School Connect Online

SCO INTERNATIONAL CODING OLYMPIAD

CLASS 7 OFFICIAL QUESTION PAPER

SCO International Coding Olympiad | Question Paper Set A

Designed for Class 7 learners to build confident coding foundations through C programming, XML data thinking, Turtle graphics, and game-development logic.

- compact inline question numbers with no large question-number block
- clean question blocks with code, options, answer key, and explanations together
- global alignment with K-12 computer science, computational thinking, and safe coding practices

C Basics	XML	Turtle	Game Logic	Algorithms
Functions	Data	Events	Safety	Review

SCO International Coding Olympiad- Class 7 Question Paper Set A

Field	Official Detail
Exam Name	SCO International Coding Olympiad
Class / Grade	Class 7
Duration	60 minutes
Type of Exam	Objective type multiple-choice questions
Number of Questions	50
Answer Format	One correct option per question
Calculator	Not allowed unless specifically announced by SCO
Website Use	PDF-ready student/teacher download

Candidate guidance: Read each question carefully, select only one answer, and use the explanation section for post-exam learning and teacher-supported review. The question number is intentionally shown as a compact inline label to keep the question, code, passage, options, answer, and explanation together in one academic block.

Section Blueprint

Section	Focus Area	Learning Emphasis
Section 1	Programming Basics	C syntax, variables, input/output, arrays, XML foundations and Turtle movement
Section 2	Advanced Coding Concepts	Debugging, functions, recursion, memory safety, data formats and event logic
Section 3	Game Development Basics	Game loops, Turtle graphics, XML configuration, user input and simple simulation
Achievers Section	Higher-order reasoning	Error analysis, real-world debugging and integrated coding scenarios

Section 1: Programming Basics

Q1. Consider the following C code intended to read two integers and print their sum:

```
#include <stdio.h>
int main() {
int a, b;
printf("Enter two numbers: ");
scanf("%d %d", a, b);
printf("Sum = %d\n", a + b);
return 0;
}
```

What is the error in this code?

A) The format specifiers are incorrect.

B) The scanf function is missing the address-of operator (&).

C) The variables a and b are not declared.

D) There is no error; the code works correctly.

Answer: B

Explanation: In C, scanf() needs the memory addresses of variables so that it can store the input values. The correct statement is scanf("%d %d", &a, &b);.

Q2. Examine the following code:

```
#include <stdio.h>
int main() {
for (int i = 0; i < 5; i++); {
printf("Hello ");
}
return 0;
}
```

What error does this code contain?

A) The semicolon after the for-loop causes the loop body to be empty, and the block following it executes only once.

B) The loop condition is incorrect.

C) The printf statement is missing a newline.

D) There is no error; it prints "Hello " five times.

Answer: A

Explanation: The semicolon (;) immediately after the for loop terminates the loop. As a result, the following block is not part of the loop and executes just once, printing "Hello " a single time.

Q3. Consider the code below:

```
#include <stdio.h>
int main() {
int arr[3] = {10, 20, 30};
printf("%d", arr[3]);
return 0;
}
```

What error occurs in this code?

A) Syntax error in array declaration.

B) Out-of-bounds access (valid indices are 0–2).

C) Incorrect use of the printf format specifier.

D) There is no error; it prints the last element.

Answer: B

Explanation: The array arr has 3 elements, with valid indices 0, 1, and 2. Accessing arr[3] is out-of-bounds, leading to undefined behavior.

Q4. Consider this code:

```
#include <stdio.h>
int main() {
int *p;
*p = 20;
printf("%d", *p);
return 0;
}
```

What is the error here?

A) The pointer p is used without initialization.
B) The printf statement has an incorrect format specifier.
C) The pointer arithmetic is wrong.
D) There is no error; the code prints 20.

Answer: A

Explanation: The pointer p is declared but not initialized to point to a valid memory location. Dereferencing an uninitialized pointer results in undefined behavior.

