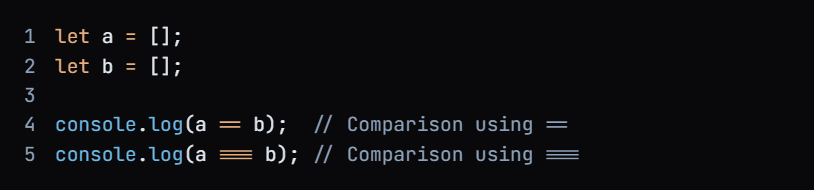
**Question 1**



**Output**

False

False

**Output Explanation**

In JavaScript, when comparing two arrays (or objects) using either **==** or **===**, the comparison is based on reference/memory rather than value.

This means that two different array instances, even if they contain the same elements, will not be considered equal. Because the value can be same but the reference can not be the same.

1. **console.log(a == b);**
   * **Output**: **false**
   * **Reason**: The **==** operator checks for value equality, but since **a** and **b** are two distinct array instances, they do not reference the same object in memory.
2. **console.log(a === b);**
   * **Output**: **false**
   * **Reason**: The **===** operator checks for both value and type equality. Again, since **a** and **b** are different instances, this comparison also evaluates to **false**.

**Question 2**

A computer screen with white text

Description automatically generated

**Output**

A black background with a black square

Description automatically generated with medium confidence

**Output Explanation**

Here, **b** is assigned the reference of **a**, meaning both **a** and **b** point to the same array instance in memory.

This demonstrates that when two variables reference the same object in memory, both **==** and **===** comparisons will evaluate to **true**.

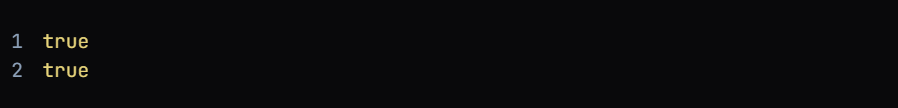
1. **console.log(a == b);**
   * **Output**: **true**
   * **Reason**: Since **a** and **b** reference the same array instance, the **==** operator evaluates to **true**.
2. **console.log(a === b);**
   * **Output**: **true**
   * **Reason**: Similarly, the **===** operator also evaluates to **true** because both **a** and **b** reference the same instance and are of the same type (array).

**Question 3**

A black background with white text

Description automatically generated

Output



Output Explanation

Here we compare the value of array not memory so the output is true in both the cases.

**Output Explanation**

1. **console.log(a[0] == b[0]);**
   * **Output**: **true**
   * **Reason**: The **==** operator checks for value equality. Since both **a[0]** and **b[0]** have the same value (**20**), this comparison evaluates to **true**.
2. **console.log(a[0] === b[0]);**
   * **Output**: **true**
   * **Reason**: The **===** operator checks for both value and type equality. Again, since both **a[0]** and **b[0]** have the same value (**20**) and are of the same type (number), this comparison also evaluates to **true**.

**Question 4**

A computer screen shot of a computer code

Description automatically generated

**Output**

**1,2,3,4**

**Output Explanation**

Reason: The spread operator (...) unpacks the elements of the array a and passes them as individual arguments to the console.log() function.

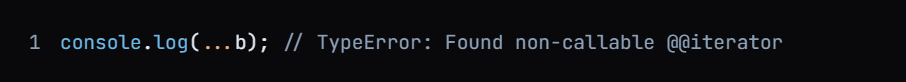
This results in the individual elements being logged to the console

(...) unpacks the elements of the array means it destructured the array and give the output in string.

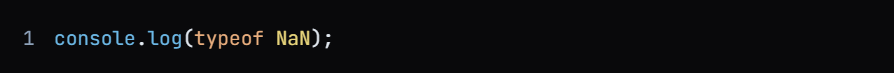
Note that the object b is not used in this example, so its presence does not affect the output.

Also, keep in mind that if you try to use the spread operator on an object (like b) that is not iterable, you will get a TypeError.

