School Connect Online

SCO INTERNATIONAL OLYMPIAD

CLASS 11 SAMPLE QUESTION PAPER

SCO International Coding Olympiad | Practice Set with Answer Key and Explanations

Designed from the official Class 11 Coding Olympiad pattern for structured practice, answer review, and school-level preparation.

- official SCO cover-page style used as an editable first page
- 50 objective questions with compact inline question numbers
- complete answer key and explanations for learning review
- PDF-ready question blocks for website upload and download

Maths	English	Science	Mental Ability	Finance Knowledge
AI	Entrepreneurship	GK	Coding	Life Skills

School Connect Online

Exam Overview

Field	Details
Exam Name	SCO International Coding Olympiad
Class / Grade	Class 11
Duration	60 minutes
Question Type	Objective multiple-choice questions
Total Questions	50 questions
Marks Pattern	One correct response per question; no negative marking in this website practice copy
Document Type	Sample Practice Paper with Answer Key

Candidate Details: Name _____ *Registration ID* _____ *School* _____

Guidelines for Candidates

1. Read every question carefully before selecting an answer.
2. Each question has only one correct option unless stated otherwise.
3. No calculator, internet search, IDE execution, or external code-running tool is allowed unless the examination authority permits it.
4. For code-output questions, trace the program step by step and check indexing, loops, data types and edge cases.
5. For SQL, web and security questions, prefer safe implementation practices such as parameterized queries, input validation and least-privilege design.
6. Submit the answer sheet only after checking question numbering and selected options.

School Connect Online

Section 1: Advanced Programming Concepts

Q1. What is the purpose of a decorator in Python?

A) It modifies the behavior of a function without changing its source code.
B) It is used to create a function that returns another function.
C) It is used to call a function multiple times.
D) It is used to display output to the console.

Answer: A) It modifies the behavior of a function without changing its source code.

Explanation:
A decorator in Python is a design pattern that allows you to modify the behavior of a function or a class method without changing its code. It's often used to add pre/post-processing to functions.

Q2. What will the following code output?

```
for (int i = 1; i <= 3; i++) {
for (int j = 1; j <= i; j++) {
printf("%d ", i);
}
}
```

A) 1 1 2 2 2 3 3 3
B) 1 1 1 2 2 3
C) 1 2 3
D) 1 2 2 3 3 3

Answer: D - 1 2 2 3 3 3

Explanation:
The outer loop prints the value of i exactly i times. For i=1 it prints 1 once, for i=2 it prints 2 twice, and for i=3 it prints 3 three times. Therefore, the output is 1 2 2 3 3 3.

Q3. What is the output of the following Python code?

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x**2 for x in numbers if x % 2 == 0]
print(squared_numbers)
```

A) [1, 9, 25]
B) [4, 16]
C) [2, 4, 6, 8, 10]
D) [1, 4, 9, 16, 25]

Answer: B) [4, 16]

Explanation:
List comprehensions in Python allow concise creation of lists. The code creates a new list with squared values of even numbers in the original list. Only 2 and 4 are even numbers, and their squares are 4 and 16.

Q4. Which of the following is the correct way to free memory allocated dynamically in C?

A) delete memory_pointer;
B) free(memory_pointer);
C) memory_pointer.free();
D) memory_pointer = null;

Answer: B) free(memory_pointer);

Explanation:

In C, memory allocated using malloc() or calloc() should be deallocated using free(). The delete keyword is used in C++.

Q5. What does the following Numpy code do?

```
import numpy as np
arr = np.array([1, 2, 3])
arr = arr * 3
```

A) Multiplies each element by 3.
B) Multiplies all elements together and stores the result in arr.
C) Adds 3 to each element.
D) Multiplies the first element by 3 and leaves others unchanged.

Answer: A) Multiplies each element by 3.

Explanation:
In Numpy, multiplying an array by a scalar (like 3) will apply the scalar multiplication element-wise to the entire array.

Q6. What does the following Swift code do?

