JS-2

Arrays, code execution , let & const and strings

Agenda

- Arrays
  - Arrays in JS
  - important methods of Arrays
- let vs const
  - stack and heap
- code execution in js
  - hoisting
  - window object
  - Execution context
- string and it's important methods

Array and it's important methods

```
console.log("JS class -2");

/******
 * Arrays
 * * array don't have a strict size
 * -> add , remove elements from it
 *
 *
 * ***/

let arr = [1, 2, 3, 4]
// let arr2 = [];

// print
console.log(arr2);
 console.log("arr",arr);

// iterate
```

```javascript
for (let i = 0; i < arr.length; i++) {
    console.log("index", i, "value: ", arr[i]);
}

/**************Important methods
 * 1. add last -> push
 * 2. remove last -> pop
 * 3. add first -> unshift
 * 4. remove first -> shift
 * ***************/
// 1. push() - add element at the end of the array
arr.push(50);
// console.log("after push", arr);
// //2 .pop() - remove element from the end of the array
arr.pop();
console.log("after pop", arr);
// // 3. unshift() - add element at the start of the array
arr.unshift(5);
// console.log("after unshift", arr);
// // 4. shift() - remove element from the start of the array
arr.shift();
// console.log("after shift", arr);
console.log(arr);

//5. slice -  input->  start index  , end index
// slice a copy the array from  sidx to edix - 1
// let slicedArr = arr.slice(1, 5);
// console.log("sliced Arr",slicedArr);
// console.log("original arr", arr);

//6. splice-> input -> start index , delete count
// splice original array me se element remove kr deta h
// const spliedArray = arr.splice(3, 2);
// console.log("removed elements", spliedArray);
// console.log("after splice", arr);

//7. indexOf - find the index of the element in the array
// console.log("index of 5", arr.indexOf(5));
// console.log("index of 30", arr.indexOf(30));
// //8. includes
// console.log("is element present", arr.includes(10));

//9. join - join the array elements with the specified separator
// let fruits = ["apple", "oranges", "banana"];
// let str = fruits.join("+");
// console.log("string:", str);



// function advancedManipulation(words) {
// let firstWord = words.shift();
// words.unshift("new");
// words.unshift(firstWord);
// // remove
// words.splice(2, 1);

// // join
// let joinedStr = words.join(",");
// return joinedStr
// }

// let words = ["apple", "banana", "cherry", "date"];
// let result = advancedManipulation(words);
// console.log(result);
```

