ES6 - Adv. Javascript MCQ's for Quiz :

Topic 1 – var,let & const:

Q1: What is the main difference between var, let, and const?
a) All are block scoped
b) var is block scoped, let is function scoped
c) var is function scoped; let and const are block scoped
d) All are function scoped

Q2: What happens if you redeclare a var variable in the same scope?
a) Syntax error
b) TypeError
c) It's allowed
d) It's converted to const

Q3: Which of the following will throw an error?

```
let x = 5;
let x = 10;
```

a) No error, x is updated
b) Error due to redeclaration
c) x becomes 10
d) It works like var

Q4: What is the output?

```
console.log(x);
var x = 10;
```

a) 10
b) undefined
c) ReferenceError
d) null

Q5: Which of the following is true for const?
a) It can be reassigned later
b) It must be initialized at declaration
c) It behaves like var
d) It can be redeclared

Q6: What is the output?

```
{
  let x = 100;
}
console.log(x);
```

a) 100
b) undefined
c) ReferenceError
d) null

Q7: Which declaration is best for a value that should not change?
a) var
b) let
c) const
d) dynamic

Q8: What is the output?

```
const arr = [1, 2, 3];
arr.push(4);
console.log(arr);
```

a) Error
b) [1, 2, 3]
c) [1, 2, 3, 4]
d) undefined

Q9: Which variable is accessible outside the loop?

```
for (var i = 0; i < 3; i++) {}
console.log(i);
```

a) 3
b) ReferenceError
c) undefined
d) 0

Q10: What is the result?

```
console.log(a);
let a = 10;
```

a) 10
b) undefined
c) ReferenceError
d) null

Correct Answers with Explanations:

Q1 – Correct Answer: c
Explanation: var is function scoped, while let and const are block scoped — they can’t be accessed outside {} blocks.

Q2 – Correct Answer: c
Explanation: var allows redeclaration in the same scope, unlike let or const.

Q3 – Correct Answer: b
Explanation: let cannot be redeclared in the same scope, so this throws a SyntaxError.

Q4 – Correct Answer: b
Explanation: var declarations are hoisted and initialized as undefined. So console.log(x) prints undefined.

Q5 – Correct Answer: b
Explanation: const must be initialized when declared, and its value can't be reassigned.

Q6 – Correct Answer: c
Explanation: let is block scoped; x is not accessible outside the block, causing a ReferenceError.

Q7 – Correct Answer: c
Explanation: Use const for values that should not be reassigned, like fixed configurations.

Q8 – Correct Answer: c
Explanation: You can't reassign the array, but its contents can be mutated. push() works fine on const arrays.

Q9 – Correct Answer: a
Explanation: var is function scoped, so i is accessible even outside the for loop.

Q10 – Correct Answer: c

Explanation: let is hoisted but not initialized, leading to a ReferenceError if accessed before declaration.

Topic 2 – Hoisting :

Q1: What is hoisting in JavaScript?
a) Moving all functions to the bottom of the code
b) Declaring variables only when needed
c) JavaScript’s default behavior of moving declarations to the top
d) Preventing variables from being used before declaration

Q2: What is the output?

```
console.log(x);
var x = 5;
```

a) 5
b) undefined
c) ReferenceError
d) null

Q3: What is the output?

```
console.log(a);
let a = 10;
```

a) 10
b) undefined
c) ReferenceError
d) null

Q4: Which variables are hoisted and initialized as undefined?
a) var
b) let

c) const
d) All of the above

Q5: What is the output?

```
greet();
function greet() {
  console.log("Hello!");
}
```

a) Hello!
b) undefined
c) ReferenceError
d) TypeError

Q6: What is the output?

```
sayHi();
var sayHi = function () {
  console.log("Hi!");
};
```

a) Hi!
b) undefined
c) ReferenceError
d) TypeError

Q7: Which statement is true about let and const hoisting?
a) They are not hoisted
b) They are hoisted but not initialized
c) They behave like var
d) They are accessible before declaration

Q8: What is the Temporal Dead Zone (TDZ)?
a) A scope where var is not allowed
b) A time from the start of block until variable declaration using let or const
c) A memory leak in JavaScript
d) The scope of anonymous functions

Q9: What happens when you access a let variable before it’s declared?
a) Returns undefined
b) Throws ReferenceError
c) Returns null
d) Returns 0

Q10: What is the output?

```
console.log(typeof myVar);
var myVar = "test";
```

a) "test"
b) "undefined"
c) "string"
d) undefined

✅ Correct Answers with Explanations

Q1 – Correct Answer: c
Explanation: Hoisting is JavaScript’s behavior of moving declarations (not initializations) to the top of the scope.

Q2 – Correct Answer: b
Explanation: var is hoisted and initialized as undefined, so console.log(x) prints undefined.

Q3 – Correct Answer: c
Explanation: let is hoisted but not initialized, causing a ReferenceError in the Temporal Dead Zone (TDZ).

Q4 – Correct Answer: a
Explanation: Only var is initialized as undefined. let and const are hoisted but not initialized.

Q5 – Correct Answer: a
Explanation: Function declarations are fully hoisted — both their definition and body.

Q6 – Correct Answer: d
Explanation: sayHi is undefined at runtime, so calling it as a function throws a TypeError.

Q7 – Correct Answer: b
Explanation: let and const are hoisted but stay uninitialized in the TDZ until declared.

Q8 – Correct Answer: b
Explanation: TDZ is the block time span where let/const exists but can't be accessed before initialization.

Q9 – Correct Answer: b
Explanation: Accessing a let variable before declaration leads to a ReferenceError due to TDZ.

Q10 – Correct Answer: b
Explanation: myVar is hoisted and initialized as undefined, so typeof myVar returns "undefined".

Topic 3 – Template Literals:

Q1: What symbol is used for template literals in JavaScript?
a) Single quotes ' '
b) Double quotes " "
c) Backticks `
d) Angle brackets < >

Q2: What is the output?

```
let name = "Ali";
console.log(`Hello, ${name}!`);
```

a) Hello, ${name}!
b) Hello, Ali!
c) Hello, "Ali"!
d) Hello, undefined!

Q3: Template literals allow:
a) String interpolation
b) Multi-line strings
c) Expression evaluation
d) All of the above

Q4: What is the output?

let a = 5;
let b = 10;
console.log(`Sum: ${a + b}`);

a) Sum: 5 + 10
b) Sum: ${a + b}
c) Sum: 15
d) Sum: undefined

Q5: Which of the following will output a multi-line string?

let msg =
a) "Line 1\nLine 2"
b) 'Line 1\nLine 2'
c) `Line 1
Line 2`
d) All of the above

Q6: What is the output?

```javascript
let item = "book";
let price = 250;
console.log(`You bought a ${item} for Rs. ${price}.`);
```

a) You bought a book for Rs. 250.
b) You bought a ${item} for Rs. ${price}.
c) undefined
d) Error

Q7: Which of the following is NOT valid inside a template literal?
a) Variables
b) Functions
c) Expressions
d) if statements

Q8: What is the output?

```
let x = 3;
let y = 4;
console.log(`${x > y ? "X" : "Y"} is greater`);
```

a) X is greater
b) Y is greater
c) 3 is greater
d) Error

Q9: Which of the following is equivalent to this template literal?

```
`Age is ${age}`
```

a) "Age is " + age
b) 'Age is age'
c) Age is age
d) "Age is ${age}"

Q10: What is the output?

```
let str = `Hello
World`;
console.log(str);
```

a) HelloWorld
b) Hello World
c) Hello
World
d) Error

✅ Correct Answers with Explanations

Q1 – Correct Answer: c
Explanation: Template literals use backticks (`) instead of single or double quotes.

