Java Script

Complete Notes

JavaScript is high level, object-oriented, multi-paradigm language, for dynamic web page development

Basic –

Variables ; the variable can contain only in letters and dollar and underscore and should start with small case,and must be in camel case.

And **constant** should be in uppercase.

Data Types ;

In java script **the value is either primitive or object.**

Object -> let score = {

Marks : 20,

Grade : b

}

And

primitive type : let score = 20;

7 primitive types :

Number, Boolean, string, undefined, null, bigInt -> for storing large number released in ES2020, symbol (ES2015)-> for storing data that cannot be changed ,

We can get the type of variable in the using keyword :

Let m = 'true';

Console.log(typeof m); // boolean

// note

typeof null - > object // type of null return object as it is bug in the language

let and const is released in the ES6;

var is old and traditional and not used.

Let is used when the value needs to be changed during programming.

Const is used to define the constant whose value is not changed in throughout the program.

We can use direct assignment like

firstName = 'mayur'; // we should avoid this type of declaration.

Template literal for String:

This is use if we have a complex string to represent as a variable values in it.

To use this we have to use the tilt symbol (``) i.e below esc key.

Example : ` I am ${name} and I live in ${city}`;

We can have a multi-line string in older version i.e., by using

\n\ -> 'string \n

Multi \n

Lines'

But using tilt the multi line can be automatically used (``)

` string

Multi

Line `

By default the input is treated as a string

== (lose equal) vs === (strict equal)

18 == '18 ' == true

18 === '18' == 'false' -> strictly checks if the type is also same for comparison

Loose equal It requires type conversion so always convert the type to match the condition.

But for strict conversion there is no type conversion.

Type conversation vs coercion :

Type conversation is explicit type casting : where we manually type cast.

Type coercion is implicit type casting

We can do only specific conversation in js :

If we convert the number into string then it is a NaN (not a number).

And type of NaN is Number.

IMP the js convert all other arithmetic operator into number (/, *,>,<, -).

And for the string concatenation the + operator convert the number into string.

i.e

('I am ' + 23 + 'year old ') -> I am 23 year old.

('20' – '10' - 3); -> 7

(' 20' / 4); -> 5

➔ **'12'+12/2*2+10/5 // o/p "12122"**

After the complete development of the js in new version use the use babel to transpile and polyfill (converting back to es5 so that compatible for all users).

Strict mode :

It is used to strictly check the variable defined, which are changed later in the program, are same and if not it logs it to the console for help to find the error.

To use the strict mode use the:

'use strict' -> it should be the first line of the js file and should not have any code prior to it. But can have the comments as js ignores the comment.

Example:

```
'use strict' // if we remove this line then we cannot see the error on
console
At line hasPermissio is misspelled
let hasPermission  = false ;
const access = true ;

if(access) hasPermissio = true ; // the spelling is misspelled
if(hasPermission) console.log('caa visit');
```

Function:

It is used to reuse the code.

Example :

Function logger(){

Console.log('this is logged to the console ');

}

logger(); // calling running invoking

logger(23); // this will still execute as the function is without the parameter

function with Parameter :

function fruitJuice(apples, oranges){

return ` juice with ${apples} apples and  ${oranges} oranges.`

}

fruitJuice(5, 2); // this will call the function with passing the values to function.

We have normal function declaration and the expression function declaration.

i.e

```
const age1 = calcAge(1996); // for function declartion we can call it b
efore
declaring it.
console.log(age1 + ' -- ');
//function declaration
function calcAge(birthYear){
return 2020 - birthYear ;
}
// console.log(age2(1996)); //Erorr but for the expression it needs to
be
declare first and then it must be called

//function expression
const age2 = function( birthYear ){
return 2020 - birthYear ;
}
```

```
console.log(age2(1996)); // here is the right way to call the function expression as function is also a value.
```

Arrow Function:

We write this type of function in the were we need one line of code and the main drawback is we cannot use the, this keyword in arrow function.

Example :

Const age = birthYear => 2021 – birthYear ;

Console.log(age(1996)); // 24

If we are returning a operation on first line then there is not need to write the return ;

For using the multiple arguments in the arrow function we need to specify the parameters in the ().

The trick in vs code is to select the words and add the right parenthesis.

```
let age = birthYear =>   2021 - birthYear ;
console.log(age(2020));

const yearForRet = (birth , firstName) => {
    const age = 2021 - birth ;
    return ` ${ firstName} has ${ 65- age} years left in retirement `;
}

console.log(yearForRet(1996, 'mayur'));
```

calling function from the function to another function.

```
function cutFruit(fruit){
    return fruit * 4 ;
}
function makeJjuice(apple, orange ){
    const applePic = cutFruit(apple);
    const orangePic = cutFruit(orange);

    return  `this is the Juice for  ${applePic} pieces of apple and ${o
rangePic} pieces of orange `
}

console.log(makeJjuice(2,3));
```

Arrays:

We can store any type of data in an array may it be number, string or even an object or other array.

Const array = [1, 'mayur', {Eng., it dep}];

Console.log(array[0]) // to print " 1 ";

If we declare the array as a const then we cannot change the entire array but, we can change the elements at the index.

Array methods:

1. **Push();** it adds the elements to the end of the array and return the length of the array.
2. **Unshift('element'):** it adds the elements to the start of the array. And return the length of the array.
3. **Pop() :** it removes the element from the end of the array and returns the popped element of the array.
4. **Shift():** it removes the element from the start of the array and the element that is removed.
5. **Indexof('element'):** it checks the element is present and returns the index of the element, if not present it returns **-1;**
6. **Includes('element') :** it returns a Boolean value if the element is present and it is introduce in the ES6 version. It checks for strict equality.

Objects:

We use to grouped the different variables(property) together in a key : value pair.

Example:

```
const object = {
  firstName: "mayur",
  lastName: "zende",
  age: 23,
  friends: {1: "donna", 2: "mike"},
};
```

note

In arrays we use square brackets and In objects we use the curly bracket.

The order of data that is stored, may be different in object.

Imp:

- We can access the object property using a '.' , or by using a [].
- log(Object.name); // using . operator
- log(object['name']); // using a [] we can write a expression to retrieve the values or passing the dynamic value to check the nested changing values like for loop iteration.
- **Const key = "name ";**
- **Log(object['first'+key]); // doing the concate operation and then accessing the value.**
- **Log(object['last'+key]) ;**
- Also we are passing a dynamic input to assess the property of the object then it is not possible to access it with the " . " property we need to use the " [] " for it.

We can also use the dot notation or [] to assign the value to the object property.

Object.location = 'India';

Object['city'] = 'pune';

Object methods:

We can have the function in the object and call it.

```
Const obj = {
Name  : 'mayur',
birthyear : '1996',
calcAge : function( birthYear){
return 2021 – birthyear;
}
// using the this keyword means current object notation.
CalcAge : function(){
Return 2021 – this..birthyear ; // this is used to point to the current object
}
// new es6 syntax to write the function is
// calcAge(birthYear) { return 2021  - birthYear}
// we can also use the this to assign the value so no need to repeat the calculation
calcAge : function(){
```

```
this.age = 2021 – this.birthyear ;
return this.age; // return obj.age;
}
}
Log( obj.calcAge(1996));
Log(obj['clacAge'](1996));
Log(obj.age); // still use as the obj.age
```

LOOPS :

For is used when we know the size and want to iterate.