reference and value types

In js there are two types of data types

- reference types
- value types

value type

- **Number**: 8 bytes

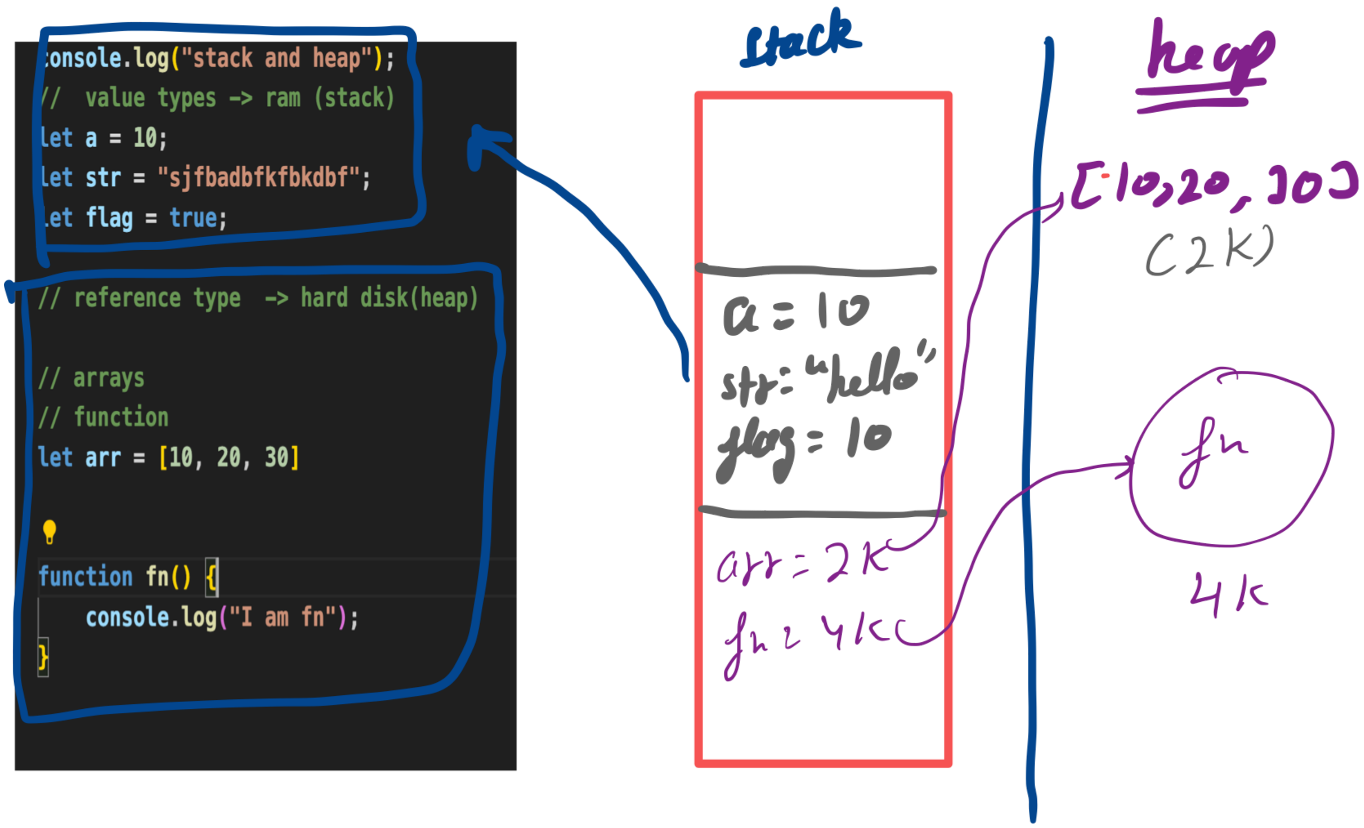
- **String**: 2 bytes per character + overhead
- **Boolean**: 1-4 bytes
- **null**: 4 bytes

