Libraries and frameworks are diff
Frameworks are more like a superset.

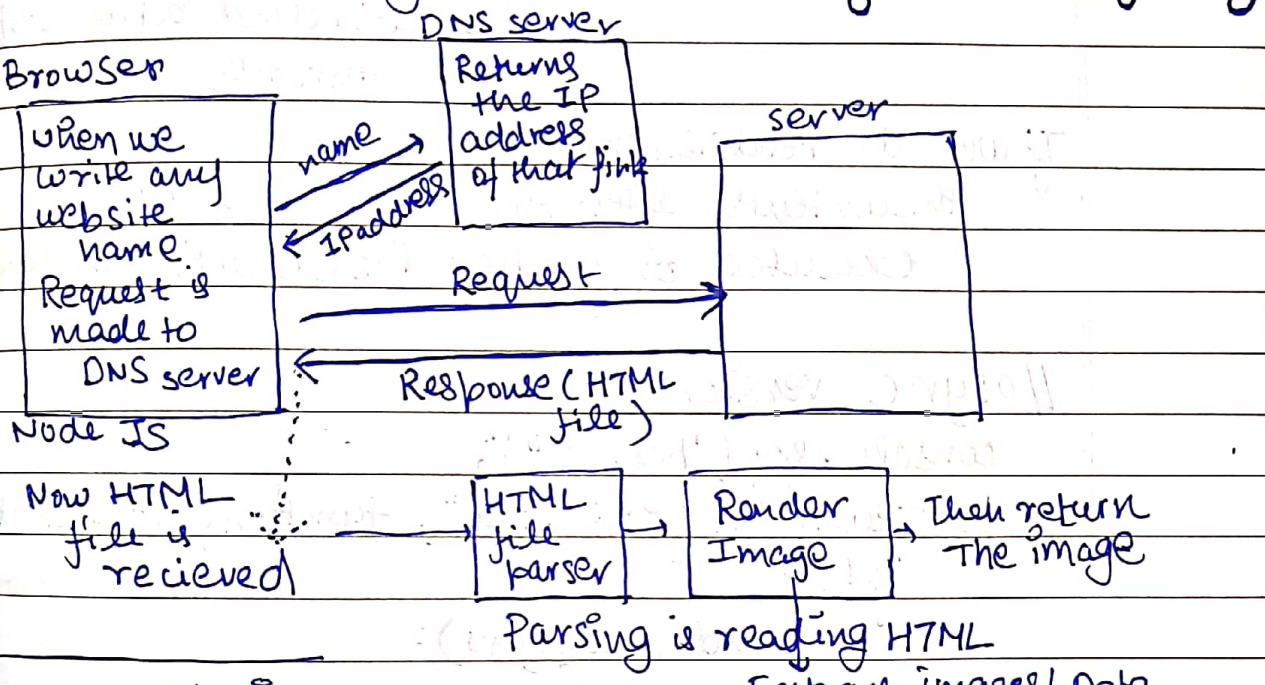
classmate Date Page

Projects Webscrape 1

① what is webscraping → Technique to collect data from raw/unstructured data and convert it into structured data.
② Requirements
③ Current Activity

Used mostly in Data Engineering

Process



Parsing is reading HTML
Extract images/ Data after reading HTML

Language → Logic
Frameworks → Implementation

Tasks
→ request | We have request module for that
→ Read/Parse HTML and extract Data | use inspect feature to specify tags
→ Save data.
↳ Excel (xlsv)
→ Files/Folder → fs module

Cheerio module

Uses case
Selecting location for pep coding nearest to industry/students.

CPU > FS > Network

Main thread - [1] [2] [3] [4]

the work for CPU is given to main thread

Rest of the work can be switched between other threads

If we do readFileSync the contents after the command are executed only after the execution is done

```
//async version
console.log("before");
fs.readFile("f1.txt", function(err, content){
    if(err)
        console.log(err);
    else
        console.log("content "+content);
});
console.log("After");
```

Output

Before
After
content : ....

async - first come first serve

if you want to write in the file read write the code in callback once you are sure that read is called (else part in above code)

npm install request

```
let request = require("request");
let url = "any url";
console.log("before");
request(url, cb);
function cb(error, response, html){
    if(error)
        console.log(error);
    else if(response && response.statusCode==404)
        console.log("Page not found")
    else
        extractData(html);
}
```

Selectors - passed into cheerio and gives data matching to it present in ~~selector~~ HTML data
- majorly according to CSS

