 1: Understanding the Concept:
Processing distinct words from a continuous text block mandates reading the file, separating strings based on whitespace, and executing character counting within those resulting segments.
Step 2: Key Formula or Approach:
The **split()** method efficiently segments string data into a list of constituent words. The **count(substring)** method tallies occurrences within those words.
Step 3: Detailed Explanation:
Open the target text file systematically in read mode.
Retrieve the string data via **read()**.
Execute the **split()** method without arguments, which natively handles multiple spaces, tabs, and newlines securely, returning an organized list of words.
Iterate over this generated list using a **for** loop.
For every word, implement the string method **word.count('e')**.
If the resulting integer count is ≥ 2, output the word seamlessly. The prompt format implies printing horizontally, so utilizing **end=' '** is appropriate.
Step 4: Final Answer:

```
def display_e_words():
    with open("Space.txt", "r") as file:
        data = file.read()
        words = data.split()
        for word in words:
            if word.count('e') >= 2:
                print(word, end=' ')
```

Quick Tip

Always use **split()** with no arguments to cleanly extract words; using **split(' ')** can dangerously yield empty strings if multiple spaces exist sequentially in the file.

30(a). A stack named FruitStack, implemented using list, contains records of some fruits. Each record is represented as a dictionary with keys 'Name', 'Origin', 'Price', and 'Expiry'. A sample record is given here :
{'Name':'Apple', 'Origin':'France', 'Price':120, 'Expiry':'12-08-2025'}
Write the following user-defined functions in Python to perform the specified operations on FruitStack :
(i) push_fruit(FruitStack, Fruit): This function takes the stack FruitStack and a new record Fruit as arguments and pushes the record stored in Fruit onto FruitStack if the Price is less than 100.
(ii) pop_fruit(FruitStack): This function pops the topmost record from the stack and returns it. If the stack is already empty, the function should display "UNDERFLOW".
(iii) display(FruitStack): This function displays all the elements of the stack starting from the topmost element. If the stack is empty, the function should display 'EMPTY STACK'.

Correct Answer: Refer to the Python code blocks below.

Solution:
Step 1: Understanding the Concept:
A Stack is a fundamental Data Structure governed by the Last-In-First-Out (LIFO) operational principle. Python lists handle stack actions excellently using native methods.
Step 2: Key Formula or Approach:
The push operation maps directly to the list **append()** method.
The pop operation maps directly to the list **pop()** method.
Validating underflow requires inspecting if **len(stack) == 0**.
Displaying LIFO involves reverse iteration via **range(len(stack)-1, -1, -1)**.
Step 3: Detailed Explanation:
Push: Access the key **'Price'** via dictionary syntax **Fruit['Price']**. If it evaluates strictly less than 100, execute **FruitStack.append(Fruit)**.
Pop: Determine emptiness utilizing **len()**. Execute print if empty. Otherwise, seamlessly **return FruitStack.pop()**.
Display: Check for emptiness. If items exist, loop backwards starting securely from the last assigned index (**len - 1**) descending to **0**, utilizing a step of **-1**. Print each item sequentially.
Step 4: Final Answer:

```
def push_fruit(FruitStack, Fruit):
    if Fruit['Price'] < 100:
        FruitStack.append(Fruit)

def pop_fruit(FruitStack):
    if len(FruitStack) == 0:
        print("UNDERFLOW")
    else:
        return FruitStack.pop()
```

```
def display(FruitStack):
    if len(FruitStack) == 0:
        print("EMPTY STACK")
    else:
        for i in range(len(FruitStack)-1, -1, -1):
            print(FruitStack[i])
```

Quick Tip

To ensure full marks in stack implementation displays, always iterate and print completely backwards to visually enforce the structural LIFO concept.

30(b). Write a Python program to accept 10 integers from the user. If the entered number is a three-digit even integer, push it onto a stack. After all inputs are taken, pop all the three-digit even integers from the stack and display them. For example, if the user enters 12, 31, 320, 457, 6, 92, 924, 220, 1, 218, then the stack should contain :
320, 924, 220, 218
and the output of the program should be :
218 220 924 320

Correct Answer: Refer to the Python code block below.

Solution:
Step 1: Understanding the Concept:
The script requires iterative input collection, conditional data filtering using mathematical operations, and executing LIFO stack manipulation routines.
Step 2: Key Formula or Approach:
A three-digit positive integer boundary is expressed mathematically as **100 ¡= num ¡= 999**.
An even integer condition is universally verified by **num % 2 == 0**.
A stack is depleted and displayed iteratively using a **while stack:** loop coupled with **pop()**.
Step 3: Detailed Explanation:
Initialize an empty list variable named **stack**.
Construct a **for** loop executing exactly 10 iterations. Inside, capture integer input.
Apply the dual condition combining limits and modulo logic seamlessly using the **and** operator.
If satisfied, **append** the integer to the list.
Following the accumulation loop, construct a **while** loop that processes as long as the stack is not empty. Extract the topmost element via **pop()** and systematically print it utilizing **end=' '** to construct horizontal output spacing matching the example.
Step 4: Final Answer:

```
stack = []

for i in range(10):
```

```
    num = int(input("Enter integer: "))

    # Check if three-digit AND even
    if 100 <= num <= 999 and num % 2 == 0:
        stack.append(num)

# Empty the stack displaying LIFO order
while stack:
    print(stack.pop(), end=' ')
```

Quick Tip

The mechanism **while stack:** acts identically to **while len(stack) ¿ 0:**, providing a clean, natively Pythonic way to systematically exhaust elements from a list.

