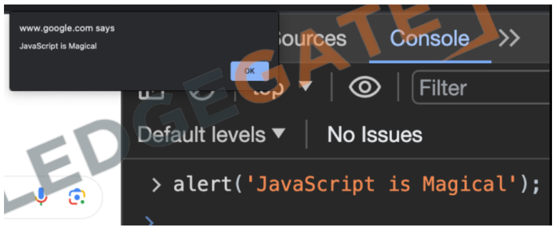
# JavaScript Journey

# Chapter:1

https://github.com/Prakash9596/JavaScript_Journey
https://prakash9596.github.io/JavaScript_Journey/

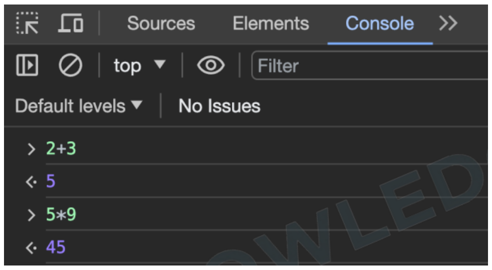
# 1.1 JS code can be executed in the console. (inspect--> console)

--> Show alert: alert('Welcome to the world of JS')



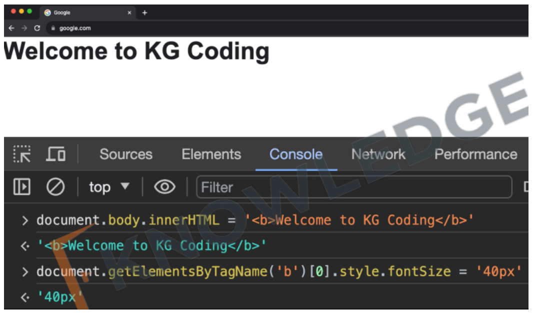
```
> alert('JavaScript is Magical');
```

--> Use as a calculator: >2+3 (result=5)



```
> 2+3
< 5
> 5*9
< 45
```

--> DOM Manipulation : Document object model
(whole webpage is document, we can change it by DOM Manipulation)



Welcome to KG Coding

```
> document.body.innerHTML = '<b>Welcome to KG Coding</b>'
< '<b>Welcome to KG Coding</b>'
> document.getElementsByTagName('b')[0].style.fontSize = '40px'
< '40px'
```

--> Change HTML: document.body.innerHTML ='<b>Welcome to the world of JS</b>'

--> Change CSS: document.getElementsByTagName('b')[0].style.fontsize = '40px'

--> Perform JS actions: document.getElementsByClassName('class_name')[1].click()

--> It can be used to create chrome extensions.

--> New forms of JS: Coffee Script & TypeScript (which will later be transpiled into JS only)

# Chapter:2

# 2.2 Arithmetic Operators (+, -, *, /, %)

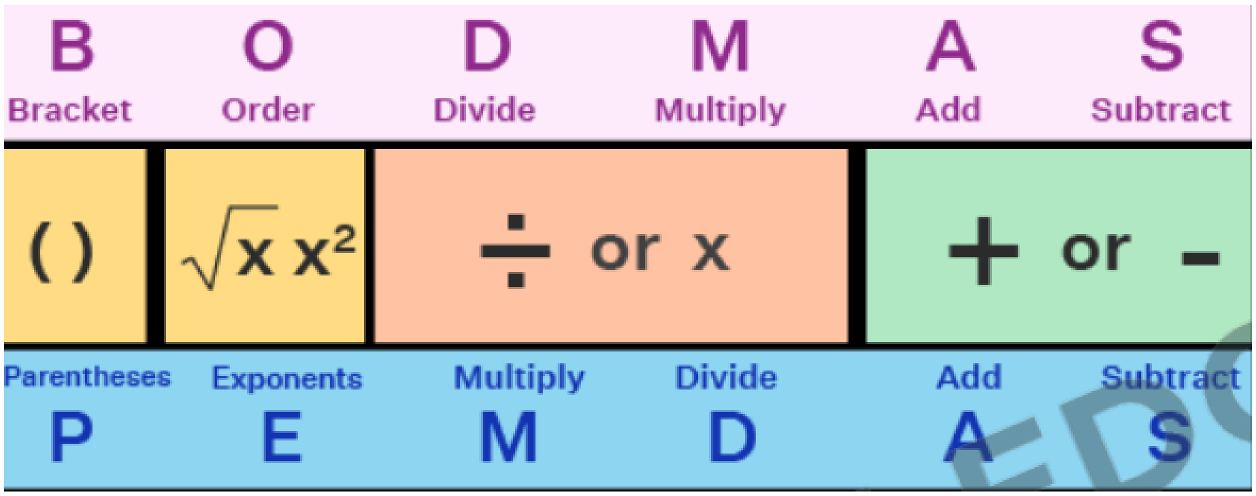
--> Order of operations (Bracket, OF, Divide/Multiply, Add/Subtract) [Left to Right]

--> Math.round(53.3) =53, Math.round(53.5) =54, Math.round(53.7) =54

--> Math.floor(53.3) =53, Math.floor(53.8) =53

--> Math.cell(53.3) =54, Math.cell(53.8) =54

--> Math.random() [it will give the value between 0-1]



# 2.3 Strings: Anything from single character to paragraph.

--> Can be defined using: " ", ' ', ` `(backticks)

[Recommended: ' ', ` `] [" " --> can be used when single quote is there in string]

-->`this is the end

  this is pakka the end` (backticks preserves the indentation)

  '₹' + (5+3) = ₹8 (1st way),

  `₹ ${5+3}` = ₹ 8 (2nd way using backticks)

--> String Concatenation: ['Hello ' + 'World' = 'Hello World'], ['Hello ' + 5 = 'Hello 5'],

['5' + '5' = 55], [5 + 5 = 10]

# 2.4 typeof operator:

[typeof 5 ='number'], [typeof 5.3 ='number'], [typeof 'hello' ='string']

# Chapter:3

# 3.5 document.getElementById('id_name');

[It will return the element which is having same ID] [old method]

--> document.getElementsByClassName('class_name');

[It will return all the elements which is having same class] [old method]

--> document.querySelector('h1')

[It will return the element which is having h1 tag] [new method]

--> document.querySelector('#id_name')

[It will return the element which is having same id] [new method]

--> document.querySelector('.class_name')

[It will return the element which is having same class] [new method]

--> document.querySelectorAll('.class_name')

[It will return all the elements which is having same class] [new method]

# 3.6 Script Tag: It can be placed --> [at the end of the body tag of HTML] --> Why at the end of the body tag? becoz page should load first then there is use of it's related action] or [in head tag --> not recommended]

```
--> way 1:    <script>
-->             alert('Hello Prakash Jha');
-->             console.log('Hello console');
-->             console.warn('I am warning');
-->             console.error('I am error');
-->             console.clear();
-->           </script>
--> way 2:    <script src=" "></script>
```

--> We can use javascript from: [1. console me jake] [2. script tag ke andar se] [3. html tags ke andar se bhi]

--> ex:3 `<body> <button id="click-me" onclick="alert('I am clicked')"> Click Me </button> </body>`

--> document.querySelector('#click-me').click()

[Clicked the button from console without touching it.]

