School Connect Online

SCO INTERNATIONAL OLYMPIAD

CLASS 11 OFFICIAL QUESTION PAPER

SCO International Coding Olympiad

Designed from the latest Class 11 Coding Olympiad paper and aligned to global computer science learning expectations.

- official SCO cover-page style used as an editable first page
- professionally formatted question blocks with compact inline question numbers
- code snippets, SQL examples, case studies, answer key and explanations
- PDF-ready layout for student, teacher and school access

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	Official Question 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. Consider the following Python code that uses nested loops and a conditional statement:

```
total = 0
for i in range(1, 5):
for j in range(1, 4):
if (i + j) % 2 == 0:
total += i * j
print(total)
```

What is the output of the code, and what is the role of the conditional statement in the loops?

A) 28; The condition filters only pairs with even sums before multiplying.
B) 30; The condition filters pairs with odd sums before multiplying.
C) 36; The condition has no effect on the result.
D) 26; The condition randomly selects pairs.

Answer: A

Explanation:

Only pairs whose sum is even are multiplied and added: (1,1)=1, (1,3)=3, (2,2)=4, (3,1)=3, (3,3)=9 and (4,2)=8. The total is 1 + 3 + 4 + 3 + 9 + 8 = 28.

Q2. Given the Python code:

```
arr = [10, 20, 30, 40, 50]
result = [arr[i] + arr[-i-1] for i in range(len(arr)//2)]
print(result)
```

What does the code print, and what is the logic behind it?

A) [60, 60] - It adds the first and last elements, then the second and second-last.
B) [60, 60, 60] - It adds all pairs symmetrically.
C) [60, 60, 60, 60] - It adds each element to itself.
D) [10, 20, 30, 40, 50] - It simply copies the array.

Answer: A

Explanation:

The list comprehension iterates for i from 0 to len(arr)//2 - 1 (i.e., 0 and 1 for a list of 5 elements). For i=0: arr[0] + arr[-1] = 10 + 50 = 60; for i=1: arr[1] + arr[-2] = 20 + 40 = 60. The result is [60, 60].

Q3. A developer is building a RESTful API to fetch user details from a MySQL database using PHP. Which HTTP method and SQL command combination should be used to securely retrieve a user's data by ID?

A) POST with DELETE
B) GET with a parameterized SELECT query
C) PUT with UPDATE
D) DELETE with SELECT

Answer: B

Explanation:

For retrieval, a REST API should use GET, and the database operation should be a SELECT query. In real implementation, the user ID must be supplied through a parameterized/prepared query so user input cannot change the SQL command.

Q4. What is the output of the following Python code?

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result = [x for x in nums if x % 2 == 0 and x > 5]
print(result)
```

A) [2, 4, 6, 8, 10]
B) [6, 8, 10]
C) [2, 4]
D) [6, 10]

Answer: B

Explanation:
The code filters nums to include only numbers that are even and greater than 5. The even numbers greater than 5 in the list are 6, 8, and 10, so the output is [6, 8, 10].

Q5. Consider the following Python code using Pandas:

```
import pandas as pd
data = {
'Class': ['X', 'Y', 'X', 'Z', 'Y', 'X'],
'Score': [80, 90, 70, 85, 95, 100]
}
df = pd.DataFrame(data)
result = df.groupby('Class')['Score'].mean().round(1)
print(result)
```

What does the code output, and what operation does it perform?
A) It outputs the total score for each class.
B) It outputs the average score for each class, rounded to one decimal place.
C) It outputs the maximum score for each class.
D) It outputs the count of scores in each class.

Answer: B

Explanation:
The code groups the data by the 'Class' column and calculates the mean of the 'Score' column for each group, then rounds the results to one decimal place. This yields the average scores per class.

Q6. Consider the following Swift code snippet:

```
let numbers = [1, 3, 5, 7, 9, 10, 12]
let evenNumbers = numbers.filter { $0 % 2 == 0 }
print(evenNumbers)
```

What is the output, and what is the role of the filter function?
A) [1, 3, 5, 7, 9] - It removes even numbers.
B) [10, 12] - It filters out odd numbers, keeping only even ones.
C) [1, 3, 5, 7, 9, 10, 12] - It does not change the array.
D) [1, 10, 12] - It randomly selects numbers.

Answer: B

Explanation:
The filter function applies the condition $0 % 2 == 0 to each element in the array, keeping only the even numbers. Therefore, the output is [10, 12].

Q7. What is the output of the following C code snippet?

