Css – Concept

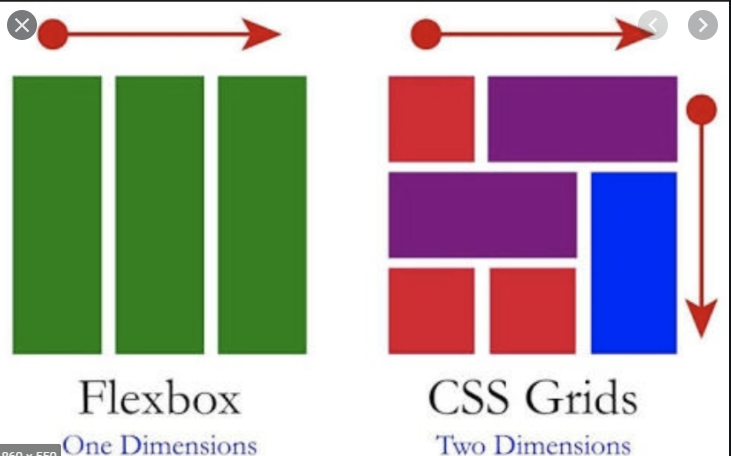
If we want to create webpage layout then we have 3 alternative to do this

1)Flex box

2)bootstrap 4

3) Css grid

But css grid and flexbox are pretty much good to do whaterver we want to do.



Flexbox can be used for one dimensional and css grid can we used for 2 dimensional.

<meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1">

To display an HTML page correctly, a web browser must know the character set used in the page.

UTF-\* it specify the character encoding for the Unicode

Viewport define that this layout is mainly for mobile device zoom in zoom out scroll all will work nicely here.

border-radius:4px;

its use to make corner of box round.

 display:inline-block;

Its use to make div side by side

If we write display:block then div will come one in one row.

When ever there is use of grid always try to use css-grid

Thus to create a css grid first step is to create a container class.

<div class=”container”>

</div>

Now first step of grid css is

.container{

**display:grid** //it says we are going to use css grid property

**grid-template-columns: 1fr 1fr 2fr** //it says how column we are going to use in grid.create 3 fraction and but first to half of 3rd one.

Similar way of saying 1fr 1fr 1fr is **repeat(3,1fr)**

There is auto property which is use to say take the least space to fill the grid

**Gap:10px;** //put the gap between grid column cells

**Grid-template-column : auto 1fr 1fr;** //here first cell will take the least space it need to fill the data it has inside.

Grid-template-column:repeat(auto-fill,minmax(200px,1fr));//

The **minmax()** [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) [function](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Functions) defines a size range greater than or equal to min and less than or equal to

.green{

    grid-column: 1/3;

}

It says this green class cell start from first cell and end at the start of 3rd cell i.e take space of two cell.

**Grid-column: 1/-1** means cover all cell distance in a first row.

**Grid-column: span 2** //means span upto two cells.

Grid-row: 1/3 // means start with first row and go upto 2 row.

Justify-self: start // means align alon vertical

Align-self:start // means align along horizontal

}

container

display

Establishes a new grid formatting context for children.

display: grid;

display: inline-grid;

display: subgrid;

grid-template

Defines the rows and columns of the grid.

grid-template-columns: 12px 12px 12px;  
grid-template-rows: 12px 12px 12px;

grid-template-columns: repeat(3, 12px);  
grid-template-rows: repeat(3, auto);

grid-template-columns: 8px auto 8px;  
grid-template-rows: 8px auto 12px;

grid-template-columns: 22% 22% auto;  
grid-template-rows: 22% auto 22%;

grid-gap

Specifies the size of column and row gutters.

grid-row-gap: 1px;  
grid-column-gap: 9px;

grid-gap: 1px 9px;

grid-gap: 6px;

justify-items

Aligns content in a grid item along the row axis.

justify-items: start;

justify-items: end;

justify-items: center;

justify-items: stretch; (default)

align-items

Aligns content in a grid item along the column axis.

align-items: start;

align-items: end;

align-items: center;

align-items: stretch; (default)

justify-content

Justifies all grid content on row axis when total grid size is smaller than container.

justify-content: start;

justify-content: end;

justify-content: center;

justify-content: stretch;

justify-content: space-around;

justify-content: space-between;

justify-content: space-evenly;

align-content

Justifies all grid content on column axis when total grid size is smaller than container.

align-content: start;

align-content: end;

align-content: center;

align-content: stretch;

align-content: space-around;

align-content: space-between;

align-content: space-evenly;

grid-auto-flow

Algorithm for automatically placing grid items that aren't explictly placed.

grid-auto-flow: row;

grid-auto-flow: column;

grid-auto-flow: dense;

children

grid-column

Determines an items column-based location within the grid.

grid-column-start: 1;  
grid-column-end: 3;

grid-column-start: span 3;

grid-column-start: 2;  
grid-column-end: 4;

grid-column: 2 / 3

grid-column: 2 / span 2

grid-row

Determines an items row-based location within the grid.

grid-row-start: 1;  
grid-row-end: 3;

grid-row-start: span 3;

grid-row-start: 2;  
grid-row-end: 4;

grid-row: 1 / 3;

grid-row: 1 / span 3;

grid-row + grid-column

Combining grid rows with grid columns.

grid-row: 1 / span 2;  
grid-column: 1 / span 2;

grid-row: 2 / span 2;  
grid-column: 2 / span 2;

justify-self

Aligns content for a specific grid item along the row axis.

justify-self: start;

justify-self: end;

justify-self: center;

justify-self: stretch; (default)

align-self

Aligns content for a specific grid item along the column axis.

align-self: start;

align-self: end;

align-self: center;

align-self: stretch; (default)

//to make any thing in center

Align-items:center

Justify-content:center

//In order to make navigation bar fixed on scroll we can do that using

.sticky{

Position:fixed;

Top:0;

Width:100%

}

**-------------------------**

**JAVASCRIPT TYPES**

**-----------------**

**1. Number**

**2. String**

**3. Boolean**

**4. Undefined**

**5. Null**

**<!-- 6. Symbol (new in ECMAScript 6) -->**

**7. Object**

**JAVASCRIPT COMPARISONS**

**-----------------**

**!==**

**===**

**>=**

**<=**

**>**

**<**

**JAVASCRIPT VARIABLES**

**-----------------**

**var**

**<!-- let (new in ECMAScript 6)-->**

**<!-- const (new in ECMAScript 6)-->**

**JAVASCRIPT CONDITIONALS**

**-----------------**

**if**

**else**

**else if**

**<!-- ternary operator -->**

**<!-- switch -->**

**JAVASCRIPT LOGICAL OPERATORS**

**-----------------**

**&&**

**||**

**!**

**JAVASCRIPT FUNCTIONS**

**-----------------**

**var a = function name() {}**

**function name() {}**

**return**

**<!-- () => (new in ECMAScript 6) -->**

**JAVASCRIPT DATA STRUCTURES**

**-----------------**

**Array**

**Object**

**JAVASCRIPT LOOPING**

**-----------------**

**for**

**while**

**do**

**forEach (new in ECMAScript 5)**

**JAVASCRIPT KEYWORDS**

**-----------------**

**break**

**case**

**catch**

**class**

**const**

**continue**

**debugger**

**default**

**delete**

**do**

**else**

**export**

**extends**

**finally**

**for**

**function**

**if**

**import**

**in**

**instanceof**

**new**

**return**

**super**

**switch**

**this**

**throw**

**try**

**typeof**

**var**

**void**

**while**

**with**

**yield**

**Javascript Advanced Topics**

**What is the scope of any variable or function defined in console**

**Ans- The scope of the variable is Root scope (window).**

function bb (){

var a = "This is the first scope"

console.log(a);

}

window.bb()

This is the first scope

**Javascript Conditionals**

If

Else

If else

Ternary operator (condition ? true : false)

Switch

**Ecmascript 6 i**s nothing but a javascript

**Let and Const --**

Const is used when we need to keep any variable assigned value const throughout the program.

