JS DAY2 TASK

1. **What will be the output?**

```
let x=5;
let y=x;
x=10;
console.log(x); //output---10
console.log(y); //output---5
```

Explanation:

`let x = 5;`: A variable `x` is declared and assigned the value `5`.

`let y = x;`: A new variable `y` is declared and assigned the value of `x` (which is `5` at this point). Note that `y` holds a **copy** of the value of `x`, not a reference.

`x = 10;`: Now, the value of `x` is updated to `10`. However, since `y` holds a copy, it still retains the original value `5`.

`console.log(x);`: This prints the current value of `x`, which is `10`.

`console.log(y);`: This prints the value of `y`, which is still `5` (unchanged by the assignment to `x`).

2. **What will be the output?**

```
let obj1 = { name: "alice"};
let obj2 = obj1;
obj1.name = "Bob";
console.log(obj1.name);  //output---Bob
console.log(obj2.name);  //output---Bob
```

Explanation:

`let obj1 = { name: "alice" };` creates an object `obj1` with a property `name` set to `"alice"`.

`let obj2 = obj1;` assigns **a reference** to the same object, not a copy, to `obj2`.

`obj1.name = "Bob";` changes the `name` property of the original object.

Since **both `obj1` and `obj2` reference the same object**, the change is reflected in both.
`console.log(obj1.name);` and **`console.log(obj2.name);`** both print `"Bob"`.
In JavaScript, **objects are passed by reference**, meaning both variables point to the same underlying data.

3.

```
let a = "hello";
let b = 42;
let c = true;
let d = {key:"value"};
let e = null;
let f = undefined;

console.log(typeof a);  //output-- string
console.log(typeof b);  //output-- number
console.log(typeof c);  //output-- boolean
console.log(typeof d);  //output-- object
console.log(typeof e);  //output-- object
console.log(typeof f);  //output-- undefined
```

Explanation:
`typeof a`
`"hello"` is a string, so `typeof a` returns `"string"`.
`typeof b`
`42` is a number, so `typeof b` returns `"number"`.
`typeof c`
`true` is a boolean, so `typeof c` returns `"boolean"`.
`typeof d`
`{ key: "value" }` is an object, so `typeof d` returns `"object"`.
`typeof e`
Even though `e` is `null`, `typeof e` returns `"object"`.
This is a **known quirk** in JavaScript—`null` is not actually an object, but `typeof` treats it as one.

`typeof f`
`f` is `undefined`, so `typeof f` returns `"undefined"`.

4.

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

console.log(numbers[2]);  //output-- 30
console.log(numbers[0]);  //output-- 10
console.log(numbers[numbers.length-1]);  //output-- 50
```

Explanation:
`numbers[2]`
Accesses the element at **index 2**, which is `30`.
`numbers[0]`
Accesses the element at **index 0**, which is `10`.
`numbers[numbers.length - 1]`
`numbers.length` gives the length of the array, which is `5`.
`numbers[5 - 1]` or `numbers[4]` accesses the **last element**, which is `50`.

5.

```
let fruits = ["apple", "banana", "mango"];
fruits[1] = "orange";

console.log(fruits) //output-- [ 'apple', 'orange', 'mango' ]
```

Explanation:
`let fruits = ["apple", "banana", "mango"];`
Creates an array named `fruits` with three elements: `"apple"`, `"banana"`, and `"mango"`.
`fruits[1] = "orange";`
Replace the element at **index 1** (which is `"banana"`) with `"orange"`.

Arrays in JavaScript are **zero-indexed**, meaning the first element is at index `0`, the second at `1`, and so on.

`console.log(fruits);`

Prints the updated array:

6.

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

console.log(matrix[1][2]); //output-- 6
console.log(matrix[2][0]); //output-- 7
```

Explanation:

`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`.

`matrix[2][0]`

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

`matrix[2][0]` accesses the element at **index 0** in the third row, which is `7`.

7.

```
let person = {
   name: "john",
   age:25,
   city: "new york"
};

console.log(person.name); //output-- john
console.log(person.age);  //output-- 25
```

Explanation:

`let person = { ... }`

Creates an **object** `person` with three properties:

`"name": "john"`
`"age": 25`
`"city": "new york"`
`console.log(person.name);`
Accesses and prints the `name` property of the `person` object, which is `"john"`.
`console.log(person.age);`
Accesses and prints the `age` property of the `person` object, which is `25`.

8.

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

console.log(car["make"]); //output-- Toyota
console.log(car["model"]); //output-- corolla
```

Explanation:
`car["make"]`
This accesses the **`"make"`** property of the `car` object, which holds the value **`"Toyota"`**.
`console.log(car["make"]);` prints **`Toyota`**.
`car["model"]`
This accesses the **`"model"`** property of the `car` object, which holds the value **`"corolla"`**.
`console.log(car["model"]);` prints **`corolla`**.

9.

```
let book = {
   title: "The Great Gatsby",
   author: "F. Scott Fitzgerald"
};

book.author = "anonymous";
console.log(book.author);  //output-- anonymous
```

Explanation:

Initial Object Creation:

The `book` object is created with two properties:

`title`: `"The Great Gatsby"`

`author`: `"F. Scott Fitzgerald"`

`book.author = "anonymous";`

This line updates the value of the `author` property from `"F. Scott Fitzgerald"` to `"anonymous"`.

In JavaScript, object properties can be modified directly.

`console.log(book.author);`

This prints the updated value of the `author` property, which is now `"anonymous"`.

10.

```
let student = {
   name: "alice",
   grade: "A"
};

student.age = 20;
console.log(student); //output-- { name: 'alice', grade: 'A', age: 20
}
```

Explanation:

Initial Object Creation:

The `student` object is created with two properties:

`name`: `"alice"`

`grade`: `"A"`

`student.age = 20;`

This line adds a new property, `age`, to the `student` object and assigns it the value `20`. In JavaScript, you can dynamically add properties to objects after they have been created.

`console.log(student);`

This prints the entire `student` object, which now includes the newly added `age` property. The output is: