Javascript\_Chai\_aur\_code\_30/10/2024 :

// we are going to read about objects here

// there are mainly 2 ways of declaring objects using the literals , using constructors

// using constructors we have the single ton

// otherwise it is going to be having many references

//both ways are going to be same , except single ton is formed

// const cricketers = {}

// the above is an empty object

const cricketers = {

batman1 : "Dhoni",

batman2 : "kohli",

bowler1 : "jaspreet",

wickKeeper : "rishab pant"

}

// these are going to be key values

// the key can be anything and the value can be anything

// there is no need of writing the key as " "

// we can write simply , "" is also fine

//

// accessing objects values using . operator

console.log(cricketers.batman1)

// we can access in other way also

// we need to use the string form of the key here

console.log(cricketers["bowler1"])

// this form is useful when we are going to have the space in keys

// like if we have the key as full name then we can use

// console.log(cricketers["full name"])

// symbols

// first we need to declare

const yellow = Symbol("yellowFruit")

const fruits = {

pink : "strawberry",

cricketers.wickKeeper = "dhoni"

// to stop chaningw e need to sue the freeze()

console.log(cricketers.wickKeeper)

Object.freeze(cricketers)

cricketers.batman2 = "yuvraj"

console.log(cricketers)

console.log(fruits)

[yellow] : "mango"

}

console.log(typeof [yellow])

console.log(typeof fruits.pink)



Functions In objects :

// functions in object

const scientists = {

name : "newton",

invention : function(){

console.log("Gravity")

}

}

console.log(scientists.invention())

scientists.invention()

console.log(scientists.invention

Gravity

undefined

Gravity

[Function: invention]

**This :**

**const countries = {**

**name : "India",**

**capital : "Delhi",**

**display: function(){**

**console.log(this.name)**

**}**

**}**

**countries.display()**

**Lecture 17 : Objects in Javascript part 2**

const japanese = new Object()

console.log(japanese)

const vocabulary = {}

console.log(vocabulary)

{}

{}

Both are same , but one is singleton using constructor

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const user = {

email : "dekshitha@gmail.com",

fullname : {

userfullname:{

firstname : "lahari",

lastname : "boreddy"

}

}

}

console.log(user)

console.log(user.email)

console.log(user.fullname)

// console.log(user.firstname)

{

email: 'dekshitha@gmail.com',

fullname: { userfullname: { firstname: 'lahari', lastname: 'boreddy' } }

}

dekshitha@gmail.com

{ userfullname: { firstname: 'lahari', lastname: 'boreddy' } }

It is nested , to access inside things we need to use nesting .’s

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const user = {

email : "dekshitha@gmail.com",

fullname : {

userfullname:{

firstname : "lahari",

lastname : "boreddy"

}

}

}

console.log(user.fullname.userfullname)

console.log(user.fullname ?.userfullname )

// this is goingt o have a question amek afte rthe fullnam e, to check if teh fullname is tehre or not

{ firstname: 'lahari', lastname: 'boreddy' }

{ firstname: 'lahari', lastname: 'boreddy' }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Combining objects

const obj1 = {1:'a', 2:"b"}

const obj2 = {

3: "c",

4 : "d"

}

const obj3 = {obj1, obj2}

console.log(obj3)

{ obj1: { '1': 'a', '2': 'b' }, obj2: { '3': 'c', '4': 'd' } }

Same problem as in arrays

const obj1 = {1:'a', 2:"b"}

const obj2 = {

3: "c",

4 : "d"

}

const obj3 = Object.assign(obj1, obj2)

//

console.log(obj3)

// way 2

const obj4 = Object.assign({}, obj1, obj2)

console.log(obj4)

{ '1': 'a', '2': 'b', '3': 'c', '4': 'd' }

{ '1': 'a', '2': 'b', '3': 'c', '4': 'd' }

The above is better , to sue assign ,and {} as a parameter

// can use as many objects as needed to combine

Using spread operator

const obj1 = {1:'a', 2:"b"}

const obj2 = {

3: "c",

4 : "d"