Q5. Review the following code:

```
#include <stdio.h>
int main() {
a = 5;
printf("%d", a);
return 0;
}
```

What is the error?

A) The variable a is not declared before use.
B) The printf function is used incorrectly.
C) The main function is declared wrongly.
D) There is no error; it works correctly.

Answer: A

Explanation: In C, all variables must be declared before they are used. Here, a is used without a declaration, causing a compilation error.

Q6. A teacher writes a program to compute the average of 5 test scores:

```
#include <stdio.h>
int main() {
int scores[5] = {80, 90, 85, 70, 95};
int total = 0;
for (int i = 0; i <= 5; i++) {
total += scores[i];
}
printf("Average = %d", total / 5);
return 0;
}
```

What is the error?

A) The loop condition should be i < 5 instead of i <= 5.
B) The total is not initialized correctly.
C) The division operator is used incorrectly.

D) There is no error.

Answer: A

Explanation: The loop iterates one time too many (i = 5 is invalid for an array of size 5). The loop condition should be i < 5.

Q7. A student writes the following code to swap two integers:

```
#include <stdio.h>
void swap(int *a, int *b) {
int temp = *a;
*a = *b;
*b = temp;
}
int main() {
int x = 10, y = 20;
swap(x, y);
printf("x = %d, y = %d", x, y);
return 0;
}
```

What is the error?

A) The swap function is written correctly.

B) The function call should pass addresses (e.g., &x, &y).

C) The function should return a value.

D) The printf statement is missing a newline.

Answer: B

Explanation: The swap function expects pointers, so the call should be swap(&x, &y). Passing x and y directly does not provide their addresses, causing the function to operate on incorrect values.

Q8. Review the following code:

```
#include <stdio.h>
int main() {
int num;
scanf("%f", &num);
printf("%d", num);
return 0;
}
```

What is the error?

A) The format specifier in scanf is incorrect for an integer.

B) The printf format specifier is wrong.

C) The variable num is not declared.

D) There is no error.

Answer: A

Explanation: The format specifier %f is used for floats. For an integer, %d should be used in scanf. The correct code is: c scanf("%d", &num);

Q9. Which of the following XML snippets is well-formed?

A)

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
```

```
<body>Don't forget me this weekend!
</note>
```

B)

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

C)

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</from>
</note>
```

D)

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
</note>
```

Answer: B

Explanation: Option B is well-formed because every opening tag has a corresponding closing tag, and the structure is properly nested. Options A and C have missing or mismatched tags, and D is incomplete.

Q10. Which of the following correctly defines an XML element with an attribute?

A)

```
<book title="XML Basics"></book>
```

B)

```
<book title=XML Basics></book>
```

C)

```
<book title='XML Basics>
</book>
```

D)

```
<book "title"="XML Basics"></book>
```

Answer: A

Explanation: Option A correctly declares an XML element with an attribute. The attribute value is enclosed in quotes. The other options have syntax errors.

Q11. Which of the following characters must be escaped in XML content?

A) <

B) >

C) &

D) All of the above

Answer: D

Explanation: In XML, the characters <, >, and & (and quotes in attributes) must be escaped to prevent them from being interpreted as markup. For example, < becomes <.

Q12. Consider this XML snippet:

```
<parent>
<child>Content</parent>
</child>
```

What is the error in this XML?

A) The elements are not properly nested.
B) The content is missing.
C) The tags are correct; there is no error.
D) The XML declaration is missing.

Answer: A

Explanation: The XML elements are incorrectly nested; the <child> element is closed after the <parent> element, which violates proper XML structure.

Q13. Which of the following is a correct self-closing tag in XML?

A) <line />
B) <line></line>
C) <line>
D) <line / >

Answer: A

Explanation: A self-closing tag in XML is written as <line /> with no space between the slash and the closing angle bracket. Option B is valid but not self-closing; C and D are incorrect.

Q14. Given an XML file with the following snippet:

```
<items>
<item id="1">Apple</item>
<item id="2">Banana</item>
<item id="3">Cherry
</items>
```

What is the error? A) The item with id "3" is missing a closing tag.
B) The items tag is not closed.
C) The attribute values are not enclosed in quotes.
D) There is no error.

Answer: A

Explanation: The third <item> element ("Cherry") does not have a closing </item> tag, which makes the XML not well-formed.

Q15. What does the following code do?