- undefined: 4 bytes
  `These are stored on ram also known stack in our case`

reference type

We stroe there address on the stack and they are actually created on heap

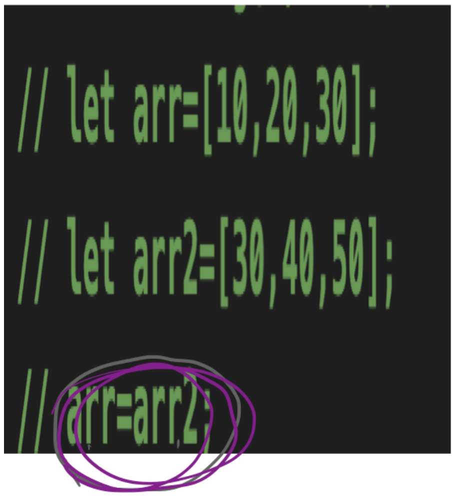
- Array
- functions
- objects



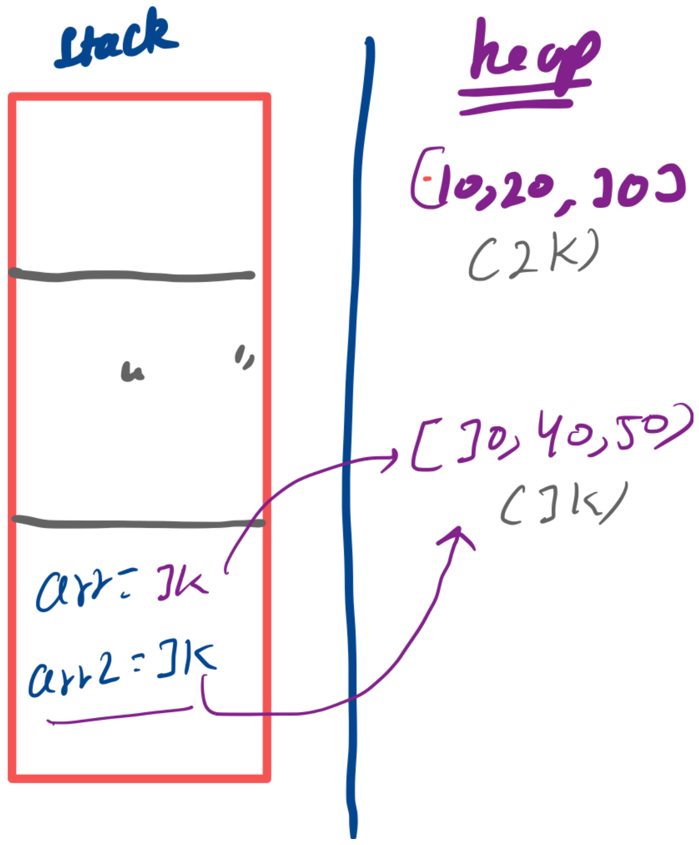
```
console.log("stack and heap");
//  value types -> ram (stack)
let a = 10;
let str = "sjfbadbfkfbkdbf";