31(a). Write the output of the following code :
def Exam2026(given):
new=[]
for ch in given[1:-1]:
if ch.isupper():
new.reverse()
elif ch not in new:
new.append(ch)
elif ch in new:
new.pop()
print(new)
Exam2026("Gold-24Medals")

Correct Answer: ['4', '2', '-', 'd', 'l', 'o']

Solution:
Step 1: Understanding the Concept:
Output tracing forces methodical line-by-line simulation of string slicing, conditional logic trees, and dynamic list modification methods.
Step 2: Key Formula or Approach:
String slice [**1:-1**] fundamentally omits the first character and the final character. **pop()** seamlessly removes the last element appended. **reverse()** inverts the entire structural order of the list.
Step 3: Detailed Explanation:
Original input parameter string: **"Gold-24Medals"**.
Slice operation **given[1:-1]**: Extracts **"old-24Medal"**.
We track the active state of **new** iteratively character by character:
1. **'o'**: Not upper. Not in **new**. Condition triggers **append**. **new** = ['o']
2. **'l'**: Not upper. Not in **new**. Append. **new** = ['o', 'l']

3. **'d'**: Not upper. Not in **new**. Append. **new = ['o', 'l', 'd']**
4. **'-'**: Not upper. Not in **new**. Append. **new = ['o', 'l', 'd', '-']**
5. **'2'**: Not upper. Not in **new**. Append. **new = ['o', 'l', 'd', '-', '2']**
6. **'4'**: Not upper. Not in **new**. Append. **new = ['o', 'l', 'd', '-', '2', '4']**
7. **'M'**: It IS structurally uppercase. **new.reverse()** executes. **new** becomes **['4', '2', '-', 'd', 'l', 'o']**
8. **'e'**: Not upper. Not in **new**. Append. **new = ['4', '2', '-', 'd', 'l', 'o', 'e']**
9. **'d'**: Not upper. It IS currently residing in **new**. Condition triggers **pop()**. Removes the final element (**'e'**). **new = ['4', '2', '-', 'd', 'l', 'o']**
10. **'a'**: Not upper. Not in **new**. Append. **new = ['4', '2', '-', 'd', 'l', 'o', 'a']**
11. **'l'**: Not upper. It IS residing in **new**. Condition triggers **pop()**. Removes the final element (**'a'**). **new = ['4', '2', '-', 'd', 'l', 'o']**
Execution terminates. Resulting list prints.
Step 4: Final Answer:
['4', '2', '-', 'd', 'l', 'o']

Quick Tip

A critical error trap: The **pop()** method natively removes the extreme rightmost element in the list, completely regardless of which element actively triggered the internal **elif ch in new:** condition.

31(b). Write the output of the following code :

```
def Exam2026(given):
new = 0
while given:
if new % 2:
new += given % 10
else:
new += given % 5
print(new, end='-')
given //= 10
Exam2026(123456)
```

Correct Answer: 1-6-10-13-15-16-

Solution:
Step 1: Understanding the Concept:
Output determination requires tracing numerical extraction using integer division algorithms alongside mutating Boolean conditional structures inside a **while** loop.
Step 2: Key Formula or Approach:
The operation **num % 10** extracts the absolute last digit. The operation **num //= 10** effectively strips off the absolute last digit.
The condition **if new % 2:** evaluates seamlessly as **True** if **new** is odd, and **False** if **new** is

even.
Step 3: Detailed Explanation:
Initiate with **given = 123456**, **new = 0**.
Iter 1: **given = 123456**. Condition **0 % 2 == 0** (**False**). **else** branch fires.
Calculation: **new += 123456 % 5**. The remainder is 1. **new** becomes **1**. Output **1-**. **given //= 10** leaves **12345**.
Iter 2: **given = 12345**. Condition **1 % 2 == 1** (**True**). **if** branch fires.
Calculation: **new += 12345 % 10**. Last digit is 5. **new = 1 + 5 = 6**. Output **6-**. **given** becomes **1234**.
Iter 3: **given = 1234**. Condition **6 % 2 == 0** (**False**). **else** branch fires.
Calculation: **new += 1234 % 5**. The remainder is 4. **new = 6 + 4 = 10**. Output **10-**. **given** becomes **123**.
Iter 4: **given = 123**. Condition **10 % 2 == 0** (**False**). **else** branch fires.
Calculation: **new += 123 % 5**. The remainder is 3. **new = 10 + 3 = 13**. Output **13-**. **given** becomes **12**.
Iter 5: **given = 12**. Condition **13 % 2 == 1** (**True**). **if** branch fires.
Calculation: **new += 12 % 10**. Last digit is 2. **new = 13 + 2 = 15**. Output **15-**. **given** becomes **1**.
Iter 6: **given = 1**. Condition **15 % 2 == 1** (**True**). **if** branch fires.
Calculation: **new += 1 % 10**. Last digit is 1. **new = 15 + 1 = 16**. Output **16-**. **given** becomes **0**.
Loop terminates since **0** is evaluated as **False**.
Step 4: Final Answer:
1-6-10-13-15-16-

> Quick Tip
>
> Modulo 5 (**num % 5**) simply requires evaluating the final digit of the number. For instance, **123456 % 5** generates the same remainder as **6 % 5**.

Data Set for Question 32
Abhishek has created a table, named STOCK, with a set of records to maintain the data of packaged milk in his shop. After creating the table, he entered the data and the table looked as follows:

Code	Type	Volume	Qty	Price
AF0.5	F	0.5	300	38.00
MF0.5	F	0.5	250	36.50
MT1.0	T	1.0	150	64.00
AT1.0	T	1.0	100	66.00
PD1.0	D	1.0	50	52.00
PT0.5	T	0.5	78	30.00

32(a)(i). Based on the data given above, write the SQL queries for the following tasks :
To display Type and the maximum Price for each Type of milk.