```
import turtle
t = turtle.Turtle()
t.forward(100)
```

A) Moves the turtle forward by 100 pixels.
B) Rotates the turtle by 100 degrees.
C) Draws a circle with a radius of 100 pixels.
D) Clears the drawing canvas.

Answer: A

Explanation: The forward(100) method moves the turtle forward 100 pixels in the direction it is currently facing.

Q16. Consider the code:

```
import turtle
t = turtle.Turtle()
```

```
t.move(50)
```

What is the error?

A) The module turtle is not imported correctly.

B) The method name is incorrect; it should be forward instead of move.

C) The code prints an error due to syntax mistakes.

D) There is no error.

Answer: B

Explanation: In the Turtle module, the correct method to move forward is forward(). There is no method named move().

Q17. A student wants to draw a square using turtle graphics. Consider the following code:

```
import turtle
t = turtle.Turtle()
for i in range(4):
t.forward(100)
t.left(45)
turtle.done()
```

What is the error, and what will the code draw?

A) It correctly draws a square.

B) The turning angle is incorrect; it would not form a square.

C) The loop should run 5 times to draw a square.

D) The forward command is invalid.

Answer: B

Explanation: A square requires a 90 degree turn after each side. A 45 degree turn changes the direction incorrectly, so the shape does not close as a square.

Q18. Consider this code:

```
import turtle
t = turtle.Turtle()
t.forward(50)
```

What potential issue might arise when running this code as a standalone script?

A) The turtle does not move.

B) The window may close immediately after executing because turtle.done() is missing.

C) The code throws a syntax error.

D) The turtle draws a circle instead.

Answer: B

Explanation: Without calling turtle.done(), the turtle graphics window may open and then immediately close after the script finishes. This prevents the user from seeing the drawing.

Q19. Consider the following code intended to draw two separate horizontal lines without a connecting line:

```
import turtle
t = turtle.Turtle()
t.penup()
t.goto(0, 0)
t.pendown()
t.forward(100)
t.penup()
t.goto(0, 50)
t.forward(100)
turtle.done()
```

What is the error in this code?

A) No error; it draws two separate lines.

B) The second line is not drawn because pendown() was not called again after moving to the new position.

C) The turtle does not move to the new starting point.

D) The code has a syntax error.

Answer: B

Explanation: After the turtle moves to the second starting point with penup(), the pen is still lifted. The program should call pendown() before t.forward(100) to draw the second line.

Q20. Consider the following code snippet:

```
import turtle
t = turtle.Turtle()
t.drawcircle(50)
t.done()
```

What is the error in this code?

A) The method name is incorrect; it should be circle() instead of drawcircle().

B) The turtle module is imported incorrectly.

C) The argument 50 is invalid.

D) There is no error; it draws a circle.

Answer: A

Explanation: The correct method to draw a circle in turtle is circle(50). The method drawcircle() does not exist, leading to an AttributeError.

Q21. Scenario: A developer writes a recursive function to compute the sum of numbers from 1 to n but

```
total = 0
def sum_recursive(n):
```

global total

```
if n == 0:
return total
```

total += n

```
return sum_recursive(n - 1)
```

print(sum_recursive(5))

print(sum_recursive(3))

What is the error in this code?

A) The function doesn't use recursion correctly.

B) The global variable total is not reset between function calls, causing cumulative results.

C) The base case is incorrect.

D) There is no error; it works as intended.

Answer: B

Explanation: Since total is declared globally and not reset, the second call to sum_recursive(3) continues accumulating from the previous call. This causes incorrect results. A proper fix is to initialize total within the function or avoid using a global variable.

Q22. Scenario: A programmer intends to create a generator that yields numbers from 0 to n-1 but mistakenly uses return inside the loop:

```
def count_numbers(n):
for i in range(n):
return i  # Incorrect: should yield i
```

print(list(count_numbers(5)))

What will the output be, and what is the error?

A) It outputs [0, 1, 2, 3, 4]; the code is correct.
B) It outputs [0]; using return stops the function at the first iteration.
C) It outputs []; no values are yielded.
D) It causes a syntax error.

Answer: B

Explanation: The use of return inside the loop causes the function to exit on the first iteration. To yield multiple values, the code should use yield instead of return.

Q23. A C programmer allocates memory for an integer array and moves a pointer inside that allocated block:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int *arr = malloc(5 * sizeof(int));
int *ptr;
ptr = arr + 3;
printf("%d\n", *ptr);
free(arr);
return 0;
}
```

What is the main problem in this code?

A) The pointer arithmetic ptr = arr + 3 is always invalid.
B) The program reads an uninitialized array element, so the printed value is indeterminate.
C) The free() function is used on the wrong pointer.
D) malloc() cannot allocate arrays.

Answer: B

Explanation: The pointer arithmetic points to the fourth integer in the allocated array, which is within bounds. However, no values were assigned to the allocated elements before reading *ptr, so the output is indeterminate.

Q24. A developer writes overloaded functions in C++:

```
#include <iostream>
using namespace std;
void show(double x) { cout << "Double: " << x << endl; }
void show(int x) { cout << "Int: " << x << endl; }
int main() {
show(5.0f);
return 0;
}
```

What issue might arise from this code?

A) It will choose show(int) due to ambiguity.
B) It will choose show(double) because a float is promoted to double.
C) It results in a compile-time error due to ambiguity between overloads.
D) It prints nothing due to incorrect overloading.

Answer: B

Explanation: A float (5.0f) is automatically promoted to double, so show(double) is called. Although overload resolution can be tricky, in this case the conversion is unambiguous.

Q25. A Swift developer writes a function to perform an asynchronous task using closures. The code may crash due to improper optional handling:

```
class Downloader {
var completion: (() -> Void)?
```

```
   func startDownload() {
   DispatchQueue.global().async {
sleep(1)
self.completion!()
   }
   }
   }
   let downloader = Downloader()
   downloader.completion = { print("Download complete") }
downloader.startDownload()
```

What is the safest correction?

A) The code is always safe.

B) Force unwrapping completion may crash if it is nil; use optional binding or optional chaining.

C) sleep(1) is not allowed in Swift.

D) The closure should always capture self strongly.

Answer: B

Explanation: Using self.completion!() force unwraps an optional. If completion becomes nil before the closure runs, the program crashes. Safer code is self.completion?() or optional binding.

Q26. A PHP developer writes a query to fetch user data but forgets to escape user input:

```
   <?php
   $username = $_GET['username'];
   $query = "SELECT * FROM users WHERE username = '$username'";
   $result = mysqli_query($conn, $query);
?>
```

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

A) The query is syntactically incorrect.

B) It is vulnerable to SQL injection; use prepared statements to fix it.

C) The $_GET variable is not allowed.

D) There is no error; it works correctly.

Answer: B

Explanation: Directly embedding user input in SQL queries without escaping or parameterization makes the query vulnerable to SQL injection attacks. The fix is to use prepared statements with parameter binding.

Q27. Consider the following query intended to retrieve the average salary by department:

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

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

A) The query is correct as written.

B) It is missing a GROUP BY clause for the non-aggregated column department.

C) The WHERE clause should be replaced with HAVING.

D) The AVG function is misused.

Answer: B

Explanation: When using an aggregate function with a non-aggregated column (department), a GROUP BY clause is required. The corrected query is: SELECT department, AVG(salary) FROM employees WHERE salary > 50000 GROUP BY department;

Q28. A programmer writes a function that sums all provided arguments:

```
   def sum_all(*args):
```

```
    total = 0
    for i in args:
total += i
    return total
print(sum_all(1, 2, "3", 4))
```

What error occurs, and why?

A) The function returns the sum as a string.

B) A TypeError occurs because "3" is a string and cannot be added to an integer.

C) The function does not accept multiple arguments.

D) There is no error; the function converts "3" to an integer.

Answer: B

Explanation: The function attempts to add an integer and a string ("3"), which results in a TypeError in Python. All arguments should be numeric (or explicitly converted) before summation.

Q29. An Objective-C developer writes:

```
@interface Car : NSObject
    @property (nonatomic, strong) NSString *model;
@end
@implementation Car
@end
    int main() {
    @autoreleasepool {
    Car *car = [Car alloc]; // Missing init
    car.model = @"Toyota";
    NSLog(@"%@", car.model);
    }
    return 0;
    }
```

What is the error in this code?

A) The property is declared incorrectly.

B) The object is not properly initialized (missing init).

C) NSLog is used incorrectly.

D) There is no error; the code works.

Answer: B

Explanation: In Objective-C, you must allocate and initialize an object using [Car alloc] init]. Failing to call init leads to undefined behavior.

Q30. A Kotlin developer writes a loop to print numbers from 1 to 10 but uses an incorrect range operator:

```
    for (i in 1...10) {
print(i)
    }
```

What is the error, and what is the correct code?

A) The range operator should be .. instead of ...; correct code: for (i in 1..10) { print(i) }.

B) The loop syntax is entirely incorrect.

C) Kotlin does not support loops.

D) There is no error; the code works.

Answer: A

Explanation: Kotlin uses .. to define a closed range. The operator ... is not valid in Kotlin, so the loop should be written as: kotlin for (i in 1..10) { print(i) }

Q31. What will be the output of the following C program?

```
#include <stdio.h>
int main() {
int x = 5;
int *ptr = &x;
printf("%d", *ptr + 2);
return 0;
}
```

A) 5
B) 7
C) 2
D) Compilation Error

Answer: B) 7

Explanation: The pointer ptr stores the address of x, and *ptr retrieves its value (5). Adding 2 results in 7.

Q32. Which of the following XML statements is incorrect?

A) <student><name>John</name></student>
B) <book><title>XML Guide</title></book>
C) <author><name>Mark</author></name>
D) <data></data>

Answer: C) <author><name>Mark</author></name>

Explanation: XML tags must be properly nested. Here, <name> is closed after </author>, which is incorrect.

Q33. What will happen if you run the following Python Turtle script?

```
import turtle
t = turtle.Turtle()
t.forward(100)
t.right(90)
t.forward(100)
```

A) Draws a straight line
B) Draws an L-shape
C) Draws a square
D) Throws an error

Answer: B) Draws an L-shape

Explanation: The turtle moves forward 100 units, turns 90 degrees right, and moves forward another 100 units.

Q34. Identify the error in the following C code:

```
#include <stdio.h>
int main() {
int arr[5];
arr[5] = 10;
printf("%d", arr[5]);
return 0;
}
```

A) No error
B) Array index out of bounds
C) Syntax error
D) Memory leak

Answer: B) Array index out of bounds

Explanation: The array arr[5] has valid indices from 0 to 4. Accessing arr[5] leads to undefined behavior.

Q35. Which of the following is not a correct feature of XML?

A) XML is case-sensitive
B) XML is used to store and transport data
C) XML tags must be predefined
D) XML supports self-descriptive structure

Answer: C) XML tags must be predefined

Explanation: Unlike HTML, XML does not have predefined tags; users define their own.

Q36. What will be the output of the following Python Turtle command?

```
import turtle
t = turtle.Turtle()
t.circle(50, 180)
```

A) A full circle
B) A semicircle
C) A square
D) An error

Answer: B) A semicircle

Explanation: t.circle(50, 180) makes the turtle draw a semicircle with radius 50.