# 3.7 Comments:

--> HTML <!-- --> , CSS // or /* */ , *JavaScript // or /**/*

# Chapter:4

# 4.8 Variables: Used to store data

--> [var a = 'Hello World'] [var b =100] [var c = true]

--> Syntax rules: [can't use keywords or reserved words as variable_name] [can't start with a number] [no special character other than $ and _]

```
-->   let money =  1; [shorthand operators below]
      money += 5; // money = money + 5;
      money -= 2; // money = money - 2;
      money *= 3; // money = money * 3;
      money /= 4; // money = money / 4;
      money++;    // money = money + 2;
```

# 4.9 HTML:

```
<button class="bag-button" onclick=" cartQuantity++;
 document.querySelector('#cart-summary').innerText = `Your cart has
${cartQuantity}items`;  ">Add to bag</button>
<h1 id ="cart-summary"></h1>
```

JS:

```
let cartQuantity =0;
document.querySelector('#cart-summary').innerText = `Your cart has ${cartQuantity}
items`;
```

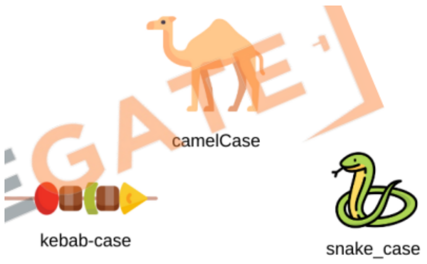
# 4.10 Naming Convention:

[camelCase: helloPrakashJha, myNameIsKhan]

[sname_case: hello_prakash_jha, my_name_is_khan]

[kebab-case: hello-prakash-jha, my-name-is-khan]

[can't use this JavaScript: no special character other than $ and _]



# 4.11 ways to create variable:

[var a=5;] --> global variable

[let a=5;] --> local variable

[const a=5;] --> [the value will be fixed and can't be changed in future, ex: pi=3.14]

# 4.(12) Scope of a variable:

--> Any variable created inside {} will remain inside {}

--> Variable can be redefined inside {}

--> Var does not follow scope

[Var will treat a redefined variable as a previously defined variable only.]

```
[let a=6; {
 let a=7;
}]
```
--> let se bne hue variable dubara se define kiye ja sakte hain ..dono alag alag treat honge.

```
[var a=6;
{
 var a=7;
}]
```
--> var se bne hue variable bhi dubara se define kiye ja skte hain ..pr dono same hi treat honge kahi se v access kre.

--> Global Scope: Accessible everywhere in the code.

--> Block Scope: Limited to a block, mainly with let and const.

# 4.13 Eval method:

console.log('45 + 45'); [output: 45 + 45]

console.log(eval('45 + 45')); [output: 90]

# 4.14 <input type="text" class="display" readonly>

--> let current_display ='';

--> document.querySelector('.display').value=current_display;

# Chapter:5

# 5.15 if-else & Boolean

Equality (== Checks value equality), (=== Checks value and type equality)

Inequality (!= Checks value inequality), (!== Checks value and type inequality)

Relational (> Greater than), (< Less than), (>= Greater than or equal to), (<= Less than or equal to)

Note: Order of comparison operators is less than arithmetic operators.

[--> console.log(5 + 5 == 4 + 6); [true]]

--> console.log(5 == 5.0); [true], --> console.log(5 === '5.0'); [true]

--> console.log(5 == 5); [true], --> console.log(5 === '5'); [true]

--> console.log(5 == '5.0'); [true], --> console.log(5 === '5.0'); [false]

--> console.log(5 == '5'); [true], --> console.log(5 === '5'); [false]

```
Syntax: if(age >= 18){
        console.log("You can drive");}
        else if (age >=10){
         console.log("Use bicycle");}
        else {
         console.log("Sleep Baby Sleep");
        }
```

# 5.16 Truthy and Falsy Values:

--> Falsy Values: 0, null, undefined, false, NaN (Not a number), ""(empty string)

--> Truthy Values: All values not listed as falsy.

--> Used in conditional statements like if.

# 5.17 If alternates:

--> Ternary Operator: condition ? trueValue : falseValue

```
   let age=12;
   let result;
   if(age > 18){
       result = 'Adult';
   }else{
       result = 'Kid';
   }
   console.log(result);

  [let age = 12;
   let result = age > 18 ? 'Adult' : 'Kid';]
```

--> Guard Operator: value || defaultValue [Use when a fallback value is needed]

```
   let age = 0;
   let finalAge = age || 15; [kuchh bhi falsy value aa jaye to by-default 15]
   console.log(age); --> 0
   console.log(finalAge); --> 15
```

--> Default Operator: value ?? fallbackValue

   [Use when you want to consider only null and undefined as falsy]

```
   let age = 0;
   let finalAge = age ?? 15; [agar null ya undefined value aa jaye to hi by-default 15]
   console.log(age); --> 0
   console.log(finalAge); --> 0
```

Chapter:6

6.18 Functions: Blocks of reusable code.

--> DRY Principle: "Don't Repeat Yourself" it Encourages code reusability.

--> Naming Rules: Same as variable names: camelCase

--> EX: alert(); Math.round(); console.log();

--> Functions Syntax:

```
function sum (x,y){
    let s = x + y;
    return s;
}
result = sum(4,5); [Function invoking/ Function Calling]
```

--> Return statement : Can Return (Value, variable, calculation)

--> Parameters: Input values that a function takes.

--> Naming Convention: Same as variable names

--> Function Parameter(jo function ko define krte waqt use krte ho) [x,y]

--> Function Argument (Jo function ko call krte waqt use krte ho) [4,5]

--> Default Value: Can set a default value for a parameter.

```
function greeting(name = 'Pyaare'){
    console.log(`Namaste ${name} Uncle`);
}
greeting('titu'); [Namaste titu Uncle]
greeting(); [Namaste Pyaare Uncle]
```

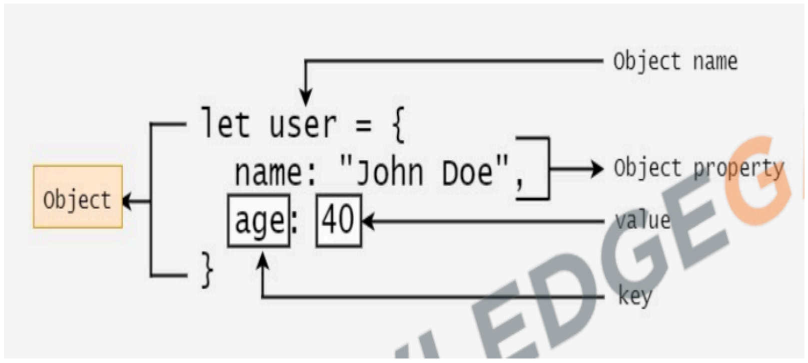
# Chapter:7

# 7.19 Objects: Groups multiple values together in key-value pairs. Organizes related data under a single name.

--> Use . operator to access values.

```
let user = {
    name: 'Milky Singh' ,
    age: 32,
    'skin-color': 'White'
};
let propertyName = age;
```

