School Connect Online

# SCO INTERNATIONAL CODING OLYMPIAD

## CLASS 10 OFFICIAL QUESTION PAPER

A professional academic document for students, teachers, schools, and parents

**Designed from Class 10 coding pathways and aligned with SCO's official Olympiad preparation, practice, reporting, and future-ready technology growth.**

- Class 10 coding readiness guidance for secondary-level learners globally
- pathways across Python, SQL, C/C++, Swift, PHP, data analytics, security, APIs, and project studios
- exam structure, question practice, answer explanations, and technology-career readiness for academic enrichment

| | | | | |
|---|---|---|---|---|
| Python | SQL | C / C++ | Swift | PHP Web |
| Data | Projects | Security | APIs | Debugging |

School Connect Online

# Class 10 - Official Question Paper - Set A

| Paper Snapshot | |
|---|---|
| **Exam Name** | SCO International Coding Olympiad |
| **Class / Grade** | Class 10 |
| **Academic Year** | 2025-26 |
| **Question Paper Set** | A |
| **Duration** | 60 minutes |
| **No. of Questions** | 50 objective-type questions |
| **Core Areas** | Advanced Programming, Application Development, Coding Projects, Data Analytics, Security, SQL, Swift, C, PHP, Python |
| **Candidate Instructions** | |
| **Total Questions** | 50 objective-type questions |
| **Duration** | 60 minutes |
| **Marking** | One correct answer for each question. Calculators are not allowed. |
| **Response Rule** | Select only one option for each question and review before final submission. |

School Connect Online

# Question Paper with Answer Key and Explanations

**Q.1**

**Consider the following code snippet:**

```
def multiply(a, b=2):
return a * b
def compute():
x = 5
return multiply(x) + multiply(x, 3)
print(compute())
```

What is the output of this code, and what does it illustrate about default parameters?

A) 30; default parameter b is always 2.

B) 25; using default parameter b=2 for the first call and b=3 for the second call.

C) 35; because the function uses b=3 for both calls.

D) 25; because x is not accessible in multiply.

**Answer: B**

**Explanation:** Inside compute(), x is 5. The first call multiply(x) uses the default b=2, so returns 5×2=10. The second call multiply(x, 3) returns 5×3=15. Their sum is 10+15 = 25.

sum is 25. This demonstrates how default parameters work when a second argument is omitted.

**Q.2**

**Consider the following Python code snippet:**

```
def binary_search(arr, target):
low, high = 0, len(arr)-1
while low <= high:
mid = (low + high) // 2
if arr[mid] == target:
return mid
```

**elif arr[mid] < target:**

```
low = mid + 1
else:
high = mid - 1
return -1
data = [2, 4, 6, 8, 10, 12, 14]
print(binary_search(data, 10))
```

What is the output, and what does this function do?

A) 3; It finds the target using linear search.

B) 4; It returns the index of the target using binary search.

C) -1; It does not find the target.

D) 5; It returns the count of numbers less than the target.

**Answer: B**

**Explanation:** The function performs a binary search on a sorted array. The target value 10 is at index 4 in the list. Hence, the function returns 4. Binary search efficiently finds the target by repeatedly dividing the search interval.

**Q.3**

A developer is designing a RESTful API endpoint that retrieves user data in JSON format. Which HTTP method and response code combination is most appropriate for successfully retrieving data?

A) POST with 201

B) GET with 200

C) PUT with 204

D) DELETE with 404

**Answer: B**

**Explanation:** A RESTful API uses GET for data retrieval. A successful GET request typically returns a 200 OK status code along with the requested data in JSON format. POST is for creating resources, PUT is for updating, and DELETE is for deletion.

**Q.4**

**Examine the following Swift code snippet:**

```
   var greeting: String? = "Hello, Swift!"
   if let message = greeting {
   print(message)
} else {
   print("No greeting available.")
}
```

What is the output, and why is optional binding used?

A) "No greeting available."; because greeting is nil.

B) "Hello, Swift!"; because optional binding safely unwraps the non-nil value.

C) "Optional("Hello, Swift!")"; because it prints the optional directly.

D) It crashes due to an unwrapped nil value.

**Answer: B**

**Explanation:** Optional binding (if let) checks whether the optional greeting contains a value. Since it does ("Hello, Swift!"), it unwraps and assigns it to message, which is then printed. This prevents runtime errors from force unwrapping a nil value.

