—JavaScript—

1.Class 1

console.log(”Hello From Chai”) —> throughj console.log we print string

To Run Java Script file → use → node filename.js

Functions →

```
function printChai()
{
    console.log("Hello From Chai");
}

function bringBrush(numOfBrush)
{
    console.log(`Hanji, le aaya ${numOfBrush} brush`);
}

function addTwo(num1, num2)
{
    return num1+num2;
}

bringBrush(4);
printChai();

var result = addTwo(1,2);
console.log(result);
console.log("addition = ",addTwo(3,2));
```

op→

Hanji, le aaya 4 brush
Hello From Chai
3
addition = 5

2. Class 2

1.let → we can update in memory value of let variable

2.const → we can not change in memory value of const variable.

1.DataTypes

1.1. Primitve DataTypes

1.String

let name = “Manish”;

2.Number

let age = 25;

3.Boolean

let isPaid = true;

4.NULL→nothing , emptiness

let favouriteClass = NULL;

5.undefined→ ab tk iske andr kch bhi nh hai null, or kch bhi nh hai

let hometown;

6.Symbol

7.BigInt

1.2. NonPrimitive DataTypes

1.Array

let skills = [”HTML”, “CSS”, “JavaScript”];

how to define Array:

1. let fruits = [”apple”, “cherry”, “banana”];
2. let intFruits = new Array(”kiwi”, “avacado”, “dragon fruit”);

→ Both are same just diffrenet of defining.

console.log(typeof fruits) → object

console.log(typeof intFruits) → object

Note→ if you want to print apple

console.log(fruits[0]): op→ apple

console.log(fruits.length); op→ 3

change value of apple to “blueberry” → fruits[0] = “blueberry”

1.Methods of Array

1. fruits.push(”new Item”);→ add last
2. fruits.unshift(”new Items”); → add in front
3. fruits.pop(); → delete from last and return that element

referred→ MDN→Array

2.Object → key, valuePair

```
let studentProfile =
{
    name:”Manish”,
    age: 26,
    isPaid : true,
    skills: [”HTML”, “CSS”, “JavaScript”],
    favouriteClass : NULL
}
```

Note→ console.log(typeof name);

op→ String

through this we can can get dataType of variable.

2.Conditionals

2.1. If else

```
let weather = “rainy”;
if(weather === “rainy”) {
    console.log(”Its raining, umbrella le janana bhai”);
} else {
    console.log(”else part hai ji”);
}
```

2.2. Nested If else

EX1:

```
let weather = “rainy”;
if(weather === “rainy”) {
    console.log(”Its raining, umbrella le janana bhai”);
} else if(weather == “cloudy”) {
    console.log(”else part hai ji”);
} else {
    console.log(”Sunny hai ji aj to”);
}
```

EX2:

```
let numberofGuest = 4;
let pizzaSize ;
if(numberofGuest ≤ 2) {
    pizzaSize = “Small”;
} else if(numberofGuest ≤ 5) {
    pizzaSize=”Medium”;
} else {
    pizzSize = “Large”;
```

}

console.log(`order pizza $PizzaSize`);

EX3:

→GradeCalculator

```
function calculateGrade(marks)
{
    let grade;

    if(marks >= 90)
    {
        grade = "A";
    }
    else if(marks >= 80)
    {
        grade = "B";
    }
    else if(marks >= 70)
    {
        grade = "C";
    }
    else if(marks >= 60)
    {
        grade = "D";
    }
    else
    {
        grade = "F";
    }

    return grade;
}

let result = calculateGrade(80);
console.log(`Student got grade ${result}`);
```

op→ Student got grade B.