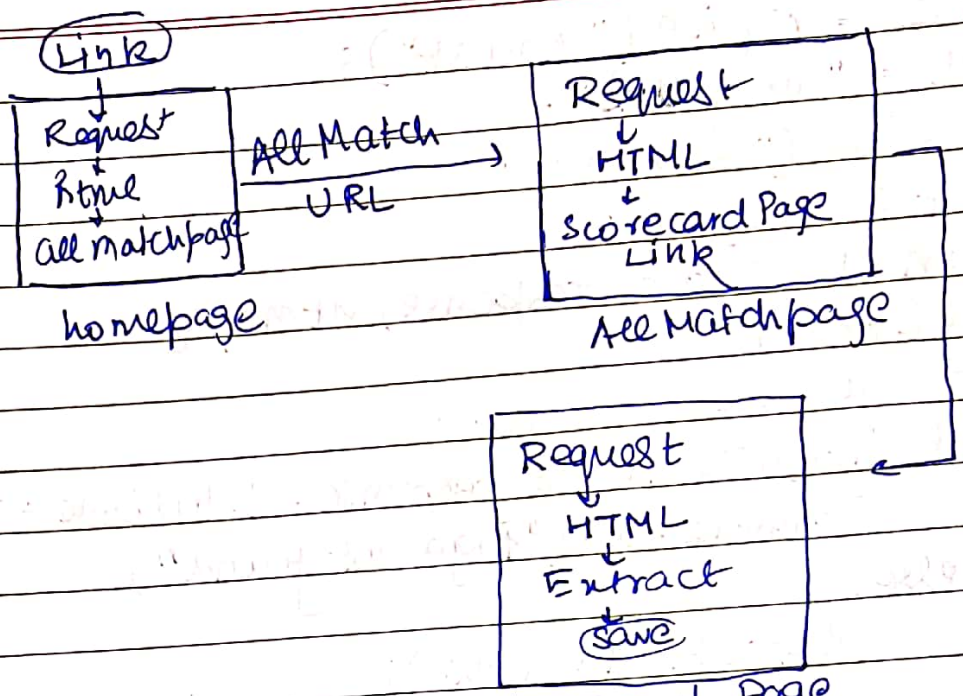
npm i cheerio

```
function extractData(html){
    let $ = cheerio.load(html);
    let elem = $("selector")
}
```

function is stored in $

cheerio.load parses the html and returns a function that is used to select elements from that html page using CSS selectors.

If unique element element is returned else array of elements is returned.




```
function extract-link(html)
{
    let $ = cheerio.load(html);
    // cheerio => access index wrap
    let scorecardElems = $("a[data-hover='Scorecard']");
    for(let i=0; i<scorecardElems.length; i++){
        let link = $(scorecardElems[i]).attr("href");
        let fullLink = "https://..." + link
    }
}
```

we have two .js files

first

```
let a = 10;
function fn(){
}
function fn1(){
}
module.exports = {
    varname : a
    fnFunction : fn
}
```

second

```
let obj = require("./first")
obj.a → undefined
obj.fn()
obj.varname
```

Video 17

npm init -y

Video's to watch

(1) Practical Intro to JS
(2) Practical Intro to NodeJS
(3) What Actually is JavaScript

Video - 11-16 (Interview questions)

(4) Practical Intro to web scraping
(5) Insights - video 18? (on selectors)

Video - 19-20 (Actual Project)

Cricinfo WebScrapper

1) Build a web scraper that scrapers details of BatsMan of each team that played in a ~~worldcup~~ IPL

2) Request package was used to make parallel requests to the server and cheerio is used to extract data out of it

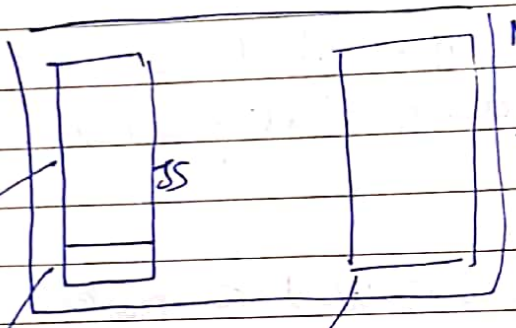
3) Node js fs module was used to create directories of each ~~item~~ team and ~~xlsx~~ module is used to create excel file for each Player

request - to learn callbacks and async functions only although depreciated

Parallel requests
- settime out
- sync / async
- Parallel

→ callbacks

Time intensive things like Network, FRW, Database, those functions are async functions mostly.