**Q.5**

**Consider the following Objective-C code snippet:**

```
#import <Foundation/Foundation.h>
   @interface Person : NSObject
@property (nonatomic, strong) NSString *name;
@end
   @implementation Person
@end
   int main(int argc, const char * argv[]) {
@autoreleasepool {
   Person *p = [[Person alloc] init];
   p.name = @"John";
NSLog(@"%@", p.name);
}
   return 0;
}
```

What does this code output, and what is the significance of alloc and init?

A) It outputs "John"; alloc allocates memory and init initializes the object.

B) It outputs nil; because the object is not properly created.

C) It outputs "John"; but alloc is unnecessary.

D) It outputs an error; because Objective-C does not support properties.

**Answer: A**

**Explanation:** In Objective-C, alloc allocates memory for a new object, and init initializes it. The code creates a Person object, sets its name property to "John", and logs "John" to the console. This demonstrates the typical object creation pattern in Objective-C.

**Q.6**

A PHP developer writes the following code to fetch a user's details securely from a MySQL database:

```
<?php
$username = $_POST['username'];
$stmt = $conn->prepare("SELECT email FROM users WHERE username = ?");
$stmt->bind_param("s", $username);
$stmt->execute();
$result = $stmt->get_result();
$row = $result->fetch_assoc();
echo $row['email'];
?>
```

What is the main benefit of using this approach?

A) It speeds up query execution by caching results.

B) It prevents SQL injection by using prepared statements.

C) It automatically converts the result to JSON.

D) It allows multiple queries to run in parallel.

**Answer: B**

**Explanation:** Using prepared statements with parameter binding ensures that the user input is properly escaped and treated as data rather than executable SQL code, thereby preventing SQL injection attacks.

**Q.7**

A database contains a table Sales(ProductID, Amount, SaleDate). Consider the following SQL query:

```
SELECT ProductID, SUM(Amount) AS TotalSales
FROM Sales
WHERE SaleDate BETWEEN '2025-01-01' AND '2025-12-31'
GROUP BY ProductID
HAVING TotalSales > 10000;
```

What does this query return?

A) It returns all products with sales totals less than 10000 in 2025.

B) It returns products and their total sales in 2025, including only those with total sales greater than 10000.

C) It returns individual sale amounts for each product.

D) It returns an error due to the use of HAVING without aggregation.

**Answer: B**

**Explanation:** The query filters sales within the year 2025, groups them by ProductID, calculates the sum of Amount for each product, and then uses the HAVING clause to include only products with total sales exceeding 10000.

**Q.8**

A developer wants to merge two dictionaries in Python and then create a new dictionary that contains only the keys present in both dictionaries with their summed values. Consider the code:

```
dict1 = {'a': 5, 'b': 3, 'c': 8}
dict2 = {'b': 7, 'c': 2, 'd': 4}
merged = {key: dict1[key] + dict2[key] for key in dict1 if key in dict2}
print(merged)
```

What is the output, and what does this code accomplish?

A) {'a': 5, 'b': 10, 'c': 10}; It incorrectly includes key 'a'.

B) {'b': 10, 'c': 10}; It sums values only for keys common to both dictionaries.

C) {'b': 7, 'c': 2}; It ignores values from dict1.

D) {'a': 5, 'd': 4}; It merges keys not in common.

**Answer: B**

**Explanation:** The dictionary comprehension iterates over keys in dict1 and checks if they exist in dict2. For common keys ('b' and 'c'), it sums their values: 'b': 3+7=10, 'c': 8+2=10. Thus, the output is {'b': 10, 'c': 10}.

**Q.9**

A Swift developer is extending the functionality of a built-in type using a protocol and an extension:

```
protocol Describable {
func describe() -> String
}
extension Int: Describable {
func describe() -> String {
return "The number is \(self)"
}
}
print(5.describe())
```

What is the output, and what does this code demonstrate?

A) "5"; It shows that integers can be directly printed.

B) "The number is 5"; It demonstrates extending a type to conform to a protocol using an extension.

C) "Error"; Because Int cannot be extended.

D) "The number is 0"; Because the default value is used.

**Answer: B**

**Explanation:** The code extends the Int type to conform to the Describable protocol by adding a describe() method. When 5.describe() is called, it prints "The number is 5". This demonstrates Swift's ability to add functionality to existing types via extensions.

**Q.10**

A data scientist is working with a DataFrame containing sales data:

```
import pandas as pd
data = {'Region': ['North', 'South', 'East', 'West', 'North'],
'Sales': [200, 150, 300, 250, 180]}
df = pd.DataFrame(data)
filtered = df[df['Sales'] > 180]
print(filtered)
```

What does this code output, and what does it illustrate about data filtering in Pandas?