**The const declaration creates a read-only reference to a value. It does not mean the value it holds is immutable—just that the variable identifier cannot be reassigned.**

Ex – const a = function(){}

Because this function is not going to change.

Const obj = {

Player:’Raj’,

Experience:100,

wizardLevel:false

}

If we try to put obj value to Boolean like

Obj = false;

It will give error but if we try ---

Obj.wizardLevel = True then it will get assigned .It will not give error.

**var and let are both used for variable declaration in javascript but the difference between them is that var is function scoped and let is block scoped.  
It can be said that a variable declared with var is defined throughout the program as compared to let.**

console.log(x1);

var x1=5;

console.log(x1);

undefined

5

Because var is declared at the top but not assigned the valsue so we are getting the undefined.

**console.log(x2);**

**let x2=5;**

**console.log(x2);**

**Uncaught ReferenceError: x2 is not defined**

Note :-- If a variable is accessed before **defining** then **JS** will show it as **not defined**, **and** if a variables is **defined** but **not** initialized I.e. **no** values is assigned it to it before accessing, then its **undefined**.

**Destructuring of object**

Const obj ={

Player:’bobby’,

Experience:100,

wizardLevel:false

}

Const Player = obj.Player;

Const Experience = obj.Experience;

Const {Player,Experience} = obj;

This will destructure the object variable and assigned to variable;

**New way of Declaring Property Variable in Object**

**Const name =”john snow”**

**Const obj = {**

**[name] : “hello”,**

**[1+3]:”hihi”**

**}**

Thus we declaring a dynamic property in obj object.

Suppose we want to declare object property and value the same then we can do something like this

**const obj = {**

**a:a,**

**b:b,**

**c:c**

**}**

Then we can do it like this **const obj = {a,b,c}**

Template String in Javascript

Const name = ‘Shally’

Const age = 24

Const Fage= 55

Const greeting = `Hello ${name} you seems to be ${age}.Your father seems like of ${Fage}`

**Closure Function**

**It is function ran --the function executed.Its never going to execute again //But its going to remember that there are reference to those variable //so the chuild scope always has access to the parent scope.**

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives you access to an outer function’s scope from an inner function.

--inner function has acess to the outer finction variable.

function init() {

var name = 'Mozilla'; // name is a local variable created by init

function displayName() { // displayName() is the inner function, a closure

alert(name); // use variable declared in the parent function

}

displayName();

}

init();

init() creates a local variable called name and a function called displayName(). The displayName() function is an inner function that is defined inside init() and is available only within the body of the init() function. Note that the displayName() function has no local variables of its own. However, since inner functions have access to the variables of outer functions, displayName() can access the variable name declared in the parent function, init().

**Currying**

Const multiply =(a,b) => a\*b

But same thing can be done like this

Const curriedMultiply =(a)=>(b)=>a\*b

In this case this will get call

curriedMultiply(5)(6) 🡺 give result 30

or some thing like this

const curry = curriedMultiply(5);

curry(6)

A curried function is a function that takes multiple arguments one at a time. Given a function with 3 parameters, the curried version will take one argument and return a function that takes the next argument, which returns a function that takes the third argument. The last function returns the result of applying the function to all of its arguments.

Example -

// logNow will be the partial of log with fixed first argument

let logNow = log(new Date());

// use it

logNow("INFO", "message"); // [HH:mm] INFO message

Now logNow is log with fixed first argument, in other words “partially applied function” or “partial” for short.

We can go further and make a convenience function for current debug logs:

let debugNow = logNow("DEBUG");

debugNow("message"); // [HH:mm] DEBUG message

**Composition**

**Function composition** is the process of combining two or more functions to produce a new function.

const compose = (...funcs) =>

initialArg => funcs.reduce((acc, func) => func(acc), initialArg);

const log = arg => {

console.log(arg);

return arg;

};

const store = key => arg => {

sessionStorage.setItem(key, JSON.stringify(arg));

return arg;

};

const getPerson = id => id === 'homer'

? ({ firstName: 'Homer', surname: 'Simpson' })

: {};

const getPersonWithSideEffects = compose(

getPerson,

log,

store('person'),

);

const person = getPersonWithSideEffects('homer');

const sum = numbers =>

numbers.reduce((total, number) => total + number, 0);

sum([2, 3, 5, 7, 9]); // => 26

remember this reduce function.Which combine all function and give one function.

So in our case it will be work like ---

getPerson(log(store(‘Person’)(‘Homer’)));

**Important**

In function try to remove side effects and enhance function purity

Make it deterministic(for a give input and we always get same output)

Ex –

What is side effects –Any thing which we are doing inside function in changing the outside variable is side effects.

Inside the scope of a fuction we always change the variable which are inside it not the outside scope variable.

**Array**

Foreach

Const array =[1,2,10,16]

Const double=[];

Const newArray = array.foreach((num)=>{

double.push(num\*2);

});

console.log(double)

/\*It will double the array values\*/

//**Map filter Reduce**

**Map** works the same way foreach works but it always return something.

It loop the array.

Const double = array.map((num)=>{return num\*2})

**Filter** loop over array and return only those element which fulfil the condition

Ex- const filterArray = array.filter(num => {

Return num>5

})

//this will return only those element which are greater than 5.

**Reduce**

Reduce return a single value .

Const reduceArray = array.reduce((accumulator,num)=>{

Return accumulator+num},//define initial value of accumulator i.e **1**);00

**Array and Its various method**

**Add items – Array.splice()**

Ex-

Let arr =[5,1,8];

Arr.splice(2,0,’tacos’) //insert at index 2

[5,1,’tacos’,8]

Month = [‘j’,’k’,’l’,’m’,’n’]

Month.splice(4,1,’z’) //replace element at index 4 by ‘z’

Month== [‘j’,’k’,’l’,’m’,’z’]

//if delete count value is greater than or equal to length of array all element will get deleted.

//if 0 or –ve no element deleted.

Return value array contain the deleted value.

**Add items – At the end of array**

Array.Push();

If we want to add one array into other we can do so

Array.prototype.push.apply(month,month2)

This month will have all the element which are present in month2.added at the end.

**Add items – At the front of array**

Arr.unshift(2);

Add – one array to another

Const arr3 = arr1.concat(arr2);

**Const arr4=**arr1.concat(arr3,arr2);

**Remove from array**

**Splice**

arr.splice(2,1) //means remove 1 element from 2nd index

**Pop**

Remove last element from the array and return that element.

//that element is also removed from array.

**Shift**

arr.shift(); // remove first element from the array and return thatbelement.

//that elent is also removed from the array.

**Slice**

**It return the shallow copy of a portion of an array.**

**Start,end and end not included.**

Const arr = [“a”,”b”,”c”,”d”,”e”];

Ex- arr.slice(2) 🡺 arr.slice(2,0) means start from 2 and goes till end.

**-ve index means start from end**

**This will always return new array containg element between index given and old array is as it is.**

**--------------------------------------------------------------------------**

**Find Items from Array**

**One item**

**Element exist --** arr.includes(1); this gives true false whether elemnt 1 exist or not**.**

**First Index- arr.indexOf(1); this gives first index of ‘1’ elemnt.If not present then -1;**

**Last index – arr.lastindexOf(1);this gives the last index of ‘1’.If not present then it will give -1**

**Element that Satisfy condition**

**Arr.find(a=>a<2);**

**Find will return the first item satisfy this condition.**

**If you need to find the firstIndex then use**

**Arr.findIndex(a=>a<2);**

**This will give the index of that element.**

**Reduce**

**Arr.reduce((acc,num)=> acc+num,intialaccvalue);**

**Find Multipele Items from array Based on Condition**

**Arr.filter(num => num>2);**

**every() // this will return true if all elemnt satisfy**

**condition**

**arr.every(num=>num>10); //note it returns true and false;**

**Some()//this will =return true if any element satisy fy condition.**

**arr.some(num=>num>10);**

**Executing each elemt of an array**

**forEach()**

**arr.forEach((ele)=>{**

**console.log(ele)**

**});**

**Map() //creates a new array with result of calling array.**

**Let map = arr.map(x=>x+1);**

**I have an array now I want to return all elemt as string**

**Join()**

**Const arr =[1,2,3]**

**Const str = arr.join()**

**This will give “1,2,3”**

**toString()**

**this will also give as a string.**

**Reverse the order of an array**

**Arr.reverse();**

**Sort the item of an array**

**Arr.sort();**

**Find the length of an array**

**Arr.length**

**Fill the array with some static value**

**arr.fill(0,2,4) // fill the arr(array) with 0 from 2nd index to 4th index**

**arr.fill(5,1);//fill with 5 from index 1 to last**

**arr.fill(6)//fill all elemnt to 6;**

**Pass by value and pass by reference**

**Primitive type are pass by value**

**Non-primitive type are pass by reference(object type)**

**Let obj = {a:’a’,b:’b’,c:’c’}**

**Let obj2=obj;**

**Obj.a =’d’;**

**Then obj2.a will also get change due to reference clone**

**So inorder to do that we can use**

**Let obj2 = Object.assign({},obj);//empty array assign obj object;**

**Now in ES6 we can do it like this**

**Let obj2 ={…obj};**

**//But remember it’s a shallow clone means only top level is clone if there is any reference inside the object it will be refer again.**

**Ex-**

**Let obj ={**

**a : ‘a’,**

**b:’b’,**

**c:{**

**deep:’this is pass as reference’**

**}**

**}**

**Let clone = Object.assign({},obj);**

**Obj.c.deep =”hahaha”**

**Then clone.c.deep will also be hahaha**

**So inorder to do Deep Clone we need to follow below steps**

**Let deepCLone = JSON.parse(JSON.stringify(obj));**

**//What is Type Coecion**

**Always use === because it compare value aswell as type**

**Otherwise due to type coercion 1==’1’ give true;**

**Object in Javascript**

Copy one object into another

let obj = {

a:"raj",

b:"moni"

}

Let copy = Object.Create(obj);

//this will create deep copy of obj;

Both reference are different;

How to assign data of one object into another object

Object.assign(target,source);//target and source are the object;

**Copy data of source object into target object;**

**Imp—How to make object non Extensible so that no one can add any property in that object.**

**Let obj={a:’raj’,b:’moni’};**

**Object.isExtensible(obj)**

* **True**

**In Order to make this object non – extensible we can do something like this—**

**1)Object.preventExtensions(obj);**

**//modify existing object property but cant add new property.But can delete existing property**

**2)Object.seal(obj)**

**//modify exiting property but cant add new property and cant delete existing property**

**3)Object.freeze(obj)**

**//cant modify existing property.cant delete cant add new property.**

**Ex-**Object.preventExtensions(person);

Object.isExtensible(person); // return false

person.surname = "Ramirez";

console.log(person.surname); // return undefined

**Please note --\**

### shallow only

All of these methods only work on object properties shallowly, meaning that just work with the direct property references.

let person = {

name: "Agustin", // Prevented, Sealed and Frozen

age: 27, // Prevented, Sealed and Frozen

address: {

// Un-prevented, un-sealed and un-frozen

country: "Argentina", // Un-prevented, un-sealed and un-frozen

city: "Corrientes", // Un-prevented, un-sealed and un-frozen

},

};

Object.preventExtensions(person)

1. *{name: "Agustin", age: 27, address: {…}}*

person.address.pincode="277001"

"277001"

person

1. *{name: "Agustin", age: 27, address: {…}}*
   1. address:
      1. city: "Corrientes"
      2. country: "Argentina"
      3. pincode: "277001"
      4. \_\_proto\_\_: Object
   2. age: 27
   3. name: "Agustin"
   4. \_\_proto\_\_: Object