```
var name: String? = "John"
print(name)
```

A) It prints John.
B) It prints Optional(John).
C) It causes a runtime error.
D) It prints an empty string.

Answer: B) It prints Optional(John).

Explanation:
In Swift, an optional (String?) represents a variable that might contain a value or nil. When printed directly, it displays Optional(value) to indicate it's an optional type.

Q7. Which type of join will you use to fetch all records from the left table and only matching records from the right table in SQL?

A) LEFT JOIN
B) RIGHT JOIN
C) FULL OUTER JOIN
D) INNER JOIN

Answer: A) LEFT JOIN

Explanation:
A LEFT JOIN returns all rows from the left table and the matching rows from the right table. If there is no match, the result is NULL for columns from the right table.

Q8. What is the main purpose of PHP sessions?

A) To store data temporarily across multiple pages.
B) To store data in the database.
C) To manage user authentication.
D) To send email notifications.

Answer: A) To store data temporarily across multiple pages.

Explanation:
Sessions in PHP are used to store user-specific data across multiple pages. Unlike cookies, sessions store data on the server.

Q9. What is the time complexity of the linear search algorithm for an array of size n?

A) O(1)
B) O(log n)
C) O(n)
D) O(n^2)

Answer: C) O(n)

Explanation:

A linear search algorithm checks each element of the array one by one. In the worst case, it will check all n elements, leading to a time complexity of O(n).

Q10. What is the result of the following C code?

```
int x = 2;
switch(x) {
case 1: printf("One");
case 2: printf("Two");
case 3: printf("Three");
default: printf("Default");
}
```

A) TwoThreeDefault
B) OneTwoThreeDefault
C) TwoThree
D) ThreeDefault

Answer: A) TwoThreeDefault

Explanation:

The switch statement does not have break statements, so once a case matches, the program continues executing all subsequent cases (fall-through). Therefore, it prints "TwoThreeDefault".

Q11. What is the result of the following Python code?

```
multiply = lambda x, y: x * y
print(multiply(4, 5))
```

A) None
B) 45
C) 20
D) 4 5

Answer: C) 20

Explanation:

A lambda function in Python is an anonymous function defined with the lambda keyword. It calculates x * y when invoked with arguments 4 and 5, returning 20.

Q12. What will be the output of the following Pandas code?

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df['C'] = df['A'] + df['B']
print(df)
```

A) Adds a new column C with values 5, 7, 9.
B) Adds a new column C with values 1, 2, 3.
C) Raises an error due to data type mismatch.
D) Adds a new column C with values 2, 4, 6.

Answer: A) Adds a new column C with values 5, 7, 9.

Explanation:

The code adds a new column C to the DataFrame by summing the corresponding values in columns A and B. The new column C contains [5, 7, 9].

Q13. What will be the output of the following Swift code?

```
let greet = { (name: String) -> String in
return "Hello, \(name)"
}
print(greet("World"))
```

A) Hello, World
B) Hello
C) World
D) Error

Answer: A) Hello, World

Explanation:

A closure in Swift is a self-contained block of code that can be passed around and used in your code. In this case, it takes a String argument and returns a greeting message.

Q14. What does the following C code do?

```
int a = 10;
int *ptr = &a;
*ptr = 20;
printf("%d", a);
```

A) Prints 10
B) Prints 20
C) Prints the address of a
D) Causes a runtime error

Answer: B) Prints 20

Explanation:

The pointer ptr holds the address of a. By dereferencing ptr with *ptr = 20, the value of a is updated to 20.

Q15. Which of the following is not a goal of database normalization?

A) Eliminate redundancy.
B) Ensure data integrity.
C) Make data retrieval faster.
D) Minimize data anomalies.

Answer: C) Make data retrieval faster.

Explanation:

The main goals of normalization are to eliminate redundancy, ensure data integrity, and minimize anomalies. Speeding up data retrieval is not a primary goal of normalization.

Section 2: Algorithm Development

Q16. What is the time complexity of the quick sort algorithm in the average case?

A) O(n log n)
B) O(n^2)
C) O(n)
D) O(log n)

Answer: A) O(n log n)

Explanation:
In the average case, quick sort performs n log n comparisons and swaps. However, in the worst case, it can take O(n^2) if the pivot selection is poor (like always selecting the smallest or largest element).

Q17. Which condition must be true for binary search to work on an array?

A) The array must be sorted in ascending order.
B) The array must contain unique elements only.
C) The array must be sorted in descending order.
D) The array should be unsorted.

Answer: A) The array must be sorted in ascending order.

Explanation:
Binary search works by repeatedly dividing the search interval in half. For it to function correctly, the array must be sorted. This allows binary search to eliminate half of the search space at each step.

Q18. Which of the following best describes the 0/1 Knapsack problem?

A) A problem that finds the maximum sum of weights that can be selected from an array.
B) A problem that determines the most valuable set of items given a weight limit.
C) A problem that minimizes the total weight while maximizing the total value.
D) A problem that sorts the items by their weight-to-value ratio.

Answer: B) A problem that determines the most valuable set of items given a weight limit.

Explanation:
The 0/1 Knapsack problem involves selecting a subset of items, each with a weight and value, to maximize the total value without exceeding the weight limit. It is solved using dynamic programming.