A) It prints all rows where Sales are less than or equal to 180.

B) It prints only rows where Sales are greater than 180, demonstrating boolean indexing.

C) It prints a single value representing the total sales.

D) It produces an error due to incorrect syntax.

**Answer: B**

**Explanation:** The code filters the DataFrame using boolean indexing, returning only the rows where the 'Sales' value is greater than 180. The output includes the rows from regions 'North' (first row) with Sales=200, 'East' with Sales=300, and 'West' with Sales=250. This demonstrates how to filter data efficiently in Pandas.

**Q.11**

What will be the output of the following Python code?

```
add = lambda x, y: x + y
multiply = lambda x, y: x * y
result = add(5, multiply(2, 3))
print(result)
```

A) 10
B) 11
C) 12
D) 15

**Answer: B**

**Explanation:** The multiply(2,3) function returns 6. Then, add(5,6) returns 11. Hence, the final output is 11.

**Q.12**

Which sorting algorithm has the worst-case time complexity of $O(n^2)$ but is generally preferred for small datasets due to its simplicity?

A) Merge Sort
B) Quick Sort
C) Bubble Sort
D) Heap Sort

**Answer: C**

**Explanation:** Bubble Sort has a worst-case time complexity of $O(n^2)$, but it is useful for small datasets or nearly sorted data because of its simplicity and ease of implementation.

**Q.13**

Which of the following is not a commonly used UI framework for developing mobile applications?

A) Flutter
B) React Native
C) Django
D) SwiftUI

**Answer: C**

**Explanation:** Django is a web development framework for building backend services, not a UI framework for mobile applications. The others (Flutter, React Native, SwiftUI) are used for mobile app UI development.

**Q.14**

What is Automatic Reference Counting (ARC) in Swift?

A) A garbage collector that automatically deletes objects.
B) A mechanism that tracks and manages memory allocation.
C) A compiler directive to manually free memory.
D) A debugging tool for memory leaks.

**Answer: B**

**Explanation:** ARC automatically tracks and manages memory allocation by deallocating objects when they are no longer referenced, preventing memory leaks.

**Q.15**

Which syntax correctly calls a method named calculateResult on an Objective-C object named calculator?

A) calculator.calculateResult();

B) [calculator calculateResult];

C) calculator->calculateResult();

D) calculateResult(calculator);

**Answer: B**

**Explanation:** Objective-C uses square bracket syntax for message passing. The correct way to call a method is [object methodName];.

**Q.16**

What will be the output of the following PHP code?

```
$data = array("name" => "Alice", "age" => 25);
$json = json_encode($data);
echo $json;
```

A) {name: Alice, age: 25}

B) {"name":"Alice","age":25}

C) [Alice, 25]

D) ['name' => 'Alice', 'age' => 25]

**Answer: B**

**Explanation:** json_encode() converts a PHP associative array into a valid JSON string. It outputs {"name":"Alice","age":25} with double-quoted keys and values.

**Q.17**

Which SQL statement retrieves data from two tables where Orders.CustomerID matches Customers.CustomerID?

A) SELECT * FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

B) SELECT * FROM Orders OUTER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

C) SELECT * FROM Orders CONNECT Customers USING(CustomerID);

D) SELECT * FROM Orders UNION Customers WHERE Orders.CustomerID = Customers.CustomerID;

**Answer: A**

**Explanation:** INNER JOIN returns only the matching rows where Orders.CustomerID = Customers.CustomerID. The other options are incorrect because SQL does not support CONNECT or OUTER JOIN without specifying LEFT or RIGHT.

**Q.18**

What does the following Python list comprehension produce?

```
numbers = [x for x in range(10) if x % 2 == 0]
print(numbers)
```

A) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

B) [0, 2, 4, 6, 8]

C) [1, 3, 5, 7, 9]

D) []

**Answer: B**

**Explanation:** The list comprehension filters out only even numbers from 0 to 9, producing [0, 2, 4, 6, 8].

**Q.19**

**Consider the following NumPy operation:**

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr.T)
```

What does arr.T do?

A) Flattens the array into a one-dimensional list.

B) Transposes the array, swapping rows and columns.

C) Returns the sum of all elements.

D) Creates a new array with doubled values.

**Answer: B**

**Explanation:** arr.T transposes the matrix, converting [[1, 2, 3], [4, 5, 6]] into [[1, 4], [2, 5], [3, 6]] by swapping rows and columns.

**Q.20**

Which function in the statistics module is used to calculate the most frequently occurring value in a dataset?

A) statistics.mean()

B) statistics.median()

C) statistics.mode()

D) statistics.stdev()

**Answer: C**

**Explanation:** The statistics.mode() function finds the most frequently occurring value in a dataset.

**Q.21**

What will be the output of the following Python function?