--> Syntax: [Object name: user] [Object Properties: name, age]
[key: name, age] [value: Milky Singh, 32] [Object: Pura ka pura]



```
let user = {
  name: "John Doe",
  age: 40
}
```

DOT Notation

--> console.log(user); [Reading the whole object]
--> console.log(user.name); [Reading the property of object]
--> console.log(user.age); [Reading the property of object]
--> console.log(user.skin-color); [Not allowed as hyphen can't be used]
--> console.log(user.propertyName); [Not allowed as this is not defined within object]
--> user.name = 'Ring EduCrafter'; [Writing/Changing the property of object]
--> console.log(user); [Reading the whole object]
--> delete user.age; [Deletion of property from the object]
--> console.log(user); [won't return age property now]

Bracket Notation

--> console.log(user); [Reading the whole object]

--> console.log(user['name']); [Reading the property of object]

--> console.log(user['age']); [Reading the property of object]

--> console.log(user['skin-color']); [allowed, output: White]

--> console.log(user['propertyName']); [allowed, output: 32]

--> user['name'] = 'Ring Educrafter'; [[Writing/Changing the property of object]]

--> console.log(user); [Reading the whole object]

--> delete user['age']; [Deletion of property from the object]

--> console.log(user); [won't return age property now]

--> Objects can contain Primitives like numbers and strings.

--> Objects can contain other objects and are called Nested Objects.

--> Functions inside an object are called methods.

```
--> let product = {
    company: 'Mango',                         --> string
    itemName: 'Cotton striped t-shirt',       --> string
    price: 861,                               --> numbers
    rating:{                                  --> object (nested object)
        stars: 4.5,
        noOfReviews: 87
    },
    displayPrice: function(){                 --> function
        return `Price of the product is ${this.price}`;
    }
}; [this mtlb is object ke andar jo v price defined the wo]
```

--> Function call : product.displayPrice();

# 7.20 Autoboxing:

--> Automatic conversion of primitives to objects. Allows properties and methods to be used on primitives.

--> Example: Strings have properties and methods like length, toUpperCase, replace etc.

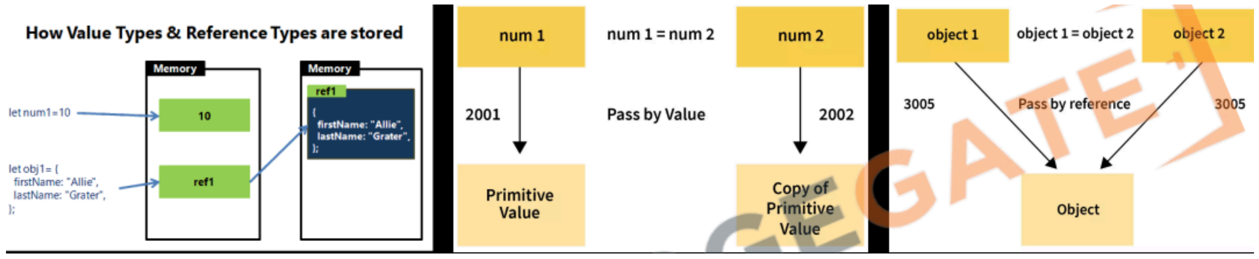
```
    console.log('Hello, My Name is Prakash Jha'.length());
    console.log('Hello, My Name is Prakash Jha'.toUpperCase());
```

# 7.21 Object References:

--> Objects work based on references, not actual data.

--> Copying an object copies the reference, not the actual object.

--> Changes to one reference affects all copies.



```
--> let a=5;
    let b=a;
    console.log(`a=${a}, b=${b}`); [a=5, b=5]
    a = 8;
    console.log(`a=${a}, b=${b}`); [a=8, b=5]
```

```
--> let x = {num: 5};
    let y = x;
    console.log(`x=${x.num}, y=${y.num}`); [a=5, b=5]
    x.num =8;
    console.log(`x=${x.num}, y=${y.num}`); [a=8, b=8]
```

--> When comparing with ==/===, you're comparing references, not content.

```
    let p = {pop: 'hello'};
    let q = {pop: 'hello'};
    console.log(p == q);
```

[false, becoz dono object alag alag memory me rakhe hain/point kar rahe hain]

# 7.22 Object Shortcuts:

1. **Destructuring**:

```
   let product = {
    company: 'Mango',
    itemName: 'Cotton striped t-shirt',
    price: 861,
    };
```

--> **Destructuring**:

way 1: let company = product.company

way 2: let {company} = product;

```
let {company} = product;
console.log(company);              [Output: Mango]
let {company, price} = product;
console.log(company);              [Output: Mango]
console.log(price);                [Output: 861]
```

2. **Property shorthand**:

```
    let price = 861;
Way 1: let product = {
        company: 'Mango',
        itemName: 'Cotton striped t-shirt',
        price: price,    --> assuming ki ye price kahin aur se aa raha hai
    };
  Way 2: let product = {
        company: 'Mango',
        itemName: 'Cotton striped t-shirt',
        price              --> assuming ki ye price kahi aur se aa raha hai
    };
```

3. **Method shorthand:**

```
  way 1: let product = {
        company: 'Mango',
        itemName: 'Cotton striped t-shirt',
        price: 861,
        displayPrice: function(){
        return `Price of the product is ${this.price}`;
        }
    };
```

way 2: let product = {
```
          company: 'Mango',
          itemName: 'Cotton striped t-shirt',
          price: 861,
          displayPrice(){
          return `Price of the product is ${this.price}`;
          }
      };
```

# 7.23 Create function isIdenticalProduct to compare two product objects.

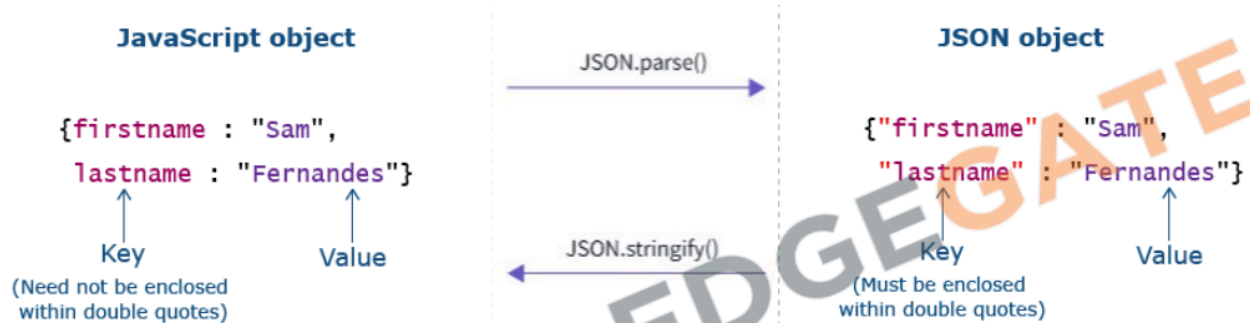
```
   function isIdenticalProduct(product1, product2){
      if(typeof product1 !== 'object' || typeof product1 !== 'object'){
         console.warn('Parameter passed was not an object');
         return false;                    --> checking inputs
      } else if(product1 == product2){
         return true; } --> checking ki same object to nahi hai
      else if ( product1.name == product2.name && product1.size == product2.size
          && product1.fit == product2.fit
          && product1['delivery-time'] == product2['delivery-time']){
          return true; } else{
         return false;
      }
   }

   let productP ={
      name: 'Jeans',
      size: 'S',
      fit: 'slim-fit',
      'delivery-time': 'same day delivery',
   } let productS ={
      name: 'Jeans',
      size: 'S',
      fit: 'slim-fit',
      'delivery-time': 'same day delivery',
   }
--> console.log(isIdenticalProduct(productP, productS)); [Output: true]
```