Q19. Which algorithm is typically used to find the shortest path from a source node to all other nodes in a weighted graph?

A) Depth-First Search (DFS)
B) Dijkstra's Algorithm
C) Bellman-Ford Algorithm
D) Kruskal’s Algorithm

Answer: B) Dijkstra's Algorithm

Explanation:
Dijkstra's algorithm is used for finding the shortest path from a source node to all other nodes in a graph with non-negative edge weights. It is commonly used in routing and network analysis.

Q20. What is the time complexity of merge sort in the worst case?

A) O(n log n)
B) O(n^2)

C) O(n)
D) O(log n)

Answer: A) O(n log n)
Explanation:
Merge sort divides the array into two halves and recursively sorts them. Merging two sorted halves requires O(n) time, and since the array is divided log n times, the overall time complexity is O(n log n).

Q21. In the Activity Selection Problem, which greedy criterion is used to select activities?

A) Select the activity with the earliest finish time.
B) Select the activity with the latest finish time.
C) Select the activity with the longest duration.
D) Select the activity with the earliest start time.

Answer: A) Select the activity with the earliest finish time.
Explanation:
The Activity Selection Problem is solved using a greedy algorithm where activities are selected based on the earliest finish time. This ensures that the maximum number of activities are selected without overlap.

Q22. Which of the following is the correct recursive formula for calculating the factorial of a number n?

A) n! = n * (n - 1)!
B) n! = n * (n + 1)!
C) n! = (n - 1)! + n
D) n! = n + (n - 1)!

Answer: A) n! = n * (n - 1)!
Explanation:
The factorial of n, denoted n!, is defined recursively as n! = n * (n - 1)! with the base case being 0! = 1.

Q23. What is the time complexity of the Divide and Conquer approach for solving the Maximum Subarray Problem?

A) O(n log n)
B) O(n^2)
C) O(n)
D) O(log n)

Answer: A) O(n log n)
Explanation:
The Maximum Subarray Problem can be solved using a Divide and Conquer approach, which divides the problem into smaller subproblems and combines the results. This approach has a time complexity of O(n log n).

Q24. Which of the following is not true about Big-O notation?

A) Big-O represents the worst-case time complexity of an algorithm.
B) Big-O represents the best-case time complexity of an algorithm.
C) Big-O is used to describe the upper bound of an algorithm's growth rate.
D) Big-O does not account for constant factors and lower-order terms.

Answer: B) Big-O represents the best-case time complexity of an algorithm.
Explanation:
Big-O notation is used to describe the worst-case time complexity of an algorithm, ignoring constant factors and lower-order terms.

Q25. Which is the time complexity of a dynamic programming approach to solve the Fibonacci sequence problem?

A) O(1)
B) O(n)
C) O(n^2)
D) O(2^n)

Answer: B) O(n)

Explanation:

A dynamic programming approach for calculating the Fibonacci sequence stores the results of subproblems and avoids redundant calculations. This reduces the time complexity to O(n).

Q26. Which of the following statements about heap sort is true?

A) Heap sort is an unstable sorting algorithm.
B) Heap sort uses a divide-and-conquer approach.
C) Heap sort requires O(n^2) time in the worst case.
D) Heap sort is a comparison-based sorting algorithm with a time complexity of O(n log n).

Answer: D) Heap sort is a comparison-based sorting algorithm with a time complexity of O(n log n).

Explanation:

Heap sort is an efficient comparison-based sorting algorithm that has a time complexity of O(n log n) for both the average and worst case.

Q27. Which approach is used to solve the N-Queens Problem?

A) Greedy approach
B) Divide and Conquer
C) Dynamic Programming
D) Backtracking

Answer: D) Backtracking

Explanation:

The N-Queens Problem is typically solved using a backtracking approach, where possible solutions are incrementally built and abandoned as soon as a conflict is detected.

Q28. Which of the following is not a method for handling collisions in a hash table?

A) Chaining
B) Open addressing
C) Linear probing
D) Quick sort

Answer: D) Quick sort

Explanation:

Quick sort is a sorting algorithm, not a method for handling collisions in a hash table. Chaining and open addressing are common techniques used to resolve collisions.

Q29. What is the time complexity of the Strassen's Matrix Multiplication algorithm?

A) O(n^2)
B) O(n^3)
C) O(n^2.81)
D) O(n log n)

Answer: C) O(n^2.81)

Explanation:
Strassen's Matrix Multiplication algorithm is an advanced divide-and-conquer algorithm that reduces the time complexity of matrix multiplication to approximately O(n^2.81), which is faster than the traditional O(n^3) approach.

Q30. Which algorithm is used to find the Minimum Spanning Tree (MST) in a graph?