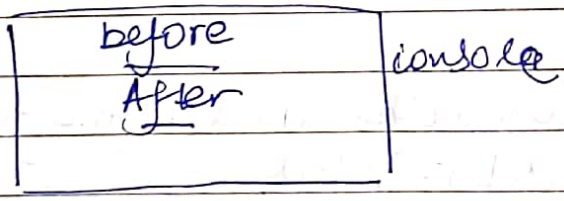


→ Simple code is executed/run in JS while features are provided by framework

if main is not there

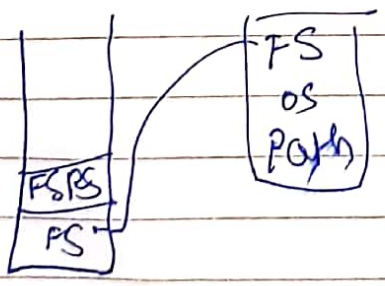
Execution context is made where everything is stored (similar to fnc call in java)

serial - fs.readfilesync
async - fs.readfile

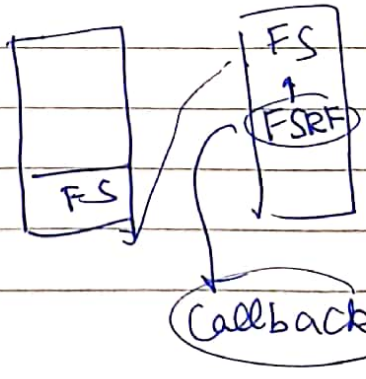


JavaScript stack

Exact word **Execution stack** where whole code is executed



In async API has the functions



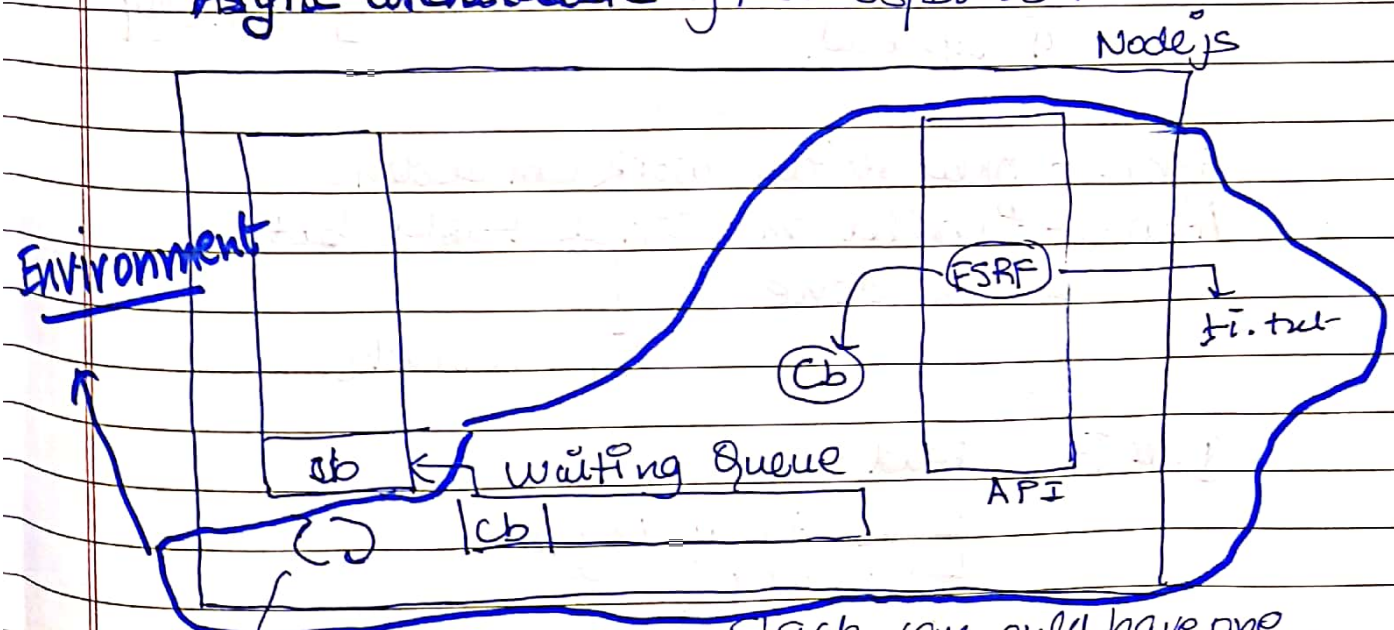
call is done in stack but run in API section