Correct Answer: SELECT Type, MAX(Price) FROM STOCK GROUP BY Type;

Solution:
Step 1: Understanding the Concept:
Grouping data records categorically to perform distinct mathematical calculations per category.
Step 2: Key Formula or Approach:
The English phrasing "for each [ColumnName]" undeniably dictates the explicit use of the **GROUP BY [ColumnName]** clause paired strictly with an aggregate function.
Step 3: Detailed Explanation:
The target attributes requested for visual display are the categorical **Type** column and a mathematically computed column.
The computed task demands extracting the absolute highest numerical value for price within a group, requiring the **MAX(Price)** aggregate function.
To structurally separate the data rows into distinct batches based strictly on milk type, append the **GROUP BY Type** clause at the very end.
Step 4: Final Answer:
SELECT Type, MAX(Price) FROM STOCK GROUP BY Type;

Quick Tip

The exact column designated inside the **GROUP BY** clause universally mirrors the primary non-aggregated column specifically listed inside the preceding **SELECT** clause.

32(a)(ii). Based on the data given above, write the SQL queries for the following tasks :
For each record, increase the Price by 0.5 where Type is 'F'.

Correct Answer: UPDATE STOCK SET Price = Price + 0.5 WHERE Type = 'F';

Solution:
Step 1: Understanding the Concept:
Modifying existing data fields (row values) stored physically within a relational database.
Step 2: Key Formula or Approach:
Data manipulation modifications uniformly utilize the **UPDATE TableName SET ColumnName = NewValue WHERE Condition;** paradigm.
Step 3: Detailed Explanation:
Specify the exact target table using **UPDATE STOCK**.

Assign the new numerical assignment parameter using the **SET** keyword. The prompt requests incrementing current values structurally by **0.5**, formulated algebraically as **SET Price = Price + 0.5**.
Restrict the update exclusively to specific rows using **WHERE Type = 'F'**. Without this restriction, every single record in the database table would improperly receive a price increase.
Step 4: Final Answer:
UPDATE STOCK SET Price = Price + 0.5 WHERE Type = 'F';

> Quick Tip
>
> Mathematical operators are perfectly valid directly inside the **SET** assignments. Never attempt to use the **ALTER** command here, as it alters structural schemas, not individual row values.

32(a)(iii). Based on the data given above, write the SQL queries for the following tasks :
To display the total value of the stock (total of Qty × Price).

Correct Answer: SELECT SUM(Qty * Price) FROM STOCK;

Solution:
Step 1: Understanding the Concept:
Performing intra-row algebraic operations mathematically combined across the entire table using an aggregate summation.
Step 2: Key Formula or Approach:
Mathematical expressions can legally reside within aggregate function arguments. **SUM(column1 * column2)** calculates the product for every row and subsequently totals them globally.
Step 3: Detailed Explanation:
The problem statement explicitly formulates the mathematical task: **Qty × Price**.
First, the database engine sequentially calculates the isolated total valuation specifically for each distinct row (**Qty * Price**).
Next, to output one consolidated grand total for the shop entirely, we envelop the mathematical expression deeply inside the **SUM()** aggregate function.
No **GROUP BY** is requested since the prompt seeks a singular unified total.
Step 4: Final Answer:
SELECT SUM(Qty * Price) FROM STOCK;

> Quick Tip
>
> Expressions inside aggregate functions are powerful tools. They completely eliminate the redundant need to create separate views or virtual calculated columns.

32(a)(iv). Based on the data given above, write the SQL queries for the following tasks :
To display the details of all records where Code starts with 'A'.

Correct Answer: SELECT * FROM STOCK WHERE Code LIKE 'A%';

Solution:
Step 1: Understanding the Concept:
String filtering applying designated wildcard sequences universally mapping partial text patterns.
Step 2: Key Formula or Approach:
Use the **SELECT *** paradigm to demand "all details" (all columns). Use the **LIKE** logical operator partnered strategically with the % percentage wildcard for string matching.
Step 3: Detailed Explanation:
To fetch every existing column attribute universally, deploy the **SELECT *** statement.
The restrictive condition focuses solely on the **Code** column.
To specify that a string must definitively commence with a literal 'A', we formulate the **LIKE** parameter as **'A%'**.
The % explicitly signifies "allow zero, one, or entirely multiple trailing random characters," mandating only the starting character check.
Step 4: Final Answer:
SELECT * FROM STOCK WHERE Code LIKE 'A%';

Quick Tip

Contrast this with the underscore wildcard _, which replaces exactly one solitary character. For instance, **LIKE '_A%'** targets strings where 'A' is the second character securely.

32(b)(i). Considering the table STOCK as given above, write the output on execution of the following queries :
SELECT Volume, Qty, Price FROM STOCK WHERE Type IN ('F', 'D');

Correct Answer: Refer to the tabular output structure in the solution block.

Solution:
Step 1: Understanding the Concept:
The **IN** operator evaluates lists of permissible values, functioning identically to executing multiple combined **OR** logical conditions seamlessly.
Step 2: Key Formula or Approach:
Sequentially scan down the target table isolating rows where the attribute **Type** definitively

equates to **'F'** OR **'D'**. From these successfully filtered rows, rigorously extract only the strictly specified columns.
Step 3: Detailed Explanation:
Let's filter sequentially row by row:
Row 1 (AF0.5): **Type = 'F'**. Valid. Extract → Volume: 0.5, Qty: 300, Price: 38.00.
Row 2 (MF0.5): **Type = 'F'**. Valid. Extract → Volume: 0.5, Qty: 250, Price: 36.50.
Row 3 (MT1.0): **Type = 'T'**. Invalid constraint.
Row 4 (AT1.0): **Type = 'T'**. Invalid constraint.
Row 5 (PD1.0): **Type = 'D'**. Valid. Extract → Volume: 1.0, Qty: 50, Price: 52.00.
Row 6 (PT0.5): **Type = 'T'**. Invalid constraint.
Construct an organized grid containing these values aligned flawlessly beneath exactly matching header labels.
Step 4: Final Answer:

Volume	Qty	Price
0.5	300	38.00
0.5	250	36.50
1.0	50	52.00

Quick Tip

Always construct SQL query output manually imitating tabular formatting. Omitting header column names routinely leads to penalizations marking exams.

