Date / /

" Everything in Javascript happens inside a an Execution Context "

→ It has two components.

1. Memory Component
   This is the place where all variables and functions are stored as a key value pairs.
   - Memory component also known as Variable Environment.

2. Code component
   This is the place where code is executed one line at a time.
   - This is also known as Thread of Execution.

Memory	code
key : value a : 10 fn : { ... }	

- two phases
  → Memory creation
  → Code execution Phase

Note :

" Javascript is a synchronous single threaded language. "

→ It means javascript can only execute one command at a time

Synchronous single thread means ∴ JS can only execute one command at a time and in a specific order.

undefined is a keyword.

* Function invocation :

Whenever we see a function name with the parenthesis, it means now the function is executed.

Functions are the heart of JavaScript.

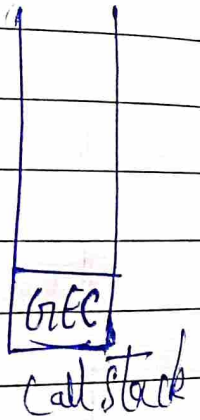
In JS, functions behaves very diffently differently than any other language.

* Callstack

Callstack handles everything to manage the execution context creation, deletion and the control, it manages the stack which is also known as Callstacks.

Call stack is a stack

- Everytime in the bottom of the stack we have our global execution context



"Call stack maintains the order of execution of execution contexts"

→ Callstack also known as

1. Execution Context stack
2. Program stack
3. Control stack
4. Runtime Stack
5. Machine Stack

* Hoisting in Javascript

Hoisting is a phenomenon in JS by which we can access variables and functions even before initilisation or put have some value in it without any error.

* "Even before the code starts executing, memory is allocated to all variables and functions"

⇒ When the variable (let's x) is not present everywhere in the memory or not initilised anywhere in the program. then it throws an error which is "not-defined"

* Undefined:

It is like a place holder which is placed inside a memory.

- It takes the memory.

It is a special keyword. It takes up own memory like a place holder which is kept for time being until the variable is assigned for some other value.

- JS is a loosely typed language.

* Scope:

means where we can access a special specific variable or a function in our code.

Scope is directly depend on the lexical environment.

Date / /

* Lexical environment
"Whenever an execution context is created a lexical Environment also created."

→ lexical env. is the local memory along with the lexical env. of its parent.

→ LE is the local memory and reference to lexical env. of its parent.

* let & const
"let & const declarations are Hoisted"
"These are in the temporal deadzone for the time being."

temporal deadzone is the time since when let variable was hoisted and till it is initilized some value the time b/w that is Known as TD

When we tried to access a variable in the temperal deadzone, it gives a Reference Error

* Window
Window is actually a global object which is created along with the global execution context.

Date / /

=> So whenever a javaScript program is run, a global object is created, a global execution context is created and along with execution context a this variable is created

this === window

=> Global space is a space where we write any code in JS which is not inside a function,

Anything which is not inside a function that is global space.

* Block scope

Block is also known as Compound Statement.

- Block is used to combine multiple JavaScript statements into one group.

=> We group multiple statements together in a block so that we can use it where javascript expects one statement. <=

=> Block scope : what all variables and functions we can access inside this block.

=> Var is a global scope,

let & const are block scope that means they are stored in a separate memory space which is reserved for this block.

* Shadowing

* Closures

"A function binds together with its lexical environment"

Function along with its lexical scope forms a closure.

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

var z = x();
console.log(z);
.
.
.
z();
```

A closure is a function that remembers its outer variables and can access them.

- All functions in javascript are closures.

- Function bundled with its lexical environment is known as closure. Whenever function is returned, even if it is vanished in execution context but it still remembers the reference it was pointing to. Its not just that function along it returns but the entire closure and that is where it becomes interesting

Date / /

> Uses of Closures
- Module Design Pattern
- Currying
- Function like once
- Memoize
- maintaining state in async world
- setTimeouts
- Iterators
- and many more..