Q2 – Correct Answer: b
Explanation: ${name} is evaluated and replaced with "Ali".

Q3 – Correct Answer: d
Explanation: Template literals support string interpolation, multi-line strings, and expressions.

Q4 – Correct Answer: c
Explanation: ${a + b} evaluates to 15, so output is Sum: 15.

Q5 – Correct Answer: d
Explanation: All options produce multi-line strings, but template literals do so more cleanly.

Q6 – Correct Answer: a
Explanation: Variables inside ${} are evaluated normally.

Q7 – Correct Answer: d
Explanation: if statements are not allowed directly inside ${}; only expressions are.

Q8 – Correct Answer: b
Explanation: Since 3 < 4, it outputs "Y is greater".

Q9 – Correct Answer: a
Explanation: Template literal ${age} works like concatenation "Age is " + age.

Q10 – Correct Answer: c
Explanation: Template literals preserve line breaks, so it prints as two separate lines.

Topic 4 – Ternary Operators :

Q1: What is the correct syntax of a ternary operator?
a) condition && value1 : value2
b) condition ? value1 : value2
c) if (condition) value1 else value2
d) condition ? : value1, value2

Q2: What is the output?

```
let age = 20;
let result = age >= 18 ? "Adult" : "Minor";
console.log(result);
```

a) Adult
b) Minor
c) true
d) Error

Q3: What is the result of this expression?

```
5 > 10 ? "Yes" : "No"
```

a) Yes
b) No
c) true
d) false

Q4: What is the output?

```
let x = 10;
let msg = x % 2 === 0 ? "Even" : "Odd";
console.log(msg);
```

a) 10
b) Odd
c) Even
d) undefined

Q5: Which of the following is equivalent to:

```
let result = condition ? "Yes" : "No";
```

a)

```
if (condition) {
  result = "Yes";
} else {
  result = "No";
}
```

b)

```
result = condition;
```

c)

```
result = "Yes" || "No";
```

d)

```
result = "No" ? condition : "Yes";
```

Q6: What is the output?

```
let score = 75;
let grade = score > 80 ? "A" : score > 60 ? "B" : "C";
console.log(grade);
```

a) A
b) B
c) C
d) Error

Q7: Can ternary operators be nested?
a) No
b) Yes, only twice
c) Yes, but should be used carefully
d) Yes, and encouraged in all situations

Q8: What is the output?

```
let num = 7;
console.log(num % 2 === 0 ? "Even" : "Odd");
```

a) Even
b) Odd
c) true
d) false

Q9: What is the output?

```
true ? false ? "A" : "B" : "C";
```

a) A
b) B
c) C
d) Error

Q10: What is the output?

```
let isOnline = false;
console.log(isOnline ? "User is online" : "User is offline");
```

a) User is online
b) User is offline
c) undefined
d) Error

✅ Correct Answers with Explanations

Q1 – Correct Answer: b
Explanation: The correct syntax is condition ? exprIfTrue : exprIfFalse.

Q2 – Correct Answer: a
Explanation: age is 20, which is >= 18, so "Adult" is returned.

Q3 – Correct Answer: b
Explanation: 5 is not greater than 10, so "No" is returned.

Q4 – Correct Answer: c
Explanation: 10 is even, so "Even" is logged.

Q5 – Correct Answer: a
Explanation: The ternary operator is a shorthand for an if...else statement.

Q6 – Correct Answer: b
Explanation: 75 is not >80 but is >60, so the second condition returns "B".

Q7 – Correct Answer: c
Explanation: Ternary operators can be nested but can hurt readability if overused.

Q8 – Correct Answer: b
Explanation: 7 is odd, so "Odd" is printed.

Q9 – Correct Answer: b
Explanation: Inner ternary false ? "A" : "B" returns "B", and true ? "B" : "C" finally returns "B".

Q10 – Correct Answer: b
Explanation: isOnline is false, so the false branch "User is offline" is returned.

Topic 5 – Short Circuits :

Q1: What does "short-circuit evaluation" mean in JavaScript?
a) Skipping functions during loops
b) Stopping evaluation once result is determined
c) Executing all conditions regardless
d) Always returning true

Q2: What is the output?

```
console.log(true || false);
```

a) true
b) false
c) undefined
d) Error

Q3: What is the output?

```
console.log(false && true);
```

a) true
b) false
c) undefined
d) Error

Q4: What is the output?

```
console.log(0 || "default");
```

a) 0
b) "default"
c) true
d) false

Q5: What is the output?

```
console.log("Hello" && 42);
```

a) Hello
b) 42
c) true
d) undefined

Q6: What is the output?

```
let x = null;
let result = x || "fallback";
console.log(result);
```

a) null
b) fallback
c) undefined
d) Error

Q7: Which values are considered falsy in JavaScript?
a) 0
b) "" (empty string)
c) null
d) All of the above

Q8: What is the output?

```
console.log("" && "second");
```

a) ""
b) second

c) undefined
d) null

Q9: What is the output?

console.log(true && false || "Yes");

a) false
b) Yes
c) true
d) undefined

Q10: Which operator returns the first truthy value?
a) &&
b) ||
c) ??
d) ==

✅ Correct Answers with Explanations

Q1 – Correct Answer: b
Explanation: Short-circuiting stops evaluation once the result is known.

Q2 – Correct Answer: a
Explanation: true || false → returns true (first truthy value).

Q3 – Correct Answer: b
Explanation: false && true → returns false (first falsy value stops evaluation).

Q4 – Correct Answer: b
Explanation: 0 is falsy, so it short-circuits to "default".

Q5 – Correct Answer: b
Explanation: "Hello" is truthy, so 42 is returned since both are truthy.

Q6 – Correct Answer: b
Explanation: null is falsy, so it moves to "fallback".

Q7 – Correct Answer: d
Explanation: 0, "", null, undefined, NaN, and false are falsy in JavaScript.

Q8 – Correct Answer: a
Explanation: "" is falsy, so the AND short-circuits and returns "".

Q9 – Correct Answer: b
Explanation: true && false → false, then false || "Yes" → returns "Yes".

Q10 – Correct Answer: b
Explanation: || returns the first truthy value.

Topic 6 – Spread & Rest Operators :

Q1: What is the syntax for the spread operator?
a) ...
b) ==>
c) &&
d) =>

Q2: What is the output?

```
const arr = [1, 2, 3];
const newArr = [...arr, 4, 5];
console.log(newArr);
```

a) [1, 2, 3]
b) [4, 5, 1, 2, 3]
c) [1, 2, 3, 4, 5]
d) undefined

Q3: What does the spread operator do?
a) Combines multiple functions
b) Spreads iterable elements (arrays, strings, etc.)

c) Adds new variables
d) Reverses arrays

Q4: What is the output?

```
const obj1 = { a: 1 };
const obj2 = { b: 2 };
const merged = { ...obj1, ...obj2 };
console.log(merged);
```

a) { a: 1 }
b) { b: 2 }
c) { a: 1, b: 2 }
d) Error

Q5: What is the output?

```
function sum(...nums) {
  return nums.reduce((a, b) => a + b, 0);
}
console.log(sum(1, 2, 3));
```

a) 6
b) NaN
c) undefined
d) 123

Q6: What is the purpose of the rest operator?
a) Splits a string
b) Collects multiple values into a single array
c) Ends a loop early
d) Ignores remaining arguments

Q7: What is the difference between spread and rest syntax?
a) Spread is only for arrays
b) Rest is for combining, spread is for separating
c) Spread collects items, rest separates
d) There is no difference

Q8: What is the output?

```
const greet = (greeting, ...names) => {
  return `${greeting} ${names.join(", ")}`;
};
console.log(greet("Hello", "Ali", "Sara"));
```

a) Hello Ali,Sara
b) Hello Ali Sara
c) Ali, Sara
d) Hello undefined

Q9: What is the output?

```
const arr = [10, 20, 30];
console.log(Math.max(...arr));
```

a) 30
b) undefined
c) NaN
d) 102030

Q10: What is the output?

```
const a = [1, 2];
const b = [3, 4];
const merged = [...a, ...b];
console.log(merged);
```

a) [1, 2, [3, 4]]
b) [1, 2, 3, 4]
c) [1, 2, 4, 3]
d) [3, 4, 1, 2]

✅ Correct Answers with Explanations

Q1 – Correct Answer: a
Explanation: The spread/rest operator uses the syntax ....

Q2 – Correct Answer: c
Explanation: [...arr, 4, 5] spreads the array into individual elements, resulting in [1, 2, 3, 4, 5].

Q3 – Correct Answer: b
Explanation: Spread takes an iterable and expands it into individual elements.

Q4 – Correct Answer: c
Explanation: Spread in objects combines all key-value pairs from both objects.

Q5 – Correct Answer: a
Explanation: The rest operator ...nums collects all arguments into an array, and .reduce() sums them: 1+2+3=6.

Q6 – Correct Answer: b
Explanation: Rest operator groups multiple values into an array.

Q7 – Correct Answer: b
Explanation: Spread “expands” values out; rest “gathers” multiple values into one.

Q8 – Correct Answer: a
Explanation: "Hello" is the first argument; "Ali" and "Sara" are collected into names.

Q9 – Correct Answer: a
Explanation: Spread breaks the array into arguments, so Math.max(...arr) is Math.max(10, 20, 30) → 30.

Q10 – Correct Answer: b
Explanation: [...a, ...b] becomes [1, 2, 3, 4].

Topic 7 – Destructuring of Array :

Q1: What is array destructuring?
a) Breaking arrays into smaller arrays
b) Assigning array elements to variables
c) Removing elements from arrays
d) Copying arrays using loops

Q2: What is the output?

```
const [a, b] = [10, 20];
console.log(a, b);
```

a) 10 20
b) 20 10
c) undefined undefined
d) Error

Q3: What is the output?

```
const [x, , y] = [1, 2, 3];
console.log(x, y);
```

a) 1 2
b) 1 3
c) 2 3
d) undefined undefined

Q4: What is the output?

```
const nums = [5];
const [first, second = 10] = nums;
console.log(first, second);
```

a) 5 undefined

b) 5 10
c) undefined 10
d) 10 5

Q5: What does the rest syntax ...rest do in destructuring?

```
const [a, ...rest] = [1, 2, 3, 4];
```

a) Adds all values
b) Skips all values
c) Collects remaining elements in rest
d) Throws error

Q6: What is the output?

```
let [a = 1, b = 2] = [];
console.log(a, b);
```

a) undefined undefined
b) 1 2
c) 0 0
d) NaN NaN

Q7: Can you skip array elements during destructuring?
a) No
b) Yes, using null
c) Yes, using commas
d) Only with numbers

Q8: What is the output?

```
const arr = [10, 20, 30];
const [, second] = arr;
console.log(second);
```

a) 10
b) 20
c) 30
d) undefined

Q9: What is the output?

```
const arr = [1, 2, 3, 4];
const [a, b, ...c] = arr;
console.log(c);
```

a) [3, 4]
b) [1, 2]
c) [4]
d) undefined

Q10: Which of the following will throw an error?

a)

```
const [a, b] = [1, 2];
```

b)

```
const [a, b] = "12";
```

c)

```
const [a, b] = {};
```

d)

```
const [a, , b] = [1, 2, 3];
```

✅ Correct Answers with Explanations

Q1 – Correct Answer: b
Explanation: Destructuring allows assigning individual values from an array into variables.

Q2 – Correct Answer: a
Explanation: [a, b] = [10, 20] → a=10, b=20.

Q3 – Correct Answer: b
Explanation: Skips the second value with a comma → x=1, y=3.

Q4 – Correct Answer: b
Explanation: First is 5, second is not provided, so it takes default value 10.

Q5 – Correct Answer: c
Explanation: The rest operator ...rest collects remaining values as an array → [2, 3, 4].

Q6 – Correct Answer: b
Explanation: No values provided, so default values are used → 1 and 2.

Q7 – Correct Answer: c
Explanation: Commas can be used to skip elements while destructuring.

Q8 – Correct Answer: b
Explanation: Skips the first element, assigns second (20) to second.

Q9 – Correct Answer: a
Explanation: a=1, b=2, c=[3, 4].

Q10 – Correct Answer: c
Explanation: You can’t destructure an object with array syntax — throws an error.

Topic 8 – Destructuring of Object :

Q1: What is object destructuring?
a) Breaking objects into arrays
b) Copying object values manually
c) Extracting values from an object into variables
d) Deleting object properties

Q2: What is the output?

```
const user = { name: "Ali", age: 25 };
const { name, age } = user;
console.log(name, age);
```

a) "Ali" 25
b) Ali undefined
c) undefined 25
d) Error

Q3: What is the output?

```
const person = { name: "Sara" };
const { name, age = 30 } = person;
console.log(name, age);
```

a) Sara undefined
b) Sara 30
c) undefined 30
d) Error

Q4: What is the output?

```
const obj = { a: 10, b: 20 };
const { a: x, b: y } = obj;
console.log(x, y);
```

a) 10 20
b) x y
c) undefined undefined
d) Error

Q5: Can you rename properties while destructuring?
a) No
b) Yes, using : syntax
c) Only with arrays
d) Only in functions

Q6: What is the output?

```
const settings = { darkMode: true };
const { darkMode, fontSize = 16 } = settings;
console.log(fontSize);
```

a) true
b) undefined
c) 16
d) false

Q7: Which of the following will throw an error?