While is used when a condition is true.

For rolling a dice and check the dice gives 6 we need the while loop for the condition.

In the debugging we can use the following commands for console log;

Console.warn() -> to see the warning.

Console.error() -> to see the red colour in the developer console.

Also we can format the objects on the developer console.:

Console.table(Object_Name);

We can use the debugger in the code to directly jump to the line of code in developer sources tab.

How javaScript works behind the scenes:

It is a highlevel, - it does not have any memory related task all is taken care by the GC

Paradigm - cam write the code according to the functional / procedural / OOP concept

Single thread – can execute only one task at time.

Interpreted / just is time compiled – only the object needed are loaded into the memory

Fist class -we can pass the function as the argument to do some operation.

Dynamic – automatically converts it into the data types

Prototype object based – everything is in objects except the datatypes like integer, Boolean

To stop the single thread execution, we use the event loop which make the large task run in the background and then it merges when the task is complete.

How the code is converted into the machine code:

Every browser has its own engine to run the code

Example the google chrome uses the v8 engine and other browser has their own engines

The js engine consists of the

Call stack – where all the execution context is present and where the code is executed.

Heap where the objects are stored in the engine.

The code is executed in the 3 phases :

1. Parse the code is converted into the AST(abstract syntax tree)
2. Compilation the code is then converted into the machine code
3. Execution the code is executed.(happens in the call stack)

The code is then executed in the **global execution context** top level high code meaning only the code outside the function is executed.

And after that the when the function is called then the code is executed.

Ex.

```
Const name = “mayur”;

Const first = () => {

Int a = 10 ;

Return a + second();

}

Function second(){

Return ‘above’ ;

}

Const x = first();
```

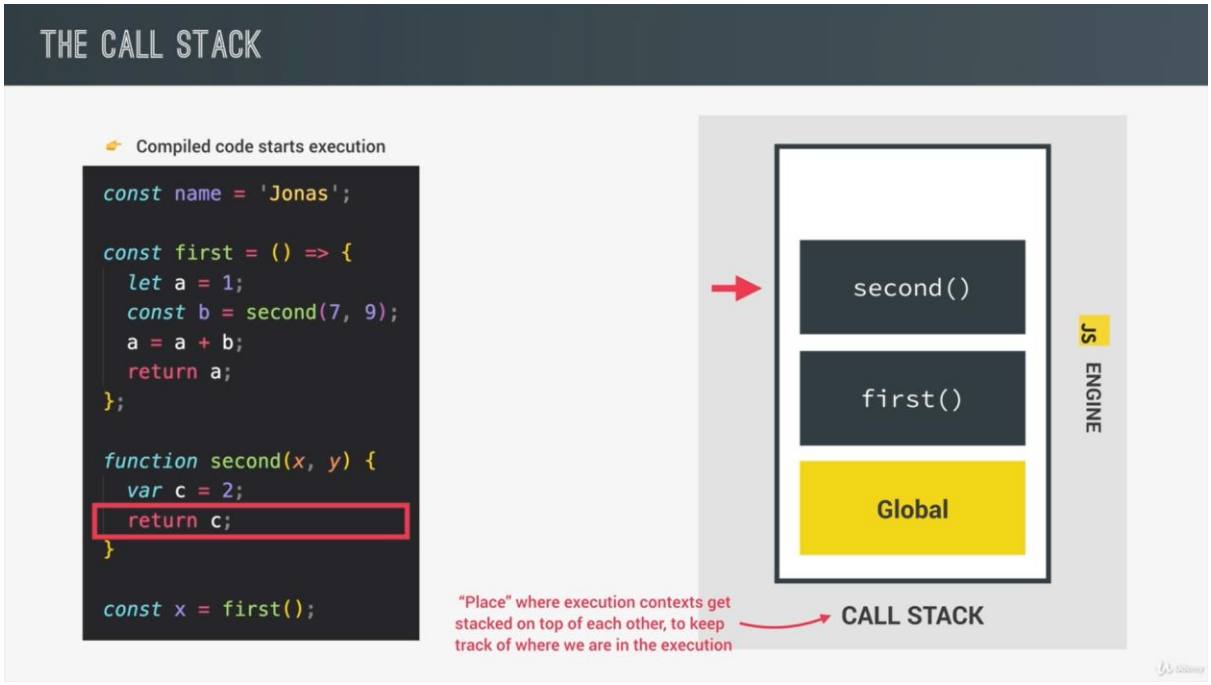
Here in the above example only the name first line will be executed as the function are not called.

The function code only will be executed whenever the function is called.

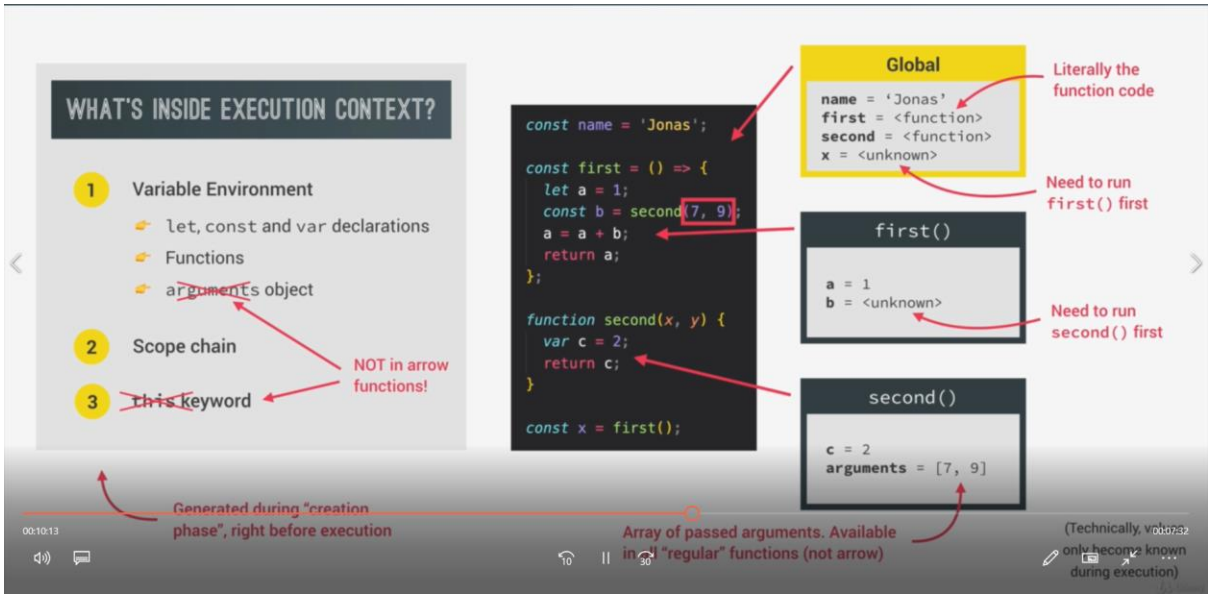
And the one execution context as per the function definition.

And waiting for the callbacks.

