



```
1  What will be the output of this code?
2
3  let x=5;
4  let y=x;
5  x=10;
6  console.log(x); // output-- 10
7  console.log(y); // output-- 5
```

- **let x = 5;**: Here this line declares a variable **x** and assigns it the value of **5**.
- **let y = x;**: Here, we declare another variable **y** and assign it the value of **x**. Since **x** is currently **5**, **y** also gets the value **5**.
- **x = 10;**: This line updates the value of **x** to **10**. And here, **y** still holds the original value of **5** because it was assigned after the **x = 5** and above the updated value so we know that js executes code line by line so **y** still holds the original value of **5**.
- **console.log(x);**: This prints the current value of **x**, which is updated as **10**.
- **console.log(y);**: This prints the value of **y**, which remains **5**, as it was set before **x** was changed.



```
1  2.What will be the output of this code?  
2  let obj1={ name:"Alice"};  
3  let obj2=obj1;  
4  obj1.name="Bob";  
5  console.log(obj1.name); // output-- Bob  
6  console.log(obj2.name); // output-- Bob
```

```
let obj1 = { name: "Alice" };
```

- It creates an object with a property **name** set to **"Alice"** and assigned it to **obj1**.

```
let obj2 = obj1;
```

- Now it makes **obj2** reference the same object as **obj1**. Means when the property's value of **obj1** changes **obj2** also changes as the reference of the **obj1**.


```
obj1.name = "Bob";
```

- By this command it updates the **name** property of the object that both **obj1** and **obj2** reference also changes

```
console.log(obj1.name);
```

```
console.log(obj2.name);
```

- When we **console.log (obj1.name)** and **console.log (obj2.name)**, both outputs we get are **"Bob"** because they are both referring to the same updated object.



```
1  3.
2      let a="hello";
3      let b=42;
4      let c=true;
5      let d={key:"value"};
6      let e=null;
7      let f=undefined;
8
9      console.log(typeof a); // output-- string
10     console.log(typeof b); // output-- number
11     console.log(typeof c); // output-- Boolean
12     console.log(typeof d); // output-- object
13     console.log(typeof e); // output-- object
14     console.log(typeof f); // output-- undefined
```

**let a = "hello";**

- **typeof a** gives output as "string" because **a**'s value is written in double quotes so if we write any thing single quotes or double quotes it is a string.

**let b = 42;**

- **typeof b** gives output as "number" because **b**'s value is a number so we get type as number.

**let c = true;**

- **typeof c** gives output as "boolean" because **c** is given a boolean value.

**let d = { key: "value" };**

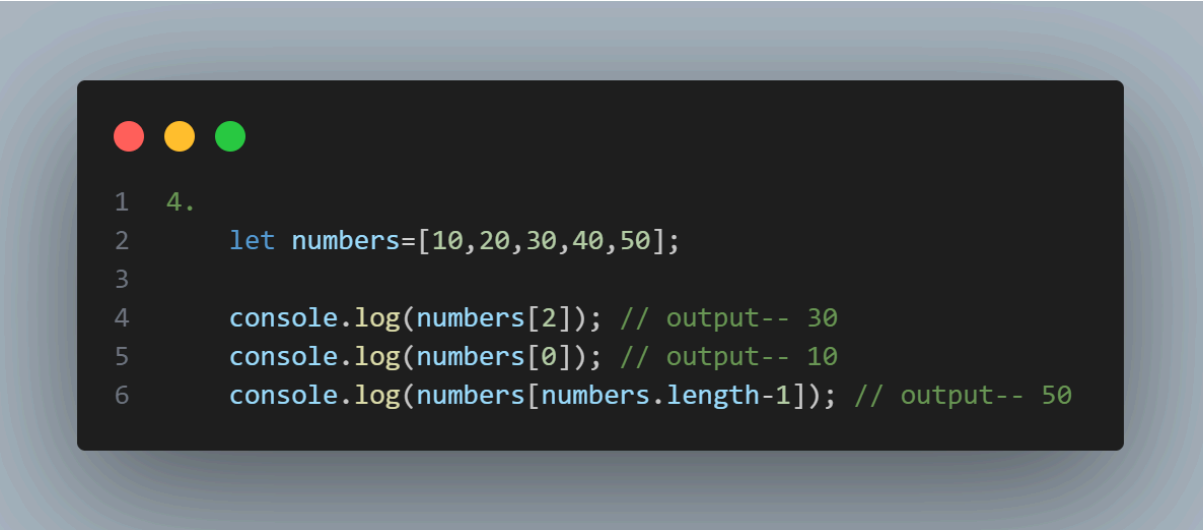
- **typeof d** gives output as "object" because we write property and its value in the flower bracket and we call it an object.

