Date :.....

MON TUE WED THU FRI SAT SUN

JavaScript

- It is lightweigh (just in time compiled) programming language.

Just in time- Compilation of code during executing of program ~~of~~ i.e. at runtime rather than before execution.

→ How to link JS code in HTML?

(a) Internal JS -

We can add JS directly to our HTML file by writing code inside the <script> tag.

<script> tag can either be placed inside the head or body tag according to the requirement.

(b) External JS:

We can write JS code in another file having an extension .js & then link this file ~~to~~ inside head tag of HTML ~~code~~ file in which we want to add this code

→ JS is a (weakly typed) language.

Date :.....

MON TUE WED THU FRI SAT SUN

☐ ☐ ☐ ☐ ☐ ☐ ☐

=> Why JS in known as lightweight programming languag

It is considered as lightweight due to the fact that it ha low CPU usage, easy to implement and has minimalist syntax.

Minimalist Syntax means, has no data types.

- Everything here is treated as an object.

=> JavaScript is Compiled or Interpreted language?

Answer is Both, previously it was ~~compiled~~ Intepreter language but after the V8, the JIT compiler was also incorporated to optimize the execution & display the result more quickly.

→ Template Literals : When we declare strings using backticks (`), this are called template literals, it has special properties, we can embed $ variables or even expression in them.

```
const greeting = `Hello ${name}`;
```

Date :.....

MON TUE WED THU FRI SAT SUN

→ Useful Number Methods -

(a) toFixed() - Round no. to a fixed no. of decimal places.

E.g. :

```
const numValue = 1.765432
numValue
const updatedValue = numValue.toFixed(2)
console.log(updateValue);
```

O/P : 1.76

(b) Converting to Number data types -

Number()

E.g.

```
const textValue = "53";
const noValue = Number(textValue)
console.log(typeof noValue);

// O/P: Number
```

→ String() : It converts anything into string.

String()

E.g.

```
const value = 53
const strValue = String(value);
console.log(typeof strValue);
```

O/P : String

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

→ Useful string methods -

(a) length of the string - length property.

e.g: const value = "abc"
console.log (value.length);

// O/P: 3

(b) Retreiving specific string character: We assume string as an array, so we can retrieve character from string using [] square bracket.

e.g.: const value = "Hello";
console.log (value[0]);
// O.P: H

(c) Check if strings contain substring: Includes () method.

e.g. const value = "Hello world";
if (value.includes ("World")) {
console.log ("Found World");
}
else {
console.log ("Not Found");
}

Includes() returns true if string contains substring, else false otherwise.

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

(d) startsWith() & endsWith():

If you want to know if a string starts or ends with particular substring. Returns true or false.

```
E.g.  const value = "mozilla";
      if (value.startsWith("moz")) {
          console.log("Moz");
      }
      else {
          console.log("No Moz");
      }
```

(e) Find position of substring in a string : IndexOf()

It takes two parameter - a) substring that we want to search
b) Optional parameter that specifies starting point of search.

It returns index of the first occurence of the substring. If string doesn't contain the substring, IndexOf() returns -1.

```
E.g.  const tagline = "MDN - Resources for developers, by
                       developers";
      console.log(tagline.indexOf("developers"))
```

O/P: 20

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

(f) Extracting substring from string: slice() method.
It takes two parameters:
(i) Index at which start extracting.
(ii) Index at which to stop extracting, this index is exclusive.

```
const type = "mozilla";
console.log(type.slice(2, 7));
```

O/P: "zilla"

We can also do, if we want to extract all the remaining characters after a certain index, then don't include second parameter.

```
const type = "mozilla";
console.log(type.slice(2));
```

O/P: "zilla"

(g) Changing Case:
(i) toLowerCase()
(ii) toUpperCase()

```
const data = "Mozilla";

console.log(data.toLowerCase()); // "mozilla"
console.log(data.toUpperCase()); // "MOZILLA"
```

Date :.....

MON TUE WED THU FRI SAT SUN

(b) Updating parts of a string:

We can replace one substring inside a string with another substring using the replace().

