15 video

First Class Function ft. Anonymous functions

Topics to cover :- function statatement
- function Expression
- function Declaration
- Anonymous Function
- Named Function Expression
- Difference bctwcen Parameter & Argumen
- First class function
- Arrow function

(1) Function statement vs function Expression

→ function a() {
console.log("a called");
}
a();
O/P a called

→ var b = function () {
console.log("b called");
}
b();
O/P b called

→ a way to create function

→ a way to create function

→ a();
function a() {
console.log("a called")
};
O/P: a called

→ b();
var b = function () {
console.log("b called");
}
O/P: TypeError: b is not a function.

* Differ in hoisting in memory creation phase of execution context exact copy of a() will be started as () is complete function while b will a variable and hence it will undefined. If you try to run as a function throws an Error.

② Function Declaration is same as function statement.
function Declaration aka function statement

③ Anonymous Function
function without a name

```
function () {
}
```

← Invalid Syntax → Syntax Error: Requires function name.

→ Do not have its own identity.
→ It is invalid statement.
→ These are used where we want to function as value.

```
var b = function () {
}
```

← function Expression

④ Named Function Expression

```
var b = function x() {
    console.log("b called");
}
b();
```

→ b called.

x(); Reference Error: x is not defined.
because x() is created as local function only. If you try to access in out of scope it will give Reference Error.

④ Argument Vs parameter

var x = function (param1, param2) { }

label identifier are called parameter these are local value value passed are called argument

x (1, 2)

⑤ First class function

passing function inside a function as an argument.

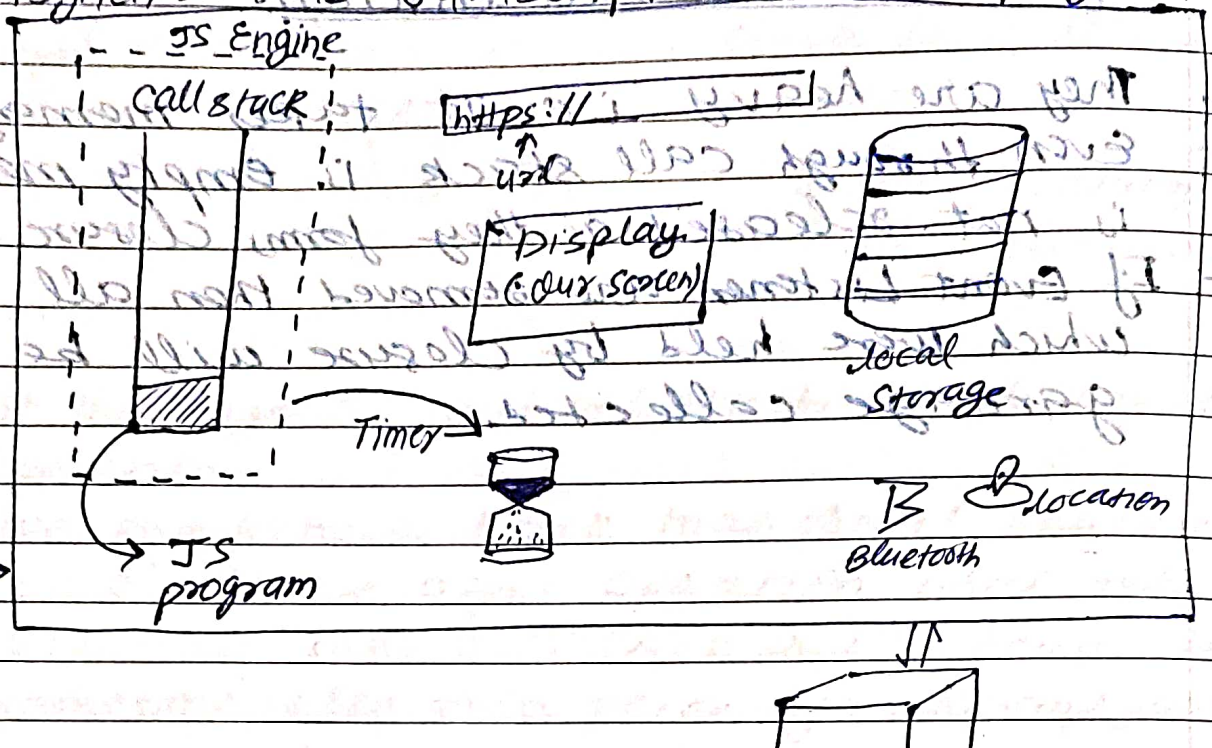
eg.:-

```
var b = function (param) {
    console.log (param);
}
function xyz () { }
b (xyz);
```

O/P function xyz () { }

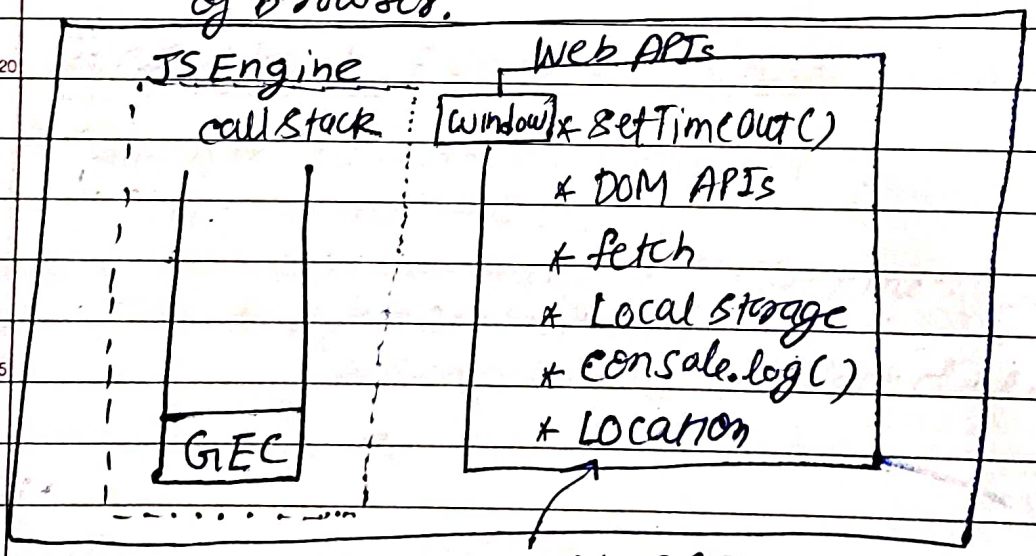
17 video

Asynchronous JavaScript & Event loop from Scratch



> The functionality related to access the website, timer, bluetooth, location etc... How can be accessed by JS Engine?