# Chapter:8

# 8.24 JSON:

1. JSON (JavaScript Object Notation): Not the same as JS object, but similar.
2. Common in network calls and data storage.
3. JSON.stringify() [used to convert object into string]
and JSON.parse() [string ko object banane ke liye]
4. Strings are easy to transport over the network.
5. JSON requires double quotes "" in Keys as well.
(Object me usually bs value hi ""/'' me hota hai)
6. JSON is data format, JS object is a data structure.

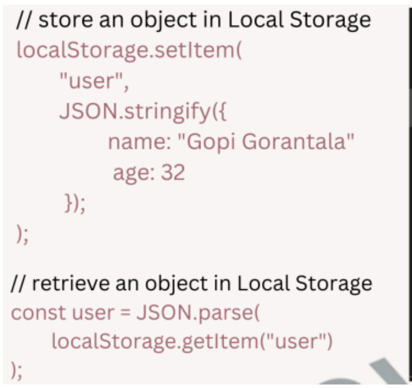


```
let product = {
   name: 'TShirt',
   price: 789,
   rating: {
     stars: 4.5,
     noOfReviews: 453,
      },
   };
   console.log(typeof product);                [object]
   console.log(product);                       [whole object print hoga]
   let str = JSON.stringify(product);          [object ko string bna diye]
   console.log(typeof str);                    [string]
   console.log(str);                           [pura string ke form me print hoga ]

   let newProduct = JSON.parse(str);           [string ko object bna rhe hain]
   console.log(typeof newProduct);             [object]
   console.log(newProduct);                    [whole object print hoga]
```

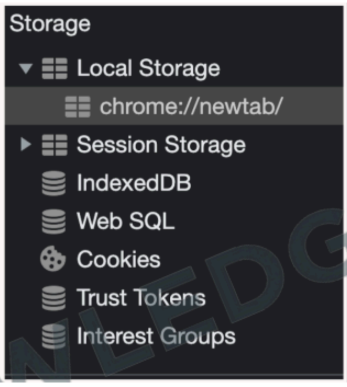
# 8.25 Local Storage:

1. Persistent data storage in the browser.
2. setItem: Stores data as key-value pairs.
3. Only strings can be stored.
4. getItem: Retrieves data based on key.
5. Other Methods: localStorage.clear(), removeItem().
6. Do not store sensitive information. Viewable in the storage console.



```
// store an object in Local Storage
localStorage.setItem(
      "user",
      JSON.stringify({
            name: "Gopi Gorantala"
             age: 32
       });
);

// retrieve an object in Local Storage
const user = JSON.parse(
      localStorage.getItem("user")
);
```



[ye same key value browser storage me store ho jayega:
application >> local storage >> url >> key-value pair]
--> localStorage.setItem('Name', 'Prakash Jha');
--> console.log(localStorage.getItem('Name')); [to get the value using key]

```
let product = {
  name: 'TShirt',
  price: 789,
  rating: {
    stars: 4.5,
    noOfReviews: 453,
  },
};
localStorage.setItem('product', JSON.stringify(product));
```
[object ko string me convert kar ke save kr rhe]
```
let product2 = JSON.parse(localStorage.getItem('product'));
```
[string ko object me convert kr rhe]
```
console.log(product2);
```

--> localStorage.removeItem('Name'); [will delete the key-value pair where key = 'Name']
--> localStorage.clear();   [will delete all the key-value pair from browser storage]

# 8.26 Date:

1. new Date() Creates a new Date object with the current date and time.
2. Key Methods:

```
 let myDate = new Date();
  console.log(myDate.getTime());             [Milliseconds since Epoch (1 jan 1970/1971)]
  console.log(myDate.getFullYear());         [4-digit year]
  console.log(myDate.getDay());              [Day of the week]
  console.log(myDate.getHours());            [Current hour]
  console.log(myDate.getMinutes());          [Current minute]
  console.log(myDate.getMilliseconds());    [Current ms]
```

3. Crucial for timestamps, scheduling, etc.

# 8.27 DOM Properties & Methods:

DOM and Element Properties

1. location 2. title 3. href 4. domain 5. innerHTML 6. innerText 7. classList

--> document.location;
--> document.title;   document.title = 'newTitle';   [title change kr diye webpage ka]
--> document.domain;
--> document.body.innerHTML ='<b>Welcome to the world of JS</b>';
--> document.querySelector('#cart-summary').innerText = `Your cart has ${cartQuantity} items`;
--> document.querySelector('.class_name_of_any_element').classList;
[will return kon-konsa class define hua h ush element ke liye]
-->document.querySelector('.class_name_of_any_element').classList.add('newly-added-class'); [to add any class]
--> document.querySelector('.class_name_of_any_element').classList.remove('class-name');
[to remove any class]

--> document.querySelector('.class_name')
[It will return the element which is having same class] [new method]
--> document.querySelectorAll('.class_name')
[It will return all the elements which is having same class] [new method]

--> document.querySelector('#click-me').click()

--> document.querySelector('.display').value=current_display;   [input type="text" ho]

--> document.querySelector('h1')

[It will return the element which is having h1 tag] [new method]

--> document.querySelector('#id_name')

[It will return the element which is having same id] [new method]

DOM and Element Methods

1. getElementById() 2. querySelector() 3. classList: add(), remove()

4. createElement() 5. appendChild() 6. removeChild() 7. replaceChild()

--> let button = document.createElement('Button');

[jaisa v element create krna chahte hain, yahan button kr rhe]

--> console input: [button], console output: [<button></button>]

--> <div class="my-div">This is the div</div>   [html code]

--> document.querySelector('.my-div').appendChild(button);   [JS Code]

button.innerText = 'Click Me';

[appendChild mtlb uske andar koi element create krna]

--> document.querySelector('.my-div').removeChild(button);

[removeChild mtlb child element delete karna]

--> document.getElementById('id_name');

[It will return the element which is having same ID] [old method]

--> document.getElementsByClassName('class_name');

[It will return all the elements which is having same class] [old method]

Conceptual:

```
.js-odd { background-color: chocolate; }
.js-even { background-color: lawngreen; }
if (noOfTimesClicked % 2 === 0) {
    button.classList.remove('js-odd');
    button.classList.add('js-even');
   } else {
    button.classList.remove('js-even');
    button.classList.add('js-odd');}
```

```
let noOfTimesClicked = localStorage.getItem('noOfTimesClicked') || 0;
    function buttonPressed() {
      noOfTimesClicked++;
      localStorage.setItem('noOfTimesClicked', noOfTimesClicked);
      updateButton();
    }
```