}

const obj3 = {

name : "deekshi",

id : "ap211"

}

const obj4 = {...obj1, ...obj2, ...obj3}

console.log(obj4)

{

'1': 'a',

'2': 'b',

'3': 'c',

'4': 'd',

name: 'deekshi',

id: 'ap211'

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const users = [

{

name : "deekshi",

email : "123@gmail.com"

},

{

name : "lahari",

email : "lahari@gmail.com"

}

]

// array having multiple objects

console.log(users)

console.log(users[1].name)

[

{ name: 'deekshi', email: '123@gmail.com' },

{ name: 'lahari', email: 'lahari@gmail.com' }

]

lahari

Keys and values ;

const ministers = {

name : "rajnath",

email : "raj@gail.com"

}

console.log(Object.keys(ministers))

// [ 'name', 'email' ]

console.log(Object.values(ministers))

// [ 'rajnath', 'raj@gail.com' ]

console.log(Object.entries(ministers))

// [ [ 'name', 'rajnath' ], [ 'email', 'raj@gail.com' ] ]

console.log(ministers.hasOwnProperty('name'))

// true

[ 'name', 'email' ]

[ 'rajnath', 'raj@gail.com' ]

**LECTURE 18 : Object de-structure and JSON API intro**

**const course = {**

**name : "js ",**

**price : 1000,**

**instructor : "hitesh"**

**}**

**// we already know the usage , or accessing the things**

**// destructuring**

**const {instructor} = course**

**console.log(instructor)**

**// hitesh**

**const {price : price2} = course**

**// changing the anme if needed**

**console.log(price2)**

Destructuring is used in props in react

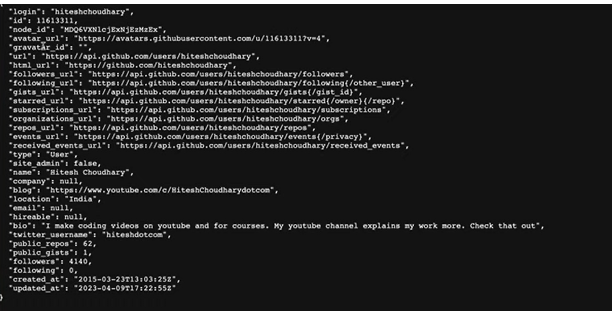
APIS : JSON

// api values aer goung to come in the form of teh JSON

// it si nothing but teh object from

// but it is not going to have any name

//



How to retrieve them

We can convert them to the object and can sue

We might get the API as the arrays also , it is going on the arrays of objects

**LECTURE 19 : Functions and parameter in javascript**

// let us strat about the functions

// the main part of the functions is for the reusability

// we can use a methods / function that is going to help in making the code repeatable

//

Declaring a function

function fun\_name(parameters){

return

}

// to execute the function we need to sue the ()

// there are many ways to use the functions , like using the variable name , suing the arrow function

// here there is no need to provide the data type for the parameters

Code

function addNumbers(num1, num2){

return num1+num2

}

console.log(addNumbers(2,4))

// if we don't return any value and call the function using a variable , then the result is going to be undefined

function addNumbers(num1, num2){

return num1+num2

}

Const result = addNumbers(2,4)

// here the result value is undefined

// after return if we do something then that is useless

Returning strings :

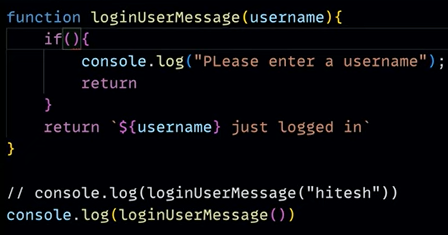
function greetings (name){

return `hello ${name}`

}

console.log(greetings("lahari"))

Example codes :



**Lecture 20 : Functions with objects and array in javascript**

// arrays and functions using arrays , objects

// accepting multiple values in a function as an array

// using rest operator

function calculatePrice(...num){

return num

}

console.log(calculatePrice(200,320,450))

