__/__/____

- Difference between var, let, const

- scope : var has functional scope ; let & const has block scope

- updation : only var & let can updated ; const wont be updated.

- redeclared : only var can be re-declared ; let & const wont.

- declaration : var & let can be declared without initialization
  const can not be declared without initialization.

__/__/____

- What is default, spread and rest operator.

- Defaulit Parameters : It is used to give the default values to the arguments, if no parameter is provide in the function call.

```
function add (a, b = 3) {
    return a+b
}

Console.log(add (5,2)) ;      // 7
console.log(add (5)) ;        // 8
```

- Spread Operator : It generally spreads data of array/list.

```
console.log (Math.min (1,2,3,-1)) ;      // -1

let arr = [1, 2, 3, -1] ;
console.log (Math.min (arr)) ;           // NaN
console.log (Math.min (...arr)) ;        // -1
```

__/__/____

- Rest Operator : It allows a "fun" to accept an indefinite number of arguments if we are not sure how many arguments will receive.

```
function myfun(a, b, ...moreArgs)
{
    console.log("a", a);
    console.log("b", b);
    console.log("Args", moreArgs);
}
myfun("one", "two", "three", "four", "five", "six");

// a one
// b two
// Args(4) ["three", "four", "five", "six"];
```

__/__/____

★ Shallow Copy & Deep Copy

- Shallow Copy: When a reference variable is copied into a new reference variable, a shallow copy of the referenced object is created.
  - In simple words, a reference variable mainly stores the address of the object it refers to.
  - When a new "reference variable" is assigned the value of the "old reference variable", the "address stored" in the "old reference variable" is "copied" into the "new one".
  - This means both the "old and new reference variable point" to the "same object in memory".

built in:
. spread operator, concat, slice, object.create(), object.assign()

- Deep Copy:
  - It makes a copy of all the members of the old object, allocates seperate memory locations for the new object, and then assigns the copied members to the new object.
  - In this way, both the objects are independent of each other and in case of any modification to either one, the other is not affected.
  - If one of the object is deleted, the other still remains in the memory.

JSON.stringify(), JSON.parse()

__/__/____

What is hoisting ?

- Hoisting is the process of moving declarations of variable, functions, or classes to the top of their scope before code execution.
- In simple word, a variable can be ~~declared~~ used before it has been declared.

```
a = 1;
console.log(a);
var a;
```

- Only 'var' can be accessible in the process of 'hoisting'.

__/__/____

★ What is async/await ?

- **async :**
  - This keywords contains Asynchronous Execution.
  - The return value will be the "Resolved value for the promise".

- **await :**
  - It makes execution thread to wait for asynchronous task/promise to resolve before proceeding further.
  - It introducing synchronous behaviour to the appl^n. Even the promises will be executed asynchronously.

```
function returnPromises () {
    return new Promise (resolve) => {
        setTimeout(() => {
            console.log ("Promise Executed");
            resolve ("sample Data");
        }, 3000);
    });
}

async function ExecuteFunction () {
    var getData = await returnPromises ();
    console.log (getData);
}
ExecuteFunction();
```

__/__/____

★ What is Promise ?

- Promise is an object that may returns a resolved value or not resolved value.
- A promise is an object which can be returned synchronously from an asynchronous function.
- A Promise must be in one of three possible state.

  - Fulfilled : (resolve() was called)
  - Rejected : (reject() was called)
  - Pending : not yet fulfilled or rejected.

```
const wait = time => new Promise((resolve) => {
    setTimeout(resolve, time)
});

wait(3000).then(() => console.log('Hello'));
```

__/__/____

★ What is callback function ?