```
#include <stdio.h>
int main() {
int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};
printf("%d\n", arr[1][1]);
return 0;
```

}
A) 1
B) 2
C) 5
D) 6

Answer: C

Explanation:
In the two-dimensional array, arr[1][1] refers to the second row, second column, which contains the value 5.

Q8. Given the following PHP code:

```
<?php
$grades = array("Alice" => 85, "Bob" => 92, "Charlie" => 78);
foreach($grades as $name => $score) {
if($score > 80) {
echo $name . " ";
}
}
?>
```

What is the output?
A) Alice Bob Charlie
B) Bob Charlie
C) Alice Bob
D) Bob

Answer: C

Explanation:
The code loops over the associative array $grades. It echoes the name if the score is greater than 80. "Alice" (85) and "Bob" (92) meet the condition, while "Charlie" (78) does not. Thus, the output is "Alice Bob".

Q9. Consider two tables, Students(StudentID, Name) and Grades(StudentID, Score). What does the following SQL query do?

```
SELECT s.Name, AVG(g.Score) AS AverageScore
FROM Students s
JOIN Grades g ON s.StudentID = g.StudentID
GROUP BY s.Name
HAVING AVG(g.Score) >= 75;
```

A) Lists all students with their total scores.
B) Lists students with an average score of 75 or higher, along with their average score.
C) Deletes students with scores below 75.
D) Updates student names with their average scores.

Answer: B

Explanation:
The query performs an inner join between Students and Grades on StudentID, groups the results by student name, calculates the average score for each student, and then filters the groups to include only those with an average score of at least 75. It returns the student names and their average scores for students whose grouped average is at least 75; no ORDER BY clause is present, so no sorting is guaranteed.

Q10. A developer is implementing session management in a PHP web application to track user login status. Which of the following code snippets correctly starts a session and stores a user's ID in the session variable?

A)
session_start();
$_SESSION['user_id'] = 101;
B)
session();
$_SESSION['user_id'] = 101;
C)
start_session();
$_SESSION['id'] = 101;
D)
session_start();
$SESSION['user_id'] = 101;

Answer: A

Explanation:

In PHP, to use sessions, you must call session_start() at the beginning. Then, you can assign values to the $_SESSION superglobal array. Option A correctly uses session_start() and then sets $_SESSION['user_id'] = 101. Option D is incorrect due to the missing underscore in $SESSION.

Q11. Which of the following is not a valid loop control statement in Python?

A) continue
B) break
C) exit
D) pass

Answer: C - exit

Explanation:

Q12. In C programming, what is the output of the following code?

```
#include <stdio.h>
int main() {
int arr[] = {10, 20, 30, 40};
printf("%d", *(arr + 2));
return 0;
}
```

A) 10
B) 20
C) 30
D) 40

Answer: C - 30

Explanation:

Q13. Which HTTP status code indicates that the client must authenticate itself to get the requested response?

A) 200
B) 301
C) 403
D) 401

Answer: D - 401

Explanation:

Q14. Which of the following Python modules is used for serializing and deserializing objects?

A) pickle
B) json
C) os
D) re

Answer: A - pickle
Explanation:

Q15. Which of the following is NOT a supervised learning algorithm?

A) Linear Regression
B) Decision Trees
C) K-Means Clustering
D) Random Forest

Answer: C - K-Means Clustering
Explanation:

Q16. In Python, which function is used to calculate the standard deviation of an array?

A) numpy.std()
B) numpy.var()
C) numpy.mean()
D) numpy.median()

Answer: A - numpy.std()
Explanation:

Q17. Which keyword is used to declare a constant in Swift?

A) const
B) let
C) static
D) final

Answer: B - let
Explanation:

Q18. What will be the output of the following C program?