Code 2 :

function calculatePrice(val1, val2, ...num){

return num

}

console.log(calculatePrice(200,320,450,90,1000))

Code : passing an object

const user = {

username : "lahari",

price : 1090

}

function handleObject(anyobject){

console.log(`username is ${anyobject.username} and price is ${anyobject.price}`)

}

handleObject(user)

In another way :

const user = {

username : "lahari",

price : 1090

}

function handleObject(anyobject){

console.log(`username is ${anyobject.username} and price is ${anyobject.price}`)

}

handleObject({username:"sai", price : 2345})

Function passing an array :

const myArray = [23,40,21,78]

function returnSecondValue(getArray){

return getArray[1]

}

console.log(returnSecondValue(myArray))

Another way :

const myArray = [23,40,21,78]

function returnSecondValue(getArray){

return getArray[1]

}

console.log(returnSecondValue([2,1,3,4]))

Lecture 21 : **Global and local scope in javascript**

Code 1 :

// let , var , const have different scopes

let a = 10

const b = 45

var c = 90

console.log(a)

console.log(b)

console.log(c)

Code :

// curly bases is the scope , it is going to declare the scope , when the curly bases are with the functions , if else , then that is called as scope

//

if(true){

// the variables in the scope should be available there only, that means the things are following the scope

let a = 8

const b = 45

var c = 90

}

CODE:

if(true){

// the variables in the scope should be available there only, thta means the things are following the scope

let a = 8

const b = 45

var c = 90

}

// the above a, b , c should not be taken outside , they aer not allowed outside

//

console.log(c)

// console.log(a)

// console.log(b)

// the above 2 are going to show error , therefore they are correct

// but if the things are global , then they should be accesses always

// if there are 2 types of values , then they are going to take the inner value inside

// for any outside outside values are used

**LECTURE 22 : Scope level and mini hoisting in javascript**

// we need to know about nested scoping here

// we are going to read about the closures

function one(){

const username = "lahari"

function two(){

const website = "youtube"

console.log(username)

}

console.log(website)

two()

}

one()

This is error the website is not accessible outside

Chote bade se le sakta hai , but the opposite is not possible

If else scoping :

**if(true){**

**const username = "lahari"**

**if(username==="lahari"){**

**const website = "youtube"**

**console.log(username+ website)**

**}**

**console.log(website)**

**// this shows error**

**}**

**console.log(username)**

**// this shows error**

**Calling function using variable**

// function usage way 2

const addone = function(num){

return num+1

}

console.log(addone(45))

Difference in these two ways :

console.log(addTwo(34))

function addTwo(num){

return num + 2

}

// the above way can be accessed before declaration

// the below way can't be possible before declaration

// console.log(addone(45))

// const addone = function(num){

// return num+1

// }

This and arrow functions :

// arrow functions and the this keyword

// this is going to say about the current context

const user = {

username : "lahari",

price : 999,

welcomeMessage : function(){

console.log(`${this.username}, welcome to website`)

}

}

user.welcomeMessage()

CODE 2 :

// arrwo functions and the rhis keyword

// this is going to say about teh current context

const user = {

username : "lahari",

price : 999,

welcomeMessage : function(){

console.log(`${this.username}, welcome to website`)

}

}

user.welcomeMessage()

// this is also going to run

user.username = "durga"

user.welcomeMessage()

CODE :

// arrwo functions and the rhis keyword

// this is going to say about teh current context

const user = {

username : "lahari",

price : 999,

welcomeMessage : function(){

console.log(`${this.username}, welcome to website`)

console.log(this)

}

}

user.welcomeMessage()

// this is also going to run

user.username = "durga"

user.welcomeMessage()

lahari, welcome to website

{

username: 'lahari',

price: 999,

welcomeMessage: [Function: welcomeMessage]

}

durga, welcome to website

{

username: 'durga',

price: 999,

welcomeMessage: [Function: welcomeMessage]