A) Dijkstra's Algorithm
B) Kruskal's Algorithm
C) Bellman-Ford Algorithm
D) Floyd-Warshall Algorithm

Answer: B) Kruskal's Algorithm

Explanation:
Kruskal's algorithm is used to find the Minimum Spanning Tree (MST) of a graph. It works by sorting all edges and adding the smallest edge that does not form a cycle.

Section 3: Coding Projects

Q31. Web-based Application - Scalable E-Commerce Backend Architecture

Case Study:
You are designing the backend architecture of a large-scale e-commerce platform. The system needs to handle millions of users accessing the website simultaneously, making purchases, adding products to carts, and checking out. The platform also requires real-time order tracking, inventory management, and personalized recommendations.
Which architecture pattern is the most suitable for efficiently scaling and managing the traffic, ensuring high availability, and decoupling the critical components such as user sessions, orders, and inventory?
A) Monolithic Architecture with a single database for all services.
B) Microservices Architecture with each component (user management, orders, payments) as an independent service.
C) Single-page web application with server-side rendering and no caching.
D) Client-server architecture with all services hosted on the same machine.

Answer: B) Microservices Architecture with each component (user management, orders, payments) as an independent service.

Explanation:
Microservices allow each component of the application to scale independently, providing better fault tolerance, high availability, and easier maintenance. This is essential for an e-commerce platform that needs to handle high traffic, complex workflows, and frequent updates.

Q32. Data Science - Building a Recommendation Engine

Case Study:
A music streaming service wants to build a recommendation system that suggests personalized songs to users based on their listening history, genre preferences, and ratings. The data is stored in a relational database, and you need to build an algorithm that generates suggestions for each user.
Which of the following advanced algorithmic techniques would you use to create a collaborative filtering recommendation system?
A) Clustering-based K-means algorithm to categorize songs into genres.
B) Content-based filtering using the term frequency-inverse document frequency (TF-IDF) on song lyrics.
C) Matrix factorization using Singular Value Decomposition (SVD) for latent factor modeling.
D) Decision trees to classify songs into "liked" or "disliked" categories.

Answer: C) Matrix factorization using Singular Value Decomposition (SVD) for latent factor modeling.

Explanation:

Matrix factorization (SVD) is a commonly used method for collaborative filtering, which predicts a user's preferences based on patterns of ratings and interactions between users and items. Unlike content-based approaches, this method discovers latent relationships between users and songs.

Q33. C Programming - Multi-threaded Processing for Data Analysis

Case Study:

You are tasked with implementing a multi-threaded program in C to process large datasets and perform statistical analysis on them. The dataset contains thousands of records of various measurements, and the analysis requires calculating the mean, median, and standard deviation.

Which of the following would be the most efficient way to parallelize the computation of statistical metrics using multiple threads in C?

A) Split the dataset into equal parts, and compute the mean, median, and standard deviation for each part in separate threads. Finally, combine the results in the main thread.

B) Use a single thread to compute all statistical metrics for the dataset sequentially.

C) Use a thread to compute each statistical metric (mean, median, standard deviation) on the entire dataset.

D) Use recursion to split the dataset into halves and compute the results in multiple recursive calls.

Answer: A) Split the dataset into equal parts, and compute the mean, median, and standard deviation for each part in separate threads. Finally, combine the results in the main thread.

Explanation:

In multi-threaded processing, the dataset should be divided into smaller parts for concurrent computation. This allows for parallelization of the computation. Afterward, results are combined efficiently in the main thread to get the final statistical metrics.

Q34. SQL Database - Optimizing Query Performance for Large Data

Case Study:

You are working on a large-scale database used by a financial institution to store transactions, customer data, and account balances. The system needs to support complex queries that aggregate data over long time periods, while ensuring that queries execute efficiently even as the data grows.

Which of the following indexing strategies would help optimize query performance on a large dataset when performing complex aggregation queries?

A) Create indexes on every column of the database to speed up query retrieval.

B) Use partial indexes on columns that are frequently used in WHERE clauses or JOIN conditions.

C) Use a full-text index on all textual columns to speed up aggregation.

D) Avoid using indexes altogether to reduce the overhead of index maintenance during data updates.

Answer: B) Use partial indexes on columns that are frequently used in WHERE clauses or JOIN conditions.

Explanation:

Creating partial indexes on frequently queried columns (such as those used in WHERE or JOIN conditions) helps to optimize query performance by reducing the amount of data scanned. Overusing indexes or using full-text indexing can lead to performance degradation, especially on large datasets.