- It is function that is passed into another function as an arguments, which is then triggered when the other function is triggered.

```
const display = (str) => {
    console.log (str);
}
const myFun = () => {
    display ('Hi');
}
```

__/__/____

★ What is call(), apply(), and bind() ?

call() : call() invokes the function and allows you to pass in arguments one by one.

apply() : apply() invokes the function and allows you to pass in arguments as an array.

bind() : bind() returns a new function, allowing you to pass in a this array & any number of arguments.

```
const cad = (...user) => {
    console.log(user);
}

cad.call('prahik');
cad.apply(['prahik']);

let name = { fname : 'Prateik', lname : 'Dhotmal' }
let callName = cad.bind(name);

callName();
```

__/__/____

★ HTML

- Semantic HTML & Accessibility Guidence.

  - Semantic elements are those which are identified by their name itself.
    eg. <form>, <nav>, <header>, <footer>

  - non-semantic element eg.: ,

- Accessibility Guideline

  - Use semantic tags
  - Add heading tags (<h1>.... <h6>)
  - Use alt, ref, title attribute accordingly
  - Declare Languages (<html lang="en">)
  - Use clear content (Don't use shortcuts)

- Form Validation

  - Either use HTML5 validation or go with JS validation

  HTML5 - required, type = email, tel, number, date.

  JS - write a regex and .match() or .test() with input

__/__/____

- Media - Responsive Images, Audio, Video

- Responsive Images :

  - To increased performance

```
<img srcset="test.jpg" src="test2.jpg" alt="img">
```

  - To control design

```
<picture>
    <source srcset="test.jpg" media="(min-width:650px)">
    <img src="test.jpg" alt="test">
</picture>
```

- Audio & video

  - Audio

```
<audio control>
    <source src="test.mp3 type="audio/mpeg">
</audio>
```

  - Video

```
<video controls>
    <source src="test.mp4" type="video/mp4">
</video>
```

__/__/____

- Workers

* Web Workers

- A Web workers is a JS code that runs in the background and does not affect the page performance
- JS is a single-threaded language; it can only process one task at a time.
- If a process is likely to take a long time, the user is not expected to wait until it is completed.
- Here web workers comes into the picture. Such a long tasks performed in a background using workers.
- Types
  - shared workers
  - Dedicated workers.

* Service Workers

- It is a script that runs independently in the browser background
- It is mainly used for
  - manage application cache
  - manage push notification
  - manage API calls

__/__/____

- Cookies
  - Cookies were store in server.
  - Cookies store max 4kB of data
  - We need to ~~some~~ set expiry date for cookies.
  - Data will persist for long time.

```
document.cookie = "uname = 'Prateik ; expires = {date}" ;
```

- Local Storage
  - It is stored in browser side.
  - It store max 10 MB of data
  - Local storage were never expire automatically.
  - Data will persist till the browser window closed.

```
localStorage.setItem (key, value);

let uname = localStorage.getItem(key);

localStorage.removeItem(key);

localStorage.clear();
```

__/__/____

- Session Storage
  - Session is stored in browser side
  - Session store max 5MB of data.
  - Data will persist till the browser tab is not closed

// sessionStorage.setItem / getItem / removeItem / clear

- IndexDB
  - It stored data on client side
  - It stored data like a NoSQL DB.
  - We can set expiry for IndexDB
  - We can also pass file as a data in IndexDB.

__/__/____

- Optimize HTML to boost performance
  - Use semantic tags
  - Add CSS files at the top
  - Add JS files at the bottom
  - Use alt, title, ref accordingly
  - Reduce inline CSS/JS
  - Minify CSS/JS
  - Compress images
  - Use icon sprite
  - Use GZip on server.
- SEO Basics
  - Add meta title
  - Add meta desc
  - Add heading tags
  - Add meta keywords
  - Use alt attributes
  - 301 redirection
  - Activate all page back links
  - Page Authority
  - Domain Authority
  - Add robot.txt → sitemap.xml

__/__/____

★ CSS

- BEM - Block, Element, Modifier

It's a CSS naming convention for writing cleaner & readable CSS classes. It also aims to write independent CSS blocks that reuse later.

```
<div class='card'>
    <div class="card__header"> ... </div>
    <div class="card--active"> ... </div>
</div>
```

- .card - Block
  - .card__header - Element
  - .card--active - Modifier

Block - It is a parent component.