```
def mystery(n):
if n == 0:
return 1
return n * mystery(n - 1)
print(mystery(5))
```

A) 15

B) 25

C) 120

D) 720

**Answer: C**

**Explanation:** This function computes n! (factorial of n).

`mystery(5) = 5 * 4 * 3 * 2 * 1 = 120.`

**Q.22**

What is the output of the following Python binary search function when called with binary_search([2, 3, 5, 7, 11], 7, 0, 4)?

```
def binary_search(arr, key, low, high):
if low > high:
return -1
mid = (low + high) // 2
if arr[mid] == key:
return mid
```

**elif arr[mid] < key:**

```
return binary_search(arr, key, mid + 1, high)
```

```
    else:
    return binary_search(arr, key, low, mid - 1)
    print(binary_search([2, 3, 5, 7, 11], 7, 0, 4))
```

A) 2

B) 3

C) 4

D) -1

**Answer: B**

**Explanation:** Binary search checks the middle index (mid = 2, value=5). Since 7 > 5, it searches the right half (mid+1 to high → index 3 to 4).

```
Now mid = 3, arr[3] = 7, so it returns 3.
```

**Q.23**

What will be the output of the following Swift program?

```
class Parent {
func greet() {
    print("Hello from Parent")
}
}
    class Child: Parent {
override func greet() {
    print("Hello from Child")
}
}
    let obj: Parent = Child()
obj.greet()
```

A) Hello from Parent

B) Hello from Child

C) Compilation Error

D) None of the above

**Answer: B**

**Explanation:** Even though obj is of type Parent, due to method overriding, the function greet() in Child is executed.

**Q.24**

Which statement correctly allocates and releases an Objective-C object manually when ARC is disabled?

A)

```
MyClass *obj = [MyClass alloc] init];
[obj release];
```

B)

```
MyClass *obj = [[MyClass alloc] init];
[obj release];
```

C)

```
MyClass *obj = [MyClass new];
[obj dealloc];
```

D)

```
MyClass *obj = [[MyClass init] alloc];
[obj free];
```

**Answer: B**

**Explanation:** For manual memory management, use alloc + init and release when the object is no longer needed.

**Q.25**

What will be the output of the following PHP code?

```
$students = array("Alice" => 95, "Bob" => 80, "Charlie" => 88);
foreach($students as $name => $marks) {
echo "$name scored $marks\n";
}
```

A) Compilation Error

B) Alice scored 95
Bob scored 80
Charlie scored 88

C) [Alice, 95], [Bob, 80], [Charlie, 88]

D) None of the above

**Answer: B**

**Explanation:** The foreach loop iterates over an associative array, printing key-value pairs correctly.

**Q.26**

**Consider the following SQL tables:**

**Employees Table:**

ID | Name | Dept_ID
1 | Alice | 101
2 | Bob | 102
3 | Charlie | 103

**Departments Table:**

Dept_ID | Dept_Name
101 | HR
102 | IT
103 | Finance

What SQL query returns employee names and their department names?

A)
```
SELECT * FROM Employees JOIN Departments;
```

B)
```
SELECT Employees.Name, Departments.Dept_Name
FROM Employees INNER JOIN Departments
ON Employees.Dept_ID = Departments.Dept_ID;
```

C)
```
SELECT Employees.Name, Employees.Dept_ID
FROM Employees JOIN Departments;
```

D)
```
SELECT * FROM Employees;
```

**Answer: B**

**Explanation:** INNER JOIN is used to match Employees and Departments using Dept_ID.

**Q.27**

What does the following code output?

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
B = np.array([[2, 0], [1, 2]])
C = np.dot(A, B)
print(C)
```

A)
[[2, 0], [1, 2]]

B)
[[4, 4], [10, 8]]

C)
[[4, 4], [6, 6]]