Q37. What will happen if the following C program runs?

```
#include <stdio.h>
int main() {
char str[5] = "hello";
printf("%s", str);
return 0;
}
```

A) Prints "hello"
B) Compilation Error
C) Prints "hell"
D) Causes memory overflow

Answer: B

Explanation: The string literal "hello" needs space for five characters plus the null terminator. char str[5] is too small for a valid C string, so a conforming implementation should diagnose the initializer as too long; use char str[6] = "hello";.

Q38. In XML, what is the correct way to define an attribute?

A) <student name=John></student>
B) <student name="John"></student>
C) <student><name>John</name></student>
D) <student>name=John</student>

Answer: B) <student name="John"></student>

Explanation: Attributes in XML must be enclosed in double quotes.

Q39. Identify the error in the following Python Turtle program:

```
import turtle
t = turtle.Turtle()
t.goto(50,50)
t.penup()
```

```
t.goto(100,100)
t.pendown()
t.goto(150,150)
t.goto(50,50)
t.clear()
print(t.position())
```

A) No error
B) t.clear() should be before t.goto(50,50)
C) t.penup() and t.pendown() are incorrectly used
D) t.goto(50,50) is incorrect

Answer: A) No error

Explanation: The commands execute correctly; clear() removes all drawings, but print(t.position()) still works.

Q40. What will be the output of this C program?

```
#include <stdio.h>
void printMessage() {
printf("Hello, World!");
}
int main() {
printMessage();
return 0;
}
```

A) Compilation Error
B) Hello, World!
C) No output
D) Runtime Error

Answer: B) Hello, World!

Explanation: The function printMessage() is correctly defined and called in main(), so "Hello, World!" prints.

Achievers Section

Q41. A C program uses a switch statement to display a menu result:

```
#include <stdio.h>
int main() {
int choice = 2;
switch(choice) {
case 1: printf("New"); break;
case 2: printf("Open");
case 3: printf("Save"); break;
default: printf("Exit");
}
return 0;
}
```

What is the output and why?

A) Open
B) OpenSave
C) Save
D) Exit

Answer: B

Explanation: The program matches case 2 and prints Open. Because there is no break after case 2, execution falls through to case 3 and prints Save.

Q42. A student stores a game level in XML:

```
<level id="1">
<enemy type="ghost">
<speed>5</speed>
</level>
```

Which problem makes this XML not well-formed?

A) The level tag has an attribute.
B) The enemy element is not closed before the level element closes.
C) The speed value is numeric.
D) XML cannot store game data.

Answer: B

Explanation: XML elements must be properly nested. The <enemy> tag must be closed with </enemy> before </level>.

Q43. In Python Turtle, a student wants to make the turtle respond when the Up arrow is pressed. Which command correctly binds a keypress to a function named move_up using a Screen object named screen?

A) turtle.keypress("Up", move_up)
B) screen.onkeypress(move_up, "Up")
C) screen.press(move_up, "Up")
D) t.whenkey("Up", move_up)

Answer: B

Explanation: The Screen object supports onkeypress(function, key) to bind a keyboard event to a function. The program should also call screen.listen() to start listening for key events.

Q44. A C function is written as:

```
int area(int length, int width) {
return length * width;
}
```

Which statement correctly describes this function?

A) It returns the perimeter of a rectangle.
B) It returns an integer product representing rectangular area.
C) It cannot accept two parameters.
D) It must always return a floating-point value.

Answer: B

Explanation: The function multiplies length and width and returns an int value, which represents the area of a rectangle when both inputs are integer dimensions.

Q45. A program saves player information as:

```
<player name="Ava">
<score>120</score>
</player>
```

In this XML snippet, what is name="Ava"?

A) A child element
B) An attribute of the player element
C) A closing tag
D) A comment

Answer: B

Explanation: An XML attribute appears inside the opening tag and gives extra information about an element. Here, name="Ava" describes the player element.