**For example:**

****

**Question 5**

****

**Output Explanation**

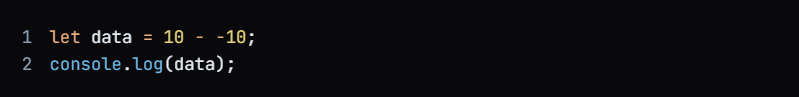
* Output: "number"
* Reason: Although NaN stands for "Not a Number", it is actually a special value in the number data type. This might seem counterintuitive, but it's because NaN is used to represent a result that cannot be expressed as a finite number, such as the result of dividing zero by zero or taking the square root of a negative number.

In JavaScript, NaN is a property of the global object, and it is a number value. Therefore, when you use the typeof operator to check the data type of NaN, it returns "number".

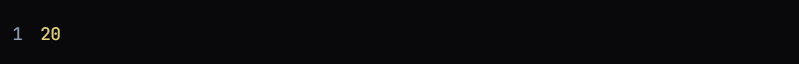
It's worth noting that NaN is the only value in JavaScript that is not equal to itself. You can verify this by using the strict equality operator (===):



**Question 6**

****

**Output**

****

**A white paper with black text

Description automatically generated**

Breaking it down:

1. Subtraction operator: The - symbol is typically used for subtraction in JavaScript.
2. Double negative (--): When you see - -10, this is actually a double negative. A double negative can be interpreted as a positive.

So, 10 - -10 is equivalent to:

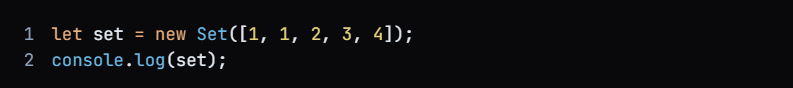
10+1010 + 1010+10

**Set Data Structure**

**A white paper with black text

Description automatically generated**

**Question 7**

****

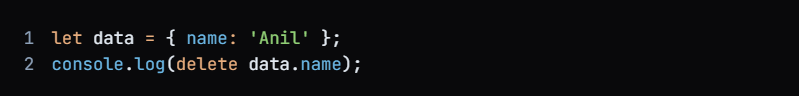
**Output**

****

**A screenshot of a white paper

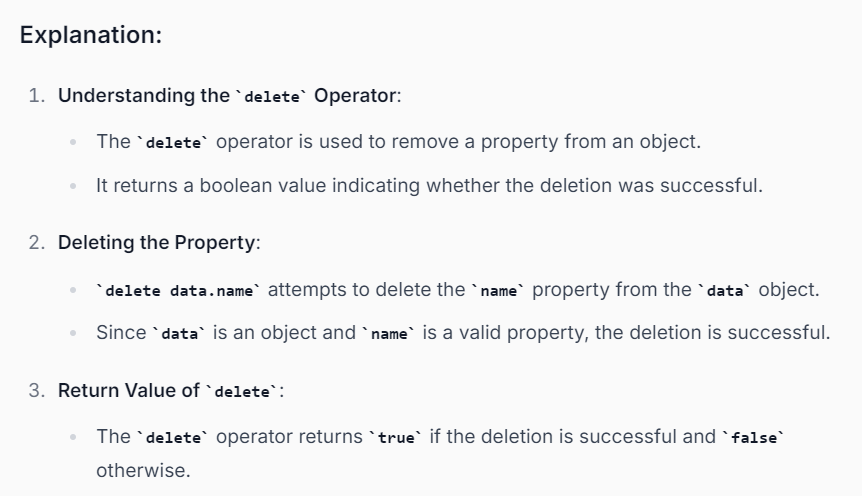
Description automatically generated**

**Question 8**

****

**Output**

****

****

**A screenshot of a computer program

Description automatically generated**

**Note:**

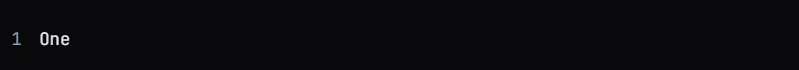
**A screenshot of a computer

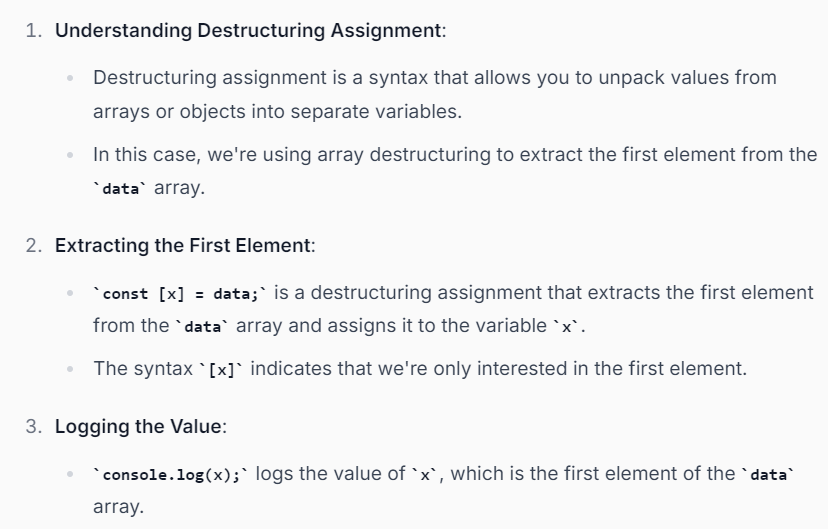
Description automatically generated**

**Question 9**

**A black background with green text

Description automatically generated**

****

****

**A screenshot of a computer program

Description automatically generated**

**Note :** If we want second element instead of getting 1st element from array then simple use ,

**Example**

A black background with white text

Description automatically generated

**Question 11**

**A screen shot of a computer

Description automatically generated**

**Output**

There are so many ways to access the object property without using .operator.

**Method 1**

A screenshot of a computer

Description automatically generated

**Method 2**

A screenshot of a computer program

Description automatically generated

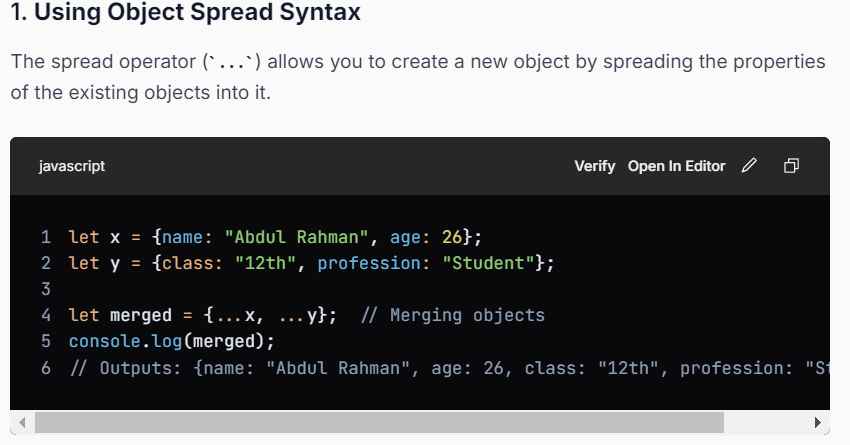
**Method 3**

A screenshot of a computer program

Description automatically generated

**Question 12**

How can we merge these two objects in a single variable?



**Question 13**

A screen shot of a computer

Description automatically generated

**Output**

A black background with text and numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

**Question 14**

A screen shot of a computer

Description automatically generated

What will be the value of skill property in final output?

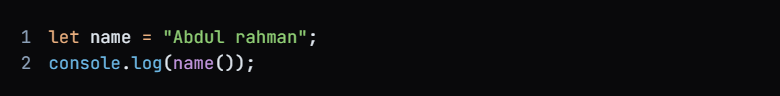
A person in a suit

Description automatically generated with medium confidence

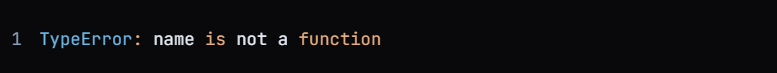
A close-up of a computer code

Description automatically generated

Question 15



Output



A screenshot of a computer error

Description automatically generated

**Question 16**

What is the Output of below code?