3.Loops

3.1.For Loops

let teas = [”masala”, “ginger”, “oolong”, “orange”, “lemon”];

for(int i=0;i<teas.length;i++) {

    console.log(`Tea at index ${h} : ${teas[h]}`);

}

ex: Sum of Array

```
let myArray = [1,2,3,4,5,6,7]

function add(myArray)
{
    let sum = 0;
    for(let i=0;i<myArray.length;i++)
    {
        sum = sum + myArray[i];
    }
    return sum;
}

console.log("Sum of Array is ",add(myArray));
```

op→ Sum of Array is 28

Challenges: Questions

```
//Problem1: Create an array containing different types of teas.
const teas = ["Green Tea", "Black Tea", "oolong Tea", "White Tea", "Herbal Tea",];
console.log(teas);

//Problem2: Add "Chamomile Tea" to the existing list of teas
teas.push("Chamomile Tea");

//Problem3: Remove "oolong Tea" from the list of tea
const index = teas.indexOf("oolong Tea");
if(index > -1)
{
    teas.splice(index, 1);
}

//Problem4: Filter the list to only include teas that are caffeinated.(herbal tea chod ke
const caffinetedTeas = teas.filter((tea) => tea !== "Herbal Tea");

//Problem 5: Sort the list of teas in alphabetical order.
console.log(teas.sort());

//Problem 6: Use for loop to print each type of tea in the array
for(let i=0;i<teas.length; i++)
{
    console.log(teas[i]);
}
```

```
//Problem 7: Use a for loop to count how many teas are caffeinated (excluding "Herbal Tea").
let count = 0;
for(let i=0;i<teas.length;i++)
{
    if(teas[i] != "Herbal Tea")
    {
        count++;
    }
}
console.log(count);

//Problem 8: Use a for loop to create a new array with all tea names in uppercase
const uppCaseTeas = [];
for(let i=0;i<teas.length;i++)
{
    uppCaseTeas.push(teas[i].toUpperCase);
}

//Problem 9: Use a for loop to find the tea name with the most characters.
let longest = "";
for(let i=0;i<teas.length;i++)
{
    if(longest.length < teas[i].length)
    {
        longest = teas[i];
    }
}
```

```
//Problem 10: Use a for loop to reverse the oreder of teas in the array

const reversedArray = [];
for(let i=teas.length-1;i>=0;i--)
{
    reversedArray.push(teas[i]);
}
console.log(reversedArray);
```

4.→*Objects - Data Structures in JS*

Data kis trh apn memory me store krte thats data structures

two ways we strore data

1.Primitive→ default like array, objects(key value pair)

2.Non-Primitive→ jo apn primitive use krke banate like stack, queue

Here We are Discussing Objects →

```
const person =
{
   x : 10,
   firtsName : "Manish",
   lastName:"Garg",
   hobbies: ["coding", "GYM"],
   isMarried: false,
   hasGf: false,
   getFullName: function()
   {
       return "Manish Garg"

   },
   address:
   {
       hno: 1,
       street: 1,
       contryCode: "IN",
       stateL: "MH",
   }
}
console.log(person.lastName);  'Garg'
console.log(person.hobbies);  [ 'coding', 'GYM' ]
console.log(person.getFullName());  'Manish Garg'
console.log(person.address)  { hno: 1, street: 1, contryCode: 'IN', stateL: 'MH' }
```

Use Cases of Objects:

1.For Grouping

Objetcs→ same concept jo class and opbject wala hai wahi ye bhi object hai.

```
const remote =
{
   color: "black",
   brand: "XYZ",
   dimesions: {height:1, width: 1},
   turnOff: function()
   {

   },
   volumeUp: function()
   {

   }

}
```

Memory Allocation → In Normal variable and Objects

```
//1.Normal-> Memory Allocation
let fname1 = "Manish";
let fname2 = fname;

console.log(fname1);  'Manish'
console.log(fname2);  'manish'

fname2 = "Piyush";

console.log(fname1);  'Manish'
console.log(fname2);  'Piyush'
//idhr copy bana rh
```

```
//2.Objects-> Memory Allocation
let p1 =
{
    fname1: "Manish",
}

let p2 = p1;
console.log(p2);  { fname1: 'Manish' }

p2.fname1 = "Piyush";
console.log(p2);  { fname1: 'Piyush' }
console.log(p1);  { fname1: 'Piyush' }
//but for objects its working as pointer
```

Copies:

1.Shallow Copy

const teaCopy = {...teas};

→upr upr copy kr rh mtlb object simple hai to har value copy ho jayegi when nested ayega uska reference lega bs.

2.Deep Copy

→string technique

→idhr har chij copy hogi reference vaigere kch nh rhga.