```
#include <stdio.h>
void main() {
int x = 5;
printf("%d", x++ * ++x);
}
```

A) 25
B) 30
C) Undefined behavior
D) Compilation error

Answer: C - Undefined behavior
Explanation:

Q19. Which function in PHP is used to send a raw HTTP header?

A) setcookie()
B) header()
C) session_start()
D) http_response_code()

Answer: B - header()

Explanation:

Q20. Which SQL clause is used to remove duplicate rows from a result set?

A) DELETE
B) UNIQUE
C) DISTINCT
D) FILTER

Answer: C - DISTINCT

Explanation:

Section 2: Algorithmic Problem Solving and Applied Programming

Q21. A developer is optimizing a computationally intensive function that calculates the nth Fibonacci number. To avoid repeated calculations, the developer implements a caching decorator. Consider the code below:

```
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 fib(n):
if n <= 1:
return n
return fib(n-1) + fib(n-2)
print(fib(10))
```

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

A) 34; It avoids recalculating already computed Fibonacci numbers by storing them.
B) 55; It avoids recalculating already computed Fibonacci numbers by storing them.
C) 89; It increases memory usage without improving speed.
D) 55; It has no effect because Python caches function calls automatically.

Answer: B

Explanation:

The 10th Fibonacci number (using 0-indexing: fib(0)=0, fib(1)=1, …) is 55. The caching decorator stores previously computed results in the dictionary memo, so repeated calls (e.g., fib(5) multiple times) are computed only once. This reduces the exponential time complexity of naive recursion to linear time.

Q22. A data scientist needs to process a very large list of numbers to compute their squares without storing the entire list of squared values in memory. Consider the code:

```
def generate_squares(n):
return (i**2 for i in range(n))
squares = generate_squares(1000000)
print(next(squares))
```

What does the code print, and what is the key advantage of using a generator expression here?

A) 0; It generates squares on-the-fly, saving memory.
B) 1; It generates all squares at once, using more memory.
C) 0; It computes all squares first then returns a generator.
D) 1; It returns a list of squares, consuming more memory.

Answer: A

Explanation:

The generator expression yields squares on demand. When next(squares) is called, it computes 0**2 = 0. Using a generator is memory efficient because it does not create a full list in memory, especially important when n is large.

Q23. A programmer implements a singly linked list in Python and needs to reverse the list in-place. The function reverse_list is given by:

```
class Node:
def __init__(self, data):
self.data = data
self.next = None
def reverse_list(head):
prev = None
current = head
while current:
nxt = current.next
current.next = prev
prev = current
current = nxt
return prev
# Construct linked list: 1 -> 2 -> 3 -> None
head = Node(1)
head.next = Node(2)
head.next.next = Node(3)
new_head = reverse_list(head)
# Traverse reversed list
result = []
while new_head:
result.append(new_head.data)
new_head = new_head.next
print(result)
```

What is the output of the code, and why is this in-place reversal algorithm efficient?

A) [1, 2, 3]; It creates a new list.
B) [3, 2, 1]; It reverses pointers without using extra space.
C) [3, 1, 2]; It randomly reorders nodes.
D) [2, 3, 1]; It uses extra memory to store nodes.

Answer: B

Explanation:

The algorithm reverses the linked list by adjusting the pointers in-place. The resulting list is [3, 2, 1]. Since it does not create new nodes or use an auxiliary data structure, it is space efficient (O(1) additional space).

Q24. A C programmer writes the following code to create and manipulate a dynamic array:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int *arr = malloc(5 * sizeof(int));
for (int i = 0; i < 5; i++) {
arr[i] = (i + 1) * 10;
}
int sum = 0;
for (int i = 0; i < 5; i++) {
sum += *(arr + i);
}
printf("%d\n", sum);
free(arr);
return 0;
}
```

What is the output of this code, and why is pointer arithmetic used effectively here?

A) 150; Pointer arithmetic allows direct element access with minimal overhead.

B) 150; Pointer arithmetic slows down the computation.

C) 120; The loop is off by one index.

D) 200; The dynamic array is not properly allocated.