↳ we have web APIs to access all the element of browser.



And many more web APIs

Web APIs

Web APIs are part of web browser
It is not part of JS

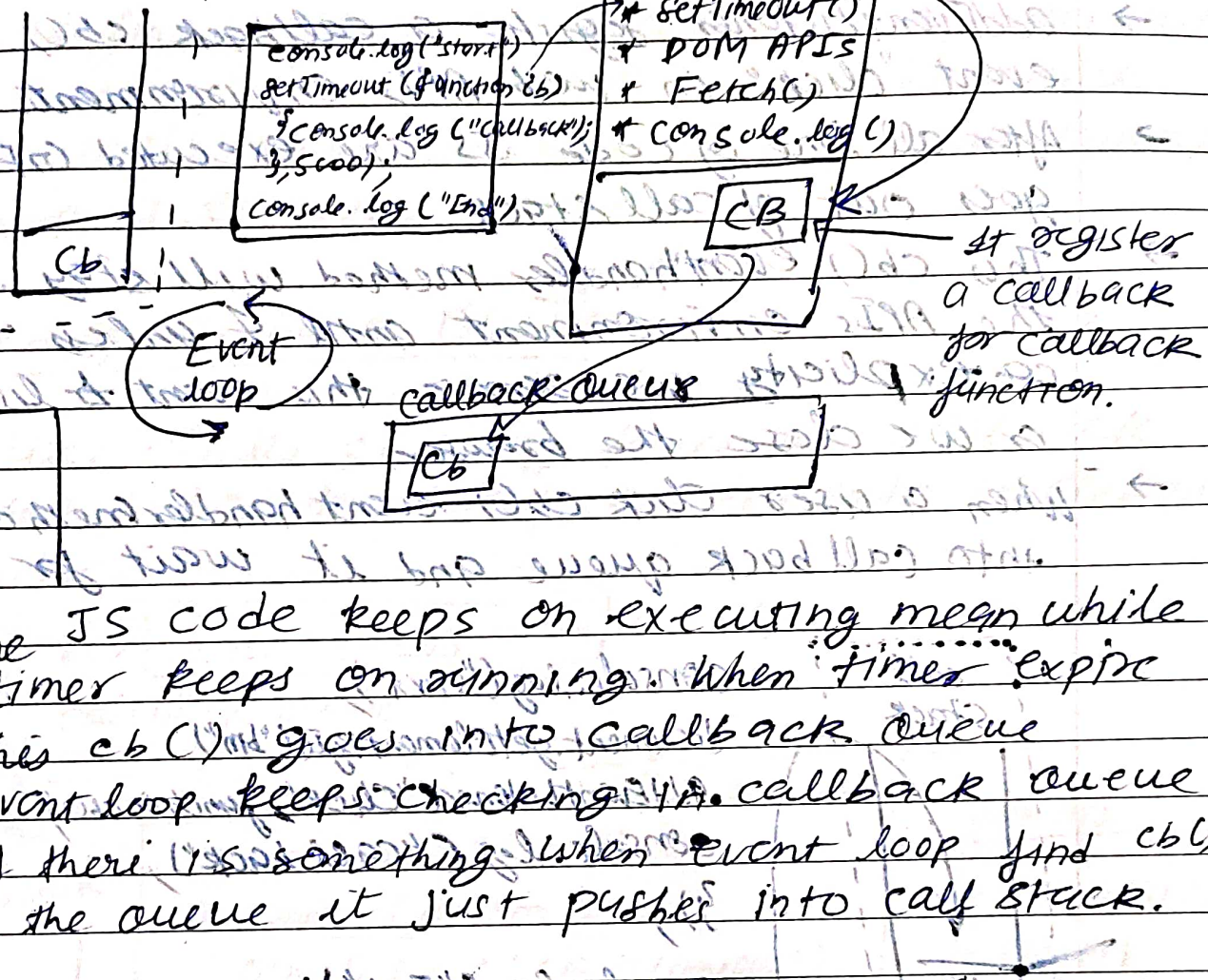
→ All web API's are for global variable.

↳ window.console.log("Me"); === console.log("Me");

↳ By default global.

