Objects

Objects are fundamental part of javascript, providing way to group related data and functions together. In JS, an object is a collection of key-value pairs, where each key is a string (or a symbol) and each value can be any data type, including other objects.
Objects can have properties and methods, making them versatile for various use cases.

→ Syntax → obj = { }

So, similar data types → Array
Multiple data types = Objects

Creating Objects

There are several ways to create objects in JS.
The most common one is Object Literals

```
Const product = {
        id : 1,
      pName : "Laptop",
};
```

[Key : Value]

```
let person = {
    name : "Vinod",
    age : 30,
    isStudent : false,
    greet : function () {
        console.log ("Welcome");
    },
};
```

when functions used as object then it is called as methods.

So, we can store string, numbers, boolean, even methods as value.

~~The~~ Accessing Properties

We can access object properties using dot notation or square bracket notation:

```
Console.log (person.age);   => 30
```

Note → If key is in the form of string or any special char. then we [ ] with " ", ' ', ` `;

i.e.
```
let person = {
    "aman" : person,
};
```
→ ~~person~~ person.aman = ✗
person [aman] ✗
person [" aman "] ✓ → person
person[' aman '] → person
person [' aman '] → person

Adding or Modifying the properties

→ As we know in our person object, there ~~is~~ is no job property. So, we can add that:

```
person.job = " web Developer ";
```

→ to modify age, `person.age = 47 ;` / `person ["age"] = 47)`

→ On console, we will see that the changes occurred

Note :- On `console.log (person.greet)` → we will get as [Function : greet]

But to print the value of function or executing the function we write `person.greet()` ~~in~~ → Welcome

//* We can add dynamic keys in an object (we can get username in react)

Dynamic keys :-? It is a key whose value is same but its key is different.

eg:- lets assume the different idType have same id as, ~~a~~ studentId, collegeId, etc, all are (A1234)

→ then we can specify a dynamic key within square bracket [ ] in object and the ~~value of~~ the key will be different as passed from outside.

eg

```
let idType = "studentId";
let student = {
    [idType] : "A123456",    // Dynamic key based on idType
    sName : "Aman",
    sAge : 29,
    greet : function () {
        console.log (
            `Hey! my ${idType} is ${student[idType]}
            and my name is ${student.sName}.`);
    },
};
student.greet();
```

output
↓

Hey! my studentId is A123456 and my name is Aman

If idType = "CollegeId", then ~~outpt~~ output:

~~Hey! my studentId~~ "

Hey! my collegeId is A123456 and my name is Aman

Data Modelling

Data Modelling is the process of creating a visual representation of an information system or its parts to show connection between data points and structures.

Think of it as creating a map for your data. Just like a map shows you how different places are connected, data modelling shows how data are related !

Purpose :- The goal is to understand what types of data we have, how they are connected ~~too~~ and how they can be organised. This helps in designing databases and systems that store and manage data efficiently.

Objects :- Objects are great for modelling real world entities. For eg:, a car, a product, or a user can be represented as objects with properties like color, brand, and username

```
let car = {
    brand : "Toyota",
    model : "Camry",
    year : 2022,
    Start : function () {
        Console.log ("Engine started!");
    },
};
```

Questions for Interview

(Q) Explain the difference between passing objects by reference and by value in Javascript. Provide an example to demonstrate each scenario.

⇒ In JS, primitive data types like numbers & strings are passed by value, while objects are passed by reference.

<u>Passing by value</u> :- when passing by value, a copy of the primitive value is created and passed to the function or assigned to a variable. Any changes made to the copy do not affect the original value.

eg:-
```
let a = 10;
const modifyValue = (x) => (x = 20);
console.log(modifyValue(a));          → 20
console.log(a);                       → 10
```

<u>Output</u>

So the original value didn't change

<u>Passing by reference</u> :- when passing by reference, a reference to the memory location of the object is passed to the function or assigned to a variable. Any changes made to the object through this reference will affect the original object.

eg:
```
let obj = { ~~a~~ name: "Aman" };
let obj1 = obj;
obj1.name = "Aman Agrawal";
console.log(obj1);              → { name: 'Aman Agrawal
console.log("original", obj);   → original { name: 'Aman
                                          Agrawal }
```

<u>Output</u>

BOOM → it changed

Q How to avoid this?

To avoid this behaviour and create a true copy of the object, we can use methods like Object.assign() or the spread operator (...);

Object.assign() is used to copy properties from one or more source objects to a target object.

eg
```
let Obj = { name: "Aman" };
let new Obj = Object.assign({}, obj);
new Obj.name = "thapa technical";
console.log(new Obj);           → { name: 'Aman' }
console.log("original", obj);   → original { name: thapa technical }
```

Q Comparison by Reference

Two objects are equal only if they refer to the same object.

→ Independent objects (even if they look alike) are not equal.

```
const Obj1 = { name: "Aman" }
const obj2 = { name: "Aman" }
const obj3 = obj1
```
so, obj1 == obj3
→ true
But obj1 == obj 2
→ false

Q Difference b/w JSON & obj

Though they look similar but

Obj = { name: 'Aman', id: 12, isMan: true }

JSON → { "name": "Aman", "id": 12, "isMan": true }

Key are in double quotes in JSON.