a)

```
const { a, b } = { a: 1 };
```

b)

```
const { a, b = 2 } = { a: 1 };
```

c)

```
const { a, a } = { a: 1 };
```

d)

```
const { a: x } = { a: 1 };
```

Q8: What is the output?

```
const student = { name: "Ali", details: { age: 20, grade: "A" } };
const { details: { grade } } = student;
console.log(grade);
```

a) A
b) undefined
c) Error
d) details

Q9: What is the output?

```
const { a = 1, b = 2 } = { a: undefined };
console.log(a, b);
```

a) undefined undefined
b) 1 2
c) 1 undefined
d) undefined 2

Q10: Which of the following is a valid use of object destructuring in function parameters?

a)

```
function greet({ name }) {
  console.log(name);
}
greet({ name: "Ali" });
```

b)

```
function greet(name) {
  console.log({ name });
}
greet("Ali");
```

c)

```
function greet(...name) {
  console.log(name);
}
greet("Ali");
```

d)

```
function greet(name = {}) {
  console.log(name.name);
}
greet("Ali");
```

✅ Correct Answers with Explanations

Q1 – Correct Answer: c
Explanation: Object destructuring lets you extract properties from an object into variables.

Q2 – Correct Answer: a
Explanation: Extracts name and age from user → "Ali", 25.

Q3 – Correct Answer: b
Explanation: age is missing, so default 30 is used.

Q4 – Correct Answer: a
Explanation: a: x renames a to x, and b: y renames b to y.

Q5 – Correct Answer: b
Explanation: You can rename properties using : syntax in destructuring.

Q6 – Correct Answer: c
Explanation: fontSize not in object, so default 16 is used.

Q7 – Correct Answer: c
Explanation: You cannot declare the same variable name (a) twice in a single destructuring statement.

Q8 – Correct Answer: a
Explanation: Nested destructuring pulls grade from details.

Q9 – Correct Answer: b

Explanation: a is undefined, so default 1 is used. b is not defined, so uses default 2.

Q10 – Correct Answer: a
Explanation: This is valid object destructuring in function parameters.

Topic 9 – Pass by value & Pass by reference :

Q1: Which data types are passed by value in JavaScript?
a) Object, Array
b) Function, Date
c) Number, String, Boolean
d) All data types

Q2: Which of the following is passed by reference?
a) Number
b) String
c) Object
d) Boolean

Q3: What is the output?

```
let a = 10;
let b = a;
b = 20;
console.log(a);
```

a) 10
b) 20
c) undefined
d) Error

Q4: What is the output?

```
let obj1 = { value: 5 };
let obj2 = obj1;
obj2.value = 10;
console.log(obj1.value);
```

a) 5
b) 10
c) undefined
d) Error

Q5: What is the output?

```
let arr1 = [1, 2];
let arr2 = arr1;
arr2.push(3);
console.log(arr1);
```

a) [1, 2]
b) [1, 2, 3]
c) [3]
d) Error

Q6: How do you clone an object to avoid reference sharing?

a) let newObj = obj;
b) let newObj = { ...obj };
c) let newObj = obj.clone();
d) let newObj = new Object(obj);

Q7: What happens if you modify a copied primitive value?

a) Original changes
b) Original remains unchanged
c) Both are deleted
d) Error

Q8: What is the output?

```
let x = { a: 1 };
let y = { ...x };
y.a = 5;
console.log(x.a);
```

a) 1
b) 5
c) undefined
d) Error

Q9: What is the output?

```
let num1 = 100;
function update(n) {
  n = n + 50;
}
update(num1);
console.log(num1);
```

a) 150
b) 100
c) undefined
d) Error

Q10: What is the output?

```
let data = { score: 90 };
function update(obj) {
  obj.score += 10;
}
update(data);
console.log(data.score);
```

a) 90
b) 100
c) undefined
d) Error

✅ Correct Answers with Explanations

Q1 – Correct Answer: c
Explanation: Primitives like numbers, strings, and booleans are passed by value.

Q2 – Correct Answer: c
Explanation: Objects (and arrays) are passed by reference.

Q3 – Correct Answer: a
Explanation: b is a separate copy of a, so changing b doesn't affect a.

Q4 – Correct Answer: b
Explanation: obj2 is a reference to obj1, so updating obj2.value also affects obj1.value.

Q5 – Correct Answer: b
Explanation: Both arr1 and arr2 point to the same array, so arr1 reflects the change.

Q6 – Correct Answer: b
Explanation: { ...obj } creates a shallow copy of obj.

Q7 – Correct Answer: b
Explanation: Primitive values are passed by value, so modifying the copy doesn't affect the original.

Q8 – Correct Answer: a
Explanation: Spread syntax creates a copy. x.a stays 1 even if y.a is changed.

Q9 – Correct Answer: b
Explanation: num1 is a primitive and passed by value, so the change inside the function doesn’t affect the original.

Q10 – Correct Answer: b
Explanation: data is an object, so it's passed by reference. Modifying score updates the original.

Topic 10 – Object Methods :

Q1: What does Object.keys(obj) return?
a) Array of values
b) Array of keys
c) Object of keys
d) String of keys

Q2: What is the output?

```
const car = { brand: "Toyota", year: 2020 };
console.log(Object.keys(car));
```

a) ["Toyota", 2020]
b) ["brand", "year"]
c) ["car"]
d) undefined

Q3: What does Object.values(obj) return?
a) Array of keys
b) Array of properties
c) Array of values
d) Object

Q4: What is the output?

```
const user = { name: "Ali", age: 22 };
console.log(Object.values(user));
```

a) ["name", "age"]
b) ["Ali", 22]
c) undefined
d) Error

Q5: What does Object.entries(obj) return?
a) Array of objects
b) Object with key-value pairs
c) Array of [key, value] pairs
d) A string

Q6: What is the output?

```
const obj = { a: 1, b: 2 };
console.log(Object.entries(obj));
```

a) [["a", "b"], [1, 2]]
b) [["a", 1], ["b", 2]]
c) ["a", 1, "b", 2]
d) Error

Q7: What does Object.freeze(obj) do?
a) Deletes all properties
b) Prevents modification
c) Copies the object
d) Makes all values undefined

Q8: What is the output?

```
const profile = { name: "Sara" };
Object.freeze(profile);
profile.name = "Ayesha";
console.log(profile.name);
```

a) "Sara"
b) "Ayesha"
c) undefined
d) Error

Q9: Which method can be used to loop over object key-value pairs?

a) for (let i of obj)
b) Object.values(obj)
c) Object.entries(obj)
d) obj.map()

Q10: What will happen if you try to add a new property to a frozen object?

```
const data = { id: 1 };
Object.freeze(data);
data.status = "active";
console.log(data.status);
```

a) "active"
b) null
c) undefined
d) Error

✅ Correct Answers with Explanations

Q1 – Correct Answer: b
Explanation: Object.keys() returns an array of enumerable property names (keys).

Q2 – Correct Answer: b
Explanation: Keys of the object car → ["brand", "year"].