}

Code :

console.log(this)

{}

In browser this is going to be different

CODE :

function chai(){

console.log(this)

}

chai()

Output :

<ref \*1> Object [global] {

global: [Circular \*1],

clearImmediate: [Function: clearImmediate],

setImmediate: [Function: setImmediate] {

[Symbol(nodejs.util.promisify.custom)]: [Getter]

},

clearInterval: [Function: clearInterval],

clearTimeout: [Function: clearTimeout],

setInterval: [Function: setInterval],

setTimeout: [Function: setTimeout] {

[Symbol(nodejs.util.promisify.custom)]: [Getter]

},

queueMicrotask: [Function: queueMicrotask],

structuredClone: [Function: structuredClone],

atob: [Getter/Setter],

btoa: [Getter/Setter],

performance: [Getter/Setter],

fetch: [Function: value],

crypto: [Getter]

}

Arrow way :

// using arrow function

const name = ()=>{

//logic

}

Code 2 :

const name = ()=>{

console.log('hello')

}

name()

// in arrow function and the normal function the difference is with the this

Explicit r3turn :

const addNumbers = (num1, num2)=>{

return num1+num2

}

console.log(addNumbers(4,5))

Implicit :

const addNumbers = (num1, num2)=> num1+num2

console.log(addNumbers(4,5))

// here we removed return and teh braces

Way 2 :

const addNumbers = (num1, num2)=> (num1+num2)

console.log(addNumbers(4,5))

//this is also ok

Then why ()

This is important to return an object

const name = ()=> ({username:"lahari"})

console.log(name())

**But when we use {} we need to sue the return word**

Arrow functions are mainly used in loops of arrays

**LECTURE 24 : Immediately Invoked Function Expressions IIFE**

// immediately invoked function expression

// we want to run the connection immediately

// /also when the child function should not pollution with the child function then we are going to use IIFE

function chai(){

console.log("heelo")

}

chai()

Way 2 :

(function chai(){

console.log("heelo")

})()

// (function def)()

Using arrow functions :

// using arrow function

(()=>{

console.log("heelo")

})()

Way 3 : problem

// named IIFE

(function chai(){

console.log("heelo")

})();

// (function def)()

(()=>{

console.log("heelo")

})()

Here the code 1 , should be end in the case of the 2 IIFE functions

Using params :

// parameters IIFE

((name)=>{

console.log(`${name} hello`)

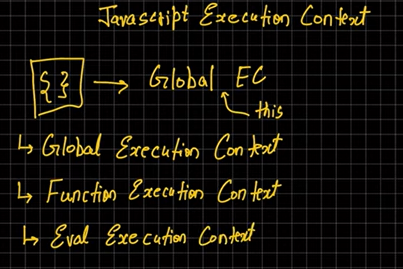
})("lahari")