32(b)(ii). Considering the table STOCK as given above, write the output on execution of the following queries :
SELECT Code, Qty FROM STOCK WHERE Price BETWEEN 30 AND 50;

Correct Answer: Refer to the tabular output structure in the solution block.

Solution:
Step 1: Understanding the Concept:
The **BETWEEN x AND y** operator tests numeric inclusiveness within boundaries seamlessly without utilizing complex $\geq$ / $\leq$ combinations.
Step 2: Key Formula or Approach:
Test every record. If $30 \leq \text{Price} \leq 50$, accept the tuple. Extract only **Code** and **Qty**.
Step 3: Detailed Explanation:
Evaluate **Price** linearly:
AF0.5: Price is 38.00. (Fits interval) → Code: AF0.5, Qty: 300
MF0.5: Price is 36.50. (Fits interval) → Code: MF0.5, Qty: 250
MT1.0: Price is 64.00. (Too high)
AT1.0: Price is 66.00. (Too high)
PD1.0: Price is 52.00. (Too high)
PT0.5: Price is 30.00. (Boundary exact, so included) → Code: PT0.5, Qty: 78

Step 4: Final Answer:

```
Code     Qty
AF0.5    300
MF0.5    250
PT0.5    78
```

> Quick Tip
>
> Be extremely careful: the **BETWEEN** operator unconditionally encompasses endpoints. Hence, **30.00** is unequivocally a valid inclusion.

32(b)(iii). Considering the table STOCK as given above, write the output on execution of the following queries :
SELECT DISTINCT Type FROM STOCK;

Correct Answer: Refer to the tabular output structure in the solution block.

Solution:
Step 1: Understanding the Concept:
Isolating unrepeated singular values specifically from a heavily populated database column.
Step 2: Key Formula or Approach:
The **DISTINCT** operational keyword strictly filters duplicate records from the final constructed output view.
Step 3: Detailed Explanation:
Extract the entire **Type** column top to bottom: **F, F, T, T, D, T**.
Run a sequential uniqueness scan:
F (New $\rightarrow$ Print)
F (Seen earlier $\rightarrow$ Ignore)
T (New $\rightarrow$ Print)
T (Seen earlier $\rightarrow$ Ignore)
D (New $\rightarrow$ Print)
T (Seen earlier $\rightarrow$ Ignore)
The resultant array yields **F**, **T**, and **D**. Output vertically beneath the standard header.
Step 4: Final Answer:

```
Type
F
T
D
```

Quick Tip

The output order generally mimics the occurrence order located naturally in the target table structurally.

32(b)(iv). Considering the table STOCK as given above, write the output on execution of the following queries :
SELECT Volume, count(*) FROM STOCK GROUP BY Volume;

Correct Answer: Refer to the tabular output structure in the solution block.

Solution:
Step 1: Understanding the Concept:
Output generation involving clustered grouping and numeric record enumeration via aggregation.
Step 2: Key Formula or Approach:
Segment rows possessing identical **Volume** quantities strictly into groups. Execute **count(*)** sequentially to dictate the population volume of each defined group.
Step 3: Detailed Explanation:
Inspect **Volume** values uniformly: There are two distinct values, **0.5** and **1.0**.
Cluster Group 1 (**Volume = 0.5**): Rows involved are AF0.5, MF0.5, PT0.5. Total rows calculated = 3.
Cluster Group 2 (**Volume = 1.0**): Rows involved are MT1.0, AT1.0, PD1.0. Total rows calculated = 3.
Present the output grid featuring the specified headers natively matching the exact SQL query syntax.
Step 4: Final Answer:

```
Volume   count(*)
0.5      3
1.0      3
```

Quick Tip

To ensure perfection, mirror the header text absolutely. Ensure you write **count(*)** exactly instead of translating it simply to 'Total' or 'Count'.

---

33. A csv file "States.csv" contains some data about all the states of India. Each record of the file contains the following data :

- **Name of the State**
- **Capital of the State**
- **Population of the State**

- **Official Language of the State**

For example, a sample record in the file is :
['Andhra Pradesh', 'Amaravati', 52221000, 'Telugu']
Write a Python program which reads the data from this file and appends all those records where population is more than 10000000 into another csv file 'More.csv'.
Note : "States.csv" also contains the Header row. The Header row should NOT be copied to "More.csv".

Correct Answer: Refer to the compiled Python code block below.

Solution:
Step 1: Understanding the Concept:
Python's built-in **csv** module securely processes comma-separated values natively representing spreadsheet layouts. Operations demand initializing reader and appending writer objects interactively.
Step 2: Key Formula or Approach:
Open source **'r'** and destination **'a'**. Execute **next(reader)** to deliberately bypass headers. Typecast the list element holding populations index securely into **int()** to perform numeric relational evaluations ($>$ **10000000**). Output valid rows via **writerow()**.
Step 3: Detailed Explanation:
1. Import the crucial **csv** library to access functionalities.
2. Utilize **with open(...)** managing dual files securely, employing **newline=''** extensively to bypass Windows-related row-spacing bugs.
3. Initialize **reader** and **writer** linked directly to active files.
4. The **next(reader)** statement efficiently fetches the first string array (header labels) immediately eliminating it from upcoming iteration paths.
5. In a **for row in reader:** iteration block, address element **row[2]** (representing Population). Convert strictly to **int** before evaluating against **10000000**.
6. When verified **True**, push the complete sequential **row** structure linearly into the file via **writer.writerow(row)**.
Step 4: Final Answer:

```
import csv

def filter_states():
    # Open target files safely in specified operational modes
    with open('States.csv', 'r') as file1, open('More.csv', 'a', newline='') as file2:
        # Create CSV engine objects
        reader = csv.reader(file1)
        writer = csv.writer(file2)

        # Omit explicitly the header array row from loop mechanics
        next(reader)

        # Perform iterative numeric extraction and copying tasks
        for row in reader:
```

```
if int(row[2]) > 10000000:
    writer.writerow(row)
```

Quick Tip

It is incredibly critical to convert CSV fields forcefully using **int()** or **float()** before performing math bounds checking, because CSV readers inherently categorize absolutely everything as pure string data natively.

