1. **How the javascript execute the code that is called execution context , we can defined in two phases:**
   1 ,2
   Example:----

10 Promises

By definition, a promise is an object that encapsulates the result of an asynchronous operation.

OR

The promises is best way to handle asynchronous operation ,it is used to find out asynchronous operation is successfully completed or not.
this is defined in three phases

1. **Pending** : indicating that the asynchronous operation is in progress.
2. **Fulfilled** :The fulfilled state indicates that the asynchronous operation was completed successfully
3. **Reject :** The rejected state indicates that the asynchronous operation failed.

The promise constructor accepts a callback function that typically performs an asynchronous operation.

If the asynchronous operation completes successfully, the executor will call the resolve() function to change the state of the promise from pending to fulfilled with a value.

In case of an error, the executor will call the `reject()` function to change the state of the promise from pending to rejected with the error reason.

ES6

Rest Operator

A rest parameter allows you to represent an indefinite number of arguments as an array.

```
function fn(a,b,...args) {
   //...
}
```

Spread Operator

spread operator that consists of three dots (. . .). The spread operator allows you to spread out elements of an iterable object such as an array, map, or set.

```
const odd = [1,3,5];
const combined = [2,4,6, ...odd];
console.log(combined);
```

Anonymous function

An anonymous function is a function without a name. The following shows how to define an anonymous function.

you often pass anonymous functions as arguments to other functions

Pass-by-value:

All function arguments are always passed by value. It means that JavaScript copies the values of the variables into the function arguments.

Or

parameters passed as an arguments create its **own copy.** So any changes made inside the function is made to the copied value not to the original value.

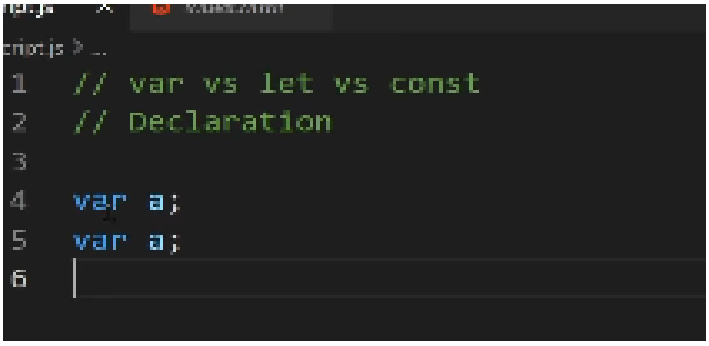
```
function square(x) {
    x = x * x;
    return x;
}

let y = 10;
let result = square(y);

console.log(result); // 100
console.log(y); // 10 -- no change
```

pass-by-reference:

In Pass by Reference,Function is called by directly passing the reference/address of the variable as an argument. So changing the value inside the function also change the original value.

Points:

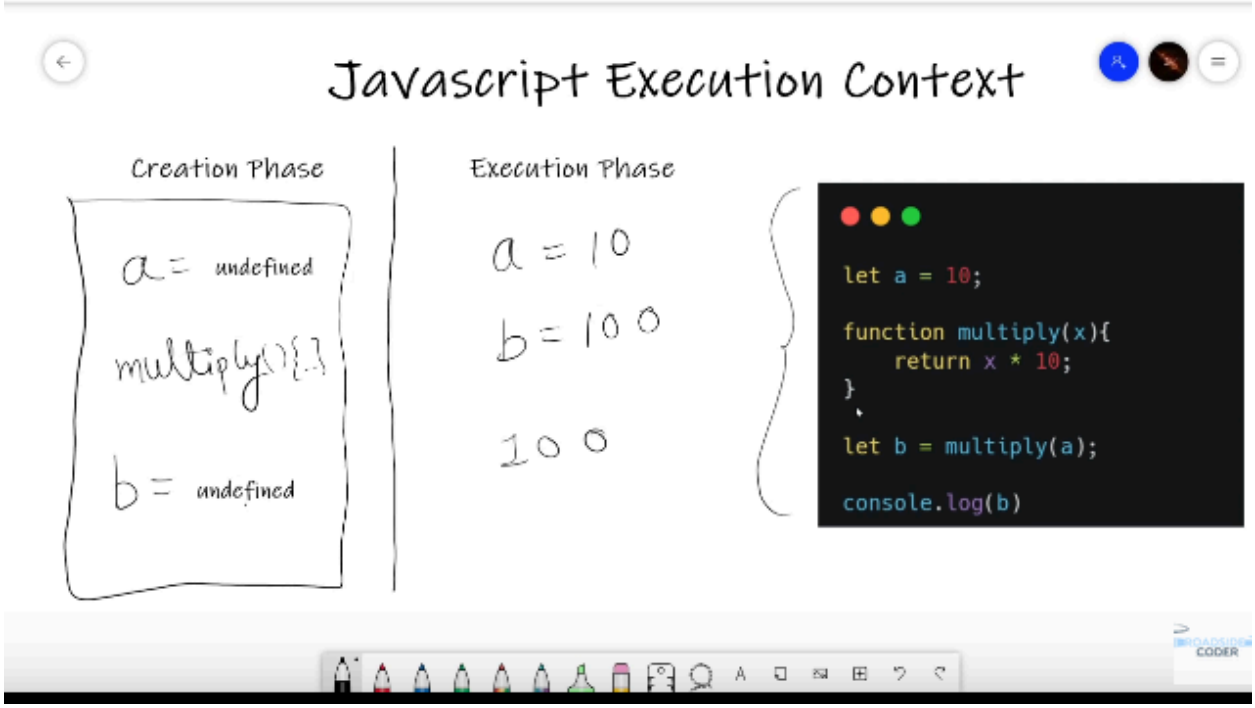
1. We can re-declare a variable by using var but we can not redeclare by using let and const