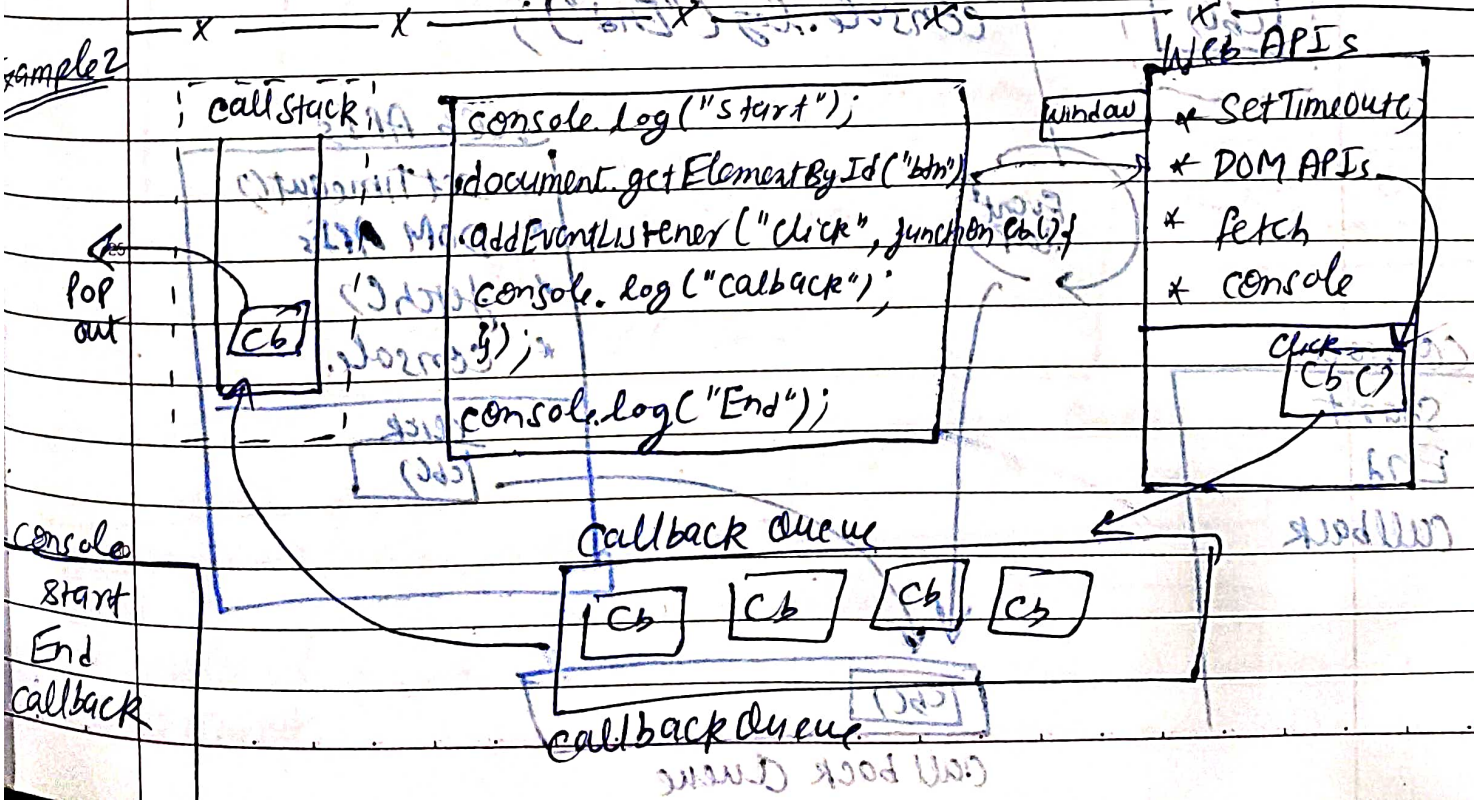
Browser wraps all its super power (like timer, &ncr access, local storage etc) into window object and makes accessible to JS engine / Call Stack through Web APIs

Example 1



→ The JS code keeps on executing meanwhile timer keeps on running. When timer expire
→ This cb() goes into callback queue
→ Event loop keeps checking in callback queue if there is something. When event loop find cb() in the queue it just pushes into call stack.

Example 2



DOM API

→ DOM is act as html source file.

→ When anyone accesses DOM API it basically access the html source file and tries to find the passed Id into DOM API.

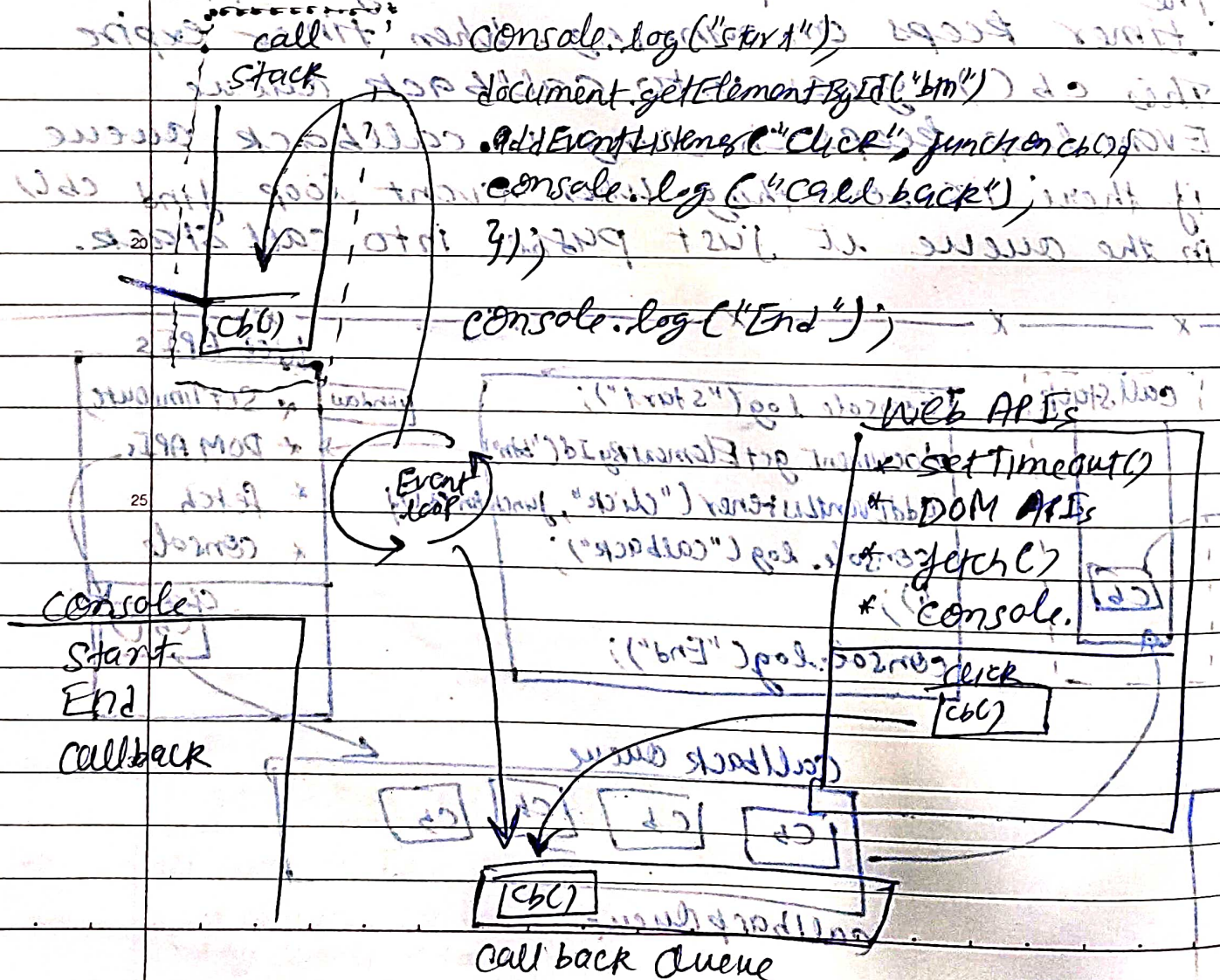
→ addEventListener registers a callback cb() on event "click" in web APIs environment.