A black background with white text

Description automatically generated

Output

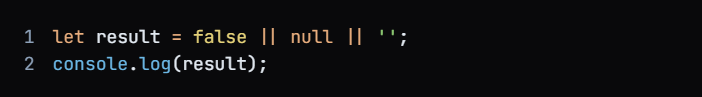
{}

1. **Logical OR (||) Operator**:
   * The **||** operator evaluates from left to right and returns the first truthy value it encounters. If all values are falsy, it will return the last value.
2. **Evaluation of Each Operand:**  
     
   false: This is a falsy value.  
   {}: This is an empty object, which is considered a truthy value.  
   null: This is also a falsy value.
3. **Evaluation Process:**  
     
   The first operand is false (falsy), so the evaluation moves to the next operand.  
   The second operand is {} (truthy). Since this is the first truthy value encountered, the evaluation stops here.

Note: This is an empty object, which is considered a truthy value in JavaScript.

**Question 17**

What will be the output?



Output

‘’

Explanation :

The **||** operator evaluates from left to right and returns the first truthy value it encounters. If all values are falsy, it will return the last value

In detail Check the explanation of Q.16

**Question 18**

What will be the output?

A black background with yellow text

Description automatically generated

Output : []

Explanation :

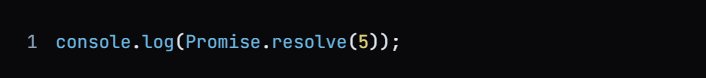
The **||** operator evaluates from left to right and returns the first truthy value it encounters. If all values are falsy, it will return the last value

In detail Check the explanation of Q.16

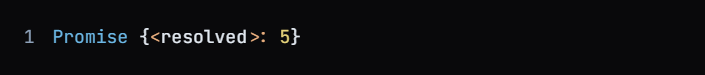
Note : This is an empty array, which is considered a truthy value in JavaScript.

**Question 19**

What will be the output?



Output



A screenshot of a computer code

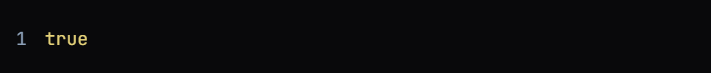
Description automatically generated

**Question 20**

What will be the output?



Output



**Explanation:**

1. **String Comparison**:
   * The **===** operator checks for both value and type equality. In this case, both sides of the operator are strings containing the same emoji character (🙂).
2. **Output of console.log()**:
   * Since both strings are identical, the comparison will evaluate to **true**.

Question 21

**What is the working of JSON.parse() ?**

1. Parse JSON to a javascript value.
2. Parse a javascript object to a JSON
3. Parse any Javascript value to a JSON
4. Parse JSON to a javascript object only

Answer

1. Parse JSON to a javascript value.

Question 22

What is the output?



Output

A screen shot of a computer

Description automatically generated

**Error Explanation**

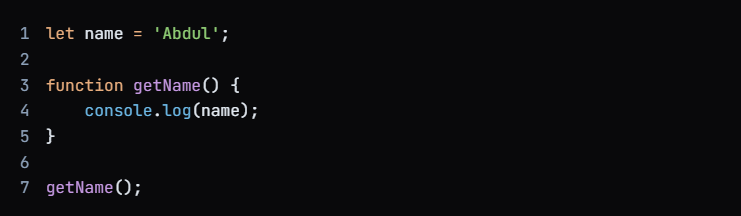
When you call **getName()**, the following happens:

1. The engine encounters **console.log(name);**. At this point, the local **name** variable is hoisted but not initialized, so it cannot be accessed.
2. This results in a **ReferenceError** stating that **name is not defined**.