The call stack keep the tack for current executing execution context.



This is how the call stack is called for the example above.

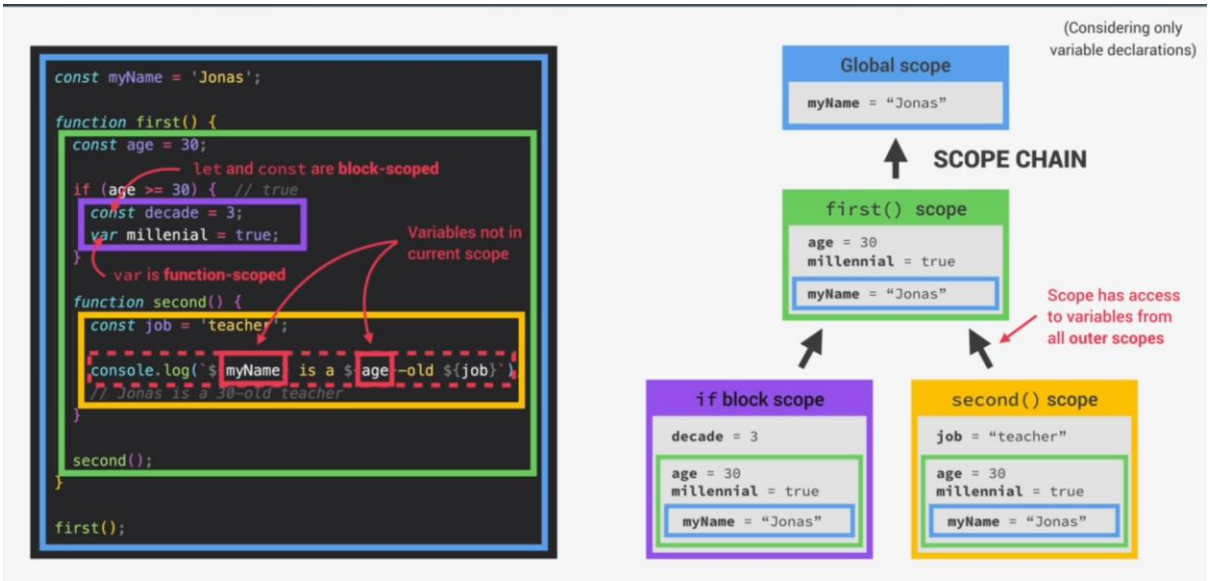


Variable scope :

Global – all over the code

Function – only in declared function

Block – in if block and then only let and const id blocked scope and the var can be accessed all over the close function even if it is in the block scope.



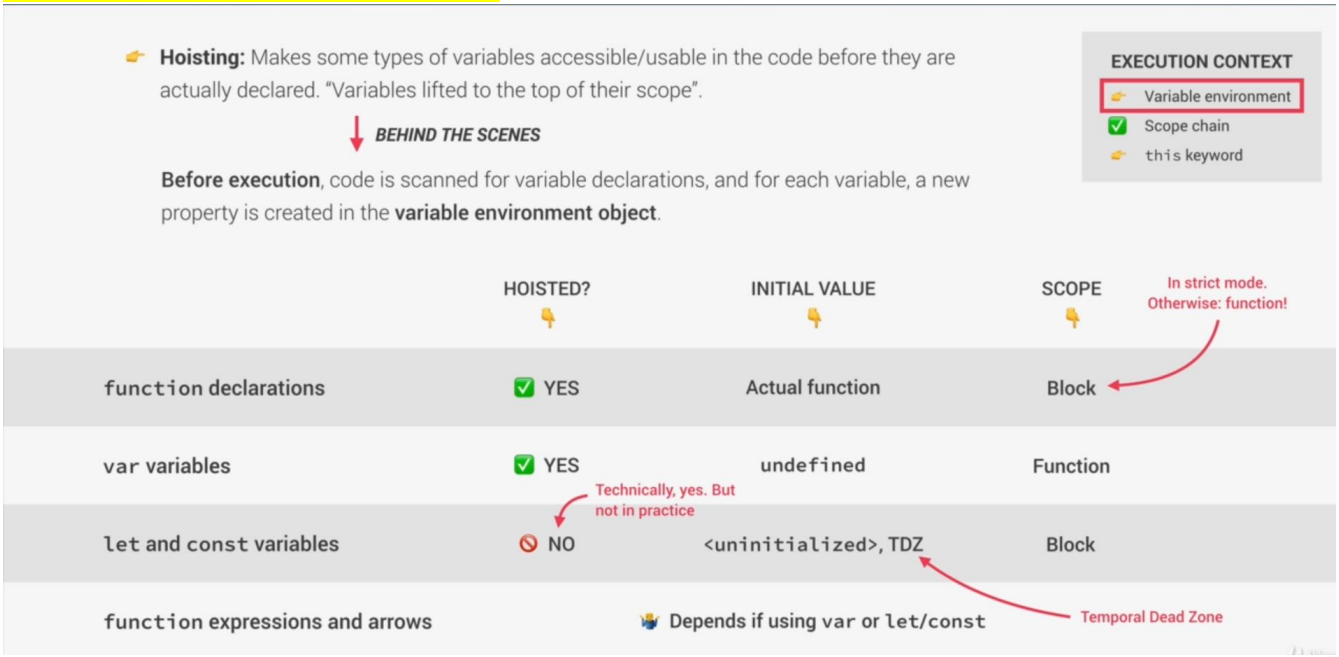
Variable declaration :

Note

The variable is define in the TDZ (tempral dead zone) – to avoid using them before defining it.

The tdz is before the variable is declared to the actullay call.

How the variables are declared in the



If we use it with the const and let then the variable is in the tdz so it can not be accessed and we get error that is accessed before initialization.

But if we declare it with the var then it is undefined as it is taken care by hostelling.

Same applies for function expression and arrow functions.

But we can access the function declaration before declaring it.

This keyword :

The this keyword in global scope point to the window and same for the arrow function.

But in function expression it show undefined.

```
"use strict";

const obj = {
  id: 0,
  year: 100,
  age: function (no) {
    console.log(this);
    return this.year - no;
  },
};
```

```
console.log(obj.age(20)); // this will call on obj object

const mayur = {
  year: 50,
};

mayur.age = obj.age; // copying the obj age function

console.log(mayur.age(30)); // this will call on the mayur object
```

```
Live reload enabled.
▶ {id: 0, year: 100, age: f}
80
▶ {year: 50, age: f}
20
>
```

We should never use the this keyword in the arrow function method as it is bound to the global scope and avoid using the variable name with the var .

But if we use it as a regular function call then it can show the called function object.

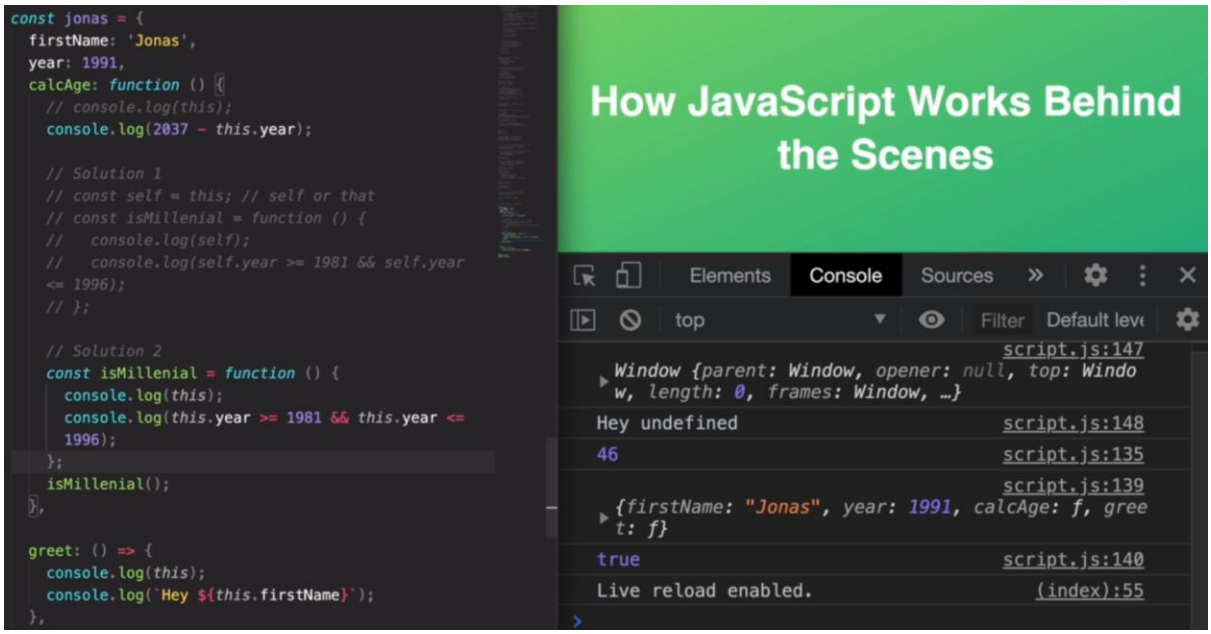
```
const obj = {
  firstName: "mayur",
  id: 0,
  year: 100,
  age: function (no) {
    console.log(this);
    return this.year - no;
  },

  greet: () => {
    console.log(`hey ${this.firstName}`);
  },
};

console.log(obj.age(20)); // this will call on obj object
console.log(obj.greet());
```

by calling using arrow function.

```
 greet: function () {
    console.log(`hey ${this.firstName}`);
  },
};
```

By using the normal function.

The arrow function inherits the parent properties



Above is the old and new way of using the this keyword in a function

The Arguments keyword :

It is used in the regular function to add more than specified arguments.

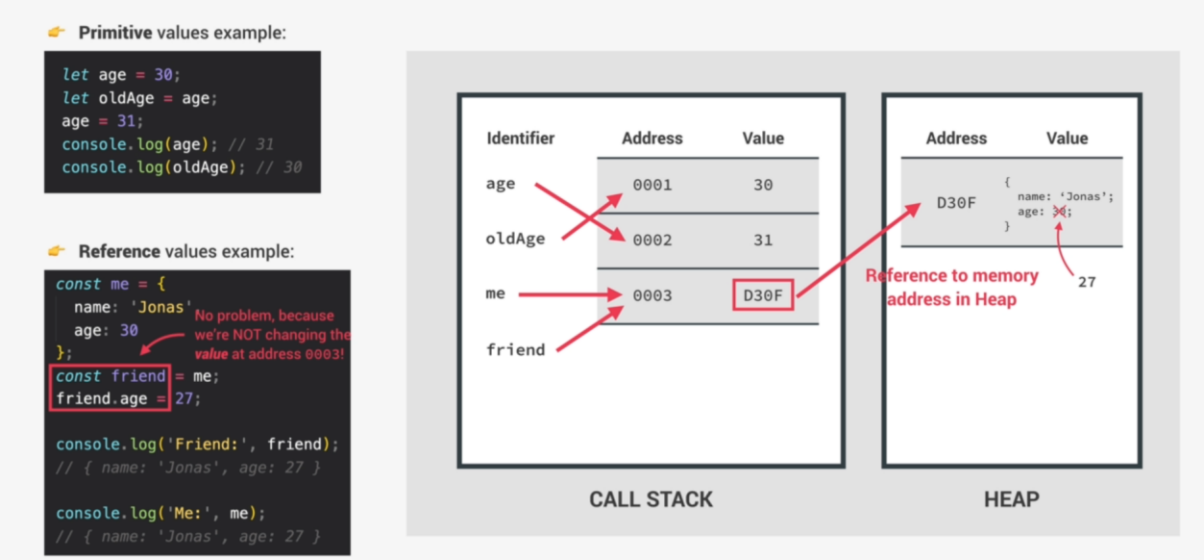
Syntax is : log(arguments);

And we can use the logic to do the operation on the size of the arguments.

But we cannot use this in the arrow function.

```
function test(first, second) {
  console.log(arguments);
}

test(2, 3);
```

Primitive types vs Object types :



When we store the value in the primitive data type then there are store on the call stack and the we can see in the above picture that the primitive type is pointing to the address and the address has the value to it.

But when we store the object the object is created in the heap area so the call stack holds the address to the heap memory address and when we duplicate the obj the call stack new entry is created but pointing to the reference of the heap area.

See the pic above to see the explanation.

To workaround we have a method :

Const copy = Object.assign({}, oldObj)

So now we change the property for the copy it will not show in the old object.

But it only a shallow copy of the object.

If we have a array in the old object and we change the new object array then the old one is also updated.

Array DE structuring:

```
const array = [2, 3, 4];
const [a, b, c] = array;

console.log(a, b, c);// 2, 3, 4
```

```
const obj = {
  firstName: "mayur",
  Id: 1,
  Arr: [10, 20, 50],
};

const [first, , second] = obj.Arr; // to add a hole to assign the va
lue
console.log(first, second); // op:- 10 50
```

this is the way to destructure : to assign the item to the variable using the array items.

We can use the nested array to ;

That is

Array = [1, [5,6]];

Const [one , two] = Array ;

Console.log(one, two);

If we have less inputs then to the received input then we can do the following :

```
const restaurant = {
  name: "mayur",
  id: 1,
  starterMenu: ["sandwich", "cake", "roll"],
  mainMenu: ["thali", "rice", "chciken"],
  openingHours: {
    mon: {
      open: 10,
      close: 22,
    },
    wed: {
      open: 10,
      close: 21,
    },
    sat: {
      open: 10,
      close: 23,
    },
  },
```

```
  order: function (starterIndex, mainIndex) {
    return `your oder is ${this.starterMenu[starterIndex]} in stater
 and mainCourse is ${this.mainMenu[mainIndex]}`;
  },
};

console.log(restaurant.order(1, 2)); // to call the function in the
object.

// the value must match with the name of the object property
// this is usefull if we need only the needed objects.
const { name, openingHours, starterMenu } = restaurant;
console.log(name, openingHours, starterMenu);
```

this is usually used when we receive the json object from the rest API call to parse the object. (give our custom name to the property of the object)

```
const {
  name: resturantName,
  openingHours: timmings,
  starterMenu: starters,
} = restaurant;
console.log(resturantName, timmings, starters);
```

to give our own name to the object so that we can reference that in the code.

```
// we can also give the default value to the object that is not
present in the object
const { menu = [], starterMenu: options = [] } = restaurant;
console.log(menu, options);
```

to do nesting arrays and nesting objects and manipulate them:

```
//mutating variables
let a = 19;
let b = 20;
const obj = {a: 7, b: 11, c: 22};
({ a, b } = obj);
console.log(a, b);
```

```
//nested objects

const {
  sat: { open: o, close: c },
} = openingHours;

console.log(o, c);
```

to call the object function by passing the object and setting default value if the parameter not passed:

```
  oderDelivery: function ({
    starterIndex = 1,
    time = "22:00",
    mainIndex = 0,
    address,
  }) {
    console.log(
      ` order placed! your order for starter ${this.starterMenu[star
terIndex]} and mainCourse
      ${this.mainMenu[mainIndex]} will be deliverd to the address ${
address} at ${time}`
    );
  },
};

restaurant.oderDelivery({
  address: "pune india ",
});
```