→ After all line of code JS are executed GFC goes out of call stack

→ This cb() eventhandler method will stay in the APIs environment until & unless explicitly we remove this event listener or we close the browser.

→ When a user click cb() event handler method goes into callback queue and it wait for its turns.

```
console.log("Start");
document.getElementById("btn")
.addEventListener("click", function cb(){
console.log("callback");
});
console.log("End");
```

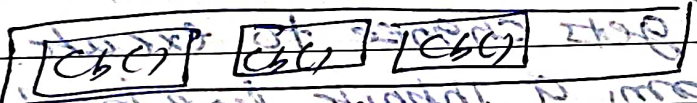


Event loop

- It has only one job
- continuously monitor the callstack & callback queue
- If it find a function in callback queue wait list it takes and pushes into call stack after call stake become empty & then callback method executed

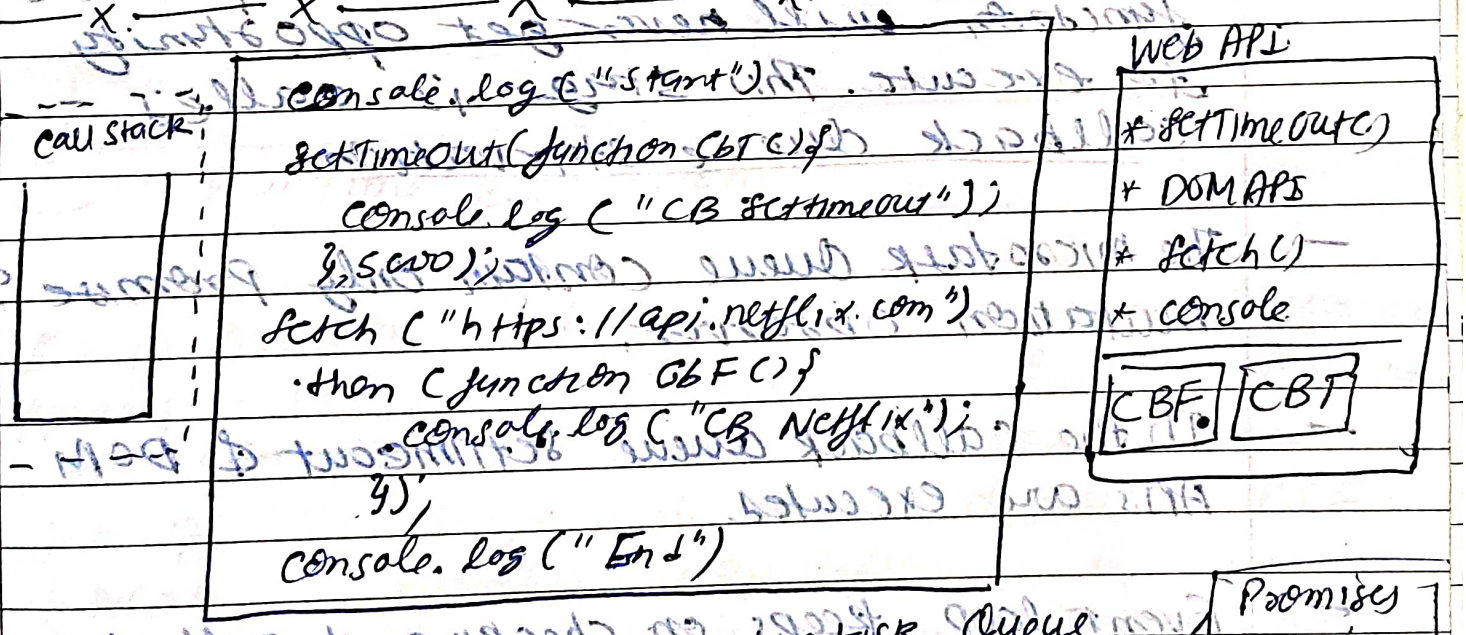
Need of callback Queue

If user clicks so many times then callback method has to executed to executed in clicked order



callback queue.