Data Set for Question 34
Assume that you are the Manager of the Loans department of a Finance House. To keep track of the loans you have created two tables : CUSTOMERS and LOANS. The sample data in these tables is given below :

Table: CUSTOMERS

C_ID	C_Name	Phone
00001	Raj Malhotra	1234567890
00003	David Xavier	3456789012
00004	Damini Iyer	3156789012
00008	Abdul	2345678901

Table: LOANS

SNo	C_ID	L_Amt	L_Date	Terms	RoI
1	00003	200000	2025-12-06	60	7.80
2	00008	2500000	2023-08-09	60	9.00
3	00001	500000	2025-08-13	48	6.00
4	00003	300000	2026-12-07	36	8.00
5	00004	600000	2026-12-07	60	6.00

34(i). Write the queries to extract the following data to create the reports : Number of records from LOANS table where Rate of Interest (RoI) is above 7.0.

Correct Answer: SELECT COUNT(*) FROM LOANS WHERE RoI ¿ 7.0;

Solution:
Step 1: Understanding the Concept:
Tracking volume enumeration leveraging aggregate database functionalities against structured limits.
Step 2: Key Formula or Approach:
The aggregate **COUNT(*)** formula comprehensively enumerates valid rows resolving a specific **WHERE** condition reliably.
Step 3: Detailed Explanation:
Specify selection of the entire **LOANS** table logically.
The parameter restricts extraction to entries where **RoI** calculates fundamentally greater than **7.0**. This creates **WHERE RoI ¿ 7.0**.
Instead of delivering data fields, the objective requires counting occurrence frequency, fulfilled perfectly via **SELECT COUNT(*)**.
Step 4: Final Answer:

SELECT COUNT(*) FROM LOANS WHERE RoI ¿ 7.0;

Quick Tip

COUNT(*) incorporates and counts every single row matching the condition perfectly, whereas **COUNT(ColumnName)** omits tuples where the designated column registers explicitly as **NULL**.

34(ii). Write the queries to extract the following data to create the reports : Names of the customers whose loan amount (L_Amt) is above 1000000.

Correct Answer: SELECT C.C_Name FROM CUSTOMERS C, LOANS L WHERE C.C_ID = L.C_ID AND L.L_Amt ¿ 1000000;

Solution:
Step 1: Understanding the Concept:
Merging information strictly distributed efficiently across dual tables executing Cartesian product filtering (Equi-Joins).
Step 2: Key Formula or Approach:
An Equi-Join dictates equating mutual foreign and primary parameters via **Table1.Key = Table2.Key** explicitly embedded within a **WHERE** construct.
Step 3: Detailed Explanation:
The **C_Name** string resides functionally inside the **CUSTOMERS** table.
The validation metric **L_Amt** dwells inside the **LOANS** table.
To synchronize tuples reliably, an intrinsic relationship join utilizing the common bridge **C_ID** is absolutely mandatory: **CUSTOMERS.C_ID = LOANS.C_ID**.
The subsequent filtering threshold constraint adds **L_Amt ¿ 1000000**.
Fusing conditions with logical **AND** and integrating table aliases simplifies text density securely.
Step 4: Final Answer:
SELECT C.C_Name FROM CUSTOMERS C, LOANS L WHERE C.C_ID = L.C_ID AND L.L_Amt ¿ 1000000;

Quick Tip

Any SQL question requiring data display from Table A combined tightly with checking conditions on Table B uniformly demands composing a table join mechanically.

34(iii). Write the queries to extract the following data to create the reports : C_ID, C_Name and Terms of all those records where Loan Date (L_Date) is after

31st December, 2024.

Correct Answer: SELECT C.C_ID, C.C_Name, L.Terms FROM CUSTOMERS C, LOANS L WHERE C.C_ID = L.C_ID AND L.L_Date ¿ '2024-12-31';

Solution:
Step 1: Understanding the Concept:
Compiling cross-table attributes seamlessly while parsing temporal database chronological strings natively.
Step 2: Key Formula or Approach:
Leverage table Equi-Joins natively identical to the previous part. Evaluate chronological parameters uniformly formatted entirely as strings **'YYYY-MM-DD'**.
Step 3: Detailed Explanation:
Identify necessary export fields mapping specific tables: **C.C_ID**, **C.C_Name**, **L.Terms**.
Formulate bridge matching: **C.C_ID = L.C_ID**.
Chronological requirement dictates events "after 31st December, 2024". Formulate string logically: **L.L_Date ¿ '2024-12-31'**.
Append parameters via **AND**.
Step 4: Final Answer:
SELECT C.C_ID, C.C_Name, L.Terms FROM CUSTOMERS C, LOANS L WHERE C.C_ID = L.C_ID AND L.L_Date ¿ '2024-12-31';

Quick Tip

MySQL parses native standard dates exactly matching lexicographical string behavior sequentially; hence placing strict apostrophes encapsulating dates perfectly formatted as YYYY-MM-DD is permanently mandatory.

34(iv)(a). Write the queries to extract the following data to create the reports : Details of all the loans in the descending order of RoI.

Correct Answer: SELECT * FROM LOANS ORDER BY RoI DESC;

Solution:
Step 1: Understanding the Concept:
Manipulating dataset visual projection arranging entries universally via numeric sorting mechanisms natively.
Step 2: Key Formula or Approach:
The logical implementation **ORDER BY ColumnName DESC** forces top-to-bottom numerical decay projection seamlessly.
Step 3: Detailed Explanation:
The instruction dictates displaying "Details of all...". Deploy **SELECT *** uniformly over the **LOANS** table schema.