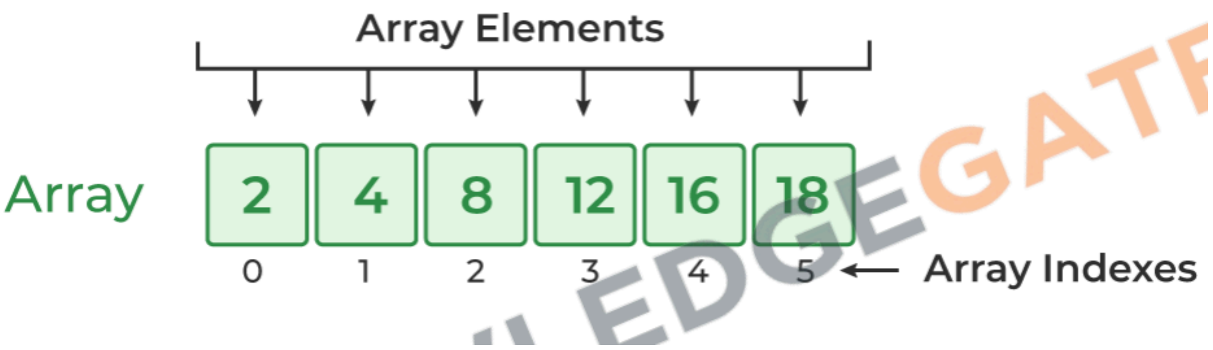
Conceptual:

```
let todoList = [
  {
    item: 'Buy Milk',
    dueDate: '2024-10-15'
  },
  {
    item: 'Go to College',
    dueDate: '2024-10-15'
  }
];

  let inputElement = document.querySelector('#todo-input');
  //target kiye user input wale element ko
  let dateElement = document.querySelector('#todo-date');
  //target kiye user selected date wale element ko
  let todoItem = inputElement.value;                          //text value store kr diye
  let todoDate = dateElement.value;                           //date value store kr diye
  todoList.push({item: todoItem, dueDate: todoDate});
  //array me text aur date object ke form me bhej diye
  inputElement.value = '';
//null kr diye taki phir se enter krne ke liye aaye
  dateElement.value = '';
```

# Chapter:9

# 9.28 Arrays & Loops:



1. An Array is just a list of values.
2. Index: Starts with 0.
3. Arrays are used for storing multiple values in a single variable.
4. Arrays can hold any value, including arrays.
5. typeof operator on Array Returns Object.
6. Arrays also use references like objects.

```
let arr= [9,8,5,7,6];
let arr2= arr;
arr2[0]=99;
console.log(arr);  [99,8,5,7,6]
```

   [mtlb dono same hi array ko point kr rha h kisi me change kr do]

7. De-structuring also works for Arrays.

```
let [a,b,c,d,e] = arr;
console.log(`a= ${a}, b= ${b}, c= ${c}, d= ${d}, e= ${e}`);
```

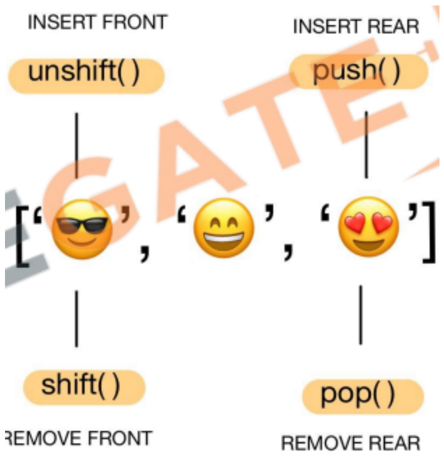
--> Syntax:

```
let myArray = [1, 'Prakash Jha', null, true, {likes: '! Million'}];
console.log(myArray);
console.log(myArray[0]);
console.log(typeof myArray);
```

[object, becoz array is also special type of object which contains list of values inside square bracket]

```
--> console.log(Array.isArray(myArray));     [true, checks if a variable is an array]
    console.log(myArray.length);              [5, returns the size of the array]
```

--> **Common Methods:**



console.log(myArray);
myArray.push('hello'); [adds element at the end]
myArray.pop(); [removes last element]
let x = myArray.pop();
[last value array se remove bhi kr dega aur whi same return bhi kr dega]

myArray.shift(); [removes first element]
myArray.unshift('kya ho rha h'); [adds element at the start]

myArray.splice(1,2); [1st index se delete karna start karo aur 2 element delete kr do]
let k= myArray.slice(2,5);
[ye 2nd_index(inclusive) se 5th_index(exclusive) tk ka element return krega]
console.log(k);
console.log(myArray.toString()); [to convert array into string]

console.log(myArray);
myArray.sort();
[sorting kr dega, string ko alphabetically aur numbers ko ascending to descending order]
console.log(myArray);

console.log(myArray.valueOf()); [ye same hi array return kr dega: useless]

let arr = ['Happiness', 'is', 'good', 'for', 'health']
console.log(arr.join(' '));
[Happiness is good for health, jis bhi tarah se join karna chaho]

# 9.29 Loops:

[Functions: Reusable blocks of code.] [Loops: Repeated execution of code.]

1. Code that runs multiple times based on a condition.
2. Loops also alter the flow of execution, similar to functions.
3. Loops automate repetitive tasks.
4. Types of Loops: for, while, do-while.
5. Iterations: Number of times the loop runs.

1. **While loop**: (Used for non-standard conditions. Jb nhi pta ho loop kitne baar chalana hai)
Remember: Always include an updation to avoid infinite loops.

```
EX:1  let num =1;                 //initialization
      while (num <=10){        //condition
        console.log(num);
        num++;                      //updation
      }
EX:2  let x = 0;              //initialization
      while (x < 1){          //condition
        console.log(x);
        x = Math.random()*2;  //updation
      }
      console.log('final value of x:' + x);
```

2. **Do While Loop**:   (Guaranteed to run at least one iteration)

```
EX:1   let num=0;        //initialization
       do {
        console.log(num);
        num++;              //updation
       }while (num >5);  //condition  (first iteration is unconditional)
```

3. For Loop: (Standard loop for running code multiple times)

```
EX:1  let nums = [4,56,8,3,-87,9,78,9,3,5,7];
      for (let i = 0; i < nums.length; i++) {  //initialization //condition //updation
        console.log(nums[i]);
       }
```

# 9.30 Break & Continue:

1. Break lets you stop a loop early, or break out of a loop
2. Continue is used to skip one iteration or the current iteration
3. In while loop remember to do the increment manually before using continue

Break:

```
let arr = [1,3,5,6,7,8,3,4];
for (let i = 0; i < arr.length; i++) {
    if (arr[i] === 8) {
      console.log(`Number found at index ${i}`);
      break;
    }
  }
```

Continue:

```
for (let i = 1; i <= 25; i++) {  //To print odd no. from 1 to 25
    if (i % 2 == 0) {
      continue;
    }
    console.log(i);
  }
```

# Chapter:10

Advance Functions:

# 10.31 Anonymous Functions As Values: //it's like a datatype only (int,string,object, array, function)

1. Functions in JavaScript are first-class citizens; they can be assigned to variables.
2. Functions defined without a name, often assigned to a variable.
3. Anonymous functions can be properties in objects.
4. This Can be passed as arguments to other functions.
5. Invoked using () after the function name or variable.
6. console.log(myFunction); and typeof myFunction will both indicate it's a function.

```
EX:1   //Anonymous function jisko sum variable me assign kiya ja rha h
let sum = function(num1, num2){
          return num1 + num2;
          };
let newSum = sum;                    //function ko as a variable use kr le rhe
console.log(newSum(4,5));
```

```
EX:2  //This Can be passed as arguments to other functions.
      // 3 no. pass kr rhe aur ek aisa function jo 2 no. ka sum kr paye

let sumThreeNumbers = function (num1, num2, num3, sum){
            let sum1 = sum(num1, num2);             // 1st call (2 no. ko sum krne wala function)
            return  sum(sum1, num3);                // 2nd call (2 no. ko sum krne wala function)
 }
 console.log(sumThreeNumbers(4,5,6,sum));

<!-- let sumThreeNumbers = function (num1, num2, num3, sumTwoNumbers){
            let sum1 = sumTwoNumbers(num1, num2);
            return  sumTwoNumbers(sum1, num3);
 }   console.log(sumThreeNumbers(4,5,6,sum));
//ye bhi theek hai kyunki sum/sumTwoNumbers to bas ek variable ke jaisa h

--> console.log(sum);                   [will return whole body of the function]
--> console.log(typeof sum);            [function]
```

# 10.32 Arrow Functions:

1. A concise way to write anonymous functions.

Form 0 : **Object create krte waqt wala form**

```
let product = {
   company: 'Mango',
   greeting: function(){
      return `Hello Sir`;
   }
};
product.greeting();
```

Form 0': Object create krte waqt wala another form

```
let product = {
   company: 'Mango',
   greeting(){
      return `Price of the product is ${this.price}`;
      }
};
product.greeting();
```

Form 1 : Normal Function

```
function sum (num1, num2){
      let s = num1 + num2;
      return s;
   }
console.log(sum(2,3));
```

Form 2: Anonymous Function

```
let sum = function(num1, num2){
         let s = num1 + num2;
         return s;
         };
console.log(sum(2,3));
```

Form 3:

1. let sum = (num1, num2) =>{
```
          let s = num1 + num2;
          return s;
          };
console.log(sum(2,3));
```

2. For Single Argument: Round brackets optional.

Form 4:

```
let square = num =>{
          let s = num *num;
          return s;
          };
console.log(square(5));
```

3. For Single Line: Curly brackets and return optional.

Form 5:

```
let square = num => num*num;
console.log(square(5));
```

4. Often used when passing functions as arguments.

# 10.33 setTimeout & setInterval

1. Functions for executing code asynchronously after a delay.
2. setTimeout runs once; setInterval runs repeatedly
3. setTimeout:

• Syntax: setTimeout(function, time)
• Cancel: clearTimeout(timerID)

```
    let alarm = function(){                                     //Anonymous Functions
    console.log('Subah ho gayi, uth jao');
     };

let timerId = setTimeout(alarm, 5000);                      //5 sec baad execute hoga
console.log(timerId);
//setTimeout(alarm(), 5000);                                //aisa nhi krna h bs argument dena h
function ko call setTimeout khud krega
clearTimeout(timerId);                                     // to cancel the alarm before execution
```

4. setInterval:

• Syntax: setInterval(function, time)
• Cancel: clearInterval(intervalID)

```
let alarm = function(){                                     //Anonymous Functions
        console.log('Subah ho gyi, uth jao');
        };
let intervalId = setInterval(alarm, 2000);                  // to repeat after every 2 sec
//clearInterval(intervalId);                                // to cancel it
```

What if you want to repeat this after every 2 sec till 10 sec   [total 5 repetition]

```
let hello = function(){
   clearInterval(intervalId);
}
setTimeout(hello, 10000);
// ye 10 sec baad hello function ko execute krke rok dega alarm ko
```

# 10.34 Event Listener

1. What Is an Event: Occurrences like clicks, mouse movement, keyboard input (e.g., birthday, functions).
2. Using querySelector to attach listeners.
3. Multiple Listeners: You can add more than one.
4. removeEventListener('event', functionVariable);

Previously used method: **In HTML**

```
<button id="my-button" onclick="console.log(`I'm clicked`)">Click Me</button>
```

Now: From JS

```
<button id="my-button" >Click Me</button> (HTML)

let btnTarget=document.querySelector('#my-button');
let status = function(){                                          //Anonymous Functions
          console.log(`I'm clicked`);
          };
let showDate = function(){
          console.log(new Date());
}
btnTarget.addEventListener('click', status);
//yahan status() mt use kr lena, nhi to bs ek baar call hoga
btnTarget.addEventListener('click', showDate);                 // adding multiple events
btnTarget.removeEventListener('click', status);
// on-click se status event remove kr rhe
```

# 10.35 For Each Loop

1. A method for array iteration, often preferred for readability.
2. Parameters: One for item, optional second for index.
3. Using return is similar to continuing in traditional loops.
4. Not straightforward to break out of a forEach loop.
5. USE THIS: When you need to perform an action on each array element and don't need to break early.

Previous method: 1

```
let arr =[2,5,'hello', 'prakash', 67];
 for(let i=0; i<arr.length; i++){
  console.log(arr[i]);
 }
```

New method: 2

```
 arr.forEach(function(element){
  console.log(element);
 });
```

Arrow function: 3

```
 arr.forEach(element => console.log(element));
```

```
//search function bnao using old method
let arr =[1,2,3,4,5,6,7,8,9];
function search(arr, num){
    for (let i=0; i< arr.length; i++){
        if(arr[i] == num){
            return i;
        }
    }
    console.log('Element not found');
}
console.log(search(arr, 5));
```

```
//Do the same with For each loop //won't work
//Using return is similar to continue in traditional loops.
let arr =[1,2,3,4,5,6,7,8,9];
arr.forEach(num => {
    if(num ===5){
        return;
    }
    console.log(`visited:${num}`);
});                                                    —> [output: 1,2,3,4,6,7,8,9]
```

# 10.36 Array Methods

1. **Filter Method:**

- Syntax: array.filter((value, index) => return true/false)
- Use: Filters elements based on condition.

```
let arr = [1,2,3,4,5,6,7,8,9];
let odds = arr.filter((num, index) => {
    if (num % 2 === 1) {
        return true;
    } else {
        return false;
        }
});
console.log(odds);  [1,3,5,7,9]

let newOdds = arr.filter( (num, index) => num % 2 === 1 );
console.log(newOdds); [1,3,5,7,9]
```

2. **Map Method:**

- Syntax: array.map((value) => return newValue)
- Use: Transforms each element.

```
let arr = [1,2,3,4,5,6,7,8,9];
let squares = arr.map(num => num*num);
console.log(squares);
```

# Chapter:11

# 11.37 Prototypes in JS

--> Kisi bhi javaScript object ke andar kuchh (properties and methods) ho skte hain.
Aur har JS objects me by-default ek special property hota hai called prototype.
Ye prototype khud me hi ek object hota hai ya (reference to an object) jiske andar
by-default kuchh (properties and methods) hote hain.