let flag = true;

// reference type  -> hard disk(heap)

// arrays
// function
let arr = [10, 20, 30]

function fn() {
    console.log("I am fn");
}
```

Assignment and reference types

In reference type like array when a refrence type is assigned -> you get the address not the whole copy

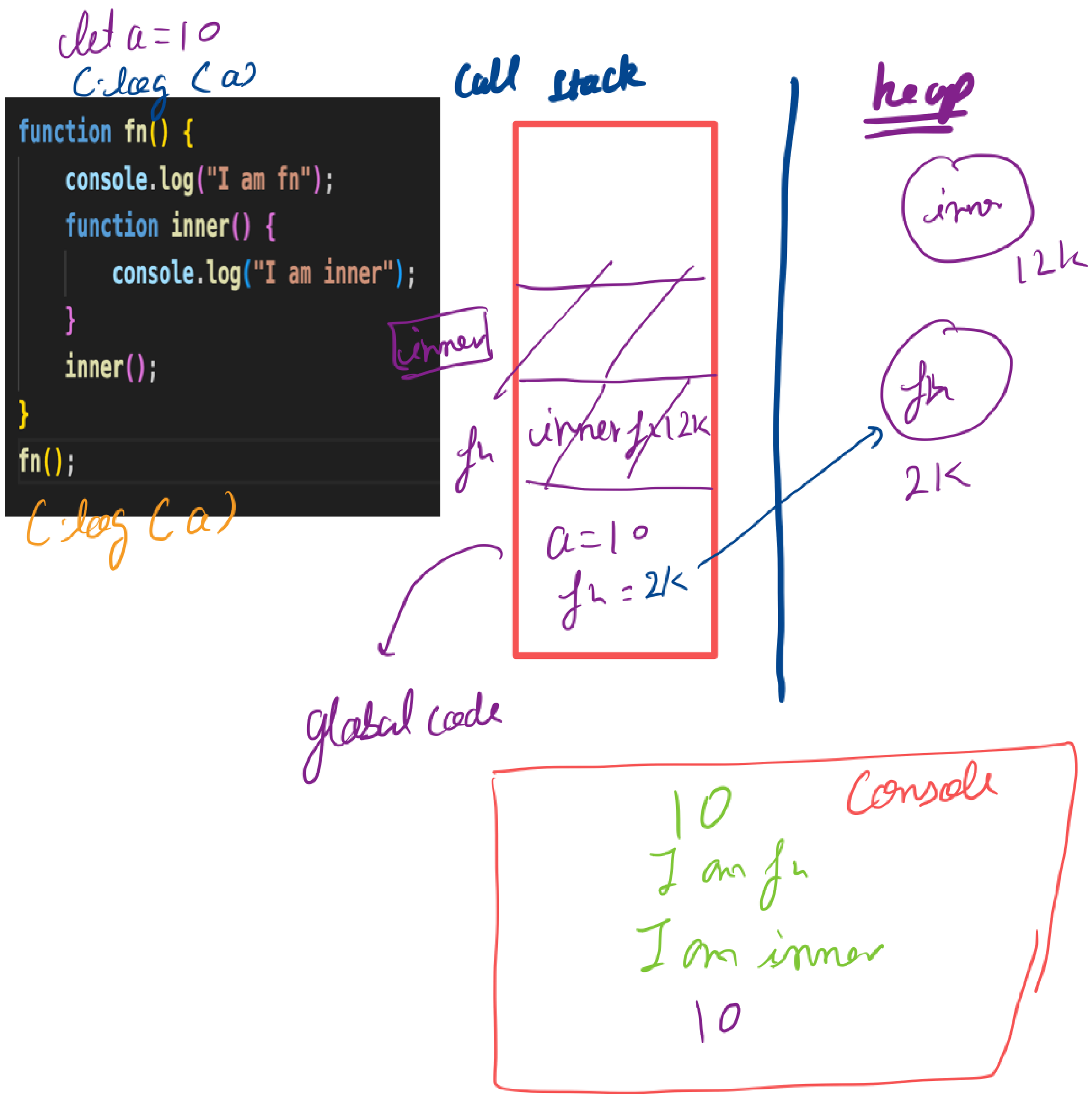


```
// let arr=[10,20,30];

// let arr2=[30,40,50];

// arr=arr2;
```