Q3 – Correct Answer: c
Explanation: Object.values() returns an array of the object's values.

Q4 – Correct Answer: b
Explanation: Values in the object → ["Ali", 22].

Q5 – Correct Answer: c

Explanation: Object.entries() returns an array of [key, value] pairs.

Q6 – Correct Answer: b
Explanation: [["a", 1], ["b", 2]] is the correct format.

Q7 – Correct Answer: b
Explanation: Object.freeze() prevents changes to properties or adding new ones.

Q8 – Correct Answer: a
Explanation: The object is frozen, so name cannot be changed.

Q9 – Correct Answer: c
Explanation: Object.entries() allows looping with for...of like:

```
for (const [key, value] of Object.entries(obj)) {}
```

Q10 – Correct Answer: c
Explanation: Frozen objects ignore property additions silently, so it returns undefined.

Topic 11 – Arrow func, Higher order func, Default Parameters :

Q1: What is the syntax of an arrow function?
a) function => () {}
b) () => {}
c) () -> {}
d) fn => () {}

Q2: What is the output?

```
const greet = () => "Hello!";
console.log(greet());
```

a) Hello
b) "Hello!"
c) undefined
d) Error

Q3: Which of the following is a higher-order function?
a) A function that returns a value
b) A function inside an object
c) A function that takes another function as an argument
d) A function without parameters

Q4: What is the output?

```
function operate(a, b, func) {
  return func(a, b);
}
console.log(operate(2, 3, (x, y) => x + y));
```

a) 5
b) 6
c) 23
d) Error

Q5: What is the output?

```
const add = (x = 5, y = 10) => x + y;
console.log(add());
```

a) 5
b) 10
c) 15
d) NaN

Q6: What is the output?

```
const square = x => x * x;
console.log(square(4));
```

a) 8

b) 16
c) 4
d) Error

Q7: What is the output?

```
const greet = name => `Hello, ${name}`;
console.log(greet("Ali"));
```

a) Hello,
b) Hello, ${name}
c) Hello, Ali
d) name

Q8: Which of the following is a valid arrow function returning an object?

a)

```
const getObj = () => { name: "Ali" };
```

b)

```
const getObj = () => ({ name: "Ali" });
```

c)

```
const getObj = () => [ name: "Ali" ];
```

d)

```
const getObj = => { name: "Ali" };
```

Q9: What is a default parameter in a function?

a) The first parameter

b) A function that never runs
c) A parameter with a default value if none is provided
d) A required value

Q10: What is the output?

```
function multiply(a, b = 2) {
  return a * b;
}
console.log(multiply(4));
```

a) 4
b) 8
c) 2
d) undefined

✅ Correct Answers with Explanations

Q1 – Correct Answer: b
Explanation: The correct arrow function syntax is () => {}.

Q2 – Correct Answer: b
Explanation: greet returns the string "Hello!" and logs it.

Q3 – Correct Answer: c
Explanation: Higher-order functions take other functions as arguments or return them.

Q4 – Correct Answer: a
Explanation: The arrow function (x, y) => x + y returns 5 when passed 2 and 3.

Q5 – Correct Answer: c
Explanation: Default values are 5 and 10, so add() returns 15.

Q6 – Correct Answer: b
Explanation: square(4) returns 4 * 4 = 16.

Q7 – Correct Answer: c
Explanation: Template literals return Hello, Ali.

Q8 – Correct Answer: b
Explanation: Use parentheses ({ ... }) to return an object from an arrow function.

Q9 – Correct Answer: c
Explanation: A default parameter is used when no argument is passed.

Q10 – Correct Answer: b
Explanation: multiply(4) uses default b = 2, so 4 * 2 = 8.

Topic 12 – Array Methods :

🔹 map() – 5 MCQs

Q1:
What does map() return?
a) The original array
b) A new transformed array
c) Boolean
d) String

Q2:
What is the output?

```
const arr = [1, 2, 3];
const doubled = arr.map(x => x * 2);
console.log(doubled);
```

a) [1, 2, 3]
b) [2, 4, 6]
c) [1, 4, 9]
d) Error

Q3:
Which of the following is true about map()?
a) It modifies the original array
b) It skips null values
c) It always returns a new array
d) It runs asynchronously

Q4:
What is the output?

```
const nums = [2, 4];
const str = nums.map(x => x.toString());
console.log(str);
```

a) [2, 4]
b) ["2", "4"]
c) "24"
d) Error

Q5:
What is the output?

```
const arr = [true, false];
const result = arr.map(val => !val);
console.log(result);
```

a) [true, false]
b) [false, true]
c) [true, true]
d) Error

🔹 filter() – 5 MCQs

Q6:
What does filter() do?
a) Modifies each element
b) Removes undefined
c) Returns elements matching condition
d) Changes values to strings

Q7:
What is the output?

```
const ages = [12, 18, 25];
const adults = ages.filter(age => age >= 18);
console.log(adults);
```

a) [12, 18]
b) [18, 25]
c) [12]
d) [25]

Q8:
What is the output?

```
const arr = [0, 1, 2];
const truthy = arr.filter(Boolean);
console.log(truthy);
```

a) [0, 1, 2]
b) [1, 2]
c) [0]
d) []

Q9:
Can filter() return an empty array?
a) Yes
b) No

Q10:
What is the output?

```
const data = ["apple", "banana", "cherry"];
const bFruits = data.filter(f => f.startsWith("b"));
console.log(bFruits);
```

a) ["banana"]
b) ["apple", "banana"]
c) ["b", "cherry"]
d) ["banana", "cherry"]

🔹 forEach() – 5 MCQs

Q11:
What does forEach() return?
a) An array
b) Nothing (undefined)
c) Boolean
d) String

Q12:
What is the output?

```
let sum = 0;
[1, 2, 3].forEach(num => sum += num);
console.log(sum);
```

a) 3
b) 6
c) 0
d) 123

Q13:
Can forEach() be used to modify array values directly?
a) Yes
b) No

Q14:
What is the output?

```
const names = ["Ali", "Sara"];
names.forEach(name => console.log(name));
```

a) AliSara
b) ["Ali", "Sara"]
c) Ali \n Sara
d) undefined

Q15:
Which is true about forEach()?
a) It returns a new array
b) It stops if condition is false
c) It is used mainly for side effects (like logging)
d) It’s faster than map()

🔹 find() – 5 MCQs

Q16:
What does find() return?
a) All matches
b) Last match
c) First match
d) Index

Q17:
What is the output?

```
const arr = [3, 6, 9];
const result = arr.find(x => x > 5);
console.log(result);
```

a) 3
b) 6
c) 9
d) [6, 9]

Q18:
Can find() return undefined?
a) Yes
b) No

Q19:
What is the output?

```
const nums = [1, 2, 3];
const result = nums.find(n => n > 10);
console.log(result);
```

a) 0
b) undefined
c) null
d) 10

Q20:
Does find() return an array?
a) Yes
b) No

🔹 reduce() – 5 MCQs

Q21:
What is the purpose of reduce()?
a) Increase array length
b) Flatten arrays

c) Accumulate to single value
d) Loop with side effects

Q22:
What is the output?

```
const nums = [1, 2, 3];
const total = nums.reduce((a, b) => a + b, 0);
console.log(total);
```

a) 6
b) 123
c) 0
d) 1

Q23:
What is the initial value in reduce() used for?
a) First loop value
b) Final result
c) Accumulator's starting value
d) Loop counter

Q24:
What is the output?

```
const str = ["a", "b", "c"];
const result = str.reduce((a, b) => a + b);
console.log(result);
```

a) abc
b) ["a", "b", "c"]
c) Error
d) undefined

Q25:
What is the output?

```
const nums = [2, 4, 6];
const result = nums.reduce((a, b) => a * b, 1);
console.log(result);
```

a) 48
b) 12
c) 24

d) 1

🔹 findIndex() – 5 MCQs

Q26:
What does findIndex() return?
a) First matching value
b) Last matching index
c) First matching index
d) All indexes

Q27:
What is the output?

```
const arr = [10, 20, 30];
console.log(arr.findIndex(x => x === 20));
```

a) 1
b) 2
c) 20
d) -1

Q28:
What is the output?

```
const fruits = ["apple", "banana"];
console.log(fruits.findIndex(f => f.startsWith("c")));
```

a) 0
b) 1
c) -1
d) undefined

Q29:
Is findIndex() similar to find() but returns index instead?
a) Yes
b) No

Q30:
What is the output?

```
const letters = ["a", "b", "c"];
console.log(letters.findIndex(l => l === "d"));
```

a) 0
b) 3
c) undefined
d) -1

✅ Correct Answers with Explanations:

🔹 map() – Answers

Q1 – b
Explanation: map() returns a new transformed array.

Q2 – b
Explanation: Each value doubled → [2, 4, 6].

Q3 – c
Explanation: map() always returns a new array; it doesn't mutate the original.

Q4 – b
Explanation: Numbers converted to strings → ["2", "4"].

Q5 – b
Explanation: true becomes false, and false becomes true → [false, true].

🔹 filter() – Answers

Q6 – c
Explanation: It returns only elements that pass the test.

Q7 – b
Explanation: Ages >= 18 → [18, 25].

Q8 – b
Explanation: Boolean truthy values → [1, 2] (0 is falsy).

Q9 – a
Explanation: Yes, if no element matches, it returns [].

Q10 – a
Explanation: Only "banana" starts with "b".

🔹 forEach() – Answers

Q11 – b
Explanation: forEach() returns undefined.

Q12 – b
Explanation: 1 + 2 + 3 = 6.

Q13 – a
Explanation: Yes, but you must mutate it manually inside the loop.

Q14 – c
Explanation: It prints:

```
Ali
Sara
```

Q15 – c
Explanation: It’s mainly used for side effects (logging, etc.), not transformation.

🔹 find() – Answers

Q16 – c
Explanation: find() returns the first matching value.

Q17 – b
Explanation: First number > 5 is 6.

Q18 – a
Explanation: Yes, if nothing matches, it returns undefined.

Q19 – b
Explanation: No number > 10 → returns undefined.

Q20 – b

Explanation: It returns a single value, not an array.

🔹 reduce() – Answers

Q21 – c
Explanation: reduce() is used to accumulate values to a single result.

Q22 – a
Explanation: 1+2+3 = 6 (with initial value 0).

Q23 – c
Explanation: Initial value sets starting value for the accumulator.

Q24 – a
Explanation: "a" + "b" + "c" = "abc".

Q25 – a
Explanation: 2 * 4 * 6 = 48.

🔹 findIndex() – Answers

Q26 – c
Explanation: It returns the first matching index.

Q27 – a
Explanation: 20 is at index 1.

Q28 – c
Explanation: No fruit starts with "c" → returns -1.

Q29 – a
Explanation: Yes, find() gives value, findIndex() gives index.

Q30 – d
Explanation: "d" not found → returns -1.

Topic 13 – Promise( ) :

Q1: What is the initial state of a JavaScript Promise?
a) fulfilled
b) pending
c) rejected
d) resolved

Q2: What are the three states of a Promise?
a) open, processing, closed
b) wait, done, fail
c) pending, fulfilled, rejected
d) start, continue, end

Q3: What is the output?

```
Promise.resolve("Success").then((msg) => {
  console.log(msg);
});
```

a) Promise {<pending>}
b) undefined
c) "Success"
d) Error

Q4: What is the output?

```
let p = new Promise((resolve, reject) => {
  reject("Error occurred");
});

p.then((res) => console.log(res))
 .catch((err) => console.log(err));
```

a) Error

b) Error occurred
c) undefined
d) Nothing

Q5: Which block is used to handle a failed Promise?
a) .then()
b) .next()
c) .catch()
d) .fail()

Q6: What is the output?

```
let p = new Promise((resolve, reject) => {
  setTimeout(() => resolve("Done!"), 1000);
});
p.then(msg => console.log(msg));
```

a) "Done!" immediately
b) Nothing
c) Error
d) "Done!" after 1 second

Q7: Can a Promise have both resolve and reject called?
a) Yes, always
b) No, only one will work
c) Yes, but only if chained
d) Only in async functions

Q8: Which method runs regardless of success or failure?
a) then()
b) catch()
c) finally()

d) all()

Q9: What is the output?

```
Promise.reject("Fail")
  .catch(err => {
    console.log("Caught:", err);
    return "Recovered";
  })
  .then(msg => console.log(msg));
```

a) Caught: Fail
b) Recovered
c) Both a and b
d) Error

Q10: What does Promise.all() do?
a) Resolves when all promises resolve
b) Rejects when any promise rejects
c) Returns an array of results
d) All of the above

✅ Answers + Explanations

Q1 – b
Explanation: A Promise starts in the pending state.

Q2 – c
Explanation: The three states are: pending, fulfilled, and rejected.

Q3 – c
Explanation: Promise.resolve() creates a resolved promise, so "Success" is logged.

Q4 – b
Explanation: The .catch() block handles the rejection and logs: "Error occurred".

Q5 – c
Explanation: .catch() handles Promise failures.

Q6 – d
Explanation: The Promise resolves after 1 second, so "Done!" is logged then.

Q7 – b
Explanation: A Promise can be settled only once — either resolved or rejected, not both.

Q8 – c
Explanation: .finally() runs regardless of the outcome.

Q9 – c
Explanation: Logs:

```
Caught: Fail
Recovered
```

Q10 – d
Explanation: Promise.all():

Resolves when all promises resolve.

Rejects if any one fails.

Returns an array of results → ✅ All of the above.