```
function outer(b) {

    function inner() {
      console.log(a, b);
    }
    let a = 10;
    return inner
}
var close = outer("Hello");
close();
```

Inner function forms a closure with its outer environment and b. as also part of a outer environment of inner function, so it forms closure with b also.

⇒ closures helps in datahiding and encapsulatio

→ Disadvantages of Closures

1. Over Memory Consumption :
Because every time closures are found, so it consumes lot of memory and sometimes on these closed variables are not garbage collected. So that means it is kind of accumulating lot of memory if we create lot of closures in our programs. Because those variables are not garbage collected till the program expires.

2) It can also lead to memory leaks.

* Garbage Collector :
Garbage collector is a program in the browser or the JS Engine which kind of freeze unutilised memory.

* Functions

→ Function statement
```
function a() {
    console.log("Hello");
}
```
The function is normally defined and can be used later on.
By this we can do Hoisting
```
b();    // error
```

→ Function Expression
```
var b = function() {
    console.log("World");
}
```

→ Disadvantages of Closures

1. Over Memory Consumption:
Because every time closures are formed, so it consumes lot of memory and sometimes on these closed variables are not garbage collected. So that means it is kind of accumulating lot of memory if we create lot of closures in our programs. Because those variables are not garbage collected till the program expires.

2) It can also lead to memory leaks.

* Garbage Collector:
Garbage collector is a program in the browser or the JS Engine which kind of freeze unutilised memory

* Functions

→ Function statement
```
function a() {
    console.log("Hello");
}
```
The function is normally defined and can be used later on.
By this we can do Hoisting

```
b();    //error
```

→ Function Expression
```
var b = function() {
    console.log("World");
}
```

Page No.
Date / /

function expression simply means to create a function and put it into the variable.

//// Diff b/w function statement and function Expression.

A function statement loads before any any code is executed. This behaviour of function statement is called hoisting, which allows a function to be used before it is defined.

A function expression associates a value with a variable, just like any other assignment statement

⊛ Function statement also known as function declaration

→ Anonymous function
Function without a name is known as Anonymous function.
- Anonymous function does not have their identity.

```
function () {

}
```
// This will show error cuz require name

Used: Anonymous function are used in a place where functions are used as value.

→ Named function:
A normal function with its name assigned to a variable.
// In this case we can't call function by its name in outer scope."

```
var b = function xyz() {
    console.log("Hii Sachin");
}
```
xyz → not error