Q35. Swift Programming - Memory Management in Mobile App Development

Case Study:

You are developing a Swift-based iOS mobile app that requires managing large amounts of data in memory. The app needs to process real-time data streams, display live updates to users, and interact with external APIs. You are concerned about memory leaks that may impact the app's performance, especially during long usage sessions.

Which of the following strategies should you implement to prevent memory leaks in a Swift iOS application that uses delegates and closures?

A) Use weak references to delegates and closures to prevent retain cycles.

B) Always use strong references for delegates to ensure that the object is not deallocated.

C) Use value types (structs) exclusively, avoiding reference types (classes) entirely.
D) Force unwrapping optionals to avoid retaining unnecessary memory.

Answer: A) Use weak references to delegates and closures to prevent retain cycles.

Explanation:

In Swift, retain cycles occur when an object retains a reference to another object, and vice versa, preventing memory from being released. Using weak references for delegates and closures helps prevent these cycles, ensuring that objects can be properly deallocated when no longer needed.

Q36. Data Science - Predictive Model for Sales Forecasting

Case Study:

You are tasked with building a predictive model to forecast sales for a retail company based on historical data. The dataset includes variables like seasonality, product categories, price changes, and advertisement spend. You are asked to choose an algorithm that will provide the most accurate predictions based on this data.
Which of the following algorithms would be the most suitable for time-series forecasting in this scenario?
A) Linear Regression with random sampling of data
B) K-Nearest Neighbors (KNN) with dynamic time warping
C) ARIMA (AutoRegressive Integrated Moving Average)
D) Neural Networks with a Recurrent Neural Network (RNN) architecture

Answer: C) ARIMA (AutoRegressive Integrated Moving Average)

Explanation:

ARIMA is widely used for time-series forecasting and works well with datasets involving time-dependent variables such as sales. It captures the seasonality, trend, and autocorrelation in time-series data. While RNNs are great for time-series, ARIMA is typically easier and more effective for financial or sales data where trends and seasonality are the primary factors.

Q37. Web Application Security - Protecting User Data

Case Study:

You are developing a web application that stores sensitive user data such as personal identification information (PII), financial details, and passwords. You need to ensure the application follows best practices for security to protect user data from potential attacks such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
Which of the following security practices would be most effective for preventing SQL injection attacks on your web application?
A) Only sanitize inputs by removing special characters and validating data types.
B) Use parameterized queries or prepared statements, with input validation.
C) Store user data in plaintext to avoid encryption overhead.
D) Allow users to input unrestricted SQL queries for advanced filtering.

Answer: B - Use parameterized queries or prepared statements, with input validation.

Explanation:

Prepared statements/parameterized queries separate SQL code from user-supplied data. This prevents user input from changing the structure or intent of the SQL command. Input validation is still useful, but it is not enough by itself for SQL injection prevention.

Q38. Case Study:

You are working on a Python-based application that processes large CSV files containing millions of records. The application needs to perform computationally expensive operations like data cleaning, filtering, and aggregations on the dataset. The task needs to be multithreaded to speed up processing and ensure the application scales well with large datasets.
Which of the following multi-threading techniques would be most efficient in Python to handle large data processing tasks in parallel?

A) Use threading module for concurrent processing of independent tasks, but with global variables.
B) Use multiprocessing module to take advantage of multiple CPU cores for parallel data processing.
C) Use asyncio for asynchronous I/O-bound operations, but not for CPU-bound tasks.
D) Use the concurrent.futures module with a ThreadPoolExecutor for maximum performance on all tasks.

Answer: B) Use multiprocessing module to take advantage of multiple CPU cores for parallel data processing.

Explanation:

The multiprocessing module in Python is best suited for CPU-bound tasks because it allows multiple processes to run on different CPU cores, effectively bypassing Python's Global Interpreter Lock (GIL). The threading module is more suited for I/O-bound tasks, while asyncio is ideal for asynchronous I/O operations but is not efficient for CPU-intensive operations.

Q39. Swift Programming - Efficient Memory Management in iOS Apps

Case Study:

You are developing an iOS mobile application in Swift that will be used by millions of users. The application is memory-intensive, and you want to optimize its performance to prevent memory leaks, especially during prolonged usage. You are using delegates and closures throughout the app.

Which memory management concept should you implement to ensure that memory leaks do not occur when using delegates and closures in Swift?

A) Use strong references to maintain ownership of delegates and closures.
B) Always use weak references for delegates and unowned references for closures to avoid retain cycles.
C) Use strong references for both delegates and closures to avoid deallocation.
D) Disable Automatic Reference Counting (ARC) to handle memory management manually.

Answer: B) Always use weak references for delegates and unowned references for closures to avoid retain cycles.