Topic 14 – Async / Awaits :

Q1: What does the async keyword do to a function?
a) Makes it synchronous
b) Converts it into a generator
c) Makes the function return a Promise
d) Delays execution

Q2: What is the output?

```
async function test() {
  return "Hello";
}
test().then(console.log);
```

a) Hello
b) Promise {<fulfilled>: "Hello"}
c) undefined
d) Error

Q3: What is the output?

```
async function test() {
  return 5;
}
test().then(res => console.log(res));
```

a) Promise {5}
b) 5
c) undefined
d) Error

Q4: What is the purpose of await?
a) Waits for a timeout
b) Blocks execution
c) Waits for a Promise to resolve
d) Used only in loops

Q5: Which statement is true about await?
a) It can be used outside any function
b) It only works inside an async function
c) It speeds up execution
d) It works with non-Promise values only

Q6: What is the output?

```
async function demo() {
  let result = await Promise.resolve("Done");
  console.log(result);
}
demo();
```

a) undefined
b) Error
c) Promise
d) Done

Q7: What is the output?

```
async function getData() {
  throw new Error("Fail");
}
getData().catch(err => console.log(err.message));
```

a) Error
b) Fail
c) undefined
d) Nothing

Q8: What will happen?

```
async function run() {
  await 42;
  console.log("After await");
}
run();
```

a) Error
b) Logs "After await"
c) Logs 42
d) Nothing

Q9: Why use try...catch with await?
a) To avoid await altogether
b) To stop loops
c) To handle rejected Promises
d) To make synchronous code faster

Q10: Which of these is equivalent to using await?

```
let result = await fetch(url);
```

a)

fetch(url).then(res => result = res);

b)

result = fetch(url);

c)

let result = url.fetch();

d)

result = await url;

✅ Answers + Explanations

Q1 – c
✅ async turns a function into one that returns a Promise automatically.

Q2 – a
✅ It logs: Hello. Even though it returns a string, it is wrapped in a Promise.

Q3 – b
✅ Logs: 5, because async wraps it in a Promise and .then() logs the value.

Q4 – c
✅ await waits for the Promise to resolve, then continues execution.

Q5 – b
✅ await only works inside async functions.

Q6 – d
✅ The promise resolves with "Done", so it logs: Done.

Q7 – b
✅ The function throws an error, which is caught by .catch() → logs "Fail".

Q8 – b
✅ await 42 wraps 42 in a resolved Promise. It waits (even if instant), then logs.

Q9 – c
✅ A rejected Promise inside await throws an error. try...catch is used to handle it safely.

Q10 – a
✅ await is equivalent to using .then() in a normal Promise.

Topic 15 – Fetch( ) :

Q1: What does fetch() return?
a) A JSON object
b) A string
c) A Promise

d) A callback

Q2: Which of the following is correct syntax for using fetch?

a) fetch.get("https://api.com")
b) fetch("https://api.com").then()
c) get("https://api.com").fetch()
d) fetch.fetch("https://api.com")

Q3: What is the output?

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
  .then(res => res.json())
  .then(data => console.log(data.title));
```

a) Error
b) The response body as text
c) The title of post 1
d) Nothing

Q4: What happens if .json() is not called on the response?
a) You get raw JSON object
b) You get a rejected Promise
c) You get a Response object
d) You get undefined

Q5: What is the correct way to catch a fetch error?

```
fetch("url")
  .then(response => response.json())
  .catch(err => console.log(err));
```

a) .catch must come before .then
b) Error must be handled inside then
c) This is correct
d) Error can’t be caught in fetch

Q6: What is the output?

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
  .then(res => res.ok)
  .then(ok => console.log(ok));
```

a) true
b) false
c) undefined
d) Response object

Q7: What does res.ok indicate?
a) If status is 200-299
b) If body is not empty
c) If JSON is valid
d) Always true

Q8: What is the output?

```
fetch("invalid_url")
  .then(res => res.json())
  .catch(err => console.log("Error:", err));
```

a) Nothing
b) JSON object
c) Network error caught
d) Syntax error

Q9: Can fetch() be used with async/await?
a) No
b) Only with callbacks
c) Yes
d) Only in Node.js

Q10: What will this log?

```
async function getData() {
  const res = await fetch("https://jsonplaceholder.typicode.com/posts/1");
  const data = await res.json();
  console.log(data.id);
}
getData();
```

a) 1
b) Error
c) undefined
d) [object Object]

✅ Answers + Explanations

Q1 – c
✅ fetch() always returns a Promise that resolves to a Response object.

Q2 – b
✅ Correct syntax: fetch("URL").then(...).

Q3 – c
✅ .then(res => res.json()) extracts the body → .then(data => ...) gets title.

Q4 – c
✅ You'll get a Response object, not the parsed data.

Q5 – c
✅ This is correct. .catch() captures network or parsing errors.

Q6 – a
✅ res.ok is true if HTTP status is 200–299.

Q7 – a
✅ res.ok is true only for successful HTTP status codes (200-299).

Q8 – c
✅ Invalid URL → fetch() fails → .catch() logs the error.

Q9 – c
✅ Yes, fetch() can be used with async/await syntax.

Q10 – a
✅ The response is parsed to JSON → then data.id is 1, so logs 1.

Topic 16 – API :

Q1: What does API stand for?
a) Application Processing Interface
b) Advanced Programming Interface
c) Application Programming Interface
d) Application Programming Integration

Q2: Which of the following is a common use of an API in web development?
a) Designing a UI
b) Storing HTML in the database
c) Communicating with a server
d) Styling pages with CSS

Q3: In REST APIs, what does GET request do?
a) Updates a resource
b) Deletes a resource
c) Creates a resource
d) Fetches a resource

Q4: What HTTP method is typically used to update an existing resource in a REST API?
a) GET
b) POST
c) PUT
d) DELETE

Q5: Which status code means a request was successful?
a) 200
b) 404
c) 500

d) 301

Q6: Which status code indicates that a resource was not found?
a) 200
b) 403
c) 404
d) 401

Q7: What is the output of this fetch API call?

```
fetch("https://api.example.com/data")
  .then(response => response.status)
  .then(status => console.log(status));
```

a) The full response data
b) The status code (like 200)
c) Error
d) undefined

Q8: Which of the following is a benefit of using APIs?
a) Slower development time
b) Code duplication
c) Scalability and integration
d) Manual data transfer

Q9: Which format is most commonly used for API responses?
a) CSV
b) XML
c) JSON
d) HTML

Q10: Which part of the API call specifies what kind of request is being made?
a) URL
b) Method (GET, POST, etc.)
c) Headers
d) Body

✅ Answers with Explanations

Q1 – c
✅ API = Application Programming Interface.

Q2 – c
✅ APIs allow communication between frontend and backend (or external servers).

Q3 – d
✅ A GET request is used to fetch data from a server.

Q4 – c
✅ PUT is used to update a resource.

Q5 – a
✅ 200 OK = successful HTTP request.

Q6 – c
✅ 404 Not Found means the server can't find the resource.

Q7 – b
✅ .status gives the HTTP status code, such as 200.

Q8 – c
✅ APIs improve integration, scalability, and reusability of services.

Q9 – c
✅ Most modern APIs return data in JSON (JavaScript Object Notation) format.

Q10 – b
✅ The method defines what type of request (GET, POST, PUT, DELETE, etc.).