**Variable Hoisting:**  
  
In JavaScript, variable declarations (but not initializations) are hoisted to the top of their scope. However, with let (and const), the variable is in a "temporal dead zone" until the declaration is reached.  
This means that when the function getName is called, the JavaScript engine sees the let name declaration at the top of the function scope but does not yet consider it initialized. Therefore, when you try to access console.log(name);, it refers to the local name, which is not yet initialized, leading to a ReferenceError.

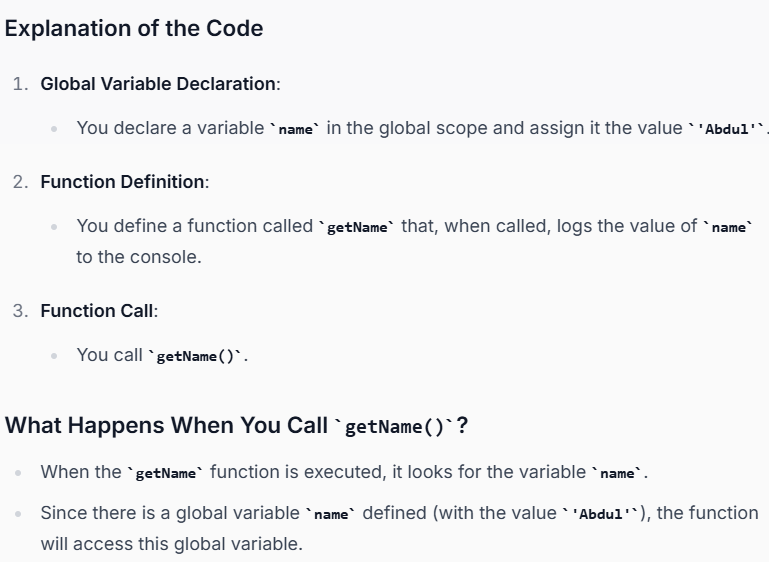
**Question 23**

What is the output ?



Output

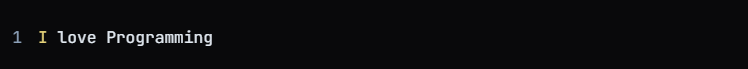




Question 24



Output

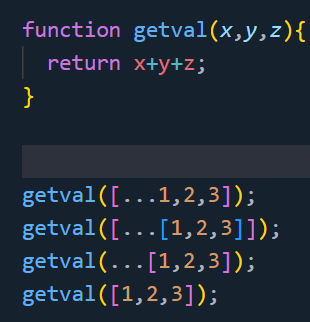


Explanation

1. **Arrow Function**:
   * The expression **(x => x)** defines an arrow function that takes one argument **x** and simply returns it. This is a concise way to create a function that does nothing but return its input.
2. **Function Invocation**:
   * The arrow function is immediately invoked with the argument **'I love'**. So, **(x => x)('I love')** executes, and it returns the string **'I love'**.
3. **Template Literal**:
   * 9jb9uLSsa;clkjas df’I,The result of the function invocation (**'I love'**) is then used inside a template literal, which is enclosed in backticks (**`**).
   * The template literal allows for embedding expressions inside the string using **${...}** syntax.

**Question 25**

Which one is correct ?

****

**Output**

Option c

**Explanation**

A screenshot of a computer code

Description automatically generated

A white text with black text

Description automatically generated with medium confidence

**Question 26**

**A computer code with text

Description automatically generated with medium confidence**

**Output**

**A blue rectangle with white dots

Description automatically generated**

**Explanation**

To understand the problem we understand that there ae two operators (! , ===) and in js the operators work from left to right

So, first ‘!’ will work then ‘===’ will work

A screenshot of a computer program

Description automatically generated

A computer code with text

Description automatically generated with medium confidence

**Question 27**

A computer screen shot of text

Description automatically generated

OutputA black and white screen with white text

Description automatically generated

Note : **isNaN()** checks if the value is Not-a-Number (NaN). In this case, **name** is a string **"ABdul Rahman"**.