It takes two parameters, (I) string we want to replace
(II) string we want to replace it with.

E.g. const browerType = "mozilla";
const updated = browerType.replace("moz", "van");

console.log(updated); // ~~vanilla~~ vanilla

- replace() in this form only changes the first occurences of the substring. If we want to change all occurences, we can use replaceAll().

E.g.: let quote = "To be or not to be";
quote = quote.replaceAll("be", "code");
console.log(quote); // To code or not to code

Date :.....

MON TUE WED THU FRI SAT SUN

☐ ☐ ☐ ☐ ☐ ☐ ☐

Arrays :

Arrays are "list-like objects", they are basically single objects that contain multiple values stored in a list.

→ Find length of array -

```
const shopping = ["bread", "milk", "cheese"];
console.log(shopping.length);
```

→ Finding the index of items in an array :

```
const birds = ["Parrot", "Falcon", "Owl"];
console.log(birds.indexOf("Falcon")); // 1
console.log(birds.indexOf("Owl")); // 2
console.log(birds.indexOf("Rabbit")); // -1
```

→ Adding Items :

```
const cities = ["Manchester"];
cities.push("Cardiff");
console.log(cities); // ["Manchester", "Cardiff"];
cities.push("Bradford", "Brighon");
console.log(cities); // ["Manchester", "Cardiff", "Bradford",
                     "Brighon"];
```

- If we want to add item at start, use unshift()

```
cities.unshift("Cardiff");
```

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

- push() returns new length of an array.

```
console.log ( cities.push ("Armsterdam")); // 5
```

→ Removing Items:

- To removing the last item from the array, use pop()

```
const cities = ["Manchester", "Cardiff"];
console.log ("Removing items is", cities.pop());

// Removing item is Cardiff
```

pop() return the item that was removed.

- To remove the first item from an array, use shift()

```
const cities = ["Manchester", "Cardiff"];
console.log ("Removing item is", cities.shift());

// Removing item is Manchester
```

- To remove an item of specific ~~itr~~ index, use splice()

```
const cities = ["Manchester", "Cardiff", "Liverpool"];
cities.splice ("cities.indexOf ("Cardiff"), 1);

console.log (cities);
// ["Manchester", "Liverpool"]
```

splice() takes two arguments, first says where to start removing items. & second says, how many items should we removed.

Date :.....

MON TUE WED THU FRI SAT SUN

☐ ☐ ☐ ☐ ☐ ☐ ☐

Accessing every item

(1) for ... of :

```
for (const city of cities) {
    console.log (city);
}
```

(2) map (): If you want to some operation of array element.

```
const num = [2, 3, 5, 7];
const double = num.map (value => {
    return value * 2;
});
console.log (double); // [4, 6, 10, 14]
```

Converting between strings & array:

- Use the split() method to split the string into an array. It takes single parameter, the character you want to seperate the string at, & returns the substrings between the seperator as items in an array.

```
const cities = "Manchester, London, Liverpool";
const arr = cities.split (", ");
console.log (arr);
```

Date :.....

MON TUE WED THU FRI SAT SUN

- Now, if you want to convert array into string, use join()

```
const arr = ["Manchester", "London", "Liverpool"];
const cities = arr.join(", ");
console.log(cities);
```

- Another way of converting array to string is toString(). its not take any parameter, & it's limiting than join() because it always use comma.

```
const arr = ["Manchester", "London", "Liverpool"];
const cities = arr.toString();
console.log(cities); // Manchester, London, Liverpool
```

Ashutosh

Date :.....

MON TUE WED THU FRI SAT SUN

React Book

JavaScript (JS)

- How JS works & Execution Context :

Note, Everything in JS happens inside an execution context.

Execution context : It is a box or container in which whole JS code is executed.

This context is divided into two part :

(1) Memory component / Variable Enviorment :
Store all variable with its value in key-value pair & also stores function.

(2) Code component / Thread of execution :
Store JS code which executes one line at a time.

Hence, JS is synchronous single threaded language.
↓ one thing at a time ↓ performs one action only.

- When we run a JS program, an execution context is created.
- When exec. context created, it will also memory phase & then code phase.