Facts - You cannot install JavaScript
- js is always wrapped inside an environment
- js doesn't have main → global execution context
- execute line by line
- async functions cannot be created/imported from the framework
- In nodejs an async function that cannot block the main execution/stack

Real life example
If three people are under a person
so you can say three ppl come under API section and the person's tasks come in stack

- callback executes in the stack only once the stack is empty

Async architecture of NodeJs/Browser



Sync and Async - Functions

↳ functions executed at once

function is called on the stack and then its implementation is done on APIs

Part of function is executed immediately (now) rest of the function is executed later (later)

call

later

callback

Parallel and serial - Tasks

Serial - called one by one

parallel - all made at once

↳ Independent tasks

Dependent tasks

① Download video from zoom
② Cut
③ Compress
④ Upload

Example can only be done seriallly

Sync - only serial work can be done

Async - Parallel and serial tasks both can be done

↓

in callback.

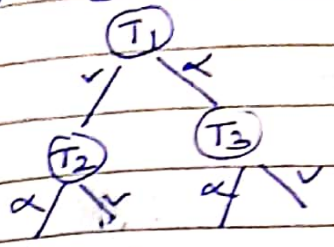
① def - Files Read

↳ seriallly (Async)
↳ parallelly

② n - files Read

↳ seriallly
↳ paralelly } → Async

③ Conditional



① Definate Files

serial using sync function

```
let fs = require("fs");
let content = fs.readFileSync("./f1.txt");
console.log("" + content);
```

Repeat for f2 and f3

serial using async function

```
fs.readFile("f1.txt", function
(err, data){
    if(err)
        console.log(err)
    else
        console.log("" + data)
```

Repeat for f2, f3

serial async (parallel basically)

```
fs.readfile("../f1.txt", function(err, data){
  if (err)
      console.log(err)
  else {
      console.log("" + data);
      fs.readFile("../f2.txt", function(err, data){
          if (err)
              console.log(err)
          else{
              console.log("" + data)
              fs.readFile("../f3.txt", function(err,
                                                data){
```

To improve this give function names

```
function cb1 = fs.readFile(fi.txt, cb)
```

define function normally like in Java/C++

ntasks

serial sync

```
let arr = ["../f1.txt", "../f2.txt", ....];
for( let i=0; i<arr.length; i++){
    content = fs.readFileSync(arr[i]);
    console.log("" + content);
}
```

parallel async

```
function cb(err, data){
    if(err){
        console.log(err)
    else
        console.log("" + data);
}
```

function is of reference type that's why only 1 is required functions are made in heap address is given in cb basically where the function is stored

Asyn serial

```
for(int i=0; i<arr.length; i++)
{
    fs.readFile( .. )
    callback{
    }  i++;
}
```

→ infinite loop stack is never empty

Actual function

```
function nFileReader(n){
    if(n == arr.length)
        return;
    fs.readFile(arr[n], function cb(err, data){
        if(err)
            console.log(err);
        else{
            console.log("" + data)
            nFileReader(n+1);
        }
    });
}
```

deadlock between callback and function call stack never empty.

Closure

Although stack is empty but the function has access to all the variables in it

Suppose, function a( ){

```
    return function b(){
    }
}
let abc = a();
```

although function call has been removed from stack but abc as inner function has access to the variables of the outer function

execution

set time out

- asynchronous function provided by environment

```
function fn(){
    console.log("hello");
}
setTimeout(fn, 3000);
```

fnc will wait for 3 seconds in API section

Javascript

Functions are first class citizens
- can be treated as a variable
  func defination can be put in a variable

```
let fncontainer=function (){
};
fncontainer() → anonymus function
```