Spread Operator :

... -> to take the individual elements from the array .

```
const array = [7, 8, 9];
const badArray = [1, 2, 3, array[0], array[1], array[2]];
console.log(badArray);
//using the spread operator
const newArray = [1, 2, 3, ...array];
console.log(newArray);
```

```
console.log(...newArray); // to take the individual elements from th
e array
```

we can also join two arrays using the spread operator.

That is

Const joinArray = [...oneArr , ...twoArr] ;

And also add the element to the array using spread Operator;

Const spOpr = [...arr , 'item'];

Example for the spread operator in the function call.

```
  orderPasta: function (ing1, ing2) {
    console.log(`here is the pasta with your choices for ${ing1} And
 ${ing2}`);
  },
};

const wish = [prompt("enter the ingridrent"), prompt("enter the ingr
ident 2")];
// console.log(wish);
// console.log(wish[0] , wish[1]);
restaurant.orderPasta(...wish); // new syntax with the es6 version
```

we can do this on the object also use of spread operator :

```
const newRestaurant = { founder: "harvey", ...restaurant, founded:
1992};

console.log(newRestaurant);

const restaurantCopy = { ...newRestaurant };
restaurantCopy.name = "mike";
console.log(restaurantCopy.name);
console.log(restaurant.name);
```

Modern operator;

Or operator:

It returns the first true value in the expression // or is every value is false then it returns the last value of the expression.

Console.log('3' || 'mayur ') // 3

Console.log(' ' || 'mayur') // mayur

Console.log(undefined || null) // null

➔ undefined || null || 0 || null || undefined // **o/p: undefined**

it return the first true value.

Ex.

Guest = 0 ;

Const guestIn = Guest || Guest = 100 ;

The output will be the 100 as the value set is 0 which make the expression false as this the bug so to avoid this we need to use the nullish coalesing operator.

Const guestIn2 = Guest ?? Guest = 100; **// this the new es6 ??**

Operator to avoid the zero value.

And operator :

It returns the last true value if all the value in the expression is true.

And if the starting value is false then the entire expression is false.

For Of loop

To loop the array using new syntax

```
for (const item of players1) console.log(item);
```

To show the index and elements together we use

```
for (const item of players1.entries()) {
  console.log(`${item[0] + 1} is ${item[1]} player`);
}
//using the array desturcting
for (const [i, el] of players1.entries()) {
  console.log(` ${i + 1} jersey no of ${el} `);
}
```

Object literals:

We can write the object property from other object.

I.e

```
 Const openingHours: {
    mon: {
      open: 10,
      close: 22,
    },
    wed: {
      open: 10,
      close: 21,
    },
    sat: {
      open: 10,
      close: 23,
    },
  },
```

Const restaurant = {

Name : ‘mayur’,

// old ways to copy the syntax was to

Openinghours : openingHours,

// in es6 we can write direct

OpeningHours

Also is same for the function

Order : function(parameter){

Return ‘this is the old way ‘

}

New way to do the thing is to

Order(parameters) {}

}

Optional chaining

```
// we use the optional chaning operator to check if the
// value is present for that object

// we are checking for the mon value or if not present we get undefi
ned
console.log(restaurant.openingHours.mon);

// here if the value is not present then we get error
console.log(restauratn.openingHours.mon.open);

// to solve this we have a optional chaning operator
console.log(restaurant.openingHours.mon?.open);
// this is then avoid the error and give undefined

// this operator goes well with the Nullish coalescing operator.
// that is ?? if not present then assign the value
const days = ["mon", "tue", "wed", "thur", "fri", "sat"];

for (const day of days) {
  const open = restaurant.openingHours[day]?.open ?? "closed";
  console.log(` the restuarnt on ${day} time is ${open}`);
}

// this also works on the method and arrays
console.log(restaurant.order?.(0, 1) ?? " method is not present");

const user = [];
console.log(uesr[0]?.name ?? "empty array");
```

Looping objects values and keys and entries

```
const openingHours = {
  mon: {
```

```
    open: 10,
    close: 22,
  },
  wed: {
    open: 10,
    close: 21,
  },
  sat: {
    open: 0,
    close: 23,
  },
};

// here we are accessing the objects key only
const properties = Object.keys(openingHours);
console.log(properties);

let openstr = `we are open for ${properties.length} days `;

for (const day of properties) {
  openstr += `${day}, `;
}
console.log(openstr);

// here we are using the values
const values = Object.values(openingHours);
console.log(values);

// to parse the entire object we need to use the following
const entries = Object.entries(openingHours);

// here we are destructing the object to use the key value
for (const [key, { open, close }] of entries) {
  console.log(` we are open on ${key} form ${open} to ${close}`);
}
```

Two set were introduce in es6 set and maps.

Sets:

It has only the unique objects and don't allow duplicate.

It can hold the different datatypes.

We do not have the iterable method in the set as it has unique value and does not have the index. So to iterate we need to use the array.

```
const orderSet = new Set(["pizza", "bread", "fries", "rolls"]);

console.log(orderSet);
console.log(new Set("Mayuresh"));

orderSet.add("pizza");
orderSet.add("omlette");
orderSet.delete("rolls");
console.log(orderSet.has("fries"));
console.log(orderSet.has("rolls"));

console.log(orderSet);
orderSet.clear();
console.log(orderSet);

console.log(new Set("mayuresh").size);
```

Maps :