LE**CTURE 25 : How does javascript execute code + call stack**

// execution context is different in different environments

// browser ahs the window object

// in every environment we have the GE



From 2:45 to 4 done reading

Started at 5: 30

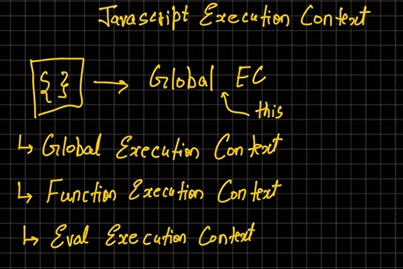
// phasess

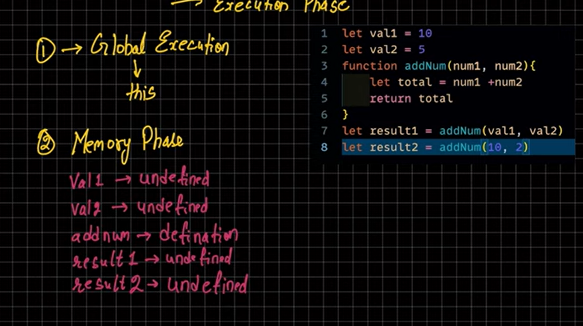
// memory creation phase

// execution phase

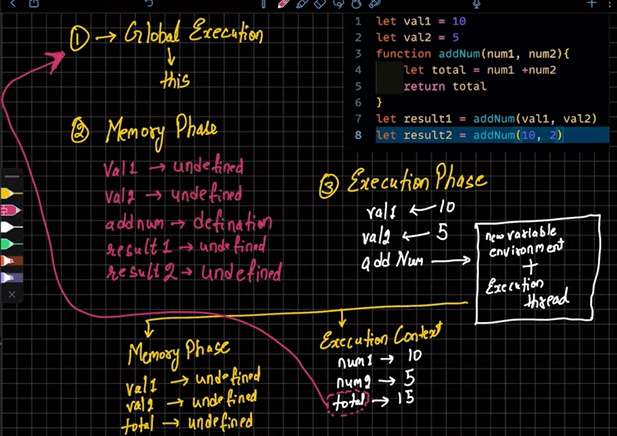
// in the memory creation phase teh jagah is going to be allocated

//

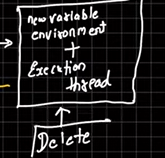
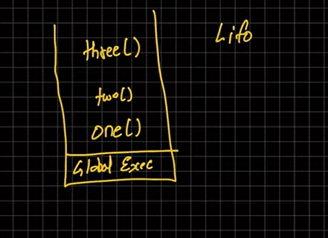