Example3



Start
End
CB Netflix
CB setTimeOut

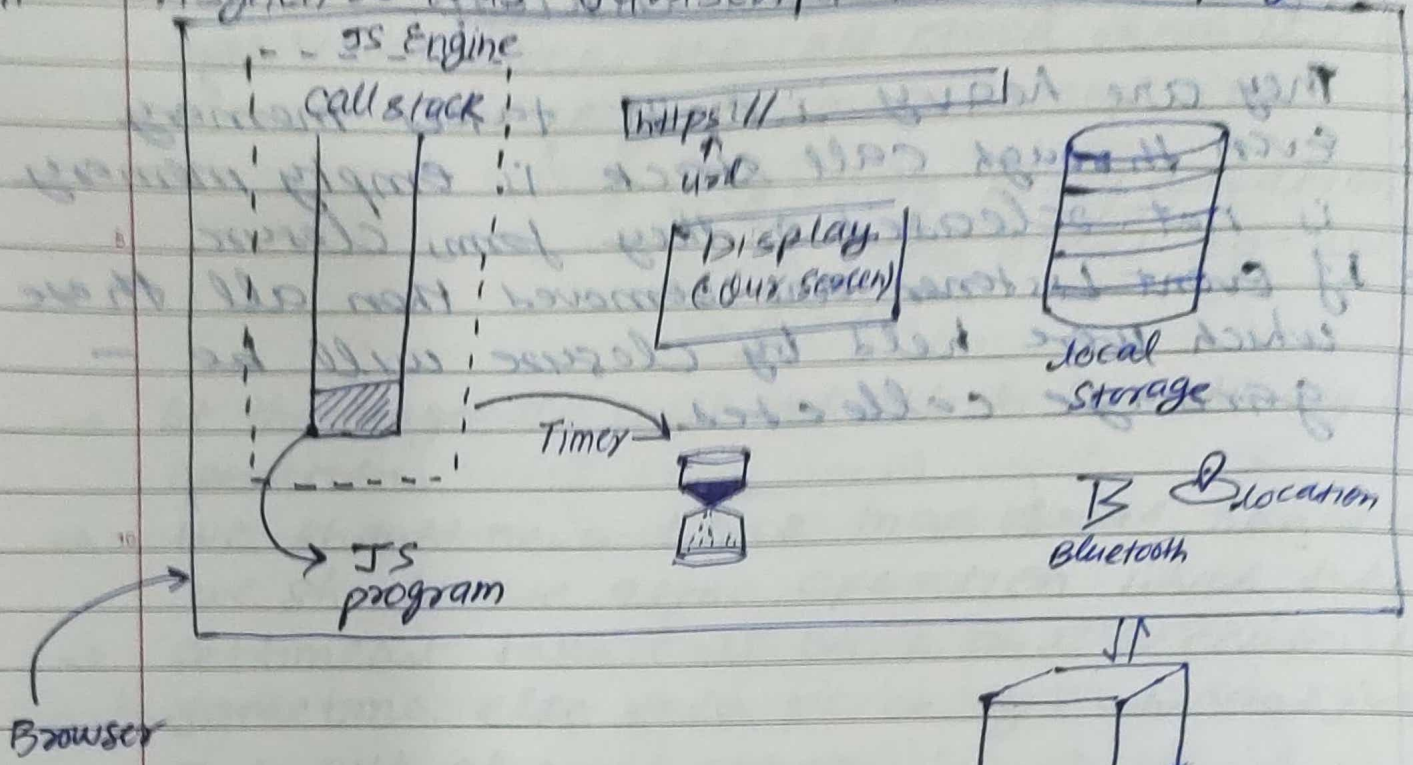
Fetch APIs : It work quite differently. fetch() method is used to make network call and request API call.
- It return a promise.
- The callback function is executed once the promise is resolved.
- JS Engine does a task at a time but browser can do multiple task at a time.

microtask Queue Vs Callback Queue

- It has higher priority than callback Queue.
- All the method of this executes then only callback queue gets chance to excute.
- If methods form a infinite loop inside microtask queue then callback / method function will never get opportunity to execute. This stage is called callback queue starvation.
- The microtask Queue contain only Promise & mutation observer.
- In the callback queue settimeout & DOM APIs are executes.
- Event loop keeps on checking if call stack is empty. If call stack is empty then only methods of microtask are pushed into call stack.
- When microtask queue become empty then only callback Queue methods are pushed into call stack

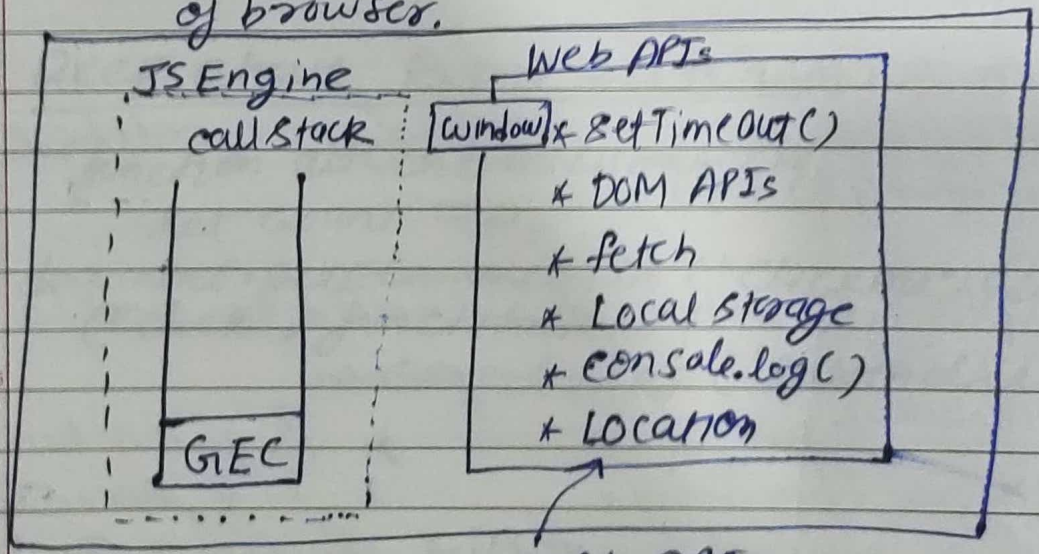
17 Video

Asynchronous JavaScript & Event loop from Scratch



> The functionality related to access the website, timer, bluetooth, location etc... How can be accessed by JS Engine?