It used to store the data in key value pair

```
maps
---------------------------------------
const rest = new Map();

rest.set("name", "funTreat");
rest.set(1, "pune india");
console.log(rest.set(2, "ny usa"));

rest
  .set("categories", ["indian", "marathi", "hydrabaadi", "western"])
  .set("open", 11)
  .set("close", 23)
```

```
  .set(true, "we are open :D")
  .set(false, "we are close :(");

console.log(rest);

console.log(rest.get(true));
console.log(rest.get("categories"));

const time = 15;
// we can also do the calculation in get method to get the key value
 form the map.
console.log(rest.get(time > rest.get("open") && time < rest.get("clo
se")));

console.log(rest.has("open"));
rest.delete("close");
// rest.clear();
console.log(rest);
console.log(rest.size);

const arr = [1, 2]; // we can set the arr as a key to the map.

// rest.set([1, 2], "array");
// console.log(rest.has([1, 2]));// it will give false as it is poin
ting towards the other  memory locaiton

rest.set(arr, "array");
console.log(rest.has(arr));

//we can also set the object as below as a key value very powerfull
feature.
rest.set(document.querySelector("h1"), "heading");
console.log(rest);

const openingHours = {
  mon: {
    open: 10,
    close: 22,
  },
  wed: {
    open: 10,
    close: 21,
```

```
  },
  sat: {
    open: 0,
    close: 23,
  },
};

// we write the map in this fashion aslo
const question = new Map([
  ["question", "which is the best progemming language in the world"]
,
  [1, "C"],
  [2, "Java"],
  [3, "JavaScript"],
  ["correct", 3],
  [true, "correct answer"],
  [false, "wrong try agian"],
]);

console.log(question);

// the openingHours object format is in the key value pair so it is
converted into the map below.
//convert the object to map
console.log(new Map(Object.entries(openingHours)));

for (const [key, value] of question) {
  if (typeof key === "number") console.log(`Answer : ${key} = ${valu
e}`);
}
const answer = 3; //Number(prompt(" enter the correct choice"));

console.log(question.get(answer === question.get("correct")));

// if (answer === question.get("correct")) {
//   console.log("right answer");
// } else {
//   console.log("wrong answer try again ");
//   Number(prompt(" enter the correct choice"));
// }
```

```
//conver map to array

console.log(...question);
console.log([...question.keys()]);
console.log([...question.values()]);
```

Arrays	Sets	Objects	Maps
When we need to store the data in consecutive manner and duplicate data	When we do not want the duplicates	Easy to use in key/value pairs (traditional store)	When we want to store the data in the key/ value format. Where keys can have any datatype
Want to iterate the values	When performance is important it is better comparing to array.	can use this keyword.	Better performance Easy to iterate
	Use to remove the duplicates form array	Can use function in the object.	Can compute size
		Easy to write and access the data with . and []	When we need to map to values.
		When working with json and can convert to map.	

Strings :

They are immutable and we do the manipulation of strings. Using the built-in function.

We can do operation on the direct strings or on the variable where the string is stored.

Function :

```
const des = "mayuresh is a smart programmer";
// -1 indicates the index form the ending
console.log(des.slice(0, des.indexOf(" "))); //mayuresh
console.log(des.slice(des.lastIndexOf(" ") + 1)); // programmer
console.log(des.slice(-2)); // er
console.log(des.slice(1, -2)); // ayuresh is a smart programm
```

the string is converted into the object behind the execution.

```
console.log(new String("mayursh"));
```

```
// String {"mayursh"}0: "m"1: "a"2: "y"3: "u"4: "r"5: "s"6: "h" leng
th: 7__proto__: String[[PrimitiveValue]]: "mayursh"

console.log(typeof new String("test"));
// and when we return the string calling function then it is convert
ed to the object and then ouput is in string
```

```
const price = "234,234&";
console.log(price.replace("&", "#").replace(",", "."));
// can combine the funtion of string in one line
// if we want to select and replace all the matching word there is n
o method
// but we can do it using the regular expression.

// replace(/door/g,'gate');
console.log("this+is+the+js".split("+"));
console.log("mayruesh zende".split(" "));

[firstName, LastName] = "mayuresh zende".split(" ");

console.log(["Mr", firstName, LastName.toUpperCase()].join(" "));

// to convert to the upper case of the starting charcter of the name
const correct = (names) => {
  const nameArr = names.split(" ");
  let arr = [];

  for (const n of nameArr) {
    // arr.push(n[0].toUpperCase() + n.slice(1));
    arr.push(n.replace(n[0], n[0].toUpperCase()));
  }

  console.log(arr.join(" "));
};

correct("mr mayursh zende");
const zen = "zende";
console.log(zen.slice(-2).padStart(8, "="));
// console;
console.log("mayuresh".padStart(20, "*").padEnd(30, "?"));
```

```
// and for the pad start the value must be greaater than the length
of the string
// to add the pad end the values must be great than then padstart to
 work.s

const maskCard = (number) => {
  const mask = number + "";
  return mask.slice(-4).padStart(mask.length, "X");
};

console.log(maskCard(12345678));
```

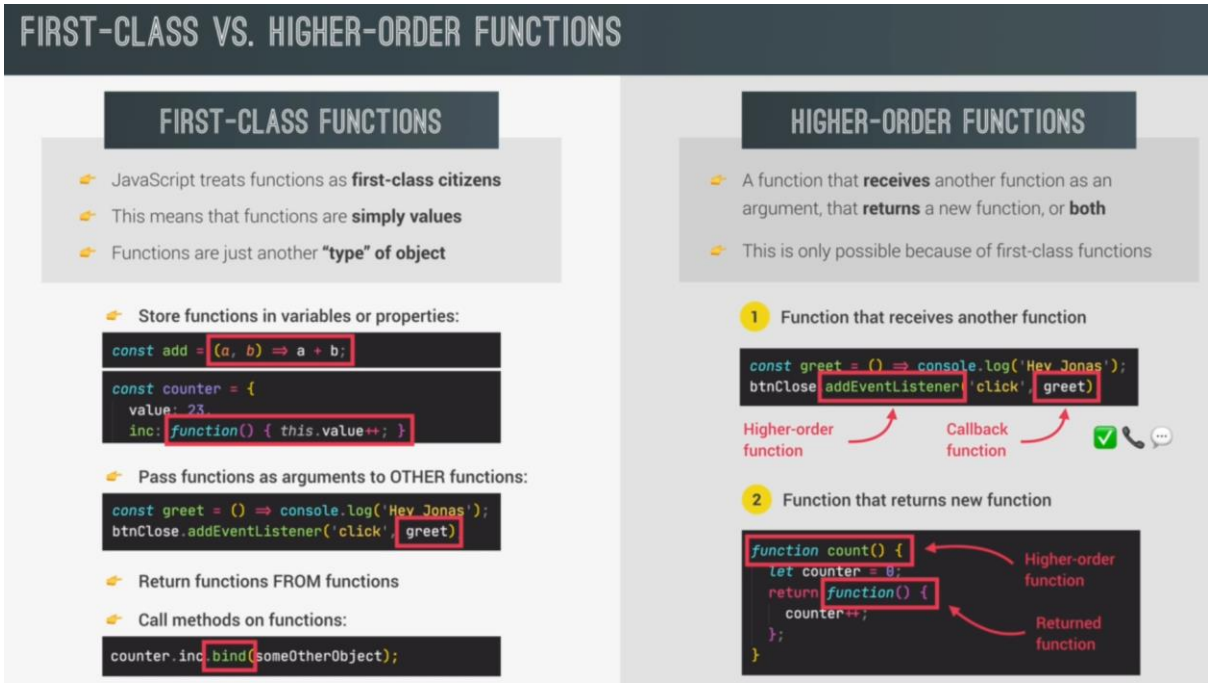
Functions:

This is the deep dive to understand in detail.