—---------------------



—----------------

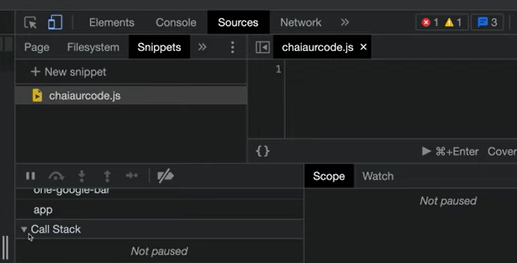
 

After doing its work

Stack concept is followed here

How to check this

Go to source in browser inspect

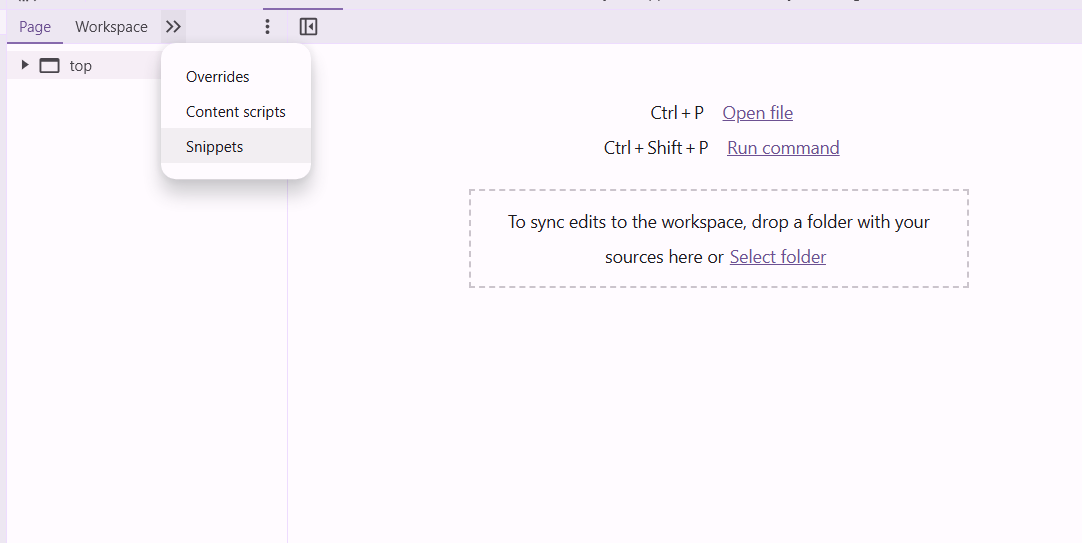


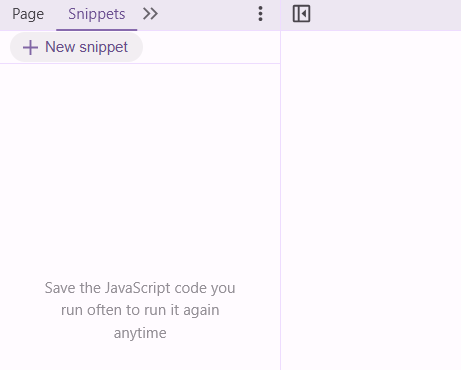
Click call stack



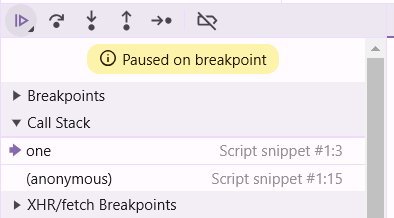
Write the above code there

Run by clicking this

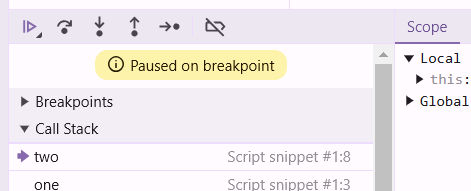




—---------------



—------------------



—----------you can do this

**Lecture 26 : Control flow in javascript in 1 shot**

If :

// if statement

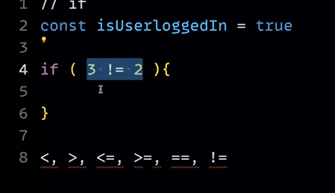
if(condition){

}

else{

}

—----------------



—----------------

Code 1:

if(2=="2"){

console.log("success")

}

Scope in if :

const score = 200

if(score>100){

const power = "fly"

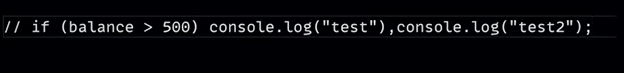
console.log(`user power is ${power}`)

}

console.log(fly)

Error

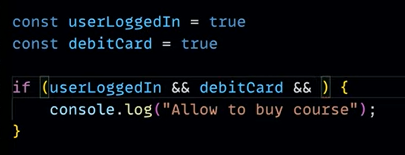
Unreadable :

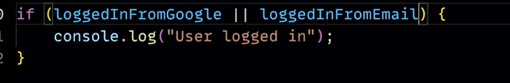


**If else is already known**

**Else if ladder**

&& and || operator :





Switch case

const month = 3

switch(month){

case 1:

console.log("January")

break

case 2:

console.log("February")

break

case 3:

console.log("March")

break

case 4:

console.log("April")

break

case 5:

console.log("may")

break

case 6:

console.log("June")

break

case 7:

console.log("July")

break

case 8:

console.log("August")

break

case 9:

console.log("September")

break

case 10:

console.log("October")

break

case 11:

console.log("November")

break

case 12:

console.log("December")

break

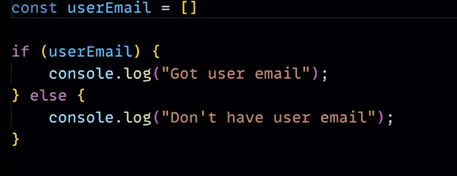
default :

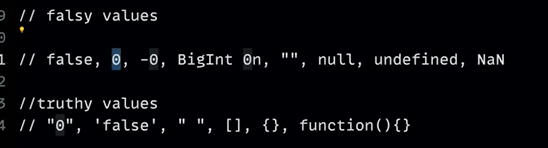
console.log("invalid")

}

Break and the default are important here

Truthy and falsy values :





Detecting empty or not using the length



false==0

true

true==0

false

false==''

true

0==''

True

**Nullish coalescing operator :**

// everything is dealt with null , undefined

const val = null ?? 10

// null is assigned

// this is going to assign the safety value , based on the safety checking

—------------------

var num= undefined ?? 23

// 23 is assigned

console.log(num)