↳ we have web APIs to access all the elements of browser.



And many more web APIs

Web APIs | Web APIs are part of web browser
It is not part of JS

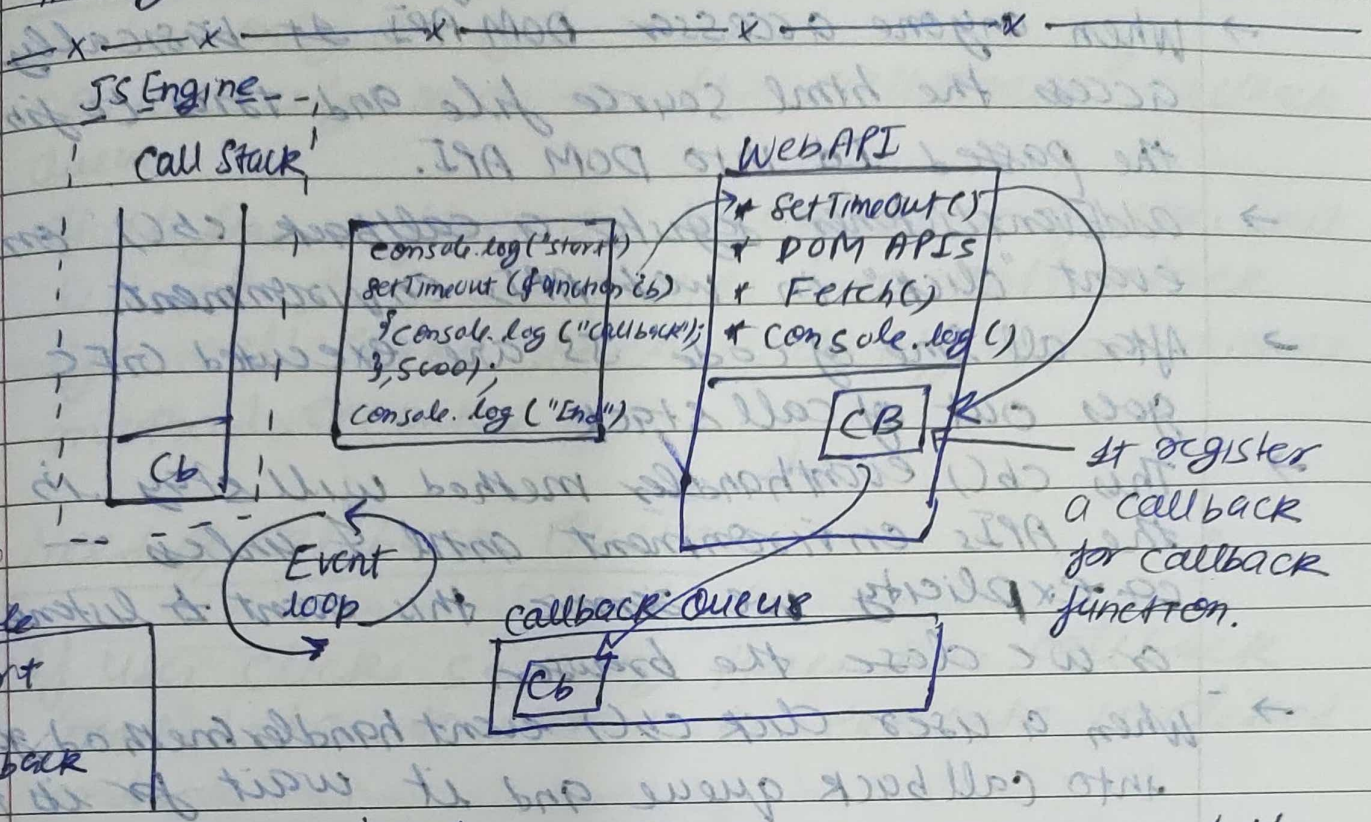
→ All web API's are for global variable.

↳ window.console.log("Me"); === console.log("Me");

↳ By default global.