Variable→Memory

we have two types of memory

1.Stack→We store all variable

→Stack grow nh kr skta , its fixed

→To jo bhi grow hote rhta like objects, array can grow vo sb Heap me jate.

→but jo object reference or address rhega vo variable me sote rhega and vo store rhega stack me. its pointer

2.Heap

- whenever we need memory dynamically then we use Heap

- like Array, Objects
- jo object reference or address rhega vo variable me sote rhega and vo store rhega stack me. its pointer

```
let p1 =
{
   fname1: "Manish",
}

let p2 = p1;//this line from here p1 and p2 both pointing to same objects
console.log(p2);

p2.fname1 = "Piyush";
console.log(p2);//op→Piyush
console.log(p1);//op→Piyush
```

TradeOFF:

1.Stack

- Read→fast
- memory→fixed

2.Heap

- Read→slower as compare to stack

→ Because first we need to go to stack memory to get the address/reference of the object, then we need to go to heap memory to access the actual object data. These 2 memory hops make heap access slower than direct stack access.

- memory→ dynamic

→Garbage Collector in JS.

Garbage Collector (GC) in JavaScript is an automatic memory management system that helps free up memory that is no longer being used by the program.

- It automatically identifies and removes objects that are no longer reachable or used in the program
- GC runs periodically in the background without developer intervention
- Main purpose is to prevent memory leaks by cleaning up unused memory

How it works:

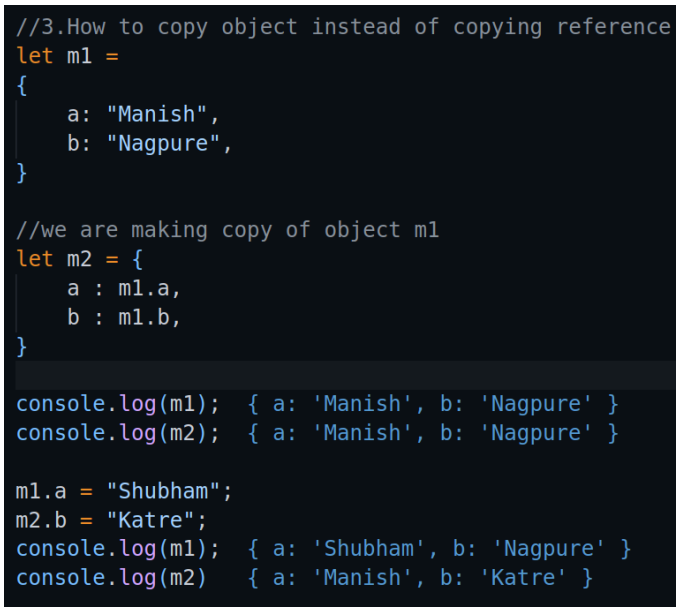
- Identifies objects that are no longer referenced by the program
- Marks these objects as eligible for garbage collection
- Removes marked objects and frees up the memory space

Note: Developers cannot directly control when garbage collection occurs, as it's handled by the JavaScript engine.

→Memory Leak in JS

A memory leak occurs when a program fails to release memory that is no longer being used, causing the application to consume more memory over time.

Ex→ like apn ne p1 ko clear kr diye and uski memory location ko bhi remove kr diye stack se. but p2 still pointing to p1's previous location but now it anything but we it still p2 is pointing at something and if we try to update it . it will update in someones else data. Thats is called memory leak. but this is not problem in languages like which have garbage collector.



```
//3.How to copy object instead of copying reference
let m1 =
{
    a: "Manish",
    b: "Nagpure",
}

//we are making copy of object m1
let m2 = {
    a : m1.a,
    b : m1.b,
}

console.log(m1);  { a: 'Manish', b: 'Nagpure' }
console.log(m2);  { a: 'Manish', b: 'Nagpure' }

m1.a = "Shubham";
m2.b = "Katre";
console.log(m1);  { a: 'Shubham', b: 'Nagpure' }
console.log(m2)   { a: 'Manish', b: 'Katre' }
```

→draw back it will take time if object is too big

```
//shortcut→using Spread Oprator
let m1 =
{
   a: "Manish",
   b: "Nagpure",
}

//we are making copy of object m1
let m2 = {
   ...m1//Spread Oprator
}
```

but using spread oprator we are just making shallow copy it means upr upr to copy kr lega simple variable rhe to but jb object ke andr object rhega tb vo andr ke object ko copy nh krga just uska address copy krga .....thats is called shallow copy

→to avoid this problem we have to make deep copy.