Topic 17 – Modules :

Q1: What is the correct keyword to export a value from a module?
a) send
b) export
c) module
d) public

Q2: Which keyword is used to bring in functionality from another module?
a) require
b) use

c) import
d) include

Q3: What is the correct way to export a function as the default export?

```
function sayHi() {
  console.log("Hi!");
}
```

a) export function sayHi();
b) export default sayHi;
c) export = sayHi;
d) default export sayHi;

Q4: What is the correct way to import the default export from a module?

```
// In utils.js
export default function greet() {
  return "Hello!";
}
```

a) import greet from './utils.js';
b) import { greet } from './utils.js';
c) import * as greet from './utils.js';
d) import './utils.js' as greet;

Q5: What will this code output?

```
// In math.js
export const x = 2;
export const y = 3;

// In main.js
import * as math from './math.js';
console.log(math.x + math.y);
```

a) 23
b) undefined
c) 5
d) NaN

Q6: What is true about ES6 module imports?
a) They are hoisted
b) They are synchronous
c) They are read-only views
d) They allow duplicate imports

Q7: What happens if you try to reassign an imported value?

```
// In config.js
export const theme = "dark";

// In main.js
import { theme } from './config.js';
theme = "light";
```

a) It changes successfully
b) It throws a SyntaxError
c) It throws a TypeError
d) It becomes undefined

Q8: How many default exports can a single module have?
a) One
b) Unlimited
c) None
d) One per function

Q9: What will happen if you try to import a non-existent named export?

```
// In math.js
export const num = 5;

// In main.js
import { sum } from './math.js';
```

a) Error at runtime
b) It imports as undefined
c) Nothing happens
d) SyntaxError at compile time

Q10: What is the output of the following code?

```
// In values.js
export let count = 0;
export function increment() {
  count++;
}

// In app.js
import { count, increment } from './values.js';
increment();
console.log(count);
```

a) 1
b) 0
c) undefined
d) Error

✅ Answers & Explanations

Q1 – b
Explanation:

export is the correct keyword used to expose values from a module.
<span style="color: orange;">It allows other modules to import them.</span>

Q2 – c
Explanation:
import is used to bring in values/functions from another module.
<span style="color: orange;">It is the ES6 way of handling modular code.</span>

Q3 – b
Explanation:
export default sayHi; exports the sayHi function as default.
<span style="color: orange;">You can then import it without curly braces.</span>

Q4 – a
Explanation:
import greet from './utils.js'; is the right way to import a default export.
<span style="color: orange;">Named exports require curly braces; default ones don't.</span>

Q5 – c
Explanation:
math.x = 2 and math.y = 3 → 2 + 3 = 5
<span style="color: orange;">Importing everything as math gives access to individual exports via dot notation.</span>

Q6 – c
Explanation:
Imported bindings are read-only views of exported values.
<span style="color: orange;">You can't reassign them directly.</span>

Q7 – c
Explanation:
Trying to reassign an imported const value throws a TypeError.
<span style="color: orange;">Imports are immutable (even if their contents can mutate).</span>

Q8 – a
Explanation:
Only one default export is allowed per file.
<span style="color: orange;">You can have many named exports but only one default.</span>

Q9 – d
Explanation:
Trying to import something that wasn't exported causes a SyntaxError at compile time.
<span style="color: orange;">You must only import names that were explicitly exported.</span>

Q10 – b
Explanation:
Even though increment() changes count, the import is a live binding,
but the imported primitive count doesn't auto-update in the local scope → stays 0.
<span style="color: orange;">This is a tricky ES module behavior.</span>

Topic 18 – Classes :

Q1: What is the correct syntax for defining a class in JavaScript?
a) function class MyClass() {}
b) class MyClass { constructor() {} }
c) MyClass = class() constructor {}
d) class = MyClass() {}

Q2: What will the following code output?

```
class Person {
  constructor(name) {
    this.name = name;
  }
}
const p = new Person("Ali");
console.log(p.name);
```

a) Ali
b) undefined
c) null
d) Person

Q3: Which keyword is used to inherit properties from a parent class?
a) inherit
b) super
c) extends
d) implements

Q4: What will the following code output?

```
class A {
  greet() {
    return "Hi from A";
  }
}
class B extends A {
  greet() {
    return super.greet() + " and B";
  }
}
const b = new B();
console.log(b.greet());
```

a) Hi from B

b) Hi from A
c) Hi from A and B
d) undefined

Q5: What is the result of the following code?

```
class X {}
console.log(typeof X);
```

a) object
b) undefined
c) function
d) class

Q6: What is the purpose of calling super() inside a subclass constructor?
a) To access the subclass fields
b) To initialize the parent class
c) To override the constructor
d) To define private fields

Q7: What happens if you omit super() in a subclass constructor?

```
class Parent {
  constructor() {
    console.log("Parent");
  }
}
class Child extends Parent {
  constructor() {
    console.log("Child");
  }
}
const c = new Child();
```

a) Only logs "Child"

b) Logs both "Parent" and "Child"
c) Throws a ReferenceError
d) Throws a SyntaxError

Q8: What is the output of the following code?

```
class Test {
  static greet() {
    return "Hello!";
  }
}
console.log(Test.greet());
```

a) Hello!
b) undefined
c) Error
d) null

Q9: What happens if you try to call a static method using an instance?

```
class MyClass {
  static sayHi() {
    return "Hi!";
  }
}
const obj = new MyClass();
console.log(obj.sayHi());
```

a) Hi!
b) undefined
c) Error
d) null

Q10: How many constructors can a JavaScript class have?
a) As many as needed

b) Only one
c) One per method
d) None

✅ Answers & Explanations

Q1 – b
Explanation:
class MyClass { constructor() {} } is the correct syntax to define a class.
<span style="color: orange;">JavaScript uses the class keyword followed by a constructor method.</span>

Q2 – a
Explanation:
Ali is passed to the constructor and stored in this.name.
<span style="color: orange;">So console.log(p.name) outputs "Ali".</span>

Q3 – c
Explanation:
extends is used for class inheritance.
<span style="color: orange;">It allows one class to inherit properties and methods from another.</span>

Q4 – c
Explanation:
super.greet() calls A’s method → returns "Hi from A"
B’s method adds " and B" → "Hi from A and B"
<span style="color: orange;">This demonstrates method overriding and super keyword usage.</span>

Q5 – c
Explanation:
In JavaScript, classes are special functions.
<span style="color: orange;">typeof X returns "function".</span>

Q6 – b
Explanation:
super() must be called in a subclass constructor to call the parent constructor.
<span style="color: orange;">It ensures the parent is correctly initialized.</span>

Q7 – d
Explanation:
If you don’t call super() in a subclass before using this, it throws a SyntaxError.
<span style="color: orange;">super() is required in constructors of child classes.</span>

Q8 – a
Explanation:
greet() is a static method, and it’s called on the class, not on an instance.
<span style="color: orange;">So Test.greet() returns "Hello!".</span>

Q9 – c
Explanation:
Static methods can’t be called on instances — only on the class itself.
<span style="color: orange;">So obj.sayHi() throws a TypeError.</span>

Q10 – b
Explanation:
JavaScript classes can have only one constructor method.
<span style="color: orange;">Multiple constructors will cause a SyntaxError.</span>

{ Prepared by : “Ismail Shah” }