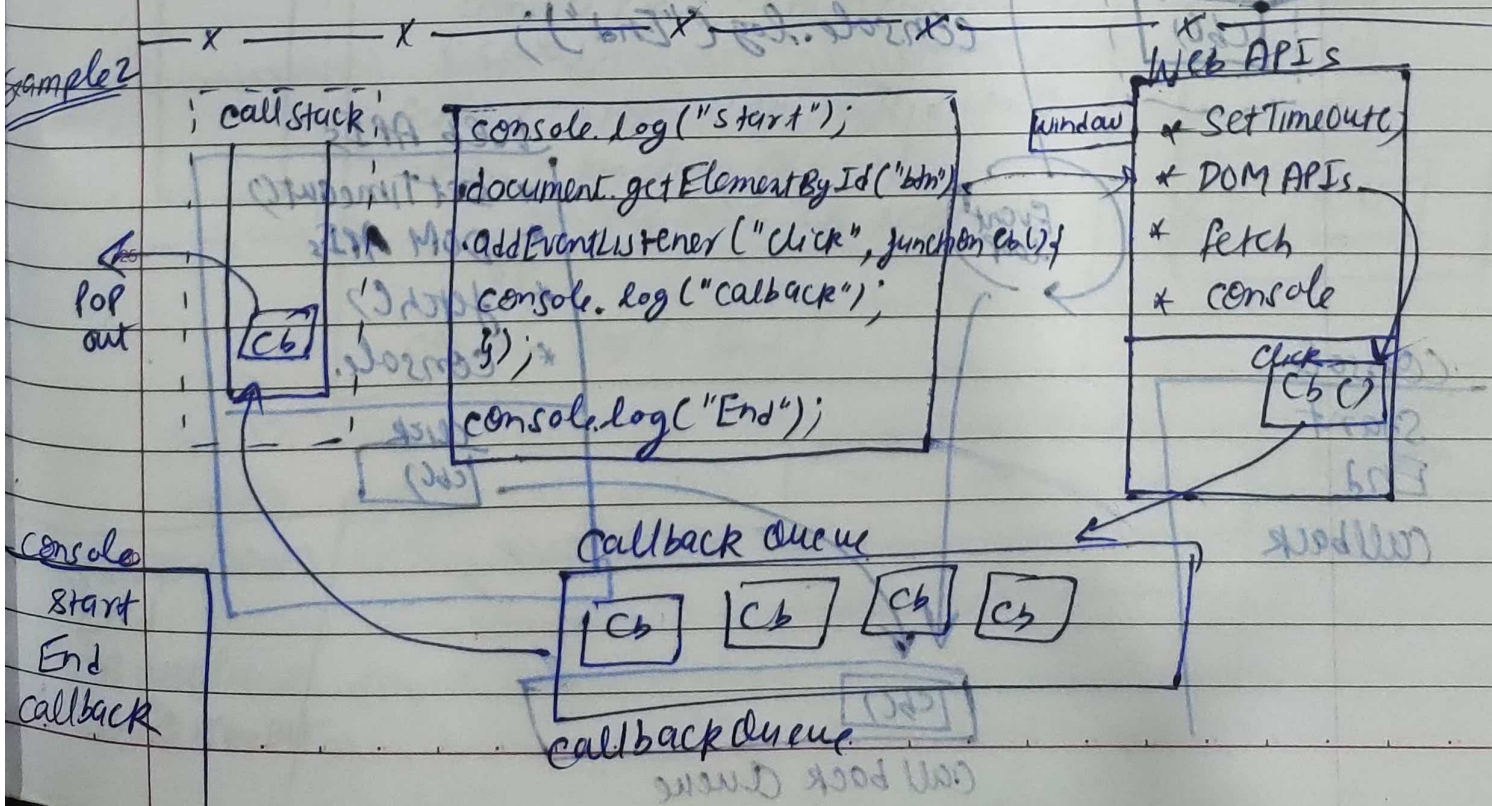
Browser wraps all its super power (like timer, &ncr access, local storage etc) into window object and makes accessible to JS engine / call stack through web APIs

Example 1



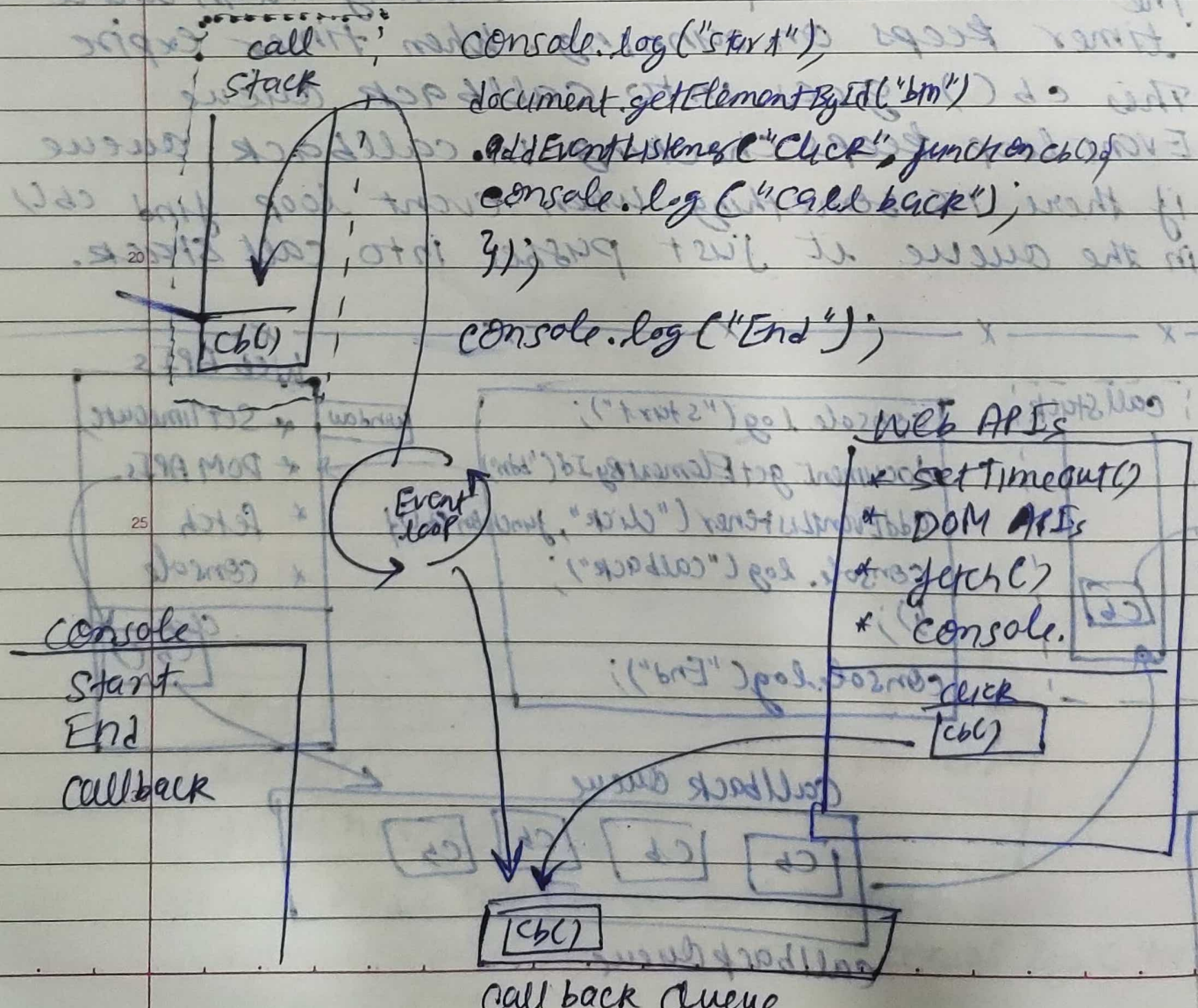
→ The JS code keeps on executing mean while timer keeps on running. When timer expire
→ This cb() goes into callback queue
→ Event loop keeps checking in callback queue if there is something when event loop find cb() in the queue it just pushes into call stack.