--> We can set prototype using _ _ proto _ _

```
const employee = {
    calcTax(){
        console.log ('Tax rate is 10%');
    }
}

const karanArjun = {
    salary: 50,000,
};

karanArjun.__proto__ = employee;
karanArjun.calcTax();
//ab karanArjun object ke andar employee object ka sara (properties and methods) aa gya
//calling calTax() using karanArjun object [o/p: Tax rate is 10%]

const karanArjun2 = {
    salary: 60,000,
    calcTax(){
        console.log ('Tax rate is 15%');
    }
};
karanArjun2.__proto__ = employee;
karanArjun2.calcTax();                          [o/p: Tax rate is 15%]
```

Note: *If the object & prototype have same method, object's method will be used.

# 11.38 Classes in JS

--> Class is a program-code template for creating objects.
  Those objects will have some (properties and methods) inside it.

--> Class is like a blueprint/template for objects.
  Agar similar tarah ka dher sara object create karna hai to sab ke liye ek template bna lenge ki wo kaisa hoga, Whi template class hai.

```
class toyotaCar {
   constructor(){
      console.log('creating new object');
   }
   start(){
      console.log('start');
   }
   stop(){
      console.log('stop');
   }
}
let fortuner = new toyotaCar();  //new is a keyword to create an object from class.
let lexus = new toyotaCar();     //same new object bana diye same class se
```

--> Constructor( ) method is : [automatically invoked by new keyword] [initializes object]
  Jb hm class ke andar koi v constructor create nhi krte hain to ye new keyword automatically create kr deta h.

--> Jb bhi hm object bnate hain to sbse pahle Constructor() call hota hai.

# 11.39 Inheritance in JS

--> Inheritance is passing down properties & methods from parent class to child class.

*If Child & Parent have the same method, child's method will be used.

[Method Overriding]

```
class parent{
    hello(){
        console.log('hello man!');
    }
}
class child extends parent {};
let obj = new child();
obj.hello();      [o/p: hello man]
```

```
class parent{
    constructor(){
        this.species = 'homo sapiens';
    }
    hello(){
        console.log('hello man!');
    }
}
class child2 extends parent {
    hello(){
        console.log('hello hello!');
    }
    greeting(){
        console.log('good morning');
    }
}
let obj2 = new child2();
obj2.hello();                      [o/p: hello hello] [method overriding]
obj2.greeting();                   [good morning] [iska apna defined function bhi use kr liye]
obj2.species;                      [homo sapiens]
```

# 11.40 Inheritance in JS

--> super Keyword

The super keyword is used to call the constructor of its parent class to access the parent's properties and methods.

```
class person{
   constructor(){
      console.log('in parent constructor');
      this.species = 'homo sapiens';
   }
   eat(){
      console.log('eat');
   }
}
class engineer {
   constructor(branch){
      super();                                  //super() is used to call parent class constructor,
      console.log('in child constructor');
      this.branch = branch;
   } //child class ka constructor call karne se pahle parent class ka constructor call krna
pdta h nhi to error aayega
   work(){
      console.log('work');
   } }
let engObj = new engineer('electrical engineer');
```

# 11.41 Error Handling (try-catch)

```
 let a =5; let b =10; console.log(`a= ${a}`); console.log(`b= ${b}`);
console.log(`a+b= ${a+b}`);
 try{
  console.log(`c= ${c}`);
 }catch(err){
   console.log(err);
 }     //fayda ye hua ki agar ish line me error aata v h to v aage ka lines execute hoga
 console.log(`a-b= ${a-b}`); console.log(`a*b= ${a*b}`);
 console.log(`a*a= ${a*a}`); console.log(`a*a*a= ${a*a*a}`);
```

# Chapter:12

# 12.42 Synchronous in JS

--> Synchronous means the code runs in a particular sequence of instructions given in the program.
Each instruction waits for the previous instruction to complete its execution.

```
console.log('1');
console.log('2');
console.log('3');
console.log('4');
console.log('5');
[o/p: 1,2,3,4,5]
```

--> Asynchronous: Due to synchronous programming, sometimes imp instructions get blocked due to some previous instructions, which causes a delay in the UI.

Asynchronous code execution allows to execute next instructions immediately and doesn't block the flow.

```
console.log('1');
console.log('2');
setTimeout(() => {
   console.log('3');
}, 4000);
console.log('4');
console.log('5');
[o/p: 1,2,4,5,3]
```

# 12.43 Callbacks

A callback is a function passed as an argument to another function.

1)
```
function sum(a, b){
    console.log(a+b);
}
function calculator(a,b, sumCallback){
    sumCallback(a,b);
}
calculator(1,2, sum);
```

2)
```
const hello = () => {
    console.log('hii');
};
setTimeout(hello, 3000);
```

--> Callback Hell: Nested callbacks stacked below one another forming a pyramid structure. (Pyramid of Doom)
This style of programming becomes difficult to understand & manage.

```
function getData(dataId, getNextData){
    setTimeout( () => {
        console.log('data:', dataId);
        if(getNextData){
            getNextData();
        }
    }, 2000);
}
getData(1, () => {
    console.log('getting data2 ....');
    getData(2, () => {
        console.log('getting data3 ....');
        getData(3, () => {
            console.log('getting data4 ....');
            getData(4);

        }); }); });
```

```
[o/p: Jb har task 2-2 sec ke baad execute krna ho
data: 1, getting data2 ....,data: 2,getting data3 ....,data: 3
getting data4 ....,data: 4]
```

# 12.44 Promises:

Promise is for “eventual” completion of task. It is an object in JS.
It is a solution to callback hell.
let promise = new Promise((resolve, reject) => { .... })
*resolve & reject are callbacks provided by JS

--> A JavaScript Promise object can be:
Pending : the result is undefined
Resolved : the result is a value (fulfilled)
Rejected : the result is an error object
resolve(result)
reject(error)

*Promise has state (pending, fulfilled) & some result (result for resolve & error for reject).

```
case 1: let promise = new Promise( (resolve, reject) => {
        console.log('I am a promise');
    })
```
[Console se promise o/p--> I'm a promise and state:pending, result:undefined,
jab tak resolve ya reject function call nhi hoga state pending hi rahega]

```
case 2: let promise = new Promise( (resolve, reject) => {
        console.log('I am a promise');
        resolve('got the product');
    })
```
[Console se promise o/p--> I'm a promise, state:resolved/fulfilled, result:got the product]

```
case 3: let promise = new Promise( (resolve, reject) => {
        console.log('I am a promise');
        reject('order has been canceled during to shipping issue');
    })
```
[Console se promise o/p--> I'm a promise, state:rejected, result:Order has been canceled ...]