**Expression 1: isNaN(name)**

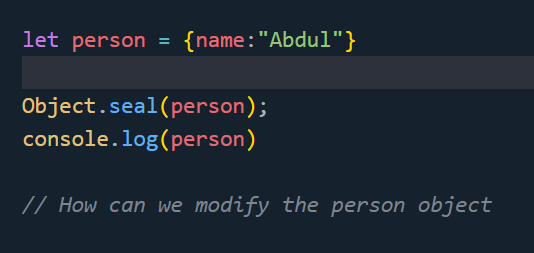
* Since **name** is a string, it cannot be converted to a number, so **isNaN(name)** returns **true**.

**Expression 2: isNaN(age)**

**age is a number 21.**

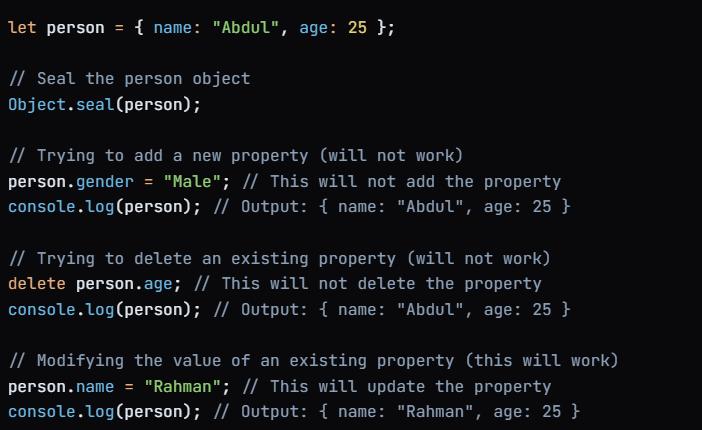
* **Since age is a valid number, isNaN(age) returns false.**

**Question 28**

****

**Explanation**

**Object.seal()** is a method in JavaScript that is used to prevent new properties from being added to an object and to prevent existing properties from being removed. However, it allows modifications to the values of existing properties.



**Question 29**

A close up of a screen

Description automatically generated

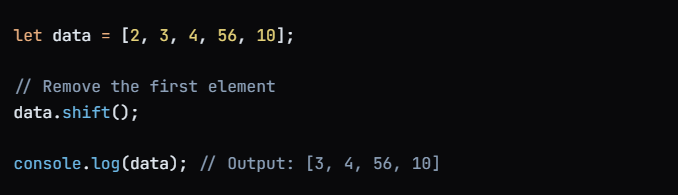
Explanation

We use shift method to remove the 1st element from array.

**Shift Method**

* The shift() method removes the first item of an array.
* The shift() method changes the original array.
* The shift() method returns the shifted element.
* If the length property is 0, undefined is returned.

**Output**



**Other Methods to remove the array element**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

Question 30

**A blue background with white text

Description automatically generated**

**Explanation**

We use pop() method to remove last element from array

**Pop Method**

* The pop() method removes (pops) **the last element** of an array.
* The pop() method changes the original array.
* The pop() method returns the removed element.

Output

A screen shot of a computer

Description automatically generated

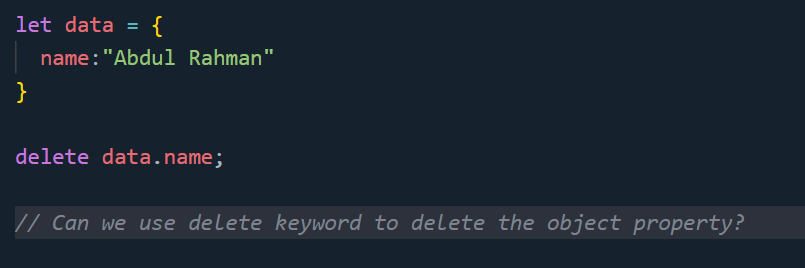
Question 31

How to check whether a number is even or odd?

A black background with text and colorful letters

Description automatically generated with medium confidence

**Question 32**



Explanation

Yes we can delete the object property using delete keyword.

