Day→12

JavaScript Output Based Question

gitHub→rajeshjha2000

(56) `console.log(Promise.resolve(5));`

Ans:> • 'Promise.resolve(value)' is a method that returns a 'Promise' object that is resolved with a given value.

• In this case, 'Promise.resolve(5)' creates a promise that is immediately resolved with the value '5'.

• When we log a promise using 'console.log', it prints the promise object itself, not the resolved value.

• The Promise Object will show it's status (whether it's pending, resolved or rejected) and the resolved value if it's resolved.

• Since the Promise is resolved immediately, the status will be 'fulfilled', and the value will be '5'.

Therefore, the output will be ⟶

⟶ Promise {5}.

↓

Promise {<fulfilled>: 5}

(57) `console.log('♡' === '♡');`

Ans:> 1. String Comparison ⟶

• In JS, the '===' operator check for strict equality, which means it checks both the value and the type of the operands.

- The two strings '♡' are identical in both value and type (they both are Strings).

2. Unicode and String Equality →

- The heart character '♡' is a unicode character, and since both strings contain exactly the same character, the comparison returns 'true'.

Therefore, the output will be →

- true

(58)
```
let name = 'Jha';
function getName () {
  console.log(name);
  let name = 'Rajesh';
}
getName ();
```

A→ 1. Variable Hoisting →

- JS uses a concept called "hoisting" where variable and function declaration are moved to the top of their scope before code execution

- However, 'let' and 'const' variables are hoisted but are not initialized. This means they are in "temporal Dead Zone" from the start of the block until the declaration is encountered.

2. Within the 'getName' Function →

- Inside the 'getName' function, the variable 'name' is declared using 'let', but this happens after the 'console.log(name)' statement.
- Due to hoisting, the 'name' variable inside 'getName' is recognized but not initialized before it is used in 'console.log(name)'.
- This leads to a 'ReferenceError' because the variable 'name' is in the 'temporal dead zone' at the time 'console.log(name)' tries to access it.

3. Global 'name' Variable →

- Even though there is a global variable 'name' with the value 'Jha', it is shadowed by the local 'name' variable within the 'getName' function
- Because the local 'name' variable is declared with let (and not yet initialized), the code does not fall back to the global 'name' variable.

Therefore, the output will be →

- ReferenceError: Cannot access 'name' before initializatio

Note →
```
• let name = 'Jha';
  function getName() {
  console.log(name);          → output - Jha
  }
  getName();
```

- Since there is no local variable 'name' declared inside the 'getName' function, it will look for the 'name' variable in it's outer scope (the global scope in this case).

(59) console.log(`${(x => x)('I love')} to program`);

Ans:> 1. Arrow Function ⟶

- `(x ⇒ x)' is an arrow function that takes a single argument `x' and returns `x'. It simply returns whatever it receives as input.

2. Function Invocation ⟶

- The arrow function is immediately invoked with the argument 'I love'.

- This means the function receives 'I love' as input and returns 'I love'.

3. Template literal ⟶

- The Template literal `${(x ⇒x) ('I love')}' evaluates as 'I love'.

- So, the whole expression inside the template literal is replaced by string 'I love'.

4. Concatenation ⟶

- The result of the template literal 'I love' is concatenated with the string "to program".

Therefore, the output will be ⟶

- I love to program

(60) Q:> Output should be 6.

```
function sumValues (x,y,z) {
    return x+y+z;
}
```

option → (A) sumValues([...1,2,3])

(B) sumValues([...[1,2,3]])

(C) sumValues (... [1,2,3])

(D) sumValues ([1,2,3])

Ans→. (A) `sumValues ([... 1,2,3]);`

- This is incorrect because `[...1,2,3]` is not valid syntax and will cause an error

- (B) `sumValues ([... [1,2,3]]);`
  - This is incorrect because `... [1,2,3]` inside an array will still produce an array, so we are essentially passing an array as a single argument, not three separate numbers.

- (C) `sumValues (... [1,2,3]);`
- This is correct. The spread operator `...` will expand the array `[1,2,3]` into individual argument `1`, `2`, `3`, which will be passed to `x`, `y`, `z` respectively.

- (D) `sumValues ([1,2,3]);`
  - This is incorrect because we are passing an array as a single argument, not individual numbers.

For more question, visit →

gitHub → rajeshjha2000