```
xyz(); // It will throw error
```

→ Parameters & Arguments

```
var b = function(par1, par2) {
    console.log("Unknown");
}

b(1, 2);
```
parameters

Arguments

→ First class function or First class citizens
The ability of use function as value
- Can be used as an Argument
- Can be executed inside a closure function &
- Can be taken as return form.

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

Page No.

Date / /

Functions are heart of JS. They are called first class citizens but first class functions because they have the ability to be stored in the variables, passed as parameters and arguments. They can also be returned in the function.

* What is callback function

function that is passed on as argument to another function is called callback function. It gives us access to the whole asyn. world in a sync. single threaded language

- setTimeout helps turns JS which is single threaded and synchronous into asynchronous.
- Event listeners can also invoke closures with scope.
- Event listeners consumes a lot of memory which can potentially slowdown the websites therefore it is good practice to remove if it is not used.

* Event loop

JavaScript is a single synchronous threaded language. It has one call stack and it can do only one thing at a time.

This callstack is present inside javascript engine and all the codes are executed in the callstack

Web APIs

callback queue

Page No.
Date

⇒ setTimeout is not a part of JavaScript
DOM APIs — local storage
console — fetch()
Location

~~To keep checking whether the callstack is empty or not~~

* Event loop

Event loop is an endless loop, which waits for tasks, executes them and then sleeps until it receives more tasks.

The event loop executes tasks from the event queque only when the callstack is empty

The event loop allows us to use callbacks and promises.

→ Callback queue :

Task Queue

Callback queue gets the ordinary callback functions coming from the setTimeout() API after the timer expires.

→ Fetch : basically goes and requests a API call.

→ Microtask Queue

All the callback functions which comes through Promises and Mutation Observer would go inside the microtask queue.

Callback queue has lesser priority than Microtask queue.

Mutation Observer interface provides the ability to watch for changes being made to the DOM tree.

Page No.
Date / /

* Starvation :

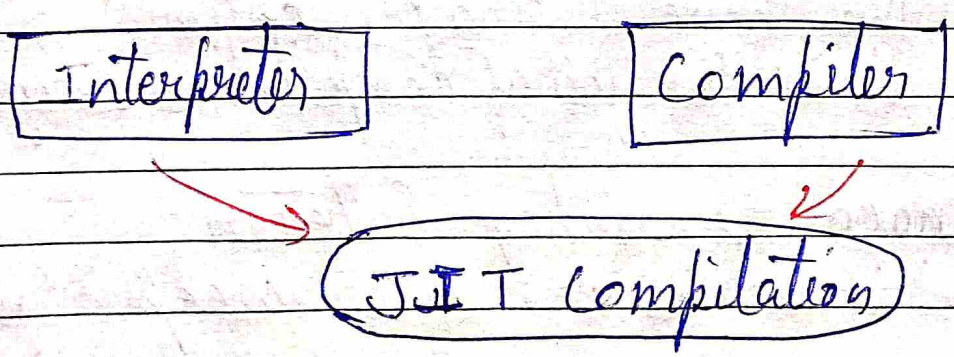
Starvation is usually caused by an overly simplistic scheduling algorithm.
Ex: If a multi tasking system always switches between the first two tasks while a third never gets to run, then the third task is being starved of CPU time.

* Asynchronous :

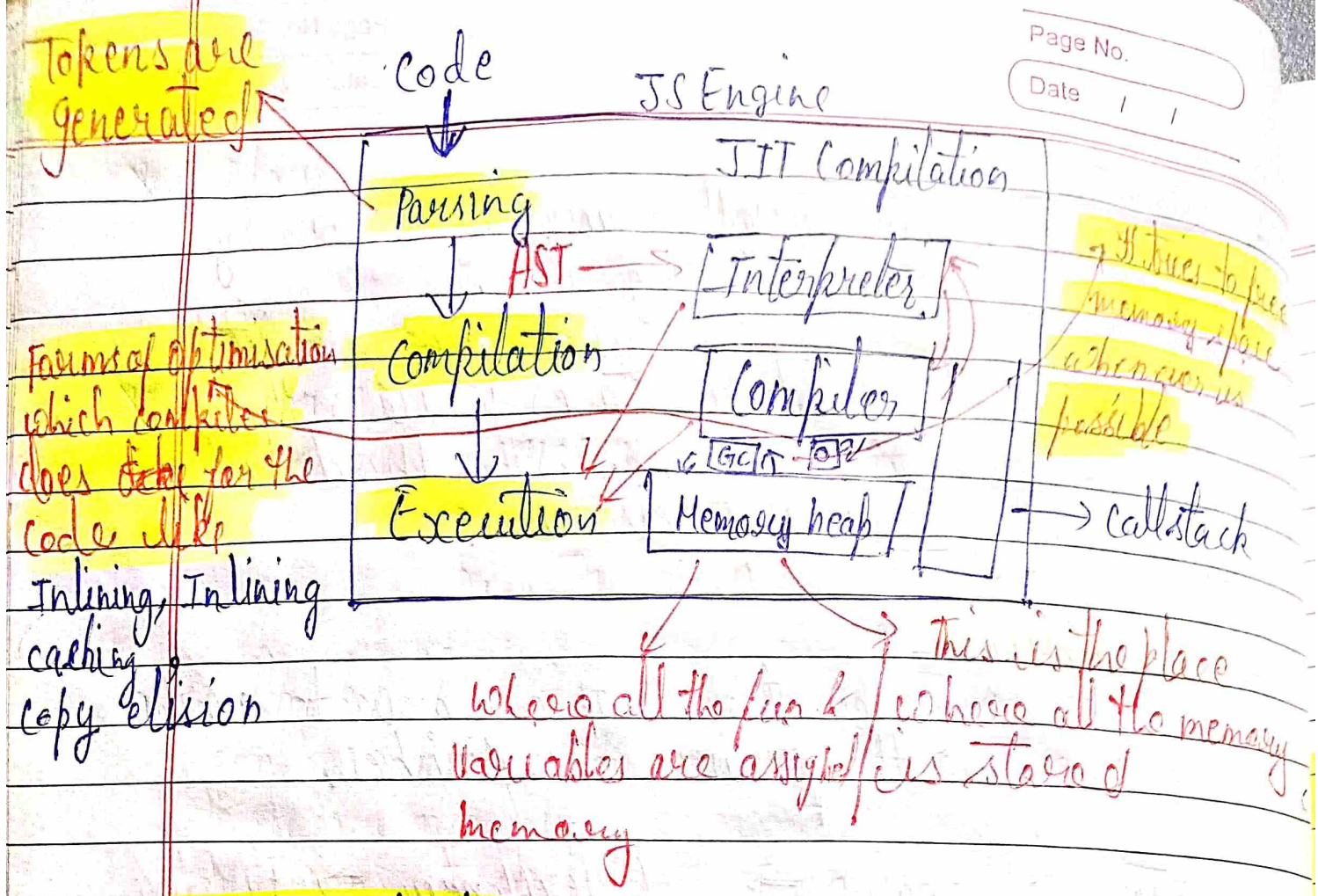
In Asy. operations we can move to another task before the previous one finishes.

⇒ ~~Fi~~ First JS Engine ⇒ SpiderMonkey ⇒ Firefox

⇒ JS Engine is not a machine.
→ It is just like a normal program like a normal code which is written in low level language.