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

- In memory phase, at start, the default value of each variable is [undefined]. For function, it stores whole code of it.

- Value of each variable is assign when code phase executed.

! FUNCTIONS ARE HEART OF JS !

- When any function, is called / invoke, a new execution context created.

- Multiple exec. context are manage using call stack.
i.e. Call stack maintains the order of execution of execution contexts.

=> Hoisting in JS:

Hoisting is a core concept relies on the way how exec. context is created.
In first phase i.e Memory allocation phase, all the variables & functions are allocated memory, even before any code is executed. All the variables are assigned 'undefined' at this instance in local memory.

=> Closure:

A function which bind together with in lexical environment.

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

closure example:

```
function x(){
    var a = 7;
    function y(){
        console.log(a);
    }
    y();
}
x();
```

Therefore, closure is the combination of function bundled together (enclosed) with reference to its surrounding state (lexical environment)

Uses of Closures:

(a) Module Design Pattern
(b) Currying
(c) Functions like once
(d) Memoize
(e) Maintaining state in async world
(f) setTimeouts
(g) Iterators.

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

→ Function Statement : A.K.a Function Declaration.

```
function a () {
    ---
}
```

→ Function expression : Function used as values.

```
var a = function x () {
    ---
}
```

- Function Statement v/s Function expression : Hoisting
i.e we can call function anywhere (even before function declaration) but if we assign function to variable & we call it before declaration then it return error.

→ Anonymous Function :
Function which do not have it's own identity.

```
var anonFun = function () {
    //....
}
```

We cannot create anonymous function using function statement, only ~~var anonFun~~ create using function expression.

Date :.....

MON TUE WED THU FRI SAT SUN

→ Named Function Expression:

When we used function as value, but it ~~is use~~ has it own identity, i.e,

```
var a = function xyz() {
    //...
}
```

but we cannot call function with its identity, it gives error, as xyz is not created in execution context.

→ Diff. b/w Parameters & Arguements:

Many times we are using both these terms as same interchangable way. IT IS WRONG!

```
function a ( param1, param2) {
    //....
}
```

Parameters

```
a ("Hello", "World")
```

Arguements

Parameters: local variable for fundtion, which holds the value

Arguements: Values passed to that function

Date :.....

MON TUE WED THU FRI SAT SUN

→ First Class Functions : A.k.a First class citizens.

In JS, when functions are used as values, function are passed to parameters, or used to return, that are called first class function.

Eg: var a = function () {
//...
}

OR

function xyz (~~function~~ param1) {
//...
}
xyz (function a () { })

OR

function xyz () {
return function a () { }
}

Date :.....

MON TUE WED THU FRI SAT SUN

→ Call back Function:

A function which passed to another function in its parameters are known as call back function.

- Callback also helps us to achieve asynchronous in JS.

E.g: function x (par1) {
    // ...
}

x (function y () { })
Call back function

Why it is called Callback?

We can call the function (y) anytime later in function x. i.e we passed call responsibility of one function to another function.

How callback get us advantage of asynchronous?

E.g. setTimeout (function () { ← callback function
    console.log ("Hello")
}, 5000);

setTimeout is asynchronous but it won't be possible without call back function.

Date :.....