Element - It is child component of parent.

Modifier - It specifies the independent style changes. It works with both Block & Element.

__/__/____

☆ Javascript

- Variable : It is container for storing the data.

- Scope : Scope determines the limit or accessibility of variable, object or functions.
  There are 3 types of scope.

  1. Block Scope : The variable declared inside the braces, were only be accessable inside that braces only (except 'var)

     ```
     {
       let x = 1;
     }
     ```

  2. Function Scope : The code which is return inside any function that function becomes the limit/scope of the function.

     ```
     function fun() {
       let x = 10;
     }
     ```

  3. Global Scope : A variable declared outside a function becomes GLOBAL. These Global variable has global scope.

     ```
     let x = 10;
     function fun() {
       console.log(x);
     }
     ```

__/__/__

- Data Types :

1. Primitive DataTypes.

- Number
- String
- ~~Object~~
- Boolean
- NULL
- Undefined
- Symbol

2. Non Primitive DataType

- Object

- Javascript Operator
  - Arithmatic operators
  - Assignment Operators
  - Comparison Operators
  - String operations
  - Logical Operator
  - Bitwise operator
  - Ternary Operator
  - Type operator

__/__/____

- Loops

1: for : loops for a block of code number of times.
2: for/in : loops through the properties of an object.
~~use for single object~~
3: for/of : loops for the values of an iterable object.
4: while } loops for a block of code while
5: do-while } specified condition is true.

- Conditional Statement

1 If...else : To specify a block of code to be executed, if a specified condition is true.

```
if(x===2)
{
console.log(2);
}
else
{
   console.log('Not 2');
}
```

__/__/____

2. Switch : It is like perform different activity depend on different condition.

```
switch (condition) {
    case 1:
        val = 1; break;
    case 2:
        val = 2;
        break;
    default
        val = 'default';
}
```

3. Ternary Operator : It is an operator which defines true or false expression based on their condition.

```
const x = 100;
const xVal = x > 100 ? 'Up' : 'Down';

console.log (x)        // Down
```

__/__/____

- Error Handling
  - Javascript provides error-handling mechanism to catch runtime errors using try-catch block.
  - It helps us to not crash our application.

```
try {
    // code that may throw an error
}
catch (e) {
    // code to be executed if an error occured.
}
finally {
    // code to be executed regardless of an
       error occurs or not
}
```

__/__/____

- Iterators

∵ It is an object which defines a sequence and return a value using 'next()' method.

'next()' method returns an object with 2 properties.

1. value : The next value in iteration sequence.
2. Done : It is return boolean value
   - true if iterator is completed
   - false if iterator produce new value.

- Iterables

It is an iterable objects like an array. The iterable allows define or customize their iteration behavior

To use @@iterator method we must use the [Symbol.iterator]

__/__/____

```
Iterable {
    [symbol.iterator] : Iterator
}
Iterator {
    next() : IResultObj
}
IResultObj {
    value : any
    done : any
}
```

- Generators : In JS regular functions return only once, to overcome this generators helps. Generator can return multiple values using "yield," one after another, on-demand.

```
function* fun() {
    yield 10;
    yield 20;
}
let gen = fun();

console.log(gen.next().value);
console.log(gen.next().value);
```

__/__/____

- Reflect
  - It is an built in object that provides method for JS operation
  - All methods are static
  - Has no constructor ; cannot use new.
  - It is used to call the related internal method.

```
let data = {
    name : 'Prateik',
    id : 93,
    hello : function (a,b) {
        console.log(`Hello, ${this.name} : ${a} ${b}`);
    }
}

console. log (Reflect.ownKeys(data)); // ['Prateik', 93, hello]
console.log (Reflect.get( alex, 'id')); // 93
console. log (Reflect.set (alex, 'id', 99)); // true
console. log (Reflect. get (alex, 'id')); // 99
```

__/__/____

- Proxy

  - It is an object which provides a way to customize operation on objects.
    - Operations are such as,
      property lookup, assignment, enumeration, and function invocation.

```
let proxy = new Proxy (target, handler);
```

- target : is an object to wrap
- handler : is an object that contains method to control the behaviour of target.

```
const user = { fname : 'abc', lname : 'pqr' }

const handler = {
    get(target, property) {
        console.log(`${property} has read`);
        return target[property];
    }
}

const proxyUser = new Proxy (user, handler);
console.log (proxyUser.fname);
// fname has read    // abc
```

__/__/____

- Object Prototype

  - Prototypes are the JS mechanism by which JS objects inherit features from one another.