The function only has the pass by value to it, in case of object also where we pass as it the reference of the object it is still the pass by value at that memory location.

```
const bookings = [];

const createBooking = function (
  fightName,
  numOfPer = 1,
  price = 100 * numOfPer
) {
  // this is the old way of writing the assign in ES6
  // numOfPer = numOfPer || 1;
  // price = 100 * numOfPer;

  const booking = {
    fightName,
    numOfPer,
    price,
  };
  console.log(booking);

  bookings.push(booking);
};
// createBooking("ABCD123");
// createBooking("new21", 2, 200);
```

```
const flight = "GH124";
const mayur = {
  name: "mayruesh",
  passport: "123456",
};

const check = function (flighName, passenger) {
  flighName = "abc123";
  passenger.name = "Mr. " + passenger.name;
  // here the object we pass is a ref that is point to main adress
  //so the values is change in the main object as well
  if (passenger.passport === "123456") {
    console.log("checked in!!!");
  } else {
    console.log("wrong passport number");
  }
};

check(flight, mayur);
console.log(mayur);
// the object is psased as the ref and value is passed as the new va
riable.
// flighName = abc123 // new variable is created and hold the value
so it is not reflected in the main flight variable
// but for object it is the ref to orignal obj.

const changePassport = function (person) {
  person.passport = Math.trunc(Math.random() * 1000000);
};

changePassport(mayur);
check(flight, mayur);
console.log(mayur);
```



Example for the functional programming. To return the function from a function.

```
const greet = function (greeting) {
  return function (name) {
    console.log(` ${greeting} ${name}`);
  };
};

const greetFun = greet('hello');
// the greetFun is a function that is holding the greet return.
greetFun('mayur');
greetFun('zende');

//greetFun(zende) ia a function and we can pass the next parameter.
// this way is very usefull in the functional programming paradigm


// to use it with arrow function.

const greetArr = (greeting) => (name) => console.log(`${greeting} ${
name}`);

greetArr("hi")("programmrs");
```

to use the this key word method call , apply, bind methods

```
const luftansa = {
  airline: "luftanse",
  aircode: "LH",
  bookings: [],
  book(flightNum, name) {
    console.log(`${name} has booked a Seat on ${this.airline}
   ,flight ${this.aircode}${flightNum}`);
    this.bookings.push({
      flight: `${this.aircode}${flightNum}`,
      name,
    });
  },
};

luftansa.book("213", "mayuresh");
luftansa.book("231", "mike");
console.log(luftansa);

const euroBook = {
  airline: "europian",
  aircode: "Eu",
  bookings: [],
};

const book = luftansa.book;
// book(23, "donna");
// will not work as this function is used in it. which points to und
efined

//call method is used in the modren JS
book.call(euroBook, "890", "shreyash");
book.call(luftansa, "343", "saksham");

// their is also a similar method but not much used

const swiss = {
  airline: "swiss air line",
  aircode: "SW",
  bookings: [],
```

```
};

const customers = ["232", "sakshi"];

// for apply method we need the array to pass the arguments.
book.apply(swiss, customers);

// we can use the call method by destructuring
book.call(swiss, ...customers);

// bind returns a new function instead of calling the function
//it binds this keyword to called object
const bookEW = book.bind(euroBook);

bookEW(...customers);

// we can also pass the argument in the bind function and so
// the need to pass the arguments is simplified.
const book77 = book.bind(swiss, "77");
book77("mayuresh zende");

// with the event listners
luftansa.planes = 300;
luftansa.buyPlane = function () {
  console.log(this);

  this.planes++;
  console.log(this.planes);
};

document
  .querySelector(".buy")
  .addEventListener("click", luftansa.buyPlane.bind(luftansa));
// luftansa.buyPlane.  bind(luftansa) this returns the lustansa obje
ct
// and then we call the function on that object
// i.e luftanas.buyplane();

const addTax = (rate, value) => value + value * rate;

console.log(addTax(0.1, 200));
```

```
const indTax = addTax.bind(null, 0.2);

console.log(indTax(300));

const addTaxRate = (rate) => {
  return (value) => {
    return value + value * rate;
  };
};

const vat2 = addTaxRate(0.3);
console.log(vat2(200));
```

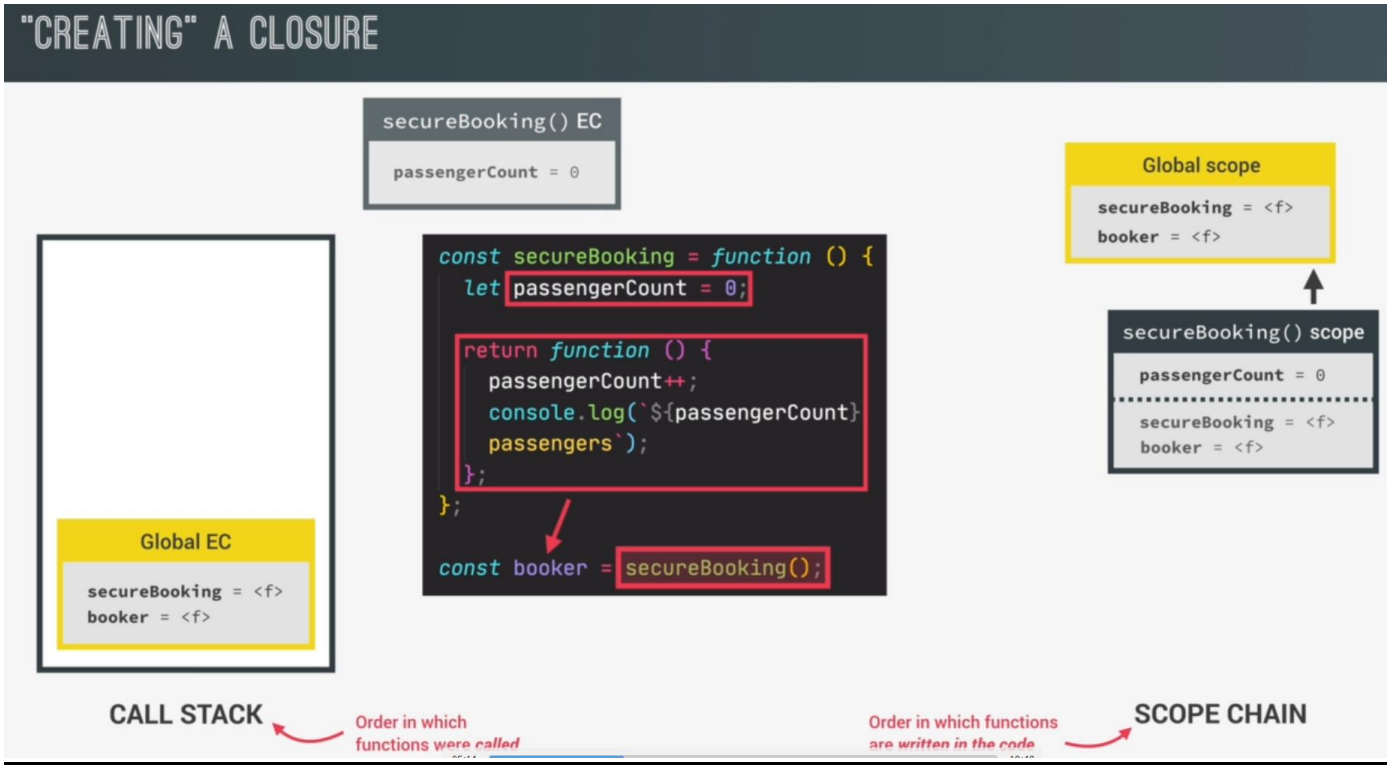
```
// immediately invoked function expression
// to execute a function only once then we use this pattern

(function () {
(function () {
  console.log("this will not execute again");
})();

(() => console.log("arrow funciton will not execute again"))();
```

To declare the private variable then we use the const or let variable to keep it in the scope so that it is used inside the scope only

Var keyword has its own scope.