We can add property inside this object property object

**Important –**

**How to check given object is Array**

**let obj = {a:1,b:'raj',c:[1,2,3]}**

**Array.isArray(obj.c)**

**//True**

**----Object few things left will get back here**

**ES7(2016)**

**Two things added**

**Array includes method**

**Var a = [1,2,3]**

**a.includes(1)//true**

**a.includes(5)//false**

**Power method**

**Var a = (b) => b\*\*2**

**a(2) -> 4**

**a(3) -> 9**

**var aa =(x)=>x\*\*3**

**aa(2) => 8**

**aa(3)=>27**

**ES8(2017 )**

**1)**

**.padstart()**

**.padEnd()**

**“turtle”.padStart(5) => “ tutle” //gives padding of 5 from start**

**Similarly for padEnd(5) //add padding at the end**

**2) Const obj =(a,b,c,d,) =>{**

**}**

**//this is still valid //trailing comma at the argument**

**3)// Object.keys – its was already present in es8**

**Object.values**

**Object.entries**

**Ex- let obj ={**

**Username0:’santa’,**

**Username1:’roni’,**

**Username2:’Mr Smith’**

**}**

**//Since this is not array we cannot use Array.map,array.foreach**

**But inorder iterate**

**Object.keys(obj).foreach((key,index)=>{**

**Console.log(obj[key]);**

**});**

**It become more easy with object.values**

**Object.value(obj).foreach((value,index)=>{**

**Console.log(value);**

**})**

**Object.entires –give aray of key value**

**Ex – let obj = {a:1,b:2,c:3}**

**Object.entries(obj) 🡪 [[a,1],[b,2],[c,3]];**

**Object.entries(obj).map(value=>{**

**Return value[1] +value[0].replace(‘username’,’’);**

**});**

**ES10(2019)**

**1)Flat()**

**Const arr = [1,[2,3,4,5],3,4,[5,6,7,8,9]]**

**arr.flat()**

**//it will flattened the array**

**[1,2,3,4,5,3,4,5,6,7,8,9] //like this**

**Please note it will flatten at 1st level and if you want to flatten at 2,3,4 or so on level then you can do it like this.**

**[1,2,3,4,[7,[8,5,6,7],9,4,6],53,5,3,22,4,55]**

**This is 3rd level nesting and if you want to flatten it then**

**arr.flat(3)//this 3 is for level of flatten**

**NOTE-Flat will also remove empty entries**

**Ex-[1,23,4,5,,,,,,]**

**[1,23,4,5] this empty comma removed**

**2)trimStart()**

**3)trimEnd() //remove blank space**

**4)fromEntries //Just like entries it convert the array into object**

**Const entry = [[‘raj’,30],[‘moni’,26],[‘hero’,22]]**

var entr =[['raj',30],['moni',26],['hero',22]]

Object.fromEntries(entr)

*{raj: 30, moni: 26, hero: 22}*

hero: 22

moni: 26

raj: 30

* 1. \_\_proto\_\_: Object

**5)try{}catch{}**

**Earlier we have to pass exception parameter**

**But now we don’t need to pass**

**Like—try{}catch(e){} //don’t require**

**LOOP IN JAVASCRIPT**

**//for loop**

**//while loop**

**// do while loop**

**//foreach loop**

**Ex- For loop**

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

// Runs 5 times, with values of step 0 through 4.

console.log('Walking east one step');

}

**Ex- do while**

let i = 0;

do {

i += 1;

console.log(i);

} while (

**3)While**

while (condition)

statement

## [break statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#break_statement)

Use the [break](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/break) statement to terminate a loop

## [continue statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration#continue_statement)

while, do-while, or for statement and continues execution of the loop with the next iteration.

//It jumps back to the loop condition and continue to the next statement.

**5)For …of**

This loop works on iterable object like array,string.