- Prototype Inheritance

  - When we ~~need~~ read property from object, and it is missing; JS automatically takes it from the prototype. This is called 'Prototype Inheritance'.

```
let animal = {
  eats : true,
};

let dog = {
  barks : true
}

dog.__proto__ = animal;  // sets dog.[[Prototype]] = animal

console.log(dog.eat);  // true
```

__/__/____

- Closure :

It is allows functions to access variables from the enclosing scope, even after it leaves the scope in which it declared

```
function first () {
    let breakfast = 'breakfast';
        return function second () {
            let lunch = 'lunch';
            return function third () {
                let dinner = 'dinner';
                    return `${breakfast} -> ${lunch} -> ${dinner}`;
            }
        }
}
first ()()();          // bf -> lunch -> dinner
```

- A closure is where an inner function has access to the outer functions variable.

__/__/____

- Hoisting

  - It is a behaviour in which a function or a variable can be used before declaration.
    - In JS, you can call a function before it is defined and you get a 'Undefined' instead of 'Uncaught ReferenceError'
    - The reason behind this is 'hoisting', where the JS interpreter always moves the variables & function to the top of the scope before the code execution.

```
myfun('Hello');

function myfun(str)
{
console.log(str);
}
```

__/__/____

- IIFE - Immediately Invoked Function Expression

  - It is common pattern used in JS to create a function & execute it immediately after it defined.

  - The purpose of using IIFE is to create a new scope for the function, which helps prevent variable name clashes & keeps the global scope clean.

```
(function () {

    var x = 10;
    console.log(x)
})();

//10
```

__/__/____

- Map
  - It is used for creating a new array from an existing one.
  - It applying a function to each one of the elements of the first array.

```
const numbers = [1, 2, 3, 4];
const doubled = numbers.map(item => item * 2);
console.log(doubled);        // 2, 4, 6, 8
```

- Reduce
  - It reduces an array of values down to just one value.
  - To get the O/P value, it runs a reducer function on each element of the array.

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce(function(result, item) {
    return result + item;
},0);

console.log(sum);        // 10
```

__/__/____

- Filters
  - This method takes each element in an array and it applies a conditional statement against it.
  - If this conditional returns true, the element gets pushed to the O/P array.
  - If the conditional return false, the element does not pushed to the output array.

```
const numbers = [1, 2, 3, 4];

const evens = numbers.filter(item => item % 2 ===0);

console.log(evens);    // [2, 4]
```

__/__/____

Array Methods

- concat() : It is used to merge 2 or more arrays.

- every() : This method test whether all elements in the array pass the test implemented by the provided function. It returns Boolean value.

- some() : This method test at least one element in the array passed the test.

- find() : This returns the first element in the provided array that ~~specifies~~ satisfies the condition. If no value satisfy "undefined" is return.

- forEach() : This method executes a provide function once for each array element.

- includes() : This method check whether an array includes certain value.

- indexOf() : This method returns the 1st index at which a given element can be found in the array, or return -1

- push() : This adds the specified element to the end of an array.

- pop() : This method removes the last element from an array.

__/__/____

- shift() : This methods removes the first element from array.

- slice() : This returns a shallow copy of a portion of an array into new array object selected from start to end.