--> Real life me hm promise create nhi krte hain, ye hme kisi 3rd party ya api se milta hai,
jisko hum bs handle kar rahe hote hain.

```
function getData(dataId, getNextdata){
  return new Promise( (resolve, reject) => {
    setTimeout( () => {
      console.log('data:', dataId);
      resolve('job completed');
      if(getNextData){
        getNextData();
      }
  }, 5000);
  })
} --> console se: result = getData(123);
--> result [o/p: Promise{state:pending till 5 sec, result: undefined} after 5 sec: data:123,
state:fulfilled ,result:job completed ]
```

--> Now ab hm promise ko use krna dekhte hain, jo ki hmko kahi aur se milega (jaise api)

```
promise.then( ( res ) => { .... } )          //fulfilled ke case me
promise.catch( ( err ) ) => { .... } )       // reject hone ke case me
```

```
case 1: const getPromise = () => {
      return new Promise ( (resolve, reject) =>{
      console.log('I am Promise');
      resolve('I am done')
  }); };
let promise = getPromise();
promise.then( (res) => {
  console.log('promise fulfilled', res);
});
```

```
Case 2: const getPromise = () => {
      return new Promise ( (resolve, reject) =>{
      console.log('I am promise');
      reject('I am gone');
  });
};let promise = getPromise();
promise.catch( (err) => {  console.log('Promise rejected', err); });
```

--> **Promise chaining**:

```
function asyncFunc1(){
    return new Promise( (resolve, reject) => {
        setTimeout( () => {
            console.log('data1');
            resolve('success');
        }, 4000)
    });
}
function asyncFunc2(){
    return new Promise( (resolve, reject) => {
        setTimeout( () => {
            console.log('data2');
            resolve('success');
        }, 4000)
    });
}
---------------------------------------------
console.log('fetching data1 ....');
let p1 = asyncFunc1();
p1.then( (res) => {
    console.log('fetching data2 ....');
    let p2 = asyncFunc2();
    p2.then( (res) => {});
});
------------------------------------------------
console.log('fetching data1 ....');
asyncFunc1().then( (res) => {
    console.log('fetching data2 ....');
    asyncFunc2().then( (res) => {});
}
);
-------------------------------------------------
```

```
function getData(dataId){
    return new Promise( (resolve, reject) => {
        setTimeout( () => {
        console.log('data:', dataId);
        resolve('success');
        }, 3000);
    });
}
getData(1)
    .then( (res) => {
    return getData(2);
    })
    .then ( (res) => {
      console.log(res);
});
```

# 12.45 Async-Await

async function always returns a promise.

async function myFunc( ) { .... }

await pauses the execution of its surrounding async function until the promise is settled.

await can be used only inside async function.

```
function getData(dataId){
    return new Promise( (resolve, reject) => {
        setTimeout( () => {
        console.log('data:', dataId);
        resolve('success');
        }, 3000);
    });
}
```

```
async function getAllData(){
    console.log('getting data1 ...');
    await getData(1);
    console.log('getting data2 ...');
    await getData(2);
    console.log('getting data3 ...');
    await getData(3);
    console.log('getting data4 ...');
    await getData(4);
} [o/p: getAllData();
Promise {<pending>}
index.html:26 data: 1
index.html:26 data: 2
index.html:26 data: 3
index.html:26 data: 4]
```

# Summary:
# Callback Hell

```
function getData(dataId, getNextData){
    setTimeout( () => {
        console.log('data:', dataId);
        if(getNextData){
            getNextData();
        }
    }, 2000);
}

getData(1, () => {
    console.log('getting data2 ....');
    getData(2, () => {
        console.log('getting data3 ....');
        getData(3, () => {
            console.log('getting data4 ....');
            getData(4);

        }); }); });
```

# Promise chaining

```
function getData(dataId){
    return new Promise( (resolve, reject) => {
        setTimeout( () => {
        console.log('data:', dataId);
        resolve('success');
        }, 3000);
    });
}
console.log('getting data1 ...');
getData(1)
    .then( (res) => {
    console.log('getting data2 ...');
    return getData(2);
    })
    .then ( (res) => {
    console.log('getting data3 ...');
    retutn getData(3);
    })
    .then( (res) => {
        console.log(res);
    });
```

# Async-Await

```
function getData(dataId){
    return new Promise( (resolve, reject) => {
        setTimeout( () => {
        console.log('data:', dataId);
        resolve('success');
        }, 3000);
    });
}
async function getAllData(){
    console.log('getting data1 ...'); await getData(1);
    console.log('getting data2 ...'); await getData(2);
    console.log('getting data3 ...'); await getData(3); }
```

--> IIFE : Immediately Invoked Function Expression. It can be used only once.
IIFE is a function that is called immediately as soon as it is defined.

Syntax: (function_definition) ();

```
(function (){
 console.log('Hello Prakash Jha');
 }) ();
```

# Chapter:13

# 13.46 fetch API (Application Programming Interface)

--> The fetch API provides an interface for fetching (sending/receiving) resources.
It uses Request and Response objects.
The fetch() method is used to fetch a resource (data).

Syntax: let promise = fetch(url, [options])

```
const URL = "https://cat-fact.herokuapp.com/facts";

const getFacts = async () => {
   console.log ('getting data ...');
   let response = await fetch(URL);
   console.log(response);
};
```

[o/p: it will return some output but that won't be completely understandable]

--> AJAX is Asynchronous JS and XML
JSON is JavaScript Object Notation
json() method: returns a second promise that resolves with the result of parsing the response body text as JSON.
(Input is JSON, Output is JS Object)

```
const URL = "https://cat-fact.herokuapp.com/facts";

const getFacts = async () => {
   console.log ('getting data ...');
   let response = await fetch(URL);
   console.log(response);
   let data = await response.json();
   console.log(data[0].text);
};
```

[o/p: Cat can be a beautiful Pet]

```
# Mini API Project 1
<button id="btn">Get a Fact</button>
<p id="fact"></p>

const URL = "https://catfact.ninja/fact";
const factPara = document.querySelector('#fact');
const btn = document.querySelector('#btn');

const getFacts = async () => {
   console.log ('getting data ...');
   let response = await fetch(URL);
   //console.log(response);
   let data = await response.json();
   factPara.innerText = data.fact;
   console.log(data.fact);
}; btn.addEventListener('click', getFacts);

# Mini API Project 2
<button id="btn">Get a Dog Image</button>
  <br><br>
  <div id ="div_img" style="width: 600px; height: 500px; background-color: lightgrey;
transition: background-image 0.1s ease; background-image: url('Dog.jpeg');">
<script>
const URL = "https://dog.ceo/api/breeds/image/random";
const imgDiv = document.querySelector('#div_img');
const btn = document.querySelector('#btn');

const getFacts = async () => {
   console.log ('getting data ...');
   let response = await fetch(URL);
   console.log(response); console.log(response.url);
   let data = await response.json();
   console.log(data.message);
   imgDiv.style.backgroundImage = `url('${data.message}')`;
   imgDiv.style.backgroundSize = "cover"; imgDiv.style.backgroundPosition = "center";
}; btn.addEventListener('click', getFacts); </script>
```

# 13.47 HTTP Request methods & Response status code:

--> GET, POST, PUT, HEAD

--> 1. Informational responses (100-199)

2. Successful responses (200-299)
3. Redirectional messages (300-399)
4. Client error messages (400-499)
5. Server error messages (500-599)

--> HTTP response headers also contain details about the responses, such as content type, HTTp status code etc.

# Completed !!!