—---------------------

val1 = null ?? 10?? 15

// 10 is assigned

console.log(val1)

Ternary operator :

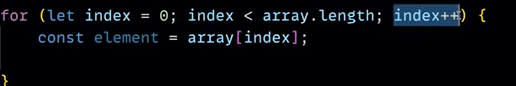
const price = 100

price >= 100 ? console.log("Don't buy"): console.log("i am buying ")

// don't but

**Lecture 27 : Loops**

# **For loop with break and continue in javascript**



For loop :

for(let i=0;i<5;i++){

console.log(`outer loop value : ${i}`)

for(let j=0;j<5;j++){

console.log(`the value for the outer loop ${i} is ${j}`)

}

}

Tables :

for(let i=1;i<11;i++){

console.log(`5\*${i}=${5\*i}`)

}

For arrays

const heros = ['siva', 'karthi', 'surya']

for(let i=0;i<heros.length;i++){

console.log(heros[i])

}

Break and continue :

for(let i=1;i<10;i++){

if(i==5){

break;

}

if(i==3){

continue

}

console.log(i)

}

Output :

1

2

4

**Lecture 28 : While and do while loop in Javascript**

let num = 5

// initialisation

while(num<10){ // consition

console.log(num)

num++ // updating variable

}

—--------------------

While

let arr = ['banana', 'apple', 'mango']

let i =0

while(i<arr.length)

{

console.log(arr[i])

i++

}

banana

apple

mango

—--------------------

Do while

let arr = ['banana', 'apple', 'mango']

let i =0

do{

console.log(arr[i])

i++

}while(i<arr.length)

banana

apple

mango

**Lecture 29 : High Order Array loops**

// for of

// this is specific for the array

const arr = [1,2,3,45,6]

for (const i of arr) {

console.log(i)

}

Output :

1

2

3

45

6

—---------------

const greetings = "Good morning"

for(const letter of greetings){

console.log(letter)

}

G

o

o

d

m

o

r

n

i

n

g

MAPS :

// map

const map = new Map()

map.set("IN", "india")

map.set("USA", "America")

map.set("FR", "France")

console.log(map)

Map(3) { 'IN' => 'india', 'USA' => 'America', 'FR' => 'France' }

They store unique value and also maintain the order

—-------------------

// map

const map = new Map()

map.set("IN", "india")

map.set("USA", "America")

map.set("FR", "France")

console.log(map)

for(const key of map){

console.log(key)

}

[ 'IN', 'india' ]

[ 'USA', 'America' ]

[ 'FR', 'France' ]

—------------

To destructure this we have

// map

const map = new Map()

map.set("IN", "india")

map.set("USA", "America")

map.set("FR", "France")

console.log(map)

for(const [key, value] of map){

console.log(key, ":-", value)

}

IN :- india

USA :- America

FR :- France

—--------------

const myObject = {

game1 : "さっか",

"game2" : "やあきゅ"

}

console.log(myObject)

{ game1: 'さっか', game2: 'やあきゅ' }

—---------------

const myObject = {

name : "lahari",

password : 1234

}

for(const [key, value] of myObject){

console.log(key , value)

}

// myObject is not iterable

For in for objects :

const myObject = {

name : "lahari",

password : 1234

}

for (const key in myObject) {

console.log(key)

}

name

Password

—----------------

const myObject = {

name : "lahari",

password : 1234

}

for (const key in myObject) {

console.log(key)

console.log(myObject[key])

}

Array for for in :

const arr = [2,3,4,"hello"]

for (const key in arr) {

console.log(key,":-",arr[key])

}

// here key is nothing but index

0 :- 2

1 :- 3

2 :- 4

3 :- hello

For each :

japanese.forEach((item , index , japanese)=>{

})

Here the for each is going to take the function as a parma ,

But there should be no name for the function

Using arrow function

const japanese = ["konnichiwa", "ohayou", "arigatou"]

japanese.forEach((item , index , japanese)=>{

console.log(item)