Explanation:

In Swift, retain cycles can occur when two objects hold strong references to each other, preventing deallocation. Using weak for delegates and unowned for closures ensures that objects are deallocated properly, preventing memory leaks. Automatic Reference Counting (ARC) is the default in Swift and should not be disabled.

Q40. PHP Web Development - Optimizing Server Performance

Case Study:

You are developing a high-traffic website using PHP as the backend language. The website needs to serve dynamic content, perform database queries, and handle user authentication in real-time. The website's performance is deteriorating as the number of concurrent users increases, causing slower page loads and database bottlenecks.

Which of the following PHP optimization strategies would be most effective in scaling your web application and improving performance for concurrent users?

A) Use a dedicated server for database queries and avoid caching.
B) Enable PHP op-code caching (e.g., using OPcache) and cache frequently accessed content.
C) Increase the server's CPU and RAM without optimizing the database queries.
D) Disable session management to reduce server load.

Answer: B) Enable PHP op-code caching (e.g., using OPcache) and cache frequently accessed content.

Explanation:

OPcache improves PHP performance by caching precompiled script bytecode, reducing the need to recompile PHP scripts with each request. Caching frequently accessed content (such as static pages or database query results) helps to offload the server and minimize database hits, leading to better performance.

Achievers Section

Section 4: Achievers Section - Error Finding and Case Analysis

Q41. Error Analysis in Python - Advanced Data Handling

Case Study:

You are given a Python program that processes large datasets using the pandas library. The program applies a filter and then performs statistical analysis on the data. However, an unexpected TypeError occurs when performing operations on certain columns, which were expected to be numeric, but seem to be strings or contain NaN values.

Code Snippet:

```
import pandas as pd
data = pd.read_csv('sales_data.csv')
# Filter rows based on sales greater than 1000
filtered_data = data[data['sales'] > 1000]
# Perform statistical analysis
mean_sales = filtered_data['sales'].mean()
std_sales = filtered_data['sales'].std()
# Display results
print("Mean Sales:", mean_sales)
print("Standard Deviation:", std_sales)
```

Which of the following changes would be necessary to fix the TypeError in the program?

A) Check for NaN values in the 'sales' column and convert them to 0.

B) Ensure the 'sales' column is converted to numeric values using pd.to_numeric() before filtering.

C) Change the filtering condition to data['sales'] >= 1000 to avoid errors.

D) Replace std_sales with std_sales = filtered_data['sales'].median() to avoid type mismatch.

Answer: B) Ensure the 'sales' column is converted to numeric values using pd.to_numeric() before filtering.

Explanation:

The error likely occurs because the 'sales' column contains non-numeric data (such as strings or NaN). Using pd.to_numeric() ensures that the column is converted to numeric values, allowing the subsequent statistical functions (mean() and std()) to work without errors. The NaN values should also be handled before performing the analysis (e.g., with fillna() or filtering out NaN values).

Q42. Error Finding in PHP Web Application

Case Study:

You are developing a PHP-based web application that uses SQL queries to retrieve and update user information in a database. After submitting a form to update user details, an error occurs, and the following error message is displayed: "Notice: Undefined index: username".

Code Snippet:

```
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
// Assume form data is sent as POST request
$username = $_POST['username'];  // The error occurs here
$email = $_POST['email'];
// SQL update query
$query = "UPDATE users SET email='$email' WHERE username='$username'";
mysqli_query($conn, $query);
}
?>
```

Which of the following would be the most effective fix for the error message "Undefined index: username"?

A) Use isset() to check if the username key exists in $_POST before accessing it.

B) Ensure the username input field in the form is named correctly as username.

C) Move the assignment of $_POST['username'] to the end of the script.

D) Use empty() instead of isset() to check for missing form data.

Answer: A) Use isset() to check if the username key exists in $_POST before accessing it.

Explanation:

The error occurs because the form may not have the username field, or the field could be missing when the form is submitted. Using isset() checks whether the username exists in the $_POST array before accessing it, which avoids the undefined index error. This ensures the script only processes the field when it exists in the form.

Q43. Error Detection in SQL Query Optimization

Case Study:

You are tasked with optimizing an SQL query that retrieves user details from a database. The current query works but is slow, especially when there are thousands of rows. It uses a JOIN operation between two tables (users and orders). The query is as follows:

```
SELECT users.username, users.email, orders.order_id, orders.date
FROM users
JOIN orders ON users.user_id = orders.user_id
WHERE orders.date BETWEEN '2021-01-01' AND '2021-12-31';
```

Which of the following changes would improve the performance of this query?

A) Add an index on the orders.date column to speed up the date range filter.