D) Compilation Error

**Answer: B**

**Explanation:** Matrix multiplication is performed using dot product:

```
C=[(1×2+2×1)(1×0+2×2)(3×2+4×1)(3×0+4×2)]=[44108]C = \begin{bmatrix} (1×2 + 2×1) & (1×0 + 2×2) \\ (3×2 + 4×1) & (3×0 + 4×2) \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 10 & 8 \end{bmatrix}C=[(1×2+2×1)(3×2+4×1)(1×0+2×2)(3×0+4×2)]=[41048]
```

**Q.28**

What will be the output of the following Python code?

```
import pandas as pd
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Score': [95, 80, 88]}
df = pd.DataFrame(data)
filtered = df[df['Score'] > 85]
print(filtered)
```

A)
Name Score
0 Alice 95
2 Charlie 88

B)
Name Score
1 Bob 80

C) Compilation Error

D) None of the above

**Answer: A**

**Explanation:** Pandas filters rows where 'Score' > 85, returning Alice and Charlie.

**Q.29**

If a dataset is skewed, which measure of central tendency is more reliable?

A) Mean

B) Median

C) Mode

D) Variance

**Answer: B**

**Explanation:** The median is more resistant to outliers and skewness than the mean.

**Q.30**

Which SQL query finds the highest salary in an Employees table?

A)

```
SELECT MAX(salary) FROM Employees;
```

B)

```
SELECT salary FROM Employees ORDER BY salary DESC LIMIT 1;
```

C) Both A and B

D) None of the above

**Answer: C**

**Explanation:** Both methods correctly find the maximum salary.

**Q.31**

A developer wishes to optimize a recursive factorial function by caching results to avoid redundant computations. Examine the following code:

```
def cache(func):
memo = {}
def wrapper(n):
if n in memo:
return memo[n]
result = func(n)
memo[n] = result
return result
return wrapper
@cache
def factorial(n):
if n == 0:
return 1
return n * factorial(n - 1)
print(factorial(6))
```

What is the output, and how does the caching decorator improve performance?

A) 720; it prevents repeated recursive calls by storing computed values

B) 720; it increases recursion depth without storing any values

C) 120; it computes a different factorial

D) 720; it adds extra overhead, making it slower

**Answer: A**

**Explanation:** The factorial of 6 is 720. The decorator caches results so that when the function is called with a previously computed value, it returns the cached result instead of performing further recursion. This transforms the exponential redundant recursion into a linear sequence of computations, greatly improving performance.

**Q.32**

A programmer writes a function to find the longest palindromic substring in a given string using a dynamic programming approach. Consider this code:

```
def longest_palindrome(s):
n = len(s)
dp = [[False] * n for _ in range(n)]
start, max_length = 0, 1
for i in range(n):
dp[i][i] = True
for i in range(n-1):
if s[i] == s[i+1]:
```

```
dp[i][i+1] = True
start = i
max_length = 2
for k in range(3, n+1):
for i in range(n-k+1):
j = i + k - 1
if s[i] == s[j] and dp[i+1][j-1]:
dp[i][j] = True
start = i
max_length = k
return s[start:start+max_length
print(longest_palindrome("babad"))
```

What is a correct output of this code, and what is the role of the dp table?

A) "bab" or "aba"; dp stores whether the substring from index i to j is a palindrome

B) "bad"; dp stores sorted order of characters

C) "aba"; dp records the count of palindromic substrings

D) "babad"; dp is not used in the final answer

**Answer: A**

**Explanation:** The function finds the longest palindromic substring. For input "babad", both "bab" and "aba" are acceptable outputs. The dp table is used to record whether the substring s[i:j+1] is a palindrome, enabling the function to build solutions for longer substrings from shorter ones.

**Q.33**

```
A C++ developer writes a function to detect cycles in a directed graph using DFS with a recursion
stack. Consider this pseudocode:
```

bool dfs(int node, vector<int> adj[], vector<bool>& visited, vector<bool>& recStack) {

if(!visited[node]) {

```
visited[node] = true;
recStack[node] = true;
for (int neighbor : adj[node]) {
```

if(!visited[neighbor] && dfs(neighbor, adj, visited, recStack))

```
return true;
else if(recStack[neighbor])
return true;
```

}

}

```
recStack[node] = false;
return false;
```

}

What is the purpose of using the recStack vector in this algorithm?

A) It stores the final cycle.

B) It keeps track of nodes in the current recursion path to detect cycles.

C) It records all visited nodes to prevent re-visitation.

D) It ensures nodes are processed in alphabetical order.