**let e = null;**

- `typeof e` gives output as "object" because `null` is programmed technically as an object type in JavaScript

`let f = undefined;`

- `typeof f` returns "undefined" because `f` has been explicitly set to `undefined`, which indicates the absence of a value. So when you call `typeof f`, the `typeof` operator checks the type of the value stored in `f`.



```
1  4.
2      let numbers=[10,20,30,40,50];
3
4      console.log(numbers[2]); // output-- 30
5      console.log(numbers[0]); // output-- 10
6      console.log(numbers[numbers.length-1]); // output-- 50
```

`let numbers = [10, 20, 30, 40, 50];`

- This creates an array with five elements: `10`, `20`, `30`, `40`, and `50`. And these elements are arranged in the indexes starting from `0` (for the first element) to `4` (for the last element).

`console.log(numbers[2]);:`


- Now this will access the element at index `2`, which is the third element in the array. Then the value at `numbers[2]` is `30`, so we get the output as `30`.

`console.log(numbers[0]);:`

- Now this will access the element at index `0`, which is the first element in the array. Then the value at `numbers[0]` is `10`, so we get the output as `10`.

`console.log(numbers[numbers.length - 1]);:`

- Here, `numbers.length` returns length of array that means the total number of elements in the array, which is `5`. `numbers.length - 1` gives us `4`, it is the index of the last element. Therefore, `numbers[numbers.length - 1]` means `numbers[4]` then it will access the element at index `4`, which is `50`, and will give the output as `50`.



```
1 5.  
2   let fruits=["apple", "banana", "mango"];  
3   fruits[1]="orange";  
4  
5   console.log(fruits); // output-- ['apple', 'orange', 'mango']
```

```
let fruits = ["apple", "banana", "mango"];
```

- Here we Declare Array where this creates an array with three elements: `"apple"`, `"banana"`, and `"mango"`. Each element is indexed as:

```
fruits[0] is "apple"
```

```
fruits[1] is "banana"
```

```
fruits[2] is "mango"
```

```
fruits[1] = "orange";
```

- Here, we're updating the value at index `1` (which was `"banana"`) to `"orange"`. After this operation, the array now looks like:

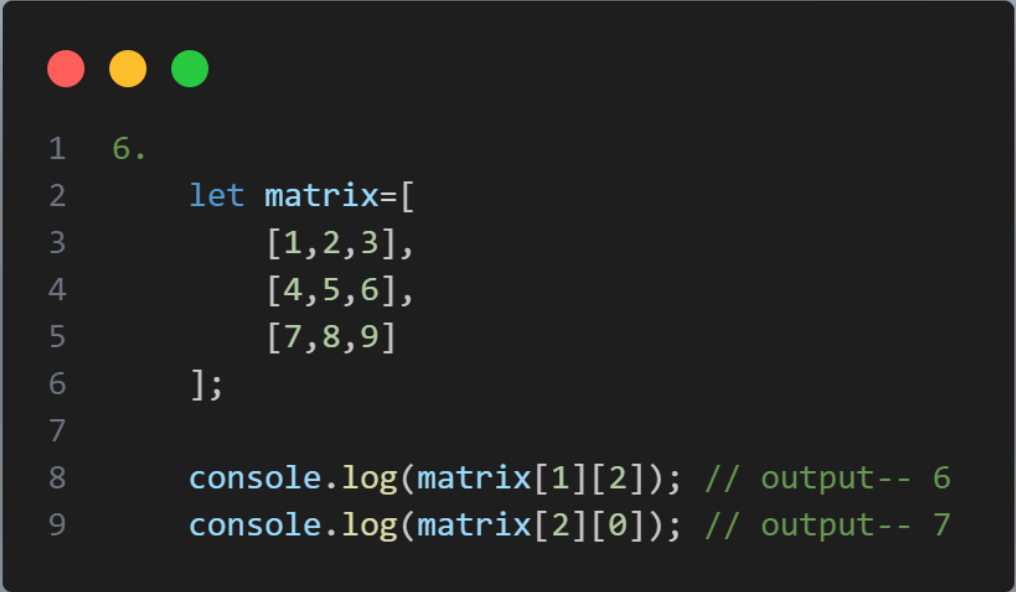
```
fruits[0] is "apple"
```

```
fruits[1] is "orange"
```

```
fruits[2] is "mango"
```

```
console.log(fruits);
```

- When we execute the `console.log(fruits)`, it give the output as `['apple', 'orange', 'mango']`, by reflecting the modification we made.



```
1 6.
2   let matrix=[
3       [1,2,3],
4       [4,5,6],
5       [7,8,9]
6   ];
7
8   console.log(matrix[1][2]); // output-- 6
9   console.log(matrix[2][0]); // output-- 7
```

```
let matrix = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ];
```

This creates a 2D array consisting of three rows:

- The first row (index 0): `[1, 2, 3]`
- The second row (index 1): `[4, 5, 6]`
- The third row (index 2): `[7, 8, 9]`

```
mconsole.log(matrix[1][2]);:
```

- `matrix[1]` refers to the **second row** (index 1): `[4, 5, 6]`.
- `matrix[1][2]` accesses the element at **index 2** in the second row, which is 6

```
console.log(matrix[2][0]);:
```

- `matrix[2]`: this refers to the **third row** (index 2): `[7, 8, 9]`.