B) Remove the JOIN and retrieve user details from the users table separately.

C) Change the BETWEEN clause to >= and <= to speed up range filtering.

D) Use DISTINCT to eliminate duplicate results from the JOIN operation.

Answer: A) Add an index on the orders.date column to speed up the date range filter.

Explanation:

The query is slow due to the filtering on orders.date for a large dataset. Adding an index on orders.date improves the performance of the BETWEEN operation, making the search more efficient. Removing the JOIN or using DISTINCT could impact the query results, while using >= and <= doesn't improve performance over BETWEEN in this case.

Q44. Advanced Algorithm Development - Sorting and Searching

Case Study:

You are developing a program that needs to sort a large list of strings alphabetically and then perform a binary search to find a specific string. The sorting is done using Merge Sort and the search is done using Binary Search.

Which of the following steps is incorrect when implementing a binary search on a sorted array of strings?

A) Ensure that the array is sorted before applying binary search.

B) If the middle element is equal to the target string, return its index immediately.

C) If the middle element is less than the target string, search the left half of the array.

D) If the middle element is greater than the target string, search the right half of the array.

Answer: C) If the middle element is less than the target string, search the left half of the array.

Explanation:

In binary search, if the middle element is less than the target string, you should search the right half of the array because the array is sorted. If the middle element is greater, you would search the left half. The algorithm works by continually narrowing down the search range based on the comparison between the middle element and the target.

Q45. Advanced Project Development - Web Application Performance

Case Study:

You are developing a web application using PHP and MySQL for a client. The application handles a lot of data and needs to run efficiently even with a high number of users. After implementing the initial version, you realize the application performance slows down when multiple users submit requests at the same time.

Which of the following strategies would most improve the performance of the application in handling multiple concurrent users?

A) Use session_start() on every page to maintain a persistent connection for each user.

B) Optimize SQL queries by using prepared statements and limiting the number of database calls per page load.

C) Disable caching for dynamic pages to ensure the latest data is always displayed.

D) Implement client-side form validation with JavaScript to reduce server-side load.

Answer: B) Optimize SQL queries by using prepared statements and limiting the number of database calls per page load.

Explanation:

The performance issue is likely due to inefficient database queries and excessive database calls. Prepared statements in MySQL improve performance by reusing the query execution plan, and limiting the number of database calls reduces unnecessary load. Session management and client-side validation improve user experience but do not directly address the core performance issue related to concurrent database queries.

Q46. Debugging a Multi-Language Web Application

Case Study:

You are developing a multi-language web application that supports English, French, and Spanish. The language preference is stored in the user's session and retrieved when rendering web pages. However, users report that their language settings reset unexpectedly.

Code Snippet (PHP + HTML):

```
<?php
session_start();
if (isset($_GET['lang'])) {
$_SESSION['lang'] = $_GET['lang'];
}
// Set default language
$lang = isset($_SESSION['lang']) ? $_SESSION['lang'] : 'en';
?>
<html>
<head>
<title>Multi-language Web App</title>
</head>
<body>
<p><?php echo ($lang == 'fr') ? 'Bonjour!' : (($lang == 'es') ? '¡Hola!' : 'Hello!'); ?></p>
</body>
</html>
```

Why does the language preference reset unexpectedly for some users?

A) The session is not being saved in a database, causing it to reset between page loads.

B) The session variable $_SESSION['lang'] is not declared as global, so it does not persist.

C) The default language is always set to English ('en'), which overrides user preferences.

D) Session handling may be inconsistent due to server configurations like session.gc_maxlifetime.

Answer: D) Session handling may be inconsistent due to server configurations like session.gc_maxlifetime.

Explanation:

PHP sessions are stored temporarily, and server settings like session.gc_maxlifetime (garbage collection timeout) may cause them to expire unexpectedly. If users experience resets, ensuring session persistence (e.g., using database-backed session storage) can prevent unintended session loss.

Q47. Memory Leak in C Programming

Case Study:

A team is working on an AI-based chatbot in C that continuously processes user inputs and responds based on predefined rules. After running for several hours, the program slows down significantly and crashes due to out-of-memory errors.