Answer: A

Explanation:

The array contains: 10, 20, 30, 40, 50. Their sum is 150. The expression *(arr + i) accesses the i-th element directly, which is efficient and equivalent to arr[i]. The code properly allocates, accesses, sums, and frees memory.

Q25. A PHP developer is tasked with retrieving user data securely from a MySQL database. The developer uses a prepared statement as follows:

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

Why is this code snippet considered secure, and what potential vulnerability does it mitigate?

A) It encrypts the database; it mitigates hardware failure.

B) It uses prepared statements to avoid SQL injection.

C) It caches user input for faster access; it mitigates performance issues.

D) It logs user data; it mitigates data loss.

Answer: B

Explanation:

The code uses a prepared statement with parameter binding, ensuring that user input is treated as data rather than executable code. This practice mitigates SQL injection vulnerabilities, which can occur when user input is directly embedded into SQL queries.

Q26. A database contains two tables: Employees(EmpID, Name, Department) and Salaries(EmpID, Salary). A data analyst needs to find the average salary for each department where the average salary exceeds 50000. Consider the query:

```
SELECT e.Department, AVG(s.Salary) AS AvgSalary
FROM Employees e
JOIN Salaries s ON e.EmpID = s.EmpID
GROUP BY e.Department
HAVING AVG(s.Salary) > 50000;
```

What does this query return, and what is the role of the HAVING clause?

A) It returns departments with exactly 50000 average salary; HAVING filters after grouping.

B) It returns departments with an average salary higher than 50000; HAVING filters the aggregated results.

C) It returns individual employee salaries; HAVING is unnecessary.

D) It updates salaries for departments; HAVING modifies data.

Answer: B

Explanation:

The query calculates the average salary for each department and then uses the HAVING clause to filter only those departments where the average salary is greater than 50000. The HAVING clause is applied after the GROUP BY operation.

Q27. A developer is creating a RESTful API endpoint in Node.js that returns JSON data. The endpoint must handle errors gracefully. Consider the pseudo-code:

```
app.get('/api/data', (req, res) => {
try {
let data = fetchDataFromDB(); // Function that may throw an error
res.status(200).json({ success: true, data });
} catch (error) {
res.status(500).json({ success: false, error: "Server Error" });
}
});
```

What is the purpose of the try-catch block in this code, and what HTTP status codes are used?

A) To handle errors during database fetch; 200 for success and 500 for server error.

B) To prevent the API from running; 404 for errors.

C) To debug the code; 301 for success.

D) To cache the data; 200 for errors.

Answer: A

Explanation:

The try-catch block is used to catch any errors that occur during the data fetch from the database. If an error occurs, the API returns an HTTP 500 status code indicating a server error. On success, it returns HTTP 200. This provides robust error handling and appropriate client responses.

Q28. A data analyst needs to filter out all prime numbers from a large list. They write the following code:

```
def is_prime(n):
if n < 2:
return False
for i in range(2, int(n**0.5) + 1):
if n % i == 0:
return False
return True
nums = list(range(1, 51))
primes = list(filter(is_prime, nums))
```

print(primes)
What does the code output, and how does the filter function work in this context?
A) A list of numbers from 1 to 50; filter returns all numbers.
B) A list of prime numbers between 1 and 50; filter uses the is_prime function to select primes.
C) A list of even numbers; filter checks for divisibility by 2.
D) An error occurs because of the prime checking logic.

Answer: B

Explanation:
The is_prime function returns True for prime numbers. The filter function applies is_prime to each number in the list and returns only those for which the function returns True. The output is a list of prime numbers between 1 and 50.

Q29. A Swift developer is tasked with converting an array of integers into their string representations and then concatenating them into one string. Consider the following code:

```
let numbers = [1, 2, 3, 4, 5]
let stringNumbers = numbers.map { String($0) }
let result = stringNumbers.joined(separator: "-")
print(result)
```

What is the output, and what does the map function accomplish here?
A) "12345"; map converts numbers to strings without separators.
B) "1-2-3-4-5"; map converts each integer to a string, and joined concatenates them with a hyphen.
C) "1,2,3,4,5"; map uses commas by default.
D) "1 2 3 4 5"; map inserts spaces between numbers.