Output

A blue rectangle with white dots

Description automatically generated

We get blank output because we have only single property in the object after deleting the property we get blank object.

**Question 33**

A screen shot of a computer

Description automatically generated

Explanations

We use ! sign to change the value

Output

False

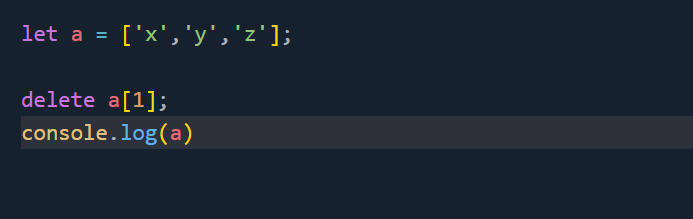
**Question 34**

What is the difference between map and forEach ?

A screenshot of a computer

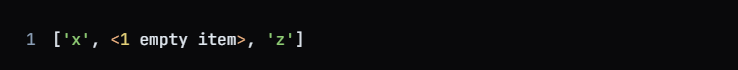
Description automatically generated

**Question 35**



What is the output of a?

Output



**Explanation**

* The **delete** operator removes the element at the specified index but does not reindex the array. Instead, it leaves an "empty" slot (or "hole") in the array.
* As a result, the original array **a** will still have a length of 3, but the second element (at index 1) will be **undefined**, and it will show as an empty slot in the console output.

OR

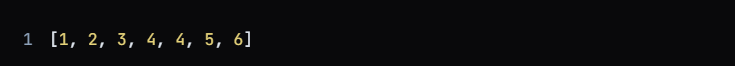
If we delete the element from an array if just create an empty slot but the length of the array remains same.

**Question 37**

A screen shot of a computer

Description automatically generated

Output



Explanation

A screenshot of a computer

Description automatically generated

Note:

A close up of a text

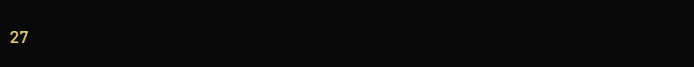
Description automatically generated

Question 38

A screen shot of a computer

Description automatically generated

Output



Explanation

In the provided code snippet, you are using the exponentiation operator (**\*\***) in JavaScript to calculate (3) raised to the power of (3):

The expression **3 \*\* 3** means (3) multiplied by itself (3) times:

**Question 39**

**A black background with a black square

Description automatically generated with medium confidence**

**What is the output of a?**

**Output**



A screenshot of a computer

Description automatically generated

Note: When we use setTimeout even we assign 0 millisecond then also it will execute in the last after the call stack will be empty.

**Question 40**

**A screen shot of a computer

Description automatically generated**

**Output**

**30**

**Explanation**

In JavaScript, variable names are case-sensitive, which means that a and A are treated as completely separate variables.

Here’s a breakdown:

* a refers to a variable with the value 2.
* A refers to a separate variable with the value 30.

**Question 41**

Can we use number as a variable in js?

**Ans:** No, in js we cannot use.

Ex: let 10, let 10A – not allow

**Question 42**

**A computer screen shot of a computer screen

Description automatically generated**

**Output**

True

**A screenshot of a computer

Description automatically generated**

**Question 43**

**A computer screen shot of a computer

Description automatically generated**

**Output**

****

**Explanation**

**A screenshot of a computer code

Description automatically generated**

**Question 44**

**A computer screen shot of a computer

Description automatically generated**

**Output**

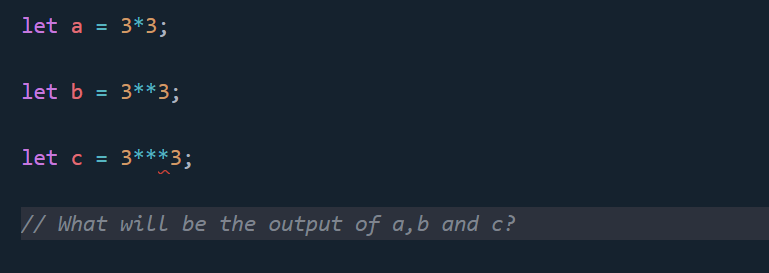
****

**Explanation**

**A screenshot of a computer program

Description automatically generated**

**Question 45**



Output

A close-up of a person

Description automatically generated

Explanation

A screenshot of a white page with black text

Description automatically generated

**Question 46**

A screen shot of a computer

Description automatically generated

**Output:**

* The output of **console.log(a);** will be **undefined**.