**Answer: B**

**Explanation:** The recStack vector is used to keep track of the nodes in the current DFS recursion path. If a node is encountered that is already in recStack, it indicates a cycle. This method is essential for detecting cycles in directed graphs.

**Q.34**

A company's database contains an Orders table with columns OrderID, CustomerID, and TotalAmount. A data analyst needs to find customers whose total order amount exceeds the overall average order amount. Which query correctly achieves this?

A)
```
SELECT CustomerID, SUM(TotalAmount) AS CustomerTotal
FROM Orders
GROUP BY CustomerID
HAVING SUM(TotalAmount) > (SELECT AVG(TotalAmount) FROM Orders);
```
B)
```
SELECT CustomerID, TotalAmount
FROM Orders
WHERE TotalAmount > (SELECT AVG(TotalAmount) FROM Orders)
GROUP BY CustomerID;
```
C)
```
SELECT CustomerID
FROM Orders
WHERE TotalAmount > (SELECT SUM(TotalAmount) FROM Orders);
```
D)
```
SELECT CustomerID, SUM(TotalAmount) AS CustomerTotal
FROM Orders
HAVING CustomerTotal > (SELECT AVG(TotalAmount) FROM Orders);
```

**Answer: A**

**Explanation:** Option A groups orders by CustomerID, calculates the total amount per customer, and then uses the HAVING clause to filter out customers whose total order amount exceeds the overall average order amount computed by the subquery. Option B incorrectly groups after the WHERE clause, and options C and D misuse aggregate functions.

**Q.35**

A PHP developer is implementing session management for a login system but finds that session variables are not persisting. Examine this code snippet:

```
<?php
session_start();
$_SESSION['user'] = 'Alice';
// ... some code that redirects to another page without calling session_start()
echo $_SESSION['user'];
?>
```

What is the likely error, and how can it be resolved?

A) Session variables are case-sensitive; change 'user' to 'User'.

B) session_start() must be called on every page that uses session variables.

C) The variable is not echoed correctly; use print instead of echo.

D) The session file is corrupted and must be deleted.

**Answer: B**

**Explanation:** For session variables to persist, session_start() must be invoked at the beginning of every page that accesses the session. Without this, the session data will not be available on the redirected page.

**Q.36**

A developer has two dictionaries representing scores from two tests:

```
test1 = {'Alice': 85, 'Bob': 90, 'Charlie': 78}
test2 = {'Alice': 88, 'Bob': 85, 'David': 92}
```

The developer wants to create a new dictionary that sums the scores for students who took both tests. Consider the following code:

```
combined = {k: test1[k] + test2[k] for k in test1 if k in test2}
print(combined)
```

**Question:**

What is the output, and what does the code achieve?

A) {'Alice': 173, 'Bob': 175}; It sums scores only for students present in both tests.

B) {'Alice': 173, 'Bob': 175, 'Charlie': 78, 'David': 92}; It merges all students.

C) {'Alice': 88, 'Bob': 85}; It replaces test1 scores with test2 scores.

D) {'Charlie': 78, 'David': 92}; It filters out common students.

**Answer: A**

**Explanation:** The comprehension iterates through keys in test1 and checks if the key exists in test2. For common students (Alice and Bob), it sums their scores: Alice: 85 + 88 = 173; Bob: 90 + 85 = 175. Therefore, the output is {'Alice': 173, 'Bob': 175}.

**Q.37**

A Swift developer writes the following code to safely unwrap an optional string:

```
var response: String? = nil
if let message = response {
print("Response: \(message)")
} else {
print("No response received")
}
```

**Question:**

What is the output of the code, and why is optional binding used here?

A) "Response: nil"; optional binding incorrectly converts nil to a string.

B) "No response received"; optional binding safely handles nil values.

C) "No response received"; but the code will crash at runtime.

D) "Response: Optional(nil)"; because it prints the optional itself.

**Answer: B**

**Explanation:** Optional binding (if let) safely unwraps the optional. Since response is nil, the else clause executes, printing "No response received". This prevents a runtime crash that could occur with force unwrapping.

**Q.38**

A C programmer writes a bubble sort function to sort an array of integers. Consider the following code:

```
#include <stdio.h>
void bubbleSort(int arr[], int n) {
for (int i = 0; i < n-1; i++) {
for (int j = 0; j < n-i-1; j++) {
if (arr[j] > arr[j+1]) {
int temp = arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
}
}
}
}
int main() {
int arr[] = {64, 34, 25, 12, 22, 11, 90};
int n = sizeof(arr)/sizeof(arr[0]);
```

```
bubbleSort(arr, n);
   for (int i = 0; i < n; i++) {
   printf("%d ", arr[i]);
}
   return 0;
}
```

**Question:**

What is the sorted output of the array, and what is the time complexity of this bubble sort algorithm?