But all we have to do only for the variable to execute only once then we can use the block and use constant instead of writing the IFFE.

Closures:



What happens in closure is it is not called explicitly but it is called implicitly.

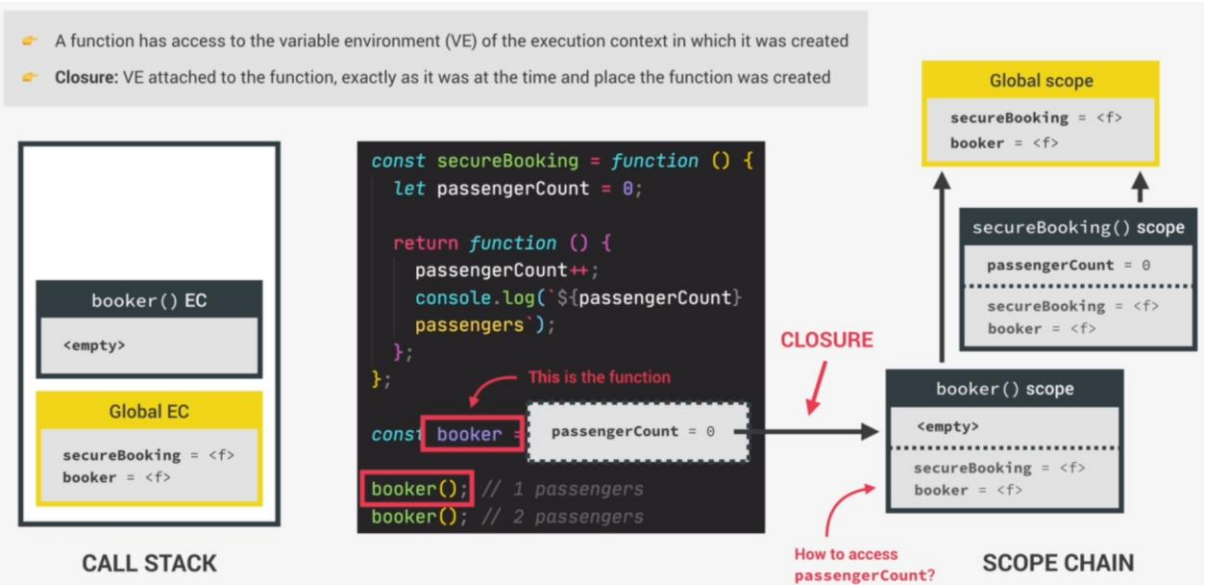
The global stack has only the secure function and it stores the return function.

When securebooking is called on the last line then call stack push the all local variables passengerCount

And when the booker then the callpassenger is poped and

Closure remembers the variables of the function at the time of the birthplace, even when the function is executed and then it uses that in the global scope when it is called.

for the above example as the securebooking is called then the passengerCount is pop from(Execution context) the call stack along with the passengercount variable but it remembered by the closure.



To see the scope of the closure use the below command

```
console.dir(booker);
```

closure has the priority over the global variables.

Array deep dive:

```
let arr = ['a', 'b', 'c', 'd', 'e'];

console.log(arr.slice(2));

console.log(arr.slice(0, -3));
console.log(arr.slice(1, -1));
console.log(arr.slice());
console.log([...arr]);
// both the slice and destructuring are same and what to use is upto
 our own.

// splice
// spice actullay change the original array but works same as the sp
lica method

// console.log(arr.splice(2));
```

```
// it is used to remove the last element in the array
// we can give the  deletecount as the second parameter

console.log(arr[(1, 2)]); // this will delete the 2 element from the
 1 position.
console.log(arr);

//reverse
// this method actually mutat the original array

arr = ['a', 'b', 'c', 'd', 'e'];
let arr2 = ['r', 'u', 'y', 'a', 'm'];

console.log(arr2.reverse());
console.log(arr2);

// concat
const letters = arr.concat(arr2);
console.log(letters); // return the concated arrat
// using the destructuring
console.log([...arr, ...arr2]);

// join
console.log(letters.join(' - '));
let arr = ['a', 'b', 'c', 'd', 'e'];

console.log(arr.slice(2));

console.log(arr.slice(0, -3));
console.log(arr.slice(1, -1));
console.log(arr.slice());
console.log([...arr]);
// both the slice and destructuring are same and what to use is upto
 our own.

// splice
// spice actullay change the original array but works same as the sp
lica method

// console.log(arr.splice(2));

// it is used to remove the last element in the array
```

```
// we can give the  deletecount as the second parameter

console.log(arr[(1, 2)]); // this will delete the 2 element from the
 1 position.
console.log(arr);

//reverse
// this method actually mutat the original array

arr = ['a', 'b', 'c', 'd', 'e'];
let arr2 = ['r', 'u', 'y', 'a', 'm'];
console.log(arr2.reverse());
console.log(arr2);

// concat

const letters = arr.concat(arr2);
console.log(letters); // return the concated arrat
// using the destructuring
console.log([...arr, ...arr2]);

// join
console.log(letters.join(' - '));
```

ForEach Loop for the array set and map

```
// ----- for each ----------------a
// the break and continue does not work with the for each loop.

const movements = [200, 450, -400, 3000, -650, -130, 70, 1300];

//the prameters must be same order
// in this the function passes the each element as the argument.
// 0: function(200)
// 0: function(450)
// 0: function(-400)

// if want to take the index as the argument then in normal for loop
 we use the let (i,movement) movements.entries()
//and in the forEach loop we need to add the first,
or all 3 parameter that is our choice
```

```
// the first is the current element and the second is index and the
third is the entire array
// the sequence is nessecarry -- the prameters must be same order
movements.forEach(function (movement, i,array) {
  if (movement > 0) {
    console.log(`movement ${i + 1}: you deposited ${movement}`);
  } else {
    console.log(`movement ${i + 1}: you withdrew ${movement}`);
  }
});

/// Map ---------------

const currencies = new Map([
  ['USD', 'United States dollar'],
  ['EUR', 'Euro'],
  ['GBP', 'Pound sterling'],
]);

currencies.forEach(function (value, key, Map) {
  console.log(`${key} : ${value}`);
});

// the set has only values and it is totaly to our to to use the val
ue
// we set can have only values and do not have the index so the defa
ult map parameters are proided into the
// forEach loop for the set
const currenciesUnique = new Set(['eur', 'usd', 'inr', 'usd', 'usd']
);

currenciesUnique.forEach(function (key, value, Map) {
  console.log(` ${key} : ${value}`);
});
```