The arranging directive explicitly references the **RoI** parameter scaling mathematically backwards. Implement this structurally utilizing the **ORDER BY RoI DESC** trailing command appended appropriately.
Step 4: Final Answer:
SELECT * FROM LOANS ORDER BY RoI DESC;

> Quick Tip
>
> Omitting the **DESC** modifier results essentially in ascending logic implicitly being utilized exclusively, corrupting grading fundamentally.

OR

34(iv)(b) Write the queries to extract the following data to create the reports : C_ID and average term for each C_ID from the LOANS table.

Correct Answer: SELECT C_ID, AVG(Terms) FROM LOANS GROUP BY C_ID;

Solution:
Step 1: Understanding the Concept:
Executing mathematical arithmetic functionally mapping categorically over clustered segments representing individuals uniquely.
Step 2: Key Formula or Approach:
Any textual presence of "for each" universally triggers establishing a **GROUP BY** structural segregation natively. Use **AVG()** for arithmetic means.
Step 3: Detailed Explanation:
Target grouping attribute specified: **C_ID**. Consequently, **GROUP BY C_ID** constitutes the query backend reliably.
Target aggregate projection: Extract **C_ID** alongside calculating the mean duration (**Terms**) assigned, deploying **AVG(Terms)**.
Combining logically forms a correct syntax.
Step 4: Final Answer:
SELECT C_ID, AVG(Terms) FROM LOANS GROUP BY C_ID;

> Quick Tip
>
> Attempting blindly to use **SELECT C_ID, AVG(Terms) FROM LOANS** ignoring the **GROUP BY** keyword immediately triggers total database calculation crashes functionally.

---

35. Peter has created a table named Account in MySQL database, SCHOOL, having following structure :

- **Stud_id - integer**
- **Sname - string**
- **Class - string**
- **Fees - float**

Help him in writing a Python program to display records of those students whose fees is less than 5000.

Note the following to establish connectivity between Python and MySQL :

- **Username - admin**
- **Password - root**
- **Host - localhost**

Correct Answer: Refer to the successfully compiled Python script below seamlessly enabling database access natively.

Solution:

Step 1: Understanding the Concept:

Establishing cross-platform relational linkage integrating Pythonic logic firmly with active MySQL environments demands deploying specialized cursor frameworks and execution pipelines properly.

Step 2: Key Formula or Approach:

Import module → Initialize Connection credentials → Instanciate Cursor → Call **execute(Query)** → Retrieve variables via **fetchall()** → Terminate securely utilizing **close()**.

Step 3: Detailed Explanation:

1. Invoke standard library component **mysql.connector**.
2. Map connectivity parameters comprehensively using **connect(host="localhost", user="admin", password="root", database="SCHOOL")**.
3. Instantiate an interactive cursor object capable of channeling SQL instructions efficiently via **cursor()**.
4. Construct accurate SQL retrieval query targeting parameters: **"SELECT * FROM Account WHERE Fees ¡ 5000"**.
5. Command transmission natively utilizing **cursor.execute(query)**.
6. Harvest total returned payload logically mapping into tuple arrays through **fetchall()**.
7. Deconstruct payload executing a **for** loop parsing tuples sequentially and output.
8. Guarantee operational security closing pipeline **conn.close()**.

Step 4: Final Answer:

```
import mysql.connector

def display_records():
    # Integrate authentication parameters
    conn = mysql.connector.connect(
        host="localhost",
        user="admin",
        password="root",
```

```
    database="SCHOOL"
)

# Instantiate querying engine cursor
cursor = conn.cursor()

# Establish target query
query = "SELECT * FROM Account WHERE Fees < 5000"

# Fire the operation
cursor.execute(query)

# Harvest incoming tuples
records = cursor.fetchall()

# Display systematically
for row in records:
    print(row)

# Purge connection safely
conn.close()
```

Quick Tip

Applying **conn.commit()** is totally unnecessary explicitly for standard **SELECT** requests, as no underlying schema tables or record parameters undergo write modifications iteratively.

---

36. NextStep is an organization which has a pool of resource persons to conduct training workshops on various topics related to ICT. The data of all its Resource Persons is stored in a binary file RESOURCES.DAT using the following record structure (each record is a tuple) :
(R_ID, R_Name, R_Expertise, Charges)
where :
R_ID - Resource Person's ID (An integer)
R_Name - Resource Person's Name (A string)
R_Expertise - Area of expertise of the Resource Person
Charges - Charges (in rupees) per hour to conduct a workshop
For example, a record in the file is :
(12, 'P. Velusami', 'Machine Learning', 5000)
In this context, write the following user defined functions in Python :
(i) Append() - To input the data of a Resource Person and write it in the file RESOURCES.DAT.
(ii) Update() - To increase the Charges of each resource person by 500.

Correct Answer: Refer to the successfully compiled functional scripts detailed precisely underneath solving both components reliably.