Answer: B

Explanation:
The map function transforms each integer in the array into its string representation. The joined(separator: "-") method then concatenates these strings, separated by hyphens. The output is "1-2-3-4-5".

Q30. A company wants to rank employees by salary within each department. Consider the following SQL query:

```
SELECT Name, Department, Salary,
RANK() OVER (PARTITION BY Department ORDER BY Salary DESC) AS RankInDept
FROM Employees;
```

What does this query accomplish, and how does the PARTITION BY clause function in this context?
A) It ranks all employees globally by salary; PARTITION BY is ignored.
B) It ranks employees within each department by salary in descending order; PARTITION BY divides the dataset into groups per department.
C) It deletes employees with the lowest salary in each department; PARTITION BY groups them for deletion.
D) It calculates the total salary per department; PARTITION BY sums the salaries.

Answer: B

Explanation:
The query uses the RANK() window function to assign a rank to each employee based on their salary within their department. The PARTITION BY Department clause groups the employees by department, so ranking is done independently within each group. The ORDER BY Salary DESC ensures that the highest salary gets rank 1.

Section 3: Practical Coding, Web and Data Applications

Q31. A developer is optimizing a computationally expensive function that calculates the nth Fibonacci number. To avoid redundant computations, the developer implements a caching decorator. Consider 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 fib(n):
if n <= 1:
return n
return fib(n - 1) + fib(n - 2)
print(fib(10))
```

What is the output of this code, and what is the primary benefit of using the caching decorator?

A) 34; caching reduces memory usage.

B) 55; caching avoids redundant recursive calls and improves performance.

C) 89; caching increases the recursion depth.

D) 55; caching makes no difference because Python caches automatically.

Answer: B

Explanation:

The 10th Fibonacci number is 55. The caching decorator stores previously computed Fibonacci numbers in the dictionary memo, thus avoiding repeated recursive calls. This transforms the exponential-time recursive algorithm into one that runs in linear time.

Q32. A data scientist needs to process a massive sequence of numbers without loading all values into memory. The scientist writes a generator function as follows:

```
def generate_numbers(n):
for i in range(n):
yield i
gen = generate_numbers(1000000)
print(next(gen))
```

What does the code print, and why are generators advantageous in this scenario?

A) 0; generators produce items one-by-one, saving memory.

B) 0; generators store all values in memory.

C) 1000000; generators produce the final value immediately.

D) An error occurs because generators cannot be iterated.

Answer: A

Explanation:

The generator yields numbers starting at 0. Calling next(gen) returns 0. Generators are memory-efficient since they yield items on demand rather than storing the entire sequence in memory.

Q33. A developer builds a RESTful API for a bookstore using Node.js and Express. One endpoint fetches a list of books. Consider the code:

```
const express = require('express');
const app = express();
app.get('/books', (req, res) => {
res.status(200).json({ success: true, data: ["Book A", "Book B", "Book C"] });
});
```

```
app.listen(3000, () => console.log("Server running on port 3000"));
```

Which HTTP method is used for retrieving the list of books, and what status code does the endpoint return on success?

A) GET; 200
B) POST; 201
C) PUT; 200
D) DELETE; 204

Answer: A

Explanation:

The endpoint is defined using app.get(), meaning it handles GET requests. It returns an HTTP status code 200 (OK) along with JSON data containing the list of books.

Q34. A C programmer writes a program to dynamically allocate an array, fill it with values, and compute their sum. Consider the code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int *arr = malloc(5 * sizeof(int));
if (arr == NULL) {
return 1;
}
for (int i = 0; i < 5; i++) {
arr[i] = (i + 1) * 10;
}
int sum = 0;
for (int i = 0; i < 5; i++) {
sum += *(arr + i);
}
printf("%d\n", sum);
free(arr);
return 0;
}
```

What is the output of this code, and why is pointer arithmetic used effectively here?

A) 150; pointer arithmetic allows efficient access to array elements.
B) 150; pointer arithmetic is used to compute the size of the array.
C) 120; the loop iterates incorrectly.
D) 200; the memory is not freed properly.