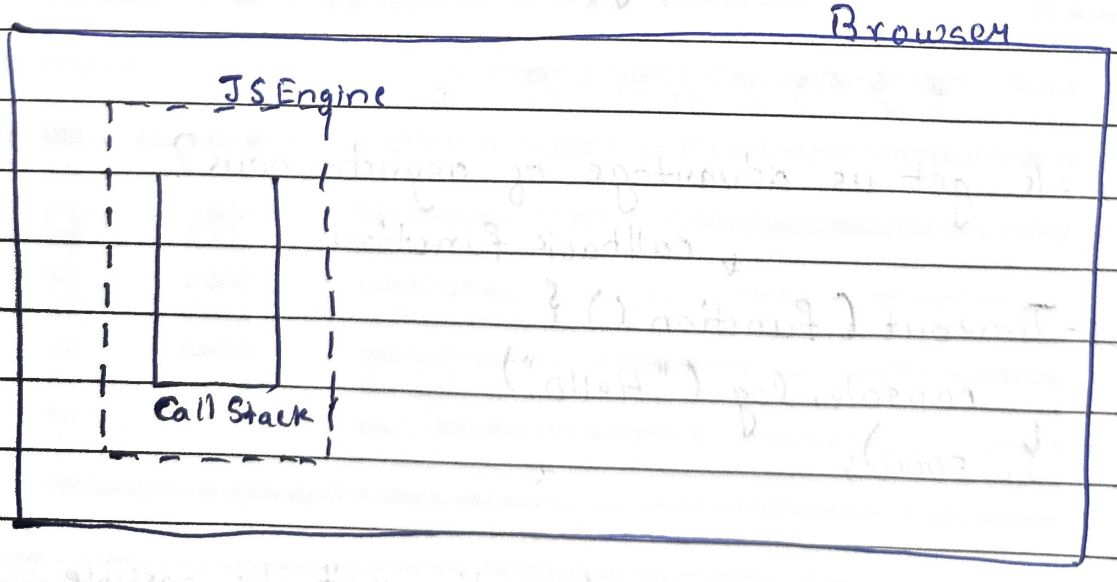
MON TUE WED THU FRI SAT SUN

→ Event Listeners:

Event listeners also use callback function for performing action on certain event.

```
document.getElementById(" ").
    addEventListener("click", function x(){
        //...
    }
);
```

- Now it is standard practice to remove eventListeners, but why?
  - It's because event listeners are heavy, they consume lot of memory which can potentially slow down the website therefore it is good practice to remove it is not used.



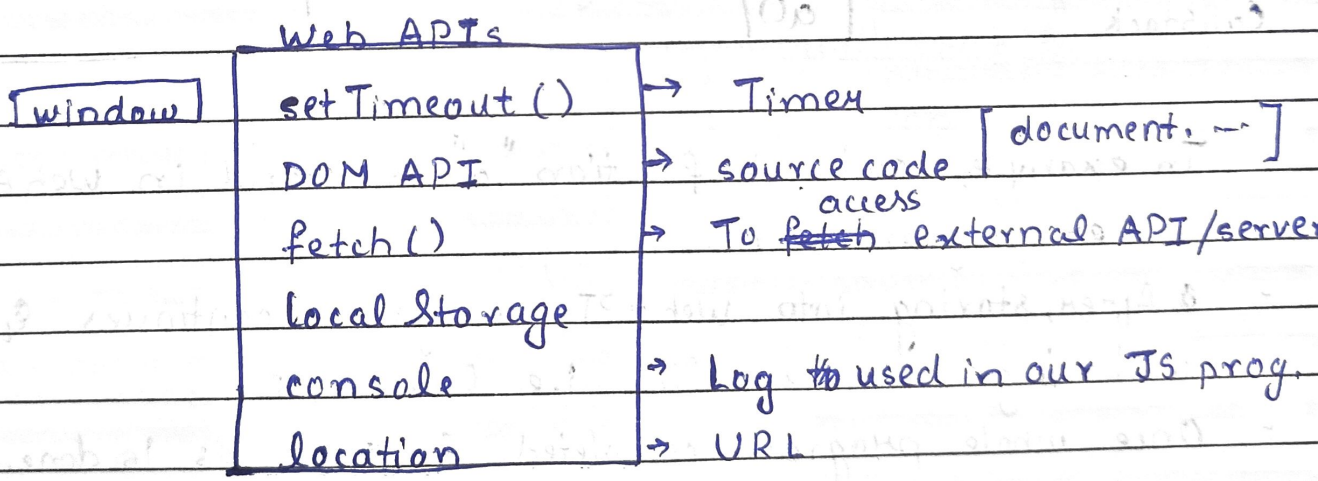
- Js program runs in call stack using JS Engine, & JS engine is installed in Browser, beoz of which we can run JS program into browser.

Date :.....