A) 11 12 22 25 34 64 90; $O(n^2)$

B) 11 12 22 25 34 64 90; O(n log n)

C) 90 64 34 25 22 12 11; $O(n^2)$

D) 11 12 22 25 34 64 90; O(n)

**Answer: A**

**Explanation:** Bubble sort arranges the array in ascending order: 11, 12, 22, 25, 34, 64, 90. Its worst-case time complexity is $O(n^2)$, making option A correct.

**Q.39**

A PHP developer needs to validate email addresses using regular expressions. Consider the following snippet:

```
   <?php
   $email = "user@example.com";
   if (preg_match("/^[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/", $email)) {
   echo "Valid email";
} else {
   echo "Invalid email";
}
   ?>
```

**Question:**

What does this code do, and which aspect of the regular expression is most critical for matching a proper domain extension?

A) It checks if the email is valid; [a-zA-Z]{2,6} ensures the domain extension is between 2 to 6 letters.

B) It checks if the email is valid; [a-zA-Z0-9] ensures numeric domain extensions.

C) It always returns "Invalid email"; the regex is incorrect.

D) It checks only for the presence of "@" in the email; the rest of the regex is redundant.

**Answer: A**

**Explanation:** The code uses preg_match to validate the email format. The portion [a-zA-Z]{2,6} ensures that the domain extension is composed of 2 to 6 letters (like com, org, museum), which is critical for proper email validation.

**Q.40**

A business intelligence analyst wants to calculate a running total of sales from a table Sales(SaleID, SaleDate, Amount). Consider the SQL query:

```
   SELECT SaleID, SaleDate, Amount,
SUM(Amount) OVER (ORDER BY SaleDate) AS RunningTotal
   FROM Sales;
```

**Question:**

What does this query do, and how does the window function improve the calculation of a running total?

A) It returns the total sales amount only; the window function is not needed.

B) It returns each sale along with a cumulative sum of sales ordered by SaleDate, allowing for a dynamic running total calculation.

C) It sorts the data by SaleID instead of SaleDate.

D) It deletes records with lower amounts.

**Answer: B**

**Explanation:** The window function SUM(Amount) OVER (ORDER BY SaleDate) computes a cumulative sum of the Amount column, ordered by SaleDate, for each row. This allows the query to return a running total along with each sale record. This approach simplifies calculations compared to self-joins or subqueries.

**Q.41**

```
values = [1, 2, 3, 4]
squares = [x**2 for x in values if y > 2]
print(squares)
```

**Question:**

What is the error in the code?

A) The exponent operator is used incorrectly.

B) The variable y is not defined in the list comprehension.

C) The list comprehension syntax is invalid.

D) The condition should use x < 2.

**Answer: B**

**Explanation:** The list comprehension uses the variable y in the condition, but y is never defined. It should likely use x instead. The error is due to the use of an undefined variable.

**Q.42**

```
def add_item(item, items_list=[]):
```

items_list.append(item)

```
return items_list
print(add_item("apple"))
print(add_item("banana"))
```

**Question:**

What unexpected behavior might this code exhibit?

A) It prints an error due to mutable default arguments.

B) It prints ["apple"] then ["banana"].

C) It prints ["apple"] then ["apple", "banana"].

D) It always prints an empty list.

**Answer: C**

**Explanation:** Using a mutable default argument (an empty list) causes the same list to be shared across function calls. The first call appends "apple," and the second call appends "banana" to the same list, resulting in ["apple", "banana"].

**Q.43**

```
#include <stdio.h>
int main() {
int numbers[4] = {10, 20, 30, 40};
for (int i = 0; i <= 4; i++) {
printf("%d ", numbers[i]);
}
return 0;
}
```

**Question:**

What is the error in the code?

A) The array is declared incorrectly.

B) The loop condition i <= 4 causes an out-of-bounds access.

C) The printf statement has a format specifier error.

D) There is no error; the code prints all numbers.

**Answer: B**

**Explanation:** Arrays in C are zero-indexed. With 4 elements, valid indices are 0 to 3. The loop condition i <= 4 iterates for i=4, leading to an out-of-bounds access.

**Q.44**

```
#include <stdio.h>
int main() {
int arr[5] = {1, 2, 3, 4, 5};
int *p = arr;
printf("%d\n", *(p++));
printf("%d\n", *p);
return 0;
}
```

**Question:**

What is the likely output, and is there an error in pointer usage?

A) 1 then 2; The pointer arithmetic is correct.

B) 2 then 3; The pointer is incremented incorrectly.

C) 1 then 1; The pointer is not incremented.

D) Undefined behavior; The pointer arithmetic is incorrect.

**Answer: A**

**Explanation:** The expression *(p++) prints the current value (1) and then increments the pointer. The next *p then points to the second element (2). The pointer arithmetic is correctly used here.

**Q.45**

```
<?php
$user = "John";
echo $User;
?>
```

**Question:**

What is the error in this code?

A) PHP variables are case-sensitive; $User is not the same as $user.

B) The echo statement is missing quotes.

C) The variable should be declared as $User initially.

D) There is no error; PHP automatically converts variable names.

**Answer: A**

**Explanation:** PHP variable names are case-sensitive. The variable is defined as $user, but the code attempts to echo $User, which is undefined, leading to an error or no output.

**Q.46**

```
SELECT id, name, COUNT(*)
FROM orders, customers
WHERE orders.customer_id = customers.customer_id
GROUP BY name;
```

**Question:**

What is the error in this SQL query?

A) The query does not use JOIN syntax.

B) The column id is ambiguous and not aggregated or grouped.

C) The WHERE clause is missing conditions.

D) The GROUP BY clause should group by both id and name.

**Answer: B**

**Explanation:** The column id appears in both tables, making it ambiguous since it is not specified which table's id to use. Additionally, if it is not included in the GROUP BY clause or aggregated, the query will raise an error.

**Q.47**

```
var message: String? = nil
print(message!)
```

**Question:**

What error does this code produce, and why?

A) It prints "nil".

B) It causes a runtime crash because force unwrapping a nil optional leads to a fatal error.

C) It compiles successfully and prints an empty string.

D) It prints "Optional(nil)".

**Answer: B**

**Explanation:** Force unwrapping (!) an optional that is nil causes a runtime crash. The code should safely unwrap the optional using optional binding (e.g., if let) instead of force unwrapping.

**Q.48**

```
def count_up(n):
for i in range(n):
return i
print(list(count_up(5)))
```

**Question:**

What is the error in this generator function?

A) The function should use yield instead of return to generate all numbers.

B) The range is incorrect.

C) The function prints the wrong type of object.

D) There is no error; it correctly generates a list.

**Answer: A**

**Explanation:** Using return inside a loop exits the function after the first iteration. To generate a sequence, yield should be used. The corrected function would use yield i inside the loop.

**Q.49**

```
<?php
$first = "Hello";
$second = "World";
echo $first + $second;
?>
```

**Question:**

What is the error in this PHP code?

A) PHP does not allow string variables.

B) The code should use the concatenation operator . instead of +.

C) The code should use commas instead of the plus operator.

D) There is no error; it outputs "HelloWorld".

**Answer: B**

**Explanation:** In PHP, the + operator is used for numeric addition. For string concatenation, the dot operator (.) should be used. Using + results in PHP converting strings to numbers (usually 0), so the output would not be as expected.

School Connect Online

# Answer Key

| Question / Answer | Question / Answer | Question / Answer | Question / Answer | Question / Answer |
|---|---|---|---|---|
| Q.1 - - | Q.1 - B | Q.2 - B | Q.3 - B | Q.4 - B |
| Q.5 - A | Q.6 - B | Q.7 - B | Q.8 - B | Q.9 - B |
| Q.10 - B | Q.11 - B | Q.12 - C | Q.13 - C | Q.14 - B |
| Q.15 - B | Q.16 - B | Q.17 - A | Q.18 - B | Q.19 - B |
| Q.20 - C | Q.21 - C | Q.22 - B | Q.23 - B | Q.24 - B |
| Q.25 - B | Q.26 - B | Q.27 - B | Q.28 - A | Q.29 - B |
| Q.30 - C | Q.31 - A | Q.32 - A | Q.33 - B | Q.34 - A |
| Q.35 - B | Q.36 - A | Q.37 - B | Q.38 - A | Q.39 - A |
| Q.40 - B | Q.41 - B | Q.42 - C | Q.43 - B | Q.44 - A |
| Q.45 - A | Q.46 - B | Q.47 - B | Q.48 - A | Q.49 - B |

School Connect Online

## Space for Rough Work