Answer: A

Explanation:

The array holds the values [10, 20, 30, 40, 50], whose sum is 150. The expression *(arr + i) accesses the i-th element of the array using pointer arithmetic, which is both efficient and equivalent to arr[i].

Q35. A PHP developer must securely query a MySQL database to retrieve user details based on their username. Consider the following code snippet:

```
<?php
$username = $_POST['username'];
$stmt = $conn->prepare("SELECT * 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 primary security advantage of this code snippet?

A) It encrypts the entire database.

B) It uses prepared statements to prevent SQL injection attacks.

C) It automatically logs user data for analysis.

D) It prevents users from accessing the database.

Answer: B

Explanation:

The code uses prepared statements with parameter binding, ensuring that user input is treated as data and not executable SQL code. This prevents SQL injection, a major security vulnerability.

Q36. A company wants to analyze employee performance by ranking employees within each department based on their performance score. Consider the SQL query:

```
SELECT Department, Name, Score,
RANK() OVER (PARTITION BY Department ORDER BY Score DESC) AS DeptRank
FROM EmployeePerformance;
```

What does this query do, and how does the PARTITION BY clause affect the ranking?

A) It ranks all employees globally regardless of department.

B) It ranks employees within each department based on their score in descending order.

C) It groups employees by department but does not rank them.

D) It updates the performance scores with ranking values.

Answer: B

Explanation:

The query uses the window function RANK() to assign a rank to each employee within their department, as defined by PARTITION BY Department. The highest score in each department gets rank 1, and so on, based on descending order of Score.

Q37. A Swift developer is building an app for drawing shapes. The developer defines a protocol Drawable that requires a method draw(). Classes such as Circle and Rectangle conform to this protocol. Consider the code:

```
protocol Drawable {
func draw() -> String
}
class Circle: Drawable {
func draw() -> String {
return "Drawing a Circle"
}
}
class Rectangle: Drawable {
func draw() -> String {
return "Drawing a Rectangle"
}
}
let shapes: [Drawable] = [Circle(), Rectangle()]
for shape in shapes {
print(shape.draw())
}
```

What is the output, and why is protocol-oriented programming beneficial in this scenario?

A) It prints "Drawing a Circle" only; protocols restrict other classes.
B) It prints "Drawing a Circle" and "Drawing a Rectangle"; protocols allow different classes to implement the same method signature.
C) It prints nothing; protocols do not work with arrays.
D) It causes a runtime error because protocols cannot be used with classes.

Answer: B

Explanation:

The array shapes contains instances of Circle and Rectangle, both conforming to the Drawable protocol. When iterating, each instance calls its own implementation of draw(), resulting in "Drawing a Circle" and "Drawing a Rectangle". Protocols promote code reusability and flexibility by defining common behavior that different types can implement.

Q38. A data scientist needs to convert a list of temperatures from Celsius to Fahrenheit using a concise approach. Consider the code:

```
temps_c = [0, 20, 30, 100]
temps_f = list(map(lambda c: (c * 9/5) + 32, temps_c))
print(temps_f)
```

What is the output, and why is the combination of lambda and map useful in this case?
A) [32.0, 68.0, 86.0, 212.0]; It provides a concise, functional way to transform each element of the list.
B) [32, 68, 86, 212]; It rounds the values automatically.
C) [0, 20, 30, 100]; It makes no transformation.
D) [32.0, 50.0, 70.0, 100.0]; It incorrectly converts temperatures.

Answer: A

Explanation:

The lambda function converts Celsius to Fahrenheit using the formula (c * 9/5) + 32. Applying this with map to each element of temps_c yields [32.0, 68.0, 86.0, 212.0]. This combination is both concise and expressive for element-wise transformations.

Q39. A developer is building a secure web application using Node.js. To manage user sessions, the developer decides to use JSON Web Tokens (JWT). Which of the following best describes a key advantage of using JWT for authentication?

A) JWTs store all user session data on the server, reducing client-side load.
B) JWTs allow stateless authentication, meaning the server does not need to store session information.
C) JWTs encrypt the user's password permanently.
D) JWTs are only used for logging out users.

Answer: B

Explanation:

JWTs allow stateless authentication by embedding user claims within the token itself. This means that the server does not need to maintain session state in memory or a database, which improves scalability and simplifies session management. Options A, C, and D do not accurately reflect the core benefit of JWT.

Q40. A database administrator is tasked with optimizing a complex query that joins multiple large tables. The query is as follows:

```
SELECT a.id, a.name, b.score
FROM Students a
JOIN Scores b ON a.id = b.student_id
WHERE b.score > 80
ORDER BY b.score DESC;
```

Which strategy is most effective for improving the performance of this query?
A) Adding an index on the Students.name column

B) Creating a composite index on Scores(score, student_id)
C) Using a full table scan for both tables
D) Removing the WHERE clause

Answer: B

Explanation:

A composite index on Scores(score, student_id) helps optimize the join on student_id and efficiently filter records where score > 80. This strategy minimizes scanning large tables and improves query performance. Options A, C, and D do not target the critical join and filter conditions.

Section 4: Achievers Section - Debugging and Error Analysis

Q41. A developer writes the following Python code to compute the square of each number in a list, but only if a condition is met. However, the code produces an error.

```
numbers = [1, 2, 3, 4, 5]
squared = [x**2 for x in numbers if y > 2]
print(squared)
```

What is the error in the code?

A) The variable x is not defined in the condition.
B) The variable y is not defined.
C) The syntax for list comprehension is incorrect.
D) The operator ** is invalid.

Answer: B

Explanation:

In the list comprehension, the condition if y > 2 refers to an undefined variable y. The intended condition might have been if x > 2. Thus, the error is due to using an undefined variable.

Q42. A developer writes a function to append a value to a list. However, when calling the function multiple times, the output is unexpected.

```
def append_item(val, my_list=[]):
my_list.append(val)
return my_list
print(append_item(1))
print(append_item(2))
```

What is the issue with this function?

A) The function does not return a list.
B) The default list is re-used across function calls, causing accumulation.
C) The list is cleared automatically on each call.
D) There is a syntax error in the function definition.

Answer: B

Explanation:

Using a mutable default argument (my_list=[]) causes the same list to be used across multiple function calls. As a result, values accumulate, and the output becomes [1] and then [1, 2] instead of starting fresh each time. The fix is to use None as the default and create a new list inside the function if needed.

Q43. Consider the following C code snippet:

```
#include <stdio.h>
int main() {
```

```
int arr[] = {10, 20, 30};
int *ptr = arr;
printf("%d\n", ptr[3]);
return 0;
}
```

What is the error in this code?

A) The array is not declared properly.

B) The pointer arithmetic is incorrect.

C) Accessing ptr[3] is out-of-bounds since the array has only three elements.

D) The program will output 0.

Answer: C

Explanation:

Arrays in C are zero-indexed. With an array of 3 elements (indices 0, 1, 2), accessing ptr[3] refers to a non-existent fourth element, resulting in undefined behavior (often a runtime error).

Q44. A C programmer writes code to allocate memory for an array and fill it with values. Examine the following code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int *arr = malloc(5 * sizeof(int));
for (int i = 0; i <= 5; i++) {
arr[i] = i * 2;
}
printf("%d\n", arr[5]);
free(arr);
return 0;
}
```

What is the error in this code?

A) The loop condition should be i < 5 instead of i <= 5, causing out-of-bound access.

B) The code uses malloc incorrectly.

C) The memory is not freed properly.

D) There is a syntax error in the for-loop declaration.

Answer: A

Explanation:

The loop runs with i <= 5, which iterates 6 times (i from 0 to 5) for an array allocated for 5 elements. This causes out-of-bound access when i is 5. The correct loop condition should be i < 5.

Q45. A database query joins two tables, but the query produces an ambiguous column error:

```
SELECT name, salary
FROM employees, departments
WHERE employees.dept_id = departments.dept_id
AND dept_id = 10;
```

What is the error, and how can it be fixed?

A) The column dept_id is ambiguous; qualify it by specifying the table name (e.g., employees.dept_id).

B) The query is missing a GROUP BY clause.

C) The query should use a JOIN keyword.