MON TUE WED THU FRI SAT SUN

- Now, Browser has various other feature,
  (a) Local storage
  (b) Timer
  (c) URL
  (d) Connection with external servers.
  (e) Geolocation

→ How to access this browser features in our javascript program?
We have several in-built function in JS engine, which can help us to ~~fetch~~ access this feature, which are,



- This are all web APIs to use / access the browser feature.
- Global object i.e "window" has all the API as function.
  E.g. 1) window.location
  2) window.setTimeout()

Date :.....

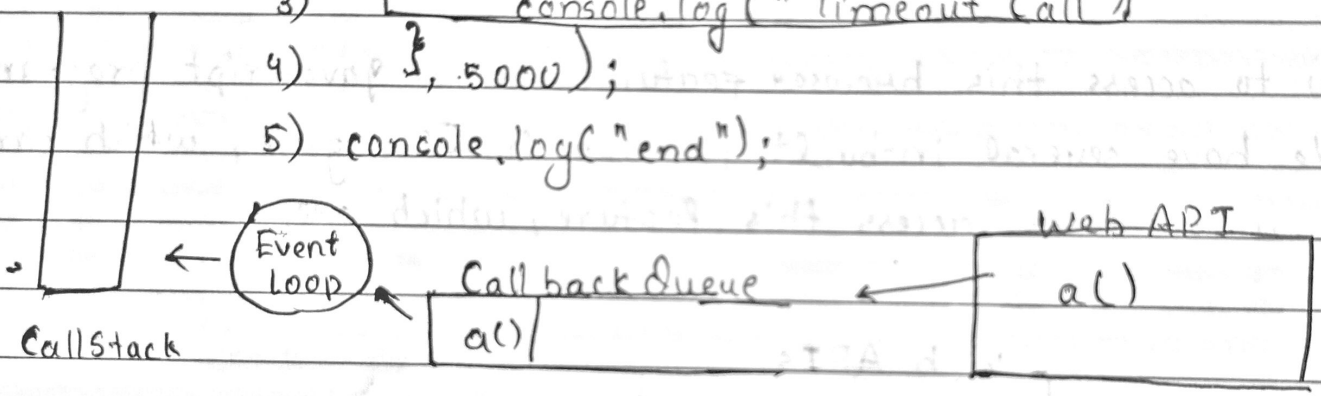
MON TUE WED THU FRI SAT SUN

→ Event loop:

Now, when there is a call back function, it first stored in Web API environment &.

E.g.

```
1) console.log("start");
2) setTimout(function a(){
3)     console.log("Timeout Call")
4) }, 5000);
5) console.log("end");
```



- In example, callback function "a" is stored in Web API when program runs.
- After, storing into Web API, program continues & it will go to next line i.e line no. 5
- Once whole program completed, & 5 seconds is done. Function a is ready to execute.
- It is moved to call back queue.
- After that, event loop continuously check call stack & callback queue.
- Once call stack is empty, it will move function a from queue to call stack to execute.

Hence, O/P:

```
start
end
Timeout Call
```

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

- Now, for promises & mutation observers, have different priority.

Promises - Call back function of fetch ()

- These function once program executing, first move to Web API env.
- But once they are ready to execute, they are moved to "Microtask Queue" which has more priority then callback Queue.
- Hence, Microtask is given priority over call back tasks.

Starvation : For an instance, if there are too many micro tasks generated, then call back queue will wait for all this to complete first, which is causing Starvation.

Conclusion :

Call Stack - It is a data structure that keeps track of function in your code.

Callback Queue - Holds tasks (callbacks/events) that are ready to execute, these tasks usually come from asynchronous operation, HTTP request, DOM request etc. AKA Task Queue

Event Loop - Responsible for continuosly checking call stack & callback queue. If call stack empty, it will move first task from queue to stack.

Date :.....
MON TUE WED THU FRI SAT SUN
☐ ☐ ☐ ☐ ☐ ☐ ☐

Microtask queue - Holds tasks that are ready to execute but has a higher priority than callback queue, tasks in here are usually Promises, mutation observers

Questions:

(1) When does the event loop actually start?
It's always running & doing its job.