Example 2



DOM API

→ DOM is act as html source file.
→ When anyone accesses DOM API it basically access the html source file and tries to find the passed Id into DOM API.
→ addEventListener registers a callback cb() on event "click" in web APIs environment.
→ After all line of code JS are executed GEC goes out of call stack.
→ This cb() event handler method will stay in the APIs environment until I unless — explicitly we remove this event to listener or we close the browser.
→ When a user click cb() event handler method goes into callback queue and it wait for its turns.



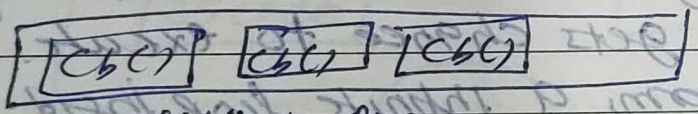
Camlin | Page
Date / /

Event loop

- It has only one job
- continuously monitor the callstack & callback queue
- If it find a function in callback queue wait list it takes and pushes into call stack after call stake become empty & then callback method executed

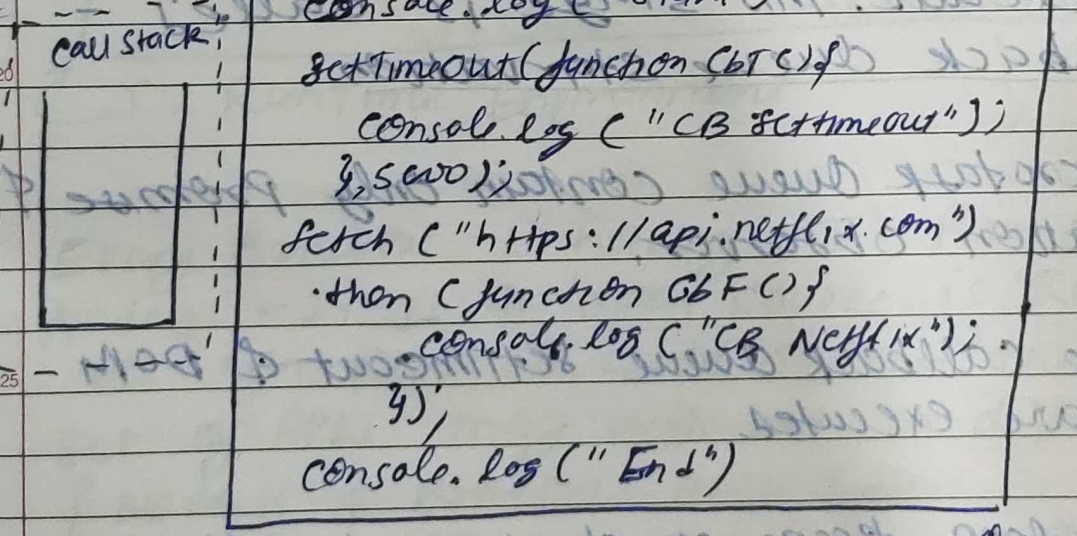
Need of callback Queue

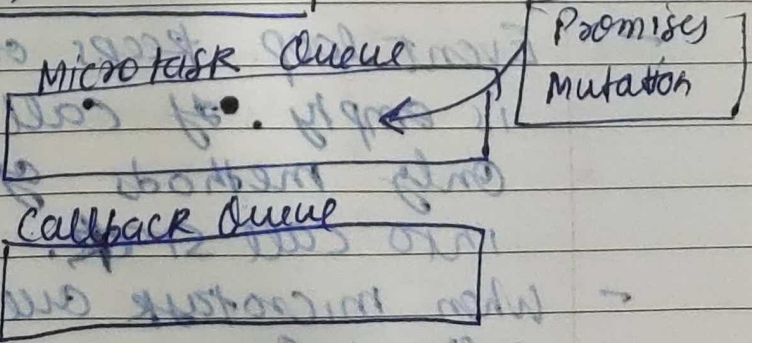
If user clicks so many times then callback method has to stored to execute in clikes order



callback queue.

Example3





Start
End
CB Netflix
CB setTimeout

Fetch APIs: It work quite differently. fetch() method is used to make network call and request API call.
- It return a promise.
- The callback function is executed once the promise is resolved.
- JS Engine does a task at a time but browser can do multiple task at a time.

Microtask Queue Vs Callback Queue

- It has higher priority than Callback Queue.
- All the method of this executed then only callback queue gets chance to excute.
- If methods form a infinite loop inside microtask queue then callback / method function will never get opportunity to execute. This stage is called - Callback Queue starvation.

- The microtask Queue contain only promise & mutation observer.
- In the callback queue setTimeout & DOM APIs are executes.

- Event loop keeps on checking if call stack is empty. If call stack is empty then only methods of microtask are pushed into call stack.
- When microtask queue become empty then only callback Queue methods are pushed into call stack.