Code excecution

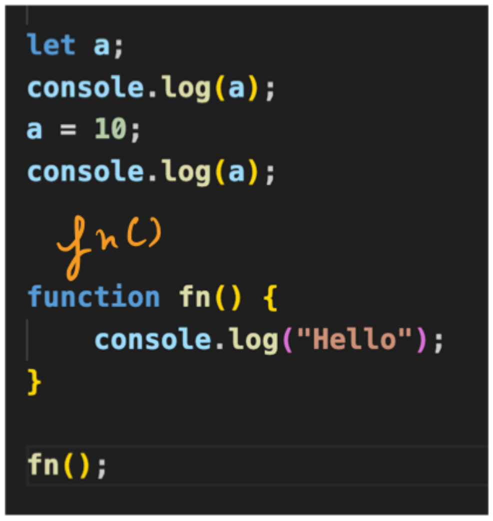
```
let a=10;
console.log(a);
function fn() {
    console.log("I am fn");
    function inner() {
        console.log("I am inner");
    }
    console.log("I am fn 2")
    inner();
    console.log("I am fn 3")
}
fn();
console.log(a);
```



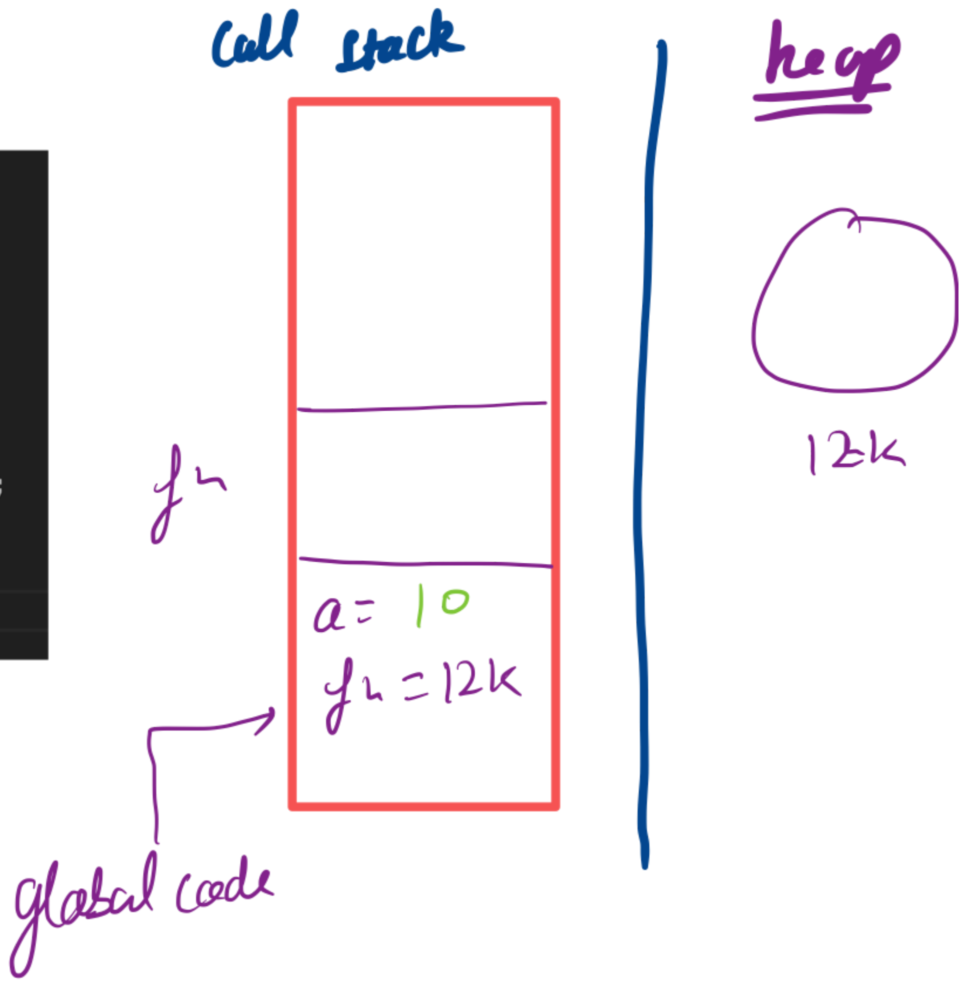
```
function fn() {
    console.log("I am fn");
    function inner() {
        console.log("I am inner");
    }
    inner();
}
fn();
```

Excecution context

```
/****
 * JS ->
 * 1. all the code is executed on call stack and inside a execution
context
 * 2. An execution context created when
 *  a. a function is called
 *  b. code execution starte for global code->
 *       global execution context
 * 3. Execution context -> it excutes in two phases
 *      a.) Execution context createion
 *         i.) memory allocation -> hositing
 *              i.) variable -> undefined
 *              ii.) function -> memory allocation in the heap
 *        ii.) window,
 *        iii .) this,
 *        iv.)  outer scope
 *
 *      b.) code execution
 * ***/
```



```
let a;
console.log(a);
a = 10;
console.log(a);
  fn()
function fn() {
    console.log("Hello");
}

fn();
```



```
undefined
10
hello
hello
```

Strings

Here are example of all the important methods and properties of strings like

- length
- toUpperCase, toLowerCase
- indexOf, includes
- substring
- split,
- charAt(), charCodeAt
- replace()

```
let text = "Hello, World!";
console.log("length",text.length); // Output: 13

// console.log(text.toUpperCase()); // Output: HELLO, WORLD!
// console.log(text.toLowerCase()); // Output: hello, world!

// console.log(text.indexOf("W")); // Output: 7
// console.log(text.indexOf("world")); // Output: -1

// text.includes("Hello"); // Output: true
// text.includes("hello"); // Output: false

// let subText = text.substring(0, 5);
// console.log(subText); // Output: Hello


// // there are some other funtions as well.

// // The`split()` method splits a string into an array of substrings
based on a specified separator.

// let words = text.split(" ");
// console.log(words); // Output: ["Hello,", "World!"]

// let joinedStrings = words.join("_");
// console.log(joinedStrings); // Output: Hello,_World!
// let text1 = "       Hello World!       ";
// let text2 = text1.trim();


let message = "HELLO WORLD";

let char = message.charAt(0);
console.log(char); // Output: H
let ascii = message.charCodeAt(0);
console.log(ascii); // Output: 72

let newText = text.replace("World", "JavaScript");
console.log(newText); // Output: Hello, JavaScript!
```