```
const num = [1,2,3,4,5]
console.log (num.slice(2));        // [3,4,5]
console.log (num.slice(2,4));      // [3,4]
```

- splice() : This changes the content of an array by removing or replacing existing element

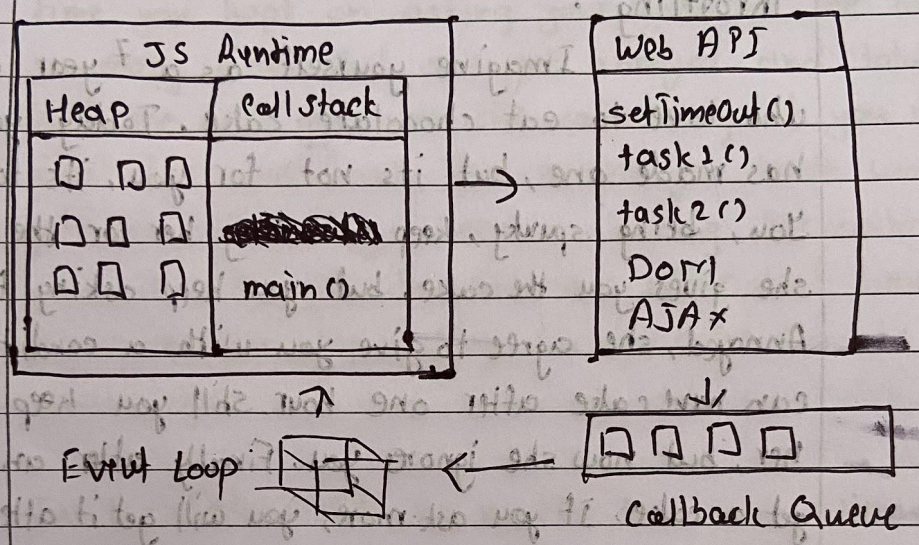
```
const num = [1,2,3,4,5]
~~console.log~~
num.splice(1,0,2.2);
console.log(num);        // [1,2.2,2,3,4,5]
```

- sort(); This methods sorts the elements of an array & returns the reference to the same array

__/__/____

★ Event Loops :

- JS is a single-threaded asynchronous programming lang.
- It means, it runs code one by one & not parallely.



- Heap - it stored variab & function memory.
- callStack - it stored the executed functions.
- Web API - It is works a JSE assistant. It handles the blocking behavior of asynchronous JS code.
- Callback Queue : This is where your asynchronous code gets pushed to, and waits for the execution.
- Event Loop : It check if the call stack is free, then lets know the callback queue to check is their any task, if yes then it move to the call stack.

__/__/____

Debouncing & Throttling

- **Throttling :**

Imagine yourself as a 7 year old boy who love to eat chocolate cake. Today your mom has made one, but its not for you, it's for the guest. You, being spunky, keep on asking her for the cake. Finally she gives you the cake, but you keep asking for more. Annoyed, she agree to give you with a condition that you can have cake after one hour. Still you keep on asking her, but now she ignores you. Finally after an hour you get a cake. If you ask more, you will get it after an hour, no matter how many times you ask her.

Throttling means, no matter how many times the user fires the event, the attached function will be executed only once in a given time interval.

eg. When a user clicks on a button, a helloWorld() function which print Hello. Now when throttling is applied with 5 seconds to this helloWorld function, no matter how many times the user click on the button, Hello will be printed only once in 5 seconds. Throttling ensures that the function executes at a regular interval.

__/__/____

- Debouncing :

Conside the same example of cake. This time you kept on asking your mom for the cake so many times that she got annoyed and told you that she will give you the cake only if you remain silent for one hour. This means you ~~have~~ won't get the cake if you keep on asking her continuously, you will only get it one hour after last time you ask, once you stop asking for the cake. This is Debouncing.

In the Debouncing, no matter how many times the user fires the event, the attached function will be executed only after the specified time once the user stops firing the event.

eg. Suppose a user is clicking a button 5 times in 1 seconds. Debouncing will not let any of these clicks execute the attached function. Once the user has stopped clicking, if debouncing time is ~~100~~ 1 seonds, the attached function will be executed after 1 seconds. Thus, to a naked eyes, debouncing behaves like grouping multiple event into one single event.