For example ⇒ Google V8 engine is written in C++



It uses both interpreter and compiler to execute the code



⇒ Garbage Collector ⇒ whenever no one function is not being used or we clear the timeout so basically it collects the all garbage and sweeps it.
It uses Mark & Sweep Algorithm

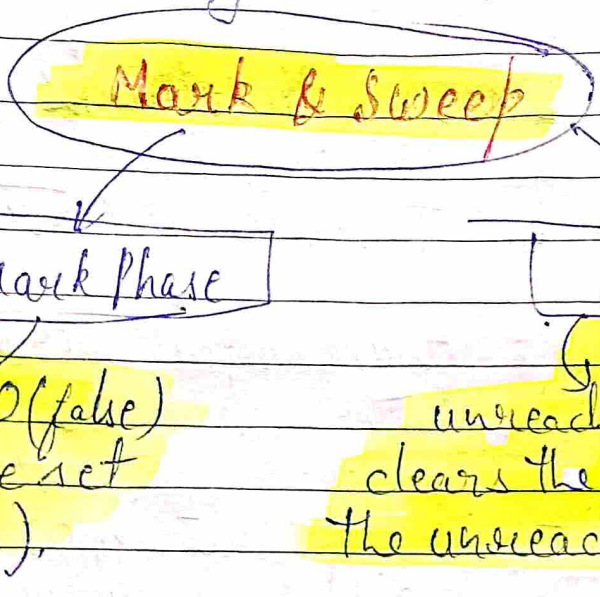
V8 ⇒ has a interpreter known as Ignition
They have turbo fan optimising compiler which does the job of compilation very fast

Orinoco ⇒ Garbage Collector
↳ uses mark & sweep algorithm

JS is JIT compiled language.

Page No.
Date / /

© Mark and Sweep algorithm is called a tracing garbage collector because it traces out the entire collection of objects that are directly or indirectly accessible by the program.



* Higher order functions :
A function which takes another function as an argument or returns a function from it is known as high order function.

* Prototype
Whenever we create any object, JS engine automatically puts this hidden property into an object and attaches an object to the orginal object.

- The value of proto can be either an object or null.

* Class
In JS, class is a type of functions.

* Callbacks
1. Issues with callbacks
   a. Callback hell
   b. Inversion of control

a. Callback hell: Asynchronous operations in JS. can be achieved through callbacks. Whenever there are multiple dependent Asynchronous operations it will result in a lot of nested callbacks. This will cause a 'pyramid of doom' like structure.

b. Inversion of control: When we give the control of callbacks being called to some other API, this may create a lot of issues. That API may be buggy, written by interns, may not call our callback and create order as in the example may call the payment call twice.

* Promises

Javascript is a single threaded, meaning that two bits of script cannot run at the same time. They have to run one after another.

A promise is an object that represents the eventual completion (or failure) of an asynchronous operation, and its resulting value.

Date / /

* Async / Await

Async make a function return a promise.
await make a function wait for a promise.

⇒ the await keyword can only be used inside an async function.
⇒ async keyword can be placed before a function like this.

- So, async ensures that the function returns a promise and wraps non-promises in it. &.

- The keyword await makes Javascript wait until that promises settles and returns its result.

* API (Application Programming Interface)
AJAX (Asynchronous Javascript And XML)
JSON (Javascript Object Notation)
↳ text format for storing and transporting data.
: Commonly used for API and Config files

⇒ Why use JSON?. JSON is language independent.
⇒ A javascript program can easily convert JSON data into JavaScript objects.

- A common use of JSON is to exchange data to/from a web server.
- When receiving data from a web server, the data is always a string.
- Parse the data with JSON.parse(), and the data becomes a Javascript object.