(2) Are only asynchronous web API callbacks are registered in web API enviroment?
YES, call back function that pass inside map, filter, reduce are'nt registered in Web API environment, only async callback functions.

(3) Does the Web API env stores only the callback fun & pushes the same callback to queue / microtask queue?
YES

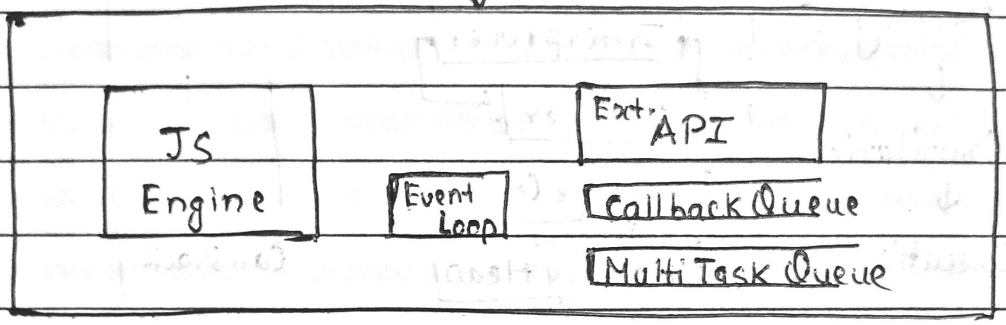
(4) How does it matter if we delay for setTimeout would be 0ms, Then callback will move to queue without any wait?
NO, Call back function need to wait until Call stack is empty.

Date :.....

MON TUE WED THU FRI SAT SUN

Javascript Architecture

Javascript Runtime Environment

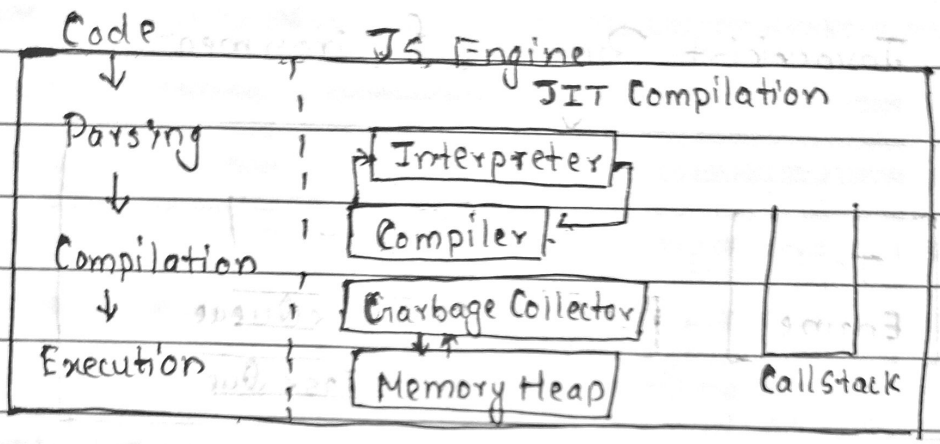


- JS runtime environment contains all elements required ~~to~~ to run JS program, i.e it contains JS engine, set of API's to connect/access outside resources, CB queue, MultiTask Queue, eventloop.
- JS engine is heart of JS runtime environment.
- Each browser / system who supports JS have its own JS engine,
  Chrome - v8 → Fastest JS Engine
  Firefox - SpiderMonkey → Oldest JS engine
  Edge - Chakora
- JS Engine is not a machine, It is a piece of code
- ~~Any~~ JS program can run anywhere, if the specific device have JS runtime env.

Date :.....

MON TUE WED THU FRI SAT SUN

→ JS Engine Architecture :



- Inside JS Engine, following process includes
  Parsing → Compilation → Execution

- Parsing breaks code into tokens & converts into AST (Abstract Syntax Tree)

- Modern JS engine follows JIT compilation, it interprets while it optimise code as much as it can, Therefore JS is both (interpreter & Compiler) language.

- Execution & Compilation are done together

- Execution has Garbage collector & other optimisation such as Inlining, copy elusion, inline caching etc.

- Garbage Collector uses Mark & Sweep algorithm into Memory Heap.