Solution:
Step 1: Understanding the Concept:
Binary operations in Python mandate exclusive deployment of the **pickle** serialization module facilitating arbitrary object mapping to physical byte strings sequentially.
Step 2: Key Formula or Approach:
For appending, file mode strictly equates to **'ab'**, subsequently triggering **pickle.dump()**.
For updating uniformly, file extraction mandates **'rb'**, looping **pickle.load()** to list memory, modifying internally, and overwriting executing **'wb'** entirely.
Step 3: Detailed Explanation:
(i) Append() Segment:
Acquire attributes manually utilizing standard **input()** and enforce typecasting constraints **int()** actively for variables **R_ID** and **Charges**.
Consolidate entities systematically into a structured tuple object format perfectly aligning with requirements.
Open binary file executing appended write access securely (**'ab'**) and trigger payload dump sequentially.
(ii) Update() Segment:
Initialize **records** empty list buffer explicitly.
Construct defensive **try...except EOFError** iteration wrapper. Open executing native read binary access safely (**'rb'**) and forcefully harvest records natively utilizing **load()** appending directly into buffer lists continuously until file culmination.
Subsequent to harvesting completion, initiate primary overwrite. Access file executing complete write binary access (**'wb'**) bypassing history.
Iterate memory buffer array precisely. Since Pythonic tuples strictly exhibit immutability implicitly, recreate structural variables manually reassembling unchanged parameters (**rec[0]**, **rec[1]**, **rec[2]**) paired efficiently with mathematically adjusted parameters (**rec[3] + 500**). Execute **dump()** natively repeatedly over records.
Step 4: Final Answer:

```
import pickle

def Append():
    # Input retrieval pipeline
    r_id = int(input("Enter ID: "))
    r_name = input("Enter Name: ")
    r_exp = input("Enter Expertise: ")
    charges = int(input("Enter Charges: "))

    # Tuple grouping systematically
    record = (r_id, r_name, r_exp, charges)

    # Binary append dump mechanics
    with open("RESOURCES.DAT", "ab") as file:
        pickle.dump(record, file)
```

```
def Update():
    records = []

    # Exhaustive harvest pipeline
    try:
        with open("RESOURCES.DAT", "rb") as file:
            while True:
                records.append(pickle.load(file))
    except EOFError:
        pass

    # Reconfiguration overwrite pipeline
    with open("RESOURCES.DAT", "wb") as file:
        for rec in records:
            # Assembly overcoming immutability seamlessly
            modified_rec = (rec[0], rec[1], rec[2], rec[3] + 500)
            pickle.dump(modified_rec, file)
```

Quick Tip

Recognize deeply that tuples refuse index assignments securely (**rec[3] = 999** will crash fundamentally). You must recreate tuple boundaries identically while applying dynamic overrides mapping parameter alterations correctly.

Data Set for Question 37

Sanjeevani is a big group of educational institutions with its head office in Hyderabad. It is planning to set up a new University in Amritsar. The Amritsar University Campus will have four blocks/buildings - ADMIN, ACADEMIC, HOSTEL, SPORTS. You, as a network expert, need to suggest the best network-related solutions for them to resolve the issues/problems mentioned in points (i) to (v), keeping in mind the distances between various blocks / buildings and other given parameters.

Block to Block distances (in Mtrs.)

FROM	TO	DISTANCE
ADMIN	ACADEMIC	60 M
ADMIN	HOSTEL	160 M
ADMIN	SPORTS	80 M
ACADEMIC	HOSTEL	40 M
ACADEMIC	SPORTS	120 M
HOSTEL	SPORTS	150 M

Number of computers in each block is as follows :

ADMIN	25
ACADEMIC	600
HOSTEL	120
SPORTS	50

37(i). Suggest the most appropriate location of the server inside the Amritsar University Campus. Justify your choice.

Correct Answer: ACADEMIC Block.

Solution:
Step 1: Understanding the Concept:
Optimum server placements structurally demand reducing cable data transmission delays practically by localizing heavy bandwidth traffic directly resolving globally accepted 80-20 mathematical deployment rules consistently.
Step 2: Key Formula or Approach:
Analyze hardware densities. Establish deployment directly targeting the single specific location actively displaying the absolute mathematical maximum computer density explicitly.
Step 3: Detailed Explanation:
Examining computer demographics presented structurally:
ADMIN $\rightarrow$ 25 computers.
ACADEMIC $\rightarrow$ 600 computers.
HOSTEL $\rightarrow$ 120 computers.
SPORTS $\rightarrow$ 50 computers.
ACADEMIC unquestionably possesses the vastly predominant absolute majority. Locating central processing operations natively inside ACADEMIC guarantees approximately maximum traffic transactions compute locally directly on ultra-fast internal LANs completely bypassing vulnerable external inter-building physical copper links perfectly.
Step 4: Final Answer:
Server Placement Location: ACADEMIC Block.
Justification: It inherently houses the absolute maximum number of networked computers (600), fundamentally adhering securely to the 80-20 bandwidth rule systematically reducing overall external physical network traffic overheads comprehensively.

Quick Tip

To safely guarantee full credit marks uniformly on server deployment placement answers routinely, literally copy the phrase "because it natively houses the absolute maximum number of computers".

37(ii). Draw the cable layout to efficiently connect various blocks within the Amritsar University Campus.

Correct Answer: A localized Star Topology architecture centered directly at the ACADEMIC Block.

Solution:
Step 1: Understanding the Concept:
Determining inter-building physical connection structures demands designing a network uniformly minimizing utilized cable lengths drastically guaranteeing hardware costs remaining systematically manageable securely establishing Star Topologies essentially.
Step 2: Key Formula or Approach:
Position the primary Hub/Server explicitly in the center natively. Establish direct linear physical connections diverging outwardly integrating remaining outer nodes individually evaluating distances precisely.
Step 3: Detailed Explanation:
The central deployment node systematically designated previously functions as ACADEMIC.
Establish connections structurally:
ACADEMIC $\leftrightarrow$ HOSTEL explicitly measures 40 Meters safely.
ACADEMIC $\leftrightarrow$ ADMIN explicitly measures 60 Meters safely.
ACADEMIC $\leftrightarrow$ SPORTS explicitly measures 120 Meters safely.
The aggregated total physical cable required mathematically equates natively to roughly 220 Meters exactly, forming an extremely efficient highly optimized localized Star Network layout smoothly eliminating redundancies functionally.
Step 4: Final Answer:
(The student should physically draw a diagram on paper designating the ACADEMIC block situated at the central focal point, drawing distinct lines connecting linearly to the outer three auxiliary blocks uniformly).
Layout Path Plan Formatted:
ACADEMIC Block $\longleftrightarrow$ HOSTEL Block
ACADEMIC Block $\longleftrightarrow$ ADMIN Block
ACADEMIC Block $\longleftrightarrow$ SPORTS Block

Quick Tip

Avoid complex Ring or Mesh Topologies structurally unless expressly demanded by highly specific prompt instructions explicitly, as they consistently increase arbitrary cable lengths exponentially without practical necessity structurally.