Const arr =[1,2,3,4]

For(let i of arr){

Console.log(i)//1,2,3,4

}

**6)For ..in loops**

**//it loop over object properties**

**//it basically not iterate but enumerate over object**

**Bcuz properties in object are enumerable**

const obj = {

'apple':10,'orange':20,'banana':1000,'vegetable':{'carrot':10,'brinjal':20}

}

for(item in obj){

console.log(item)

}

🡪

apple

orange

banana

vegetable

**Note—array is itseld an object so it can be put in for in loop like—**

**For( i in arr){**

**Console.log(i)///this will give 0 1 2 3 4 index of array**

**}**

**ES11(2020)**

In Es 11 we have new type BigInt .

In javascript infact in all language we have limitation of a max number upto which we can do calculation and if we do calculation

Bigger than this no. then it will break

In case of javasript we have

**Number.MAX\_SAFE\_INTEGER**

**9007199254740991**

**If we try to add any number with this number it will give**

**Anamaly result.**

**So in order to solve this problem in javascript we have**

1**n – type any number with n at the last means it’s a bigint**

**And we can perform calculation over this number and it will give correct result**

**typeOf(10000n)**

* **Bigint**

**2) Optional chaining operator**

**In past inorder to get property of an object what we do we check whether that object is not null then we refere its property like below**

**let pokemon= {**

**Pikachu:{**

**species:’mouse’,**

**height:’.4’,**

**weight:’6’**

**}**

**}**

**if(pokemon && pokemon.pikachu && pokemon.pikachu.weight )**

**{**

**//here we checking all null condition inorder to get the object property**

**}**

BUT we can do it like

**let f = pokemon?.pikachu?.weight**

this will give undefined if doestnot exist otherwise the property value.

**3)Optional chaining operator**

let power = **pokemon?.pikachu?.weight || ‘No weight’**

now in above condition we want if weight is not present or its false then it should ‘No weight’

Now,What if we want that if weight is zero or null then is sould return that insead of ‘No weight’

This is possible by ?? operator

SO,

?? is use instead of || or operator

**let pokemon= { Pikachu:{ species:'mouse', height:'.4', weight:'' } }**

**let weight = pokemon.Pikachu.weight || 'no weight'**

**🡪weight**

**"no weight"**

**BUT**

**let pokemon= { Pikachu:{ species:'mouse', height:'.4', weight:'' } } let weight = pokemon.Pikachu.weight ?? 'no weight'**

**🡪weight**

**""**

**?? doesn’t check if value is falsey its check whether value is null or undefined.**

**What is Module in Javascript?**

Initially there was

Inline script🡪Script tags🡪IIFE🡪broweserify🡪ES6(require js)

Initially we would like to create inline script tag in HTML

1)

<script type=”text/javascript”>

here we write our javascript but problem with it was if our html file is very long then we need to copy this script in other file also and also there is always problem of global variables.

This variables create problems.

</script>

2)Script tags

<script type=”text/javascript” src”./1.js”></script>

Now problems with it is dependency resolution -- we need to maintain its order if one script have function declaration and other script is using it then this file need to be in some specific order.

3)IIFE(Immediately invoked Function executions)

var myApp={}

(

function(){

myApp.add=function(a,b){

return a+b

}

})();

this will solve the problem of global variable problems as each script has its own scope variable which will pe present at function scope.

Now problems of script order i.e lack of dependency resolution still exist—

**Now comes – Common js + browserify**

In Js file we write in this way

module.exports = function add(a,b){

return a +b;

}

Now in other script file if we need to use this add function then we can do it like this

var add = require(“./add”)

How it works it will move all the function into one file i.e common js when need to execute.

**Now with ES6 + Webpack2**

we have MODULE – It is a building block .Each module do one independent work.

export const add =(a,b) =>a+b

import {add} from ‘./add’

export default function add(a,b){

return a+b

}

import add from ‘./add’

Thus we put import at the top of each file so each file know what they want .

**Since still each browser don’t support module bundle since it still a latest things thus comes into picture webpack which bundle each module javascript file.**

**Webpack is the module bundler.**

**GIT**