Technique for Deep Copy

1. Object→String→Copy→Object

```
let m1 =
{
   fname: "Manish",
   lname: "ch",
   adress:
   {
     h: 1,
     s: 1,
   }
}

const m1KaString = JSON.stringify(m1);
console.log(m1KaString);  '{"fname":"Manish","lname":"ch","adress":{"h":1,"s":1}}'
let m2 = JSON.parse(m1KaString);

m2.fname = "Shubham";
m2.adress.h = "hacked";

console.log(m1);  { fname: 'Manish', lname: 'ch', adress: { h: 1, s: 1 } }
console.log(m2);  { fname: 'Shubham', lname: 'ch', adress: { h: 'hacked', s: 1 } }
```

H.W→

- [ ] Read Basic Objects in JavaScript

Book

- Who Moved My Cheese
- **Focus on What Matters**

→ For Loop in Object

Use a for...in loop to print all properties of the tea object.

```
for(let it in tea)
{
  console.log(it, tea[it]);
}
```

→Practice

```javascript
//Problem 1: Create an object representing a type of tea with properties for name, type, and caffeine content.
let tea =
{
    name : "green Tea",
    type : "Bitter",
    caffeine_content : "High",
}

//Problem 2: Acess and print the named and type properties of the tea object.
console.log(tea.name);  'green Tea'
console.log(tea["type"]);  'Bitter'

//Problem 3: Add a new property origin to the tea object.
tea["origin"] = "Allahabad";
console.log(tea);  { name: 'green Tea', type: 'Bitter', caffeine_content: 'High', origin: 'Allahabad' }

//Problem 4: Change the caffeine level of the tea object to "Medium".
tea.caffeine_content = "Medium";
console.log(tea);  { name: 'green Tea', type: 'Bitter', caffeine_content: 'Medium', origin: 'Allahabad' }

//Problem 5: Remove the type property from the tea object.
delete tea.type;
console.log(tea);  { name: 'green Tea', caffeine_content: 'Medium', origin: 'Allahabad' }

//Problem 6: Check if the tea object has a property origin.
if(tea.origin)
{
    console.log("Yes Property Exist");  'Yes Property Exist'
}
else
{
    console.log("I dont have Property");
}
```

```javascript
//Problem 7: Use a for...in loop to print all properties of the tea object.
for(let it in tea)
{
   console.log(it, tea[it]);  [ 'name', 'green Tea' ], [ 'caffeine_content', 'Medium' ], [ 'origin', 'Allahabad' ]
}

//Problem 8: Create a nested object representing different types of teas and their properties.
let tea1 = {
    name: "Black Tea",
    types: {
        black: {
            flavor: "Strong",
            caffeine_content: "High"
        },
        green: {
            flavor: "Mild",
            caffeine_content: "Medium"
        },
        herbal: {
            flavor: "Fruity",
            caffeine_content: "None"
        }
    }
};
console.log(tea1);  ... green: { flavor: 'Mild', caffeine_content: 'Medium' }, herbal: { flavor: 'Fruity', caffeine_content: 'None' }

//Problem 9: Create a copy of the tea object.
const teaCopy =
{
    ...tea,
}
console.log(tea);//it shallow copy  { name: 'green Tea', caffeine_content: 'Medium', origin: 'Allahabad' }
```

5.Prototype

→Its nothing but object.which hold all the methods related to actual object, array, string etc.

Ex→ If suppose we defined Array and array have built in methods like .map, .forEach, .filter etc. so what happen all this method hold by prototype. this called prototype.

Array→ Our Main default array

Actual → Array.prototype = {all the array properties}

lets take example→ const arr = [1,2,3]

arr.__proto__.map;

here arr.__proto__ = Array.prototype;

this is way we have to use all the methods related to arr . but now we can ignore that word .prototype . we can directly use arr.map; it will work fine.