D) There is a syntax error in the SELECT statement.

Answer: A

Explanation:

Since both tables have a column named dept_id, using dept_id in the WHERE clause is ambiguous. Qualify it by specifying either employees.dept_id or departments.dept_id to resolve the ambiguity.

Q46. A PHP script intended to print a user's name is written as follows:

```
<?php
$name = "Alice";
echo $Name;
?>
```

What is the error in the code?

A) PHP variables are case-sensitive; $Name does not match $name.
B) The echo statement is incorrect.
C) The variable should be defined as $Name from the start.
D) There is no error; it prints "Alice".

Answer: A

Explanation:

PHP variable names are case-sensitive. The variable is defined as $name, but the script attempts to echo $Name, which is undefined. The correct usage is to echo $name.

Q47. A Swift developer writes code that force unwraps an optional value without ensuring it is not nil:

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

What is the error, and how can it be corrected?

A) There is no error; the code prints "nil".
B) Force unwrapping a nil optional causes a runtime crash; use optional binding (e.g., if let) to safely unwrap.
C) The variable should not be declared as optional.
D) The exclamation mark is used incorrectly; it should be removed.

Answer: B

Explanation:

Force unwrapping (!) an optional that is nil will cause a runtime crash. The correct approach is to use optional binding (e.g., if let unwrapped = greeting { ... }) to safely handle nil values.

Q48. A developer intends to create a generator to yield numbers from 0 to n-1. However, the following code does not behave as expected:

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

What is the error in this code?

A) The code correctly creates a generator.
B) Using return inside the loop causes the function to exit after the first iteration; it should use yield.
C) The loop should iterate from 1 to n.
D) The function should not convert the result to a list.

Answer: B

Explanation:

Using return inside the loop returns only the first value and exits the function. To create a generator that yields multiple values, the function should use yield instead of return.

Q49. A Node.js function is meant to fetch data asynchronously but does not call its callback:

```
function fetchData(callback) {
setTimeout(() => {
let data = "Hello, World!";
// Missing callback(data);
}, 1000);
}
fetchData((data) => {
console.log(data);
});
```

What is the error in the code?

A) The function fetchData never calls the callback, so nothing is logged.
B) The setTimeout delay is too short.
C) The callback is called incorrectly.
D) The function should return data instead of using a callback.

Answer: A

Explanation:

The error is that the callback function is never invoked inside the setTimeout. To fix this, the developer should call callback(data) inside the setTimeout block to pass the data to the callback.

Q50. A developer writes an SQL query intended to retrieve the maximum salary and the average salary for each department:

```
SELECT department, MAX(salary), AVG(salary)
FROM employees
WHERE salary > 50000;
```

However, this query fails to execute correctly. What is the error, and how should it be corrected?

A) The query is missing a GROUP BY clause to aggregate by department.
B) The WHERE clause should be replaced with HAVING.
C) The MAX function is used incorrectly.
D) The query should not use AVG with MAX.

Answer: A

Explanation:

When using aggregate functions (MAX and AVG) with a column that is not aggregated (department), the query must include a GROUP BY clause. The corrected query should be:

```
SELECT department, MAX(salary), AVG(salary)
FROM employees
WHERE salary > 50000
GROUP BY department;
```

This groups the results by department before applying the aggregate functions.

School Connect Online

Answer Key

1	2	3	4	5	6	7	8	9	10
Q1: A	Q2: A	Q3: B	Q4: B	Q5: B	Q6: B	Q7: C	Q8: C	Q9: B	Q10: A
Q11: C	Q12: C	Q13: D	Q14: A	Q15: C	Q16: A	Q17: B	Q18: C	Q19: B	Q20: C
Q21: B	Q22: A	Q23: B	Q24: A	Q25: B	Q26: B	Q27: A	Q28: B	Q29: B	Q30: B
Q31: B	Q32: A	Q33: A	Q34: A	Q35: B	Q36: B	Q37: B	Q38: A	Q39: B	Q40: B
Q41: B	Q42: B	Q43: C	Q44: A	Q45: A	Q46: A	Q47: B	Q48: B	Q49: A	Q50: A

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