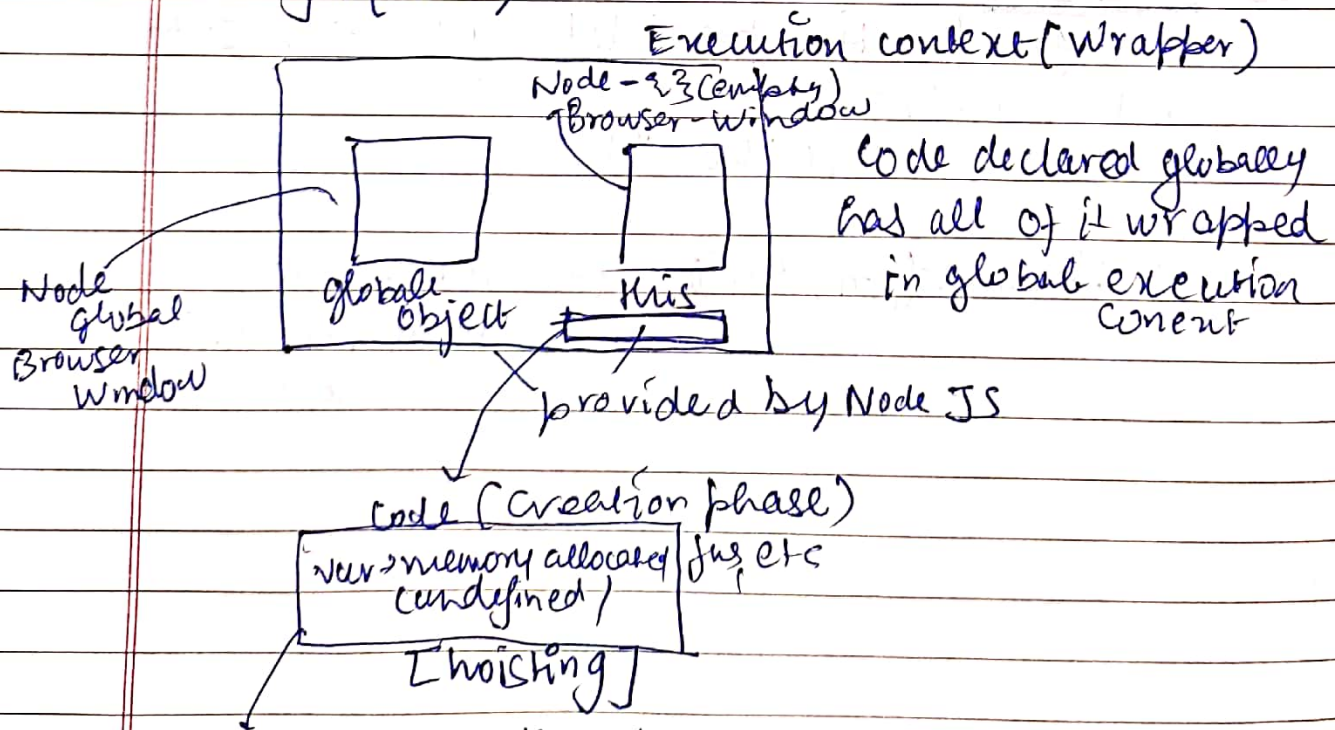
→ IIFE

```
function fn(){
}
```
] runs by its own to not do so use

```
(function(){
})();
```

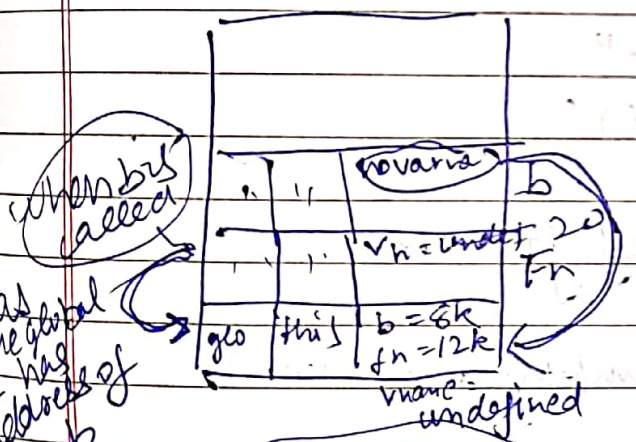
// Arrow function → syntax, react, this

```
let fn= num => num* num;
fn(num)
```



Scope - area where function / variable is found.

```
console.log("line no 1", var)
let var = 10
function b(){
    console.log("line number 5", var)
}
console.log("line no 7", var);
function fn(){
    console.log("line number 9", var);
    var varName = 20;
    b()
    console.log("line no 13", varname);
}
fn();
```



Lexical Scope → area when a variable is not defined in that scope

```
Var varname;
console.log(varnames); → undefined
varname = 20;
console.log(varname); → 20
var varname;
console.log(varname); → 20
```

as varname is already declared will not run again

~~let varName;~~

Let and const

Temporal dead zone

// code

//

console.log(varName)

let varName;

Block scope

{} → block

if (--) {

//blocks

}

Keyword	Scope	Reassign	Redeclare	TDZ (Temporal Dead Zone)
var	Func	✓	✓	✗
let	block	✓	✗	✓
const	block	✗	✗	✓

arr = [1,2,3,4,5];

arr.myprop = "Hello"

arr.Print = function() {

}

indexes work as an key for arr.

arr[95] = 100

[1, 2, 3, 4, 5, <90 empty item>, 100]