Code Snippet:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char* process_input(char* input) {
```

```
char* response = (char*)malloc(100);
strcpy(response, "Hello! How can I help you?");
return response;
}
int main() {
char input[50];
while (1) {
printf("User: ");
scanf("%s", input);
char* reply = process_input(input);
printf("Bot: %s\n", reply);
}
return 0;
}
```

What is the main reason for the memory leak in the chatbot?

A) malloc(100) is allocating memory repeatedly without being freed.

B) The response string should be initialized to NULL before allocation.

C) The process_input() function does not correctly modify the input string.

D) Using scanf("%s", input); can cause buffer overflow issues.

Answer: A) malloc(100) is allocating memory repeatedly without being freed.

Explanation:

Each call to process_input() allocates new memory, but the program never frees the old memory, causing a memory leak. Fix: Use free(reply); after printing the chatbot's response to prevent memory leaks.

Q48. SQL Query Optimization for Large Datasets

Case Study:

A company is storing millions of customer transactions in a MySQL database. The finance department needs to generate monthly reports, but the SQL queries are too slow. The following query is used:

```
SELECT customer_id, SUM(amount)
FROM transactions
WHERE transaction_date BETWEEN '2023-01-01' AND '2023-01-31'
GROUP BY customer_id;
```

Which of the following strategies will significantly speed up the execution of this query?

A) Use ORDER BY transaction_date DESC to retrieve the most recent transactions faster.

B) Create an index on the transaction_date column to improve filtering efficiency.

C) Use HAVING SUM(amount) > 0 to eliminate zero-sum transactions.

D) Run the query inside a stored procedure to precompile the SQL execution plan.

Answer: B) Create an index on the transaction_date column to improve filtering efficiency.

Explanation:

Indexes speed up WHERE conditions significantly by reducing the number of rows scanned. The absence of an index makes the query scan all rows, making it slow for large datasets.

Q49. Finding Bugs in Swift Mobile App Development

Case Study:

A mobile app written in Swift allows users to log in and fetch data from a remote server. However, users report that the app crashes intermittently when retrieving data.

Code Snippet (Swift):

```
func fetchData() {
let url = URL(string: "https://api.example.com/data")!
let task = URLSession.shared.dataTask(with: url) { data, response, error in
```

```
let json = try? JSONSerialization.jsonObject(with: data!, options: [])
print(json ?? "No data received")
}
task.resume()
}
```

What is the main issue with the above code that can cause runtime crashes?

A) The data! force-unwrapping may cause a crash if data is nil.

B) The URLSession.shared.dataTask() does not handle HTTP errors properly.

C) The JSONSerialization.jsonObject() function must use do-catch for error handling.

D) The print(json ?? "No data received") should be executed on the main thread.

Answer: A) The data! force-unwrapping may cause a crash if data is nil.

Explanation:

If data is nil (e.g., due to a network error), force-unwrapping data! causes a runtime crash. The correct approach is to safely unwrap it:

```
if let safeData = data {
let json = try? JSONSerialization.jsonObject(with: safeData, options: [])
print(json ?? "No data received")
}
```

Q50. Debugging JavaScript Web Application Performance

Case Study:

A JavaScript-based web application fetches real-time data and updates the UI every second. However, users report that after running for a while, the browser becomes unresponsive.

Code Snippet (JavaScript):

```
function fetchData() {
fetch("https://api.example.com/data")
.then(response => response.json())
.then(data => {
document.getElementById("dataContainer").innerHTML = JSON.stringify(data);
});
}
setInterval(fetchData, 1000);
```

What is the most likely reason the browser becomes unresponsive?

A) The function fetchData() is called too frequently, overloading the browser with API requests.

B) The innerHTML assignment is causing memory leaks due to excessive re-rendering.

C) The API server is rate-limiting the requests, causing delays in response time.

D) The fetchData() function should be wrapped in a try-catch block to handle errors properly.

Answer: A) The function fetchData() is called too frequently, overloading the browser with API requests.

Explanation:

Calling fetchData() every 1 second without limiting network requests overloads the browser, causing performance degradation. The correct approach is to use Debounce or Throttling to optimize API calls.

Answer Key

1	2	3	4	5	6	7	8	9	10
Q1: A	Q2: D	Q3: B	Q4: B	Q5: A	Q6: B	Q7: A	Q8: A	Q9: C	Q10: A
Q11: C	Q12: A	Q13: A	Q14: B	Q15: C	Q16: A	Q17: A	Q18: B	Q19: B	Q20: A

Q21: A	**Q22: A**	**Q23: A**	**Q24: B**	**Q25: B**	**Q26: D**	**Q27: D**	**Q28: D**	**Q29: C**	**Q30: B**
Q31: B	**Q32: C**	**Q33: A**	**Q34: B**	**Q35: A**	**Q36: C**	**Q37: B**	**Q38: B**	**Q39: B**	**Q40: B**
Q41: B	**Q42: A**	**Q43: A**	**Q44: C**	**Q45: B**	**Q46: D**	**Q47: A**	**Q48: B**	**Q49: A**	**Q50: A**

Teachers may use the individual explanations in each question block for post-test review and remediation.