- `Matrix[2][0]`: it will access the element at **index 0** in the third row, which is **7**.



```
let person = {  
  name: "Jhon",  
  age: 25,  
  agecity: "New York"  
};
```

- Here we have created an object with three properties as `name`: a string with the value "Jhon", `age`: a number with the value 25, `agecity`: a string with the value "New York".

```
console.log(person.name);:
```

- Here we are accessing the `name` property of the `person` object. it gives output as "Jhon".

```
console.log(person.age);:
```

- Here we are accessing the `age` property of the `person` object. it gives output as 25.



```
1 8.  
2   let car={  
3       make:"Toyota",  
4       model:"Corolla",  
5       year:2021  
6   };  
7   console.log(car["make"]); //output-- Toyota  
8   console.log(car["model"]); //output-- Corolla
```

```
let car = {  
    make: "Toyota",  
    model: "Corolla",  
    year: 2021  
};
```

We have Declared an object with three properties:

- Property name as `make`: a string with the value "Toyota"
- Property name as `model`: a string with the value "Corolla"
- Property name as `year`: a number with the value 2021

```
console.log(car["make"]);:
```

- Here we are accessing the `make` property of the `car` object using bracket notation. so that it gives the output as "Toyota".

```
console.log(car["model"]);:
```



- Here we are accessing the `model` property of the `car` object, also using bracket notation. so that it gives the output is `"Corolla"`.



```
1 9.
2   let book = {
3       title: "The Great Gatsby",
4       author: "F.Scott Fitzgerald"
5   };
6   book.author = "Anonymous";
7   console.log(book.author); //output-- Anonymous
```

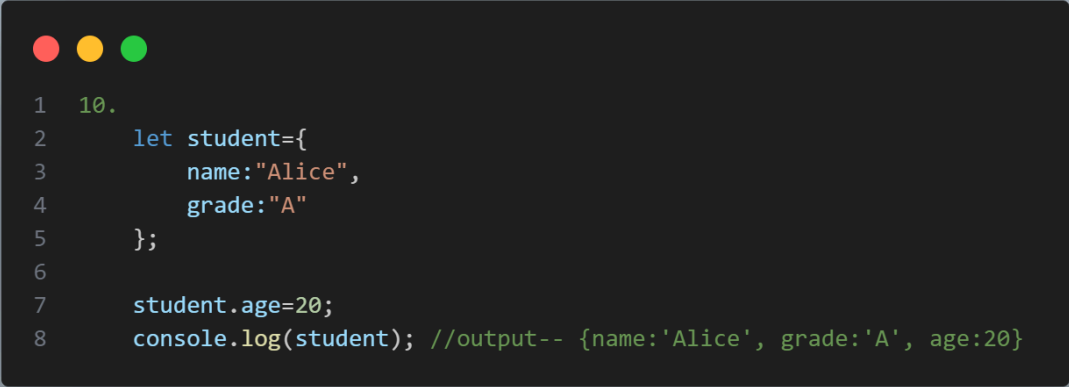
In your question, we have an object called `book` that initially contains properties for a book's title and author.

**`book.author = "Anonymous";`**

- Here, we have updated the `author` property of the `book` object. This will change the value from `"F.Scott Fitzgerald"` to `"Anonymous"`.

**`console.log(book.author);`**

- When we give `console.log(book.author)`, it gives the output as `"Anonymous"`, by reflecting the change we have made.



```
1 10.
2   let student={
3       name:"Alice",
4       grade:"A"
5   };
6
7   student.age=20;
8   console.log(student); //output-- {name:'Alice', grade:'A', age:20}
```

In the given question, we have an object called `student` that initially contains properties for the student is, name and grade.

**`student.age = 20;`**

- Here, we're adding a new property called `age` to the `student` object and assigning it a value `20`. This modifies the object to now include this new property.

**`console.log(student);`**

- When we give `console.log(student);`, it gives the output as the entire object, with including the newly added `age` property also as: `{ name: 'Alice', grade: 'A', age: 20 }`.