37(iii). Name any two wired media that can be used to connect various computers of a block inside Amritsar Campus.

Correct Answer: Twisted Pair Cable (Ethernet) and Coaxial Cable.

Solution:
Step 1: Understanding the Concept:
Connecting localized arrays of functional machines securely internally mandates the use of

physically tangible highly conductive standard physical transmission mediums structurally executing Local Area Networks accurately.
Step 2: Key Formula or Approach:
Extract basic definitions distinguishing physically "wired" (Guided mediums) separating reliably from specifically "wireless" (Unguided) methods correctly.
Step 3: Detailed Explanation:
Inside singular buildings natively, extremely expensive Optical Fiber is entirely unnecessary structurally.
The fundamental prevailing global standard executing inter-computer LAN connections essentially is the Twisted Pair Cable securely (CAT5, CAT6 Ethernet arrays perfectly).
A highly reliable alternative traditionally supporting legacy deployments systematically comprises standard Coaxial Cable.
Step 4: Final Answer:
1. Twisted Pair Cable (Ethernet/UTP/STP Cable).
2. Coaxial Cable.

Quick Tip

Listing "Wi-Fi" or "Optical Fiber" functionally results natively in marked deductions implicitly; "Wi-Fi" fails explicitly being "wireless", while "Fiber" technically links buildings externally, not computers natively internally.

37(iv). For the academic purpose, the University will provide its own 24×7 FM channel within the University Campus. Which communication medium, out of the following, is used by FM ?

(A) **Radio Waves**
(B) **Micro Waves**
(C) **Infrared Waves**

Correct Answer: (A) **Radio Waves**

Solution:
Step 1: Understanding the Concept:
Different frequencies executing on the standardized electromagnetic spectrum inherently power distinct communication technologies natively functionally depending securely on transmission characteristics essentially.
Step 2: Key Formula or Approach:
Evaluate acronyms strictly mapping FM (Frequency Modulation) securely directly aligning frequencies precisely.
Step 3: Detailed Explanation:
Infrared technology essentially relies entirely unconditionally on severely restricted short-range explicit line-of-sight limits natively (e.g., Remote controls).
Microwaves effectively manage extremely high-band secure unidirectional point-to-point physical dish networks accurately.

FM Broadcasting stations unequivocally deploy globally standard Radio Waves implicitly broadcasting unconditionally spanning omnidirectional massive geographical terrain areas completely natively.
Step 4: Final Answer:
(A) **Radio Waves**

> Quick Tip
>
> Any occurrence citing radio broadcast mechanics, conventional Walkie-Talkies, or analog AM/FM communication uniformly equates undeniably straight exclusively to Radio Waves implicitly.

37(v)(a). The students will be attending a lot of online academic sessions and workshops. These will involve audio-visual communication.
Write the full name of the protocol which will be used for such a communication through the internet.

Correct Answer: Voice over Internet Protocol (VoIP).

Solution:
Step 1: Understanding the Concept:
Real-time multimedia stream transmissions executed securely utilizing high-speed broadband network structures uniformly employ highly specific procedural architectures reliably.
Step 2: Key Formula or Approach:
Identify universally adopted standardized acronyms structurally bridging internet audio technologies.
Step 3: Detailed Explanation:
Transferring massive simultaneous synchronized multimedia effectively strictly mandates executing VoIP.
VoIP seamlessly transcodes live analog streams originating fundamentally from physical microphones and cameras directly converting entirely into digital packets routing instantly leveraging IP connectivity perfectly.
Step 4: Final Answer:
Voice over Internet Protocol (VoIP).

> Quick Tip
>
> Keywords highlighting "audio-visual", "Skype", "Zoom", "Video Conferencing", or specifically "online voice communications" perfectly signify absolute reliance uniformly on VoIP exclusively.

OR

37(v)(b). Where should a repeater be installed in Amritsar University campus to boost the signal between blocks ? Justify your answer.

Correct Answer: Specifically installed strategically between the ACADEMIC Block and the SPORTS Block.

Solution:
Step 1: Understanding the Concept:
When transmitted natively across extended lengths utilizing physical copper boundaries, digital signals suffer attenuation (signal degradation/loss). Repeaters actively resolve this.
Step 2: Key Formula or Approach:
The established physical capability baseline boundary natively constraining conventional Ethernet Twist Pair cables reliably explicitly equals universally 100 Meters accurately. Distances extending beyond absolutely demand repeater installation inherently.
Step 3: Detailed Explanation:
Critically scrutinize explicitly the exact physical cable layout designated inherently in part (ii). The active defined links structurally are:
ACADEMIC $\leftrightarrow$ HOSTEL strictly 40 M. (Safe ≤ 100)
ACADEMIC $\leftrightarrow$ ADMIN strictly 60 M. (Safe ≤ 100)
ACADEMIC $\leftrightarrow$ SPORTS strictly 120 M. (Danger > 100).
The linkage linking ACADEMIC and SPORTS decisively severely exceeds standard limitation boundaries fundamentally, making transmission practically unusable naturally without active regeneration amplification hardware deployment securely.
Step 4: Final Answer:
A repeater absolutely must systematically be explicitly installed strategically between the ACADEMIC Block and the SPORTS Block directly.
Justification: The explicit structural distance separating these two blocks measures exactly 120 meters organically, which severely exceeds universally established standard transmission limits implicitly equating to 100 meters effectively deployed for physical Ethernet wires, risking immense signal degradation.

Quick Tip

Always strictly evaluate only the explicit connections actively drawn effectively in your layout in part (ii), completely ignoring unrelated possible block distances located passively embedded in the dataset table.