})

konnichiwa

ohayou

arigatou

// for each is going to take the item , index , array\_name as paraams, it is not needed to take all

Using normal function as callback :

const japanese = ["konnichiwa", "ohayou", "arigatou"]

japanese.forEach(function(item , index , japanese){

console.log(item)

})

konnichiwa

ohayou

arigatou

Another way ;

const japanese = ["konnichiwa", "ohayou", "arigatou"]

function display(item , index , japanese){

console.log(item)

}

japanese.forEach(display)

// here we wrote only display in forEach , because that si going to return the function and that function is going to become the parameter

Array with objects :

const japanese = [

{

"konnichiwa" : "hello",

"type" : "hiragana"

},

{

"食べます" : "to eat",

"type": "kanji"

},

{

"eakon" : "AC",

"type" : "katakana"

}

]

japanese.forEach((item)=>{

console.log(item["type"])

})

hiragana

kanji

katakana

This is used in making the cards for the items , we are going to sue JSONa nd then use for each and that is it

**Lecture 30 : Filter map and reduce in javascript**

// filter , reduce , map

// for each extra

const langauges = ["telugu", "Hindi", "English", "Japanese"]

const values = langauges.forEach((item) => {

console.log(item)

});

console.log(values)

// undefined

// the foreach is not going to return anything

Filter method : it is going to return values automatically

const values = [1,2,4,5,7,8,90]

const nums = values.filter((item)=>{

return item>3

})

console.log(nums)

// [ 4, 5, 7, 8, 90 ]

We already know about the () in arrow functions

Using {} we need to use return word

This can be done in forEach also

const values = [1,2,4,5,7,8,90]

values.forEach((item)=>{

if(item>3){

console.log(item)

}

})

Output :

4

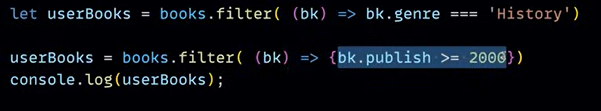
5

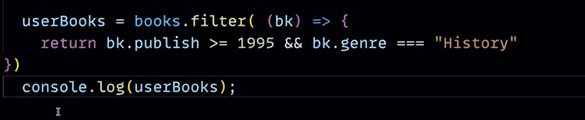
7

8

90

Real usage :





Map :

const nums = [1,2,3,45,6,6,7,8,90]

// adding 10 to all number

const values = nums.map((item)=>{

return item+10

})

console.log(values)

[

11, 12, 13, 55, 16,

16, 17, 18, 100

]

Chaining in maps :

const nums = [1,2,3,45,6,6,7,8,90]

// adding 10 to all number

const values = nums.map((item)=>{

return item+10

}).map((item)=>{

return item+5

})

console.log(values)

[

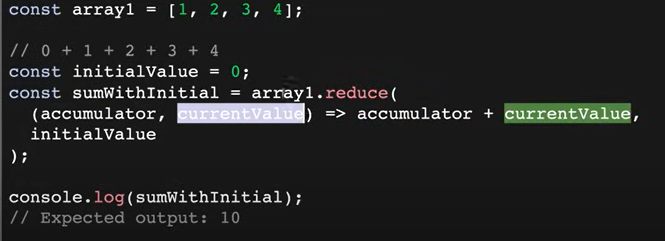
16, 17, 18, 60, 21,

21, 22, 23, 105

]

Reduce:

Example :



—---------------

const nums = [10,20,30,40]

const total = nums.reduce((acc, curr)=>{

return acc+curr

},0)

// after function we need to have the , and then initial value

console.log(total)

// 100

—--------------

const nums = [10,20,30,40]

const total = nums.reduce((acc, curr)=>{

return acc+curr

}, 1000)

// after function we need to have the , and then initial value

console.log(total)

// 1100

Real value

We Have the array of objects and then we have the prices of products , by accessing them we can find the sum of prices

DOM :