Q46. A Turtle program draws a triangle using this loop:

```
for i in range(3):
t.forward(80)
t.left(120)
```

What mathematical idea is used in choosing 120 degrees?

A) A triangle has 120 degree interior angles.
B) The exterior turn for an equilateral triangle is 360 / 3 = 120 degrees.
C) Turtle always turns 120 degrees by default.
D) The angle is chosen randomly.

Answer: B

Explanation: To draw a regular polygon with turtle graphics, the turn angle is commonly 360 divided by the number of sides. For a triangle, 360 / 3 = 120 degrees.

Q47. A C program contains:

```
int values[4] = {2, 4, 6, 8};
int *p = values;
printf("%d", *(p + 2));
```

What will be printed?

A) 2
B) 4
C) 6
D) 8

Answer: C

Explanation: p points to the first element. p + 2 points to the third element, values[2], which stores 6.

Q48. Which order best describes the main cycle of a simple game loop?

A) Save database -> exit -> render -> input
B) Read input -> update game state -> render graphics
C) Render graphics -> shut down -> read input
D) Compile program -> write XML -> stop game

Answer: B

Explanation: A game loop generally reads input, updates the game state according to rules and physics, and then renders the updated frame to the screen.

Q49. A team wants game designers to change enemy speed and level name without editing the C source code. Which storage approach is most suitable?

A) Hardcode all values in the C program.
B) Store configuration values in an external XML file.
C) Delete the level settings after each run.
D) Use only printed instructions.

Answer: B

Explanation: External XML configuration separates data from code. Designers can update values in the XML file without recompiling the program.

Q50. A student writes this XML comment:

// This level is difficult

Why is it incorrect?

A) XML comments must use <!-- comment text -->.

B) XML does not allow any comments.
C) Comments must be written in Python syntax.
D) The comment must be placed after the root element.

Answer: A

Explanation: XML comments use the syntax <!-- comment text -->. The // form is used in languages such as C, C++, JavaScript, and Java, not XML.

Compact Answer Key

Q.No	Answer	Section
1	B	Section 1: Programming Basics
2	A	Section 1: Programming Basics
3	B	Section 1: Programming Basics
4	A	Section 1: Programming Basics
5	A	Section 1: Programming Basics
6	A	Section 1: Programming Basics
7	B	Section 1: Programming Basics
8	A	Section 1: Programming Basics
9	B	Section 1: Programming Basics
10	A	Section 1: Programming Basics
11	D	Section 1: Programming Basics
12	A	Section 1: Programming Basics
13	A	Section 1: Programming Basics
14	A	Section 1: Programming Basics
15	A	Section 1: Programming Basics
16	B	Section 1: Programming Basics
17	B	Section 1: Programming Basics
18	B	Section 1: Programming Basics
19	B	Section 1: Programming Basics
20	A	Section 1: Programming Basics
21	B	Section 1: Programming Basics
22	B	Section 1: Programming Basics
23	B	Section 1: Programming Basics
24	B	Section 1: Programming Basics
25	B	Section 1: Programming Basics
26	B	Section 1: Programming Basics
27	B	Section 1: Programming Basics
28	B	Section 1: Programming Basics
29	B	Section 1: Programming Basics
30	A	Section 1: Programming Basics
31	B	Section 1: Programming Basics
32	C	Section 1: Programming Basics
33	B	Section 1: Programming Basics
34	B	Section 1: Programming Basics
35	C	Section 1: Programming Basics
36	B	Section 1: Programming Basics
37	B	Section 1: Programming Basics
38	B	Section 1: Programming Basics
39	A	Section 1: Programming Basics
40	B	Section 1: Programming Basics
41	B	Achievers Section
42	B	Achievers Section

43	B	Achievers Section
44	B	Achievers Section
45	B	Achievers Section
46	B	Achievers Section
47	C	Achievers Section
48	B	Achievers Section
49	B	Achievers Section
50	A	Achievers Section