2. We can re-initialisation using var & let but in case of const , we can not do that

```
// var vs let vs const
// Re-Initialisation

var a = 5;
a = 6;
```

Execution context



how js execution context works

When we try to execute js code there have two phase : creation phase and execution phase

Creation phase: In execution phase 3 things happen **1=>** it creates a global or window object then setups a memory heap for storing variable and fn references and initialize the fn and variable undefined.

Execution context:

JavaScript has a feature called ***hoisting*** where a function can sometimes be used before it is initialized. You can only do this if you declare functions with the `function` syntax.

```
function createFunction() {
  return f;
  function f(a, b) {
    const sum = a + b;
    return sum;
  }
}
const f = createFunction();
console.log(f(3, 4)); // 7
```

In this example, the function is returned before it is initialized. Although it is valid syntax, it is sometimes considered bad practice as it can reduce readability.

Closures

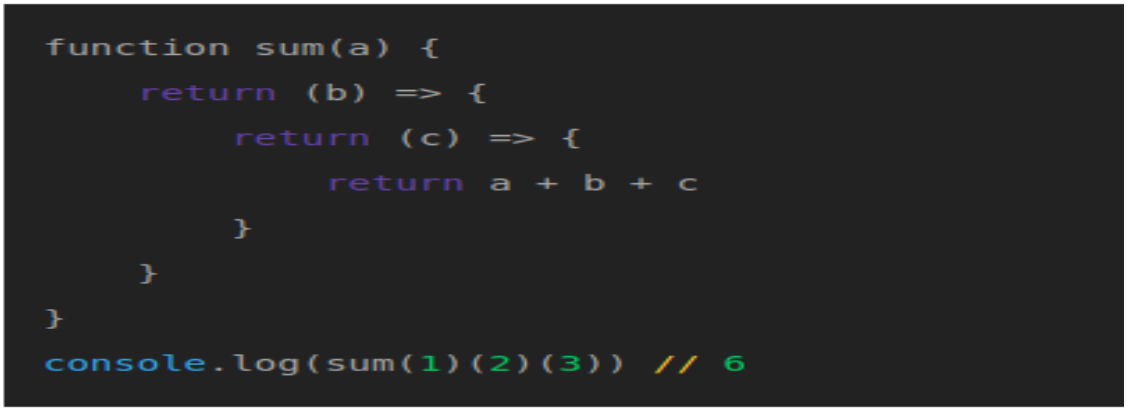
An important topic in JavaScript is the concept of ***closures***. When a function is created, it has access to a reference to all the variables declared around it, also known as it's ***lexical environment***. The combination of the function and its enviroment is called a ***closure***. This is a powerful and often used feature of the language.

```
function createAdder(a) {
  function f(b) {
    const sum = a + b;
    return sum;
  }
  return f;
}
const f = createAdder(3);
console.log(f(4)); // 7
```

Function currying

Currying in JavaScript is a process that allows you to transform a function with multiple arguments into a sequence of nesting functions.

Advantage
Currying helps you avoid passing the same variable again and again.
It helps to create a higher order function.



```
function sum(a) {
    return (b) => {
        return (c) => {
            return a + b + c
        }
    }
}
console.log(sum(1)(2)(3)) // 6
```

Type coercion:

Type coercion is the automatic conversion of values from one data type to another during certain operation or comparisons.

It is used during string and number concatination

It is used while using comparison operator.

What is difference be normally handling promises and using async/await

In the case of normal promise ,the program will not wait for handle promise and code will executed b/c javascript wait for none.

```
const p = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Promise Resolved Value!!");
  }, 10000);
});

// await can only be used inside an async function
// async function handlePromise() {
//    const val = await p;
//    console.log(val);
// }
// handlePromise();

function getData() {
  // JS Engine will not wait for promise to be resolved
  p.then((res) => console.log(res));
  console.log("Namaste JavaScript");
}

getData();
```

Lazy Loading:

Lazy loading is a technique used to improve the performance of web pages by loading resources only when they are needed.

This means that non-critical resources such as images, JavaScript, and CSS are loaded only when they are required, rather than being loaded all at once.

Local state and Global state:

Local state and global state are two ways of managing data in React applications. Local state is specific to one or a few components and is accessible only to their child components.

Global state is needed by multiple components and is accessible throughout the entire application. Local state is usually implemented using the useState hook or the this.state object in class components. Global state can be implemented using various libraries or techniques, such as React Context, Redux, MobX, etc.

Advantage of Custom hooks in React js:

- They help avoid code duplication and make the code more readable and maintainable.
- They allow you to extract complex logic from components and encapsulate it in reusable functions.
- They enable you to share stateful logic between components without using higher-order components or render props.
- They make it easier to test and debug your stateful logic.

Var and let can be updated or redeclare but const can’t be

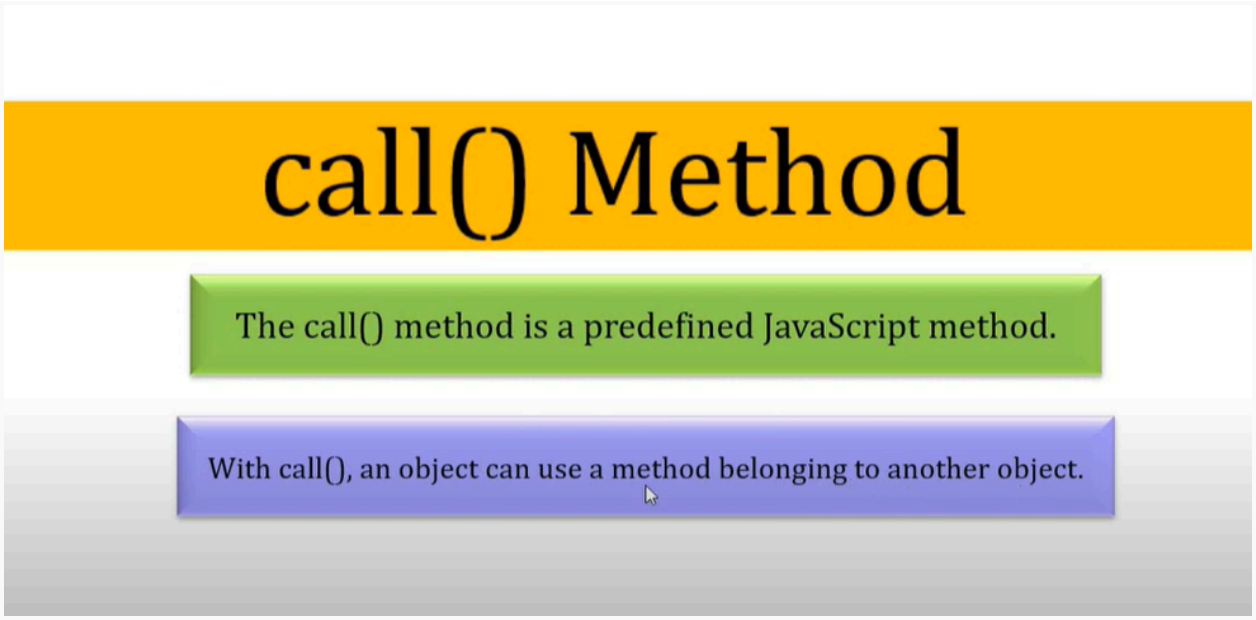
Function hosting

```
// Q8 - Function Hoisting

functionName();
console.log(x);

function functionName() {
  console.log("Roadside Coder");
}

var x = 5;
```

The output of this snippet is 3 times 0 reason of the scoping

```
// Ques 4: Block scope and setTimeout

for (var i = 0; i < 3; i++) {
  setTimeout(function log() {
    console.log(i); // What is logged?
  }, 1000);
}
```

Call method




```
const youtuber1 = {
    name: "ThapaTechnical",
    content : "Programming",
    feature: function(rating,){
        console.log(`very friendly way of teaching. ${this.name} is
            my fav ${this.content} channel. I will love to give ${
            rating} star.`);
    }
}

// youtuber1.feature(5);

const youtuber2 = {
    name : "kuchbhi",
    content : "programming&vlog"
}

youtuber1.feature.call(youtuber2,5);
```