**Explanation:**

1. When the **console.log(a);** line is executed, the variable **a** has been declared (due to hoisting), but it has not yet been initialized with a value. Therefore, it will output **undefined**.
2. After the **var a;** line, **a** remains **undefined** unless it is assigned a value later in the code.

**Question 47**

**A computer screen shot

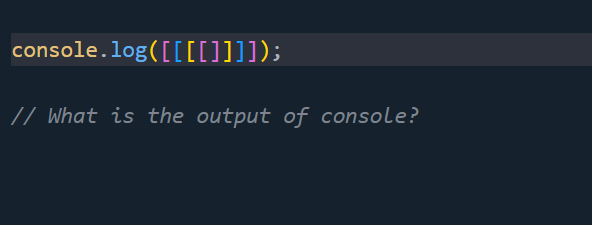
Description automatically generated**

**Output**

**A white background with black text

Description automatically generated**

**Question 48**

****

**Output**

**A screenshot of a computer program

Description automatically generated**

**Question 49**

How to find the OS name usings js?

**Output**

**A white background with black text

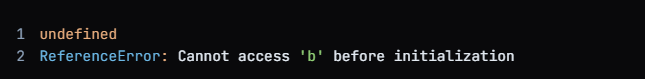
Description automatically generated**

**Question 50**

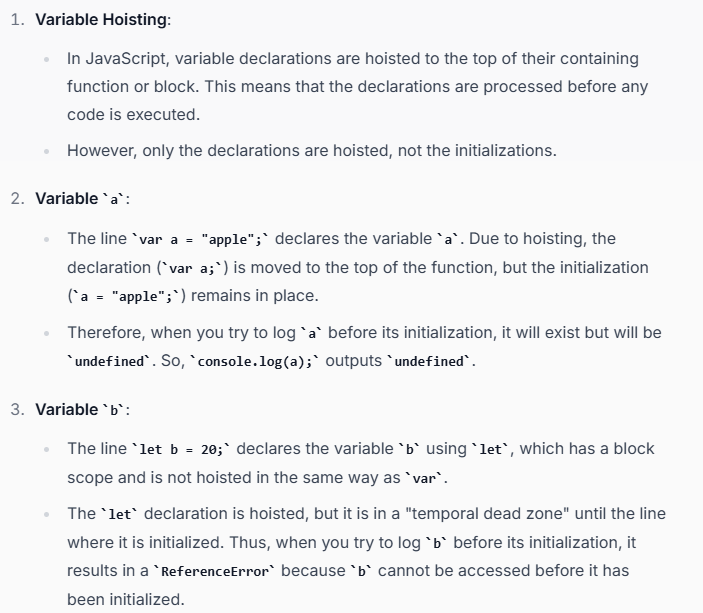
**A screen shot of a computer code

Description automatically generated**

**Output**

****

**Explanation**

****

**Question 51**

**A screen shot of a computer screen

Description automatically generated**

**Output**

**A blue background with black lines

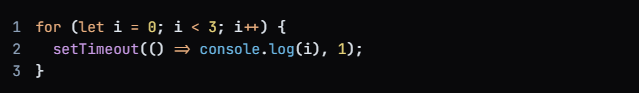
Description automatically generated**

**Explanation**

**A screenshot of a computer

Description automatically generated**

**Question 52**

****

**Output**

**A black background with a black square

Description automatically generated with medium confidence**

**Explanation**

This is because let creates a new block scope for each iteration of the loop, so each callback retains its own reference to the value of i at the time the callback was created.