* Class :- No need to use prototype.

```
class Vehicle {
    constructor (nw, price) {
        this.nw = nw;
        this.price = price;
    }
    getPrice () {
        return this.price;
    }
}

var vehicle1 = new Vehicle (2, 300);
vehicle1.
=> Vehicle {nw: 2, price: 300}
```

* Class Expression and Hoisting.

- can not call class without 'new' keyword.
- does not support hoisting.

* Inheritance using class.

```
class Car extends Vehicle {
    constructor (nD) {
        super (4, 300)
        this.nD = nD;
    }
}

var c = new Car (4);
```

point to parent func/class

Promises :-

3 steps ⇒ Pending , fullfilled , Rejected.
(settle)

- var promise = new Promise((resolve, reject) => {});
  console.log(promise);
  ⇒ Promise {pending}

- var promise2 = new Promise((resolve, reject) => {
      resolve("resolved");
  });
  console.log(promise);
  ⇒ Promise {<fullfilled>: "resolved"}

or Promise.resolve('resolved').then(data => console.log(data))

- var promise3 = new Promise((resolve, reject) => {
      reject("Error Occured");
  });
  console.log(promise);
  ⇒ Promise {<rejected>: "Error Occured"}
  Uncaught

To get data : (when fullfilled) ⇒ .then
(when rejected) .catch

```
var promise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve({message: "resolved"});
    }, 3000);
});

promise
    .then((data) => {
        console.log(data);
    })
    .catch((error) => {
        console.log('error', error);
    });
```

fullfilled

rejected

* Callbacks: function pass as argument to another function & invoked later.

```
function greet(name, callback){
    console.log('Hi ${name}');
    callback();
}
function askQuestion() {
    console.log('How are you?');
}
greet('John', askQuestion);
```

⇒ Hi John
How are you?

- Callback function takes it time for calculation but execution of of code goes on.

```
console.log(1);
var promise = new
setTimeout(function() {console.log(2)}, 1000);
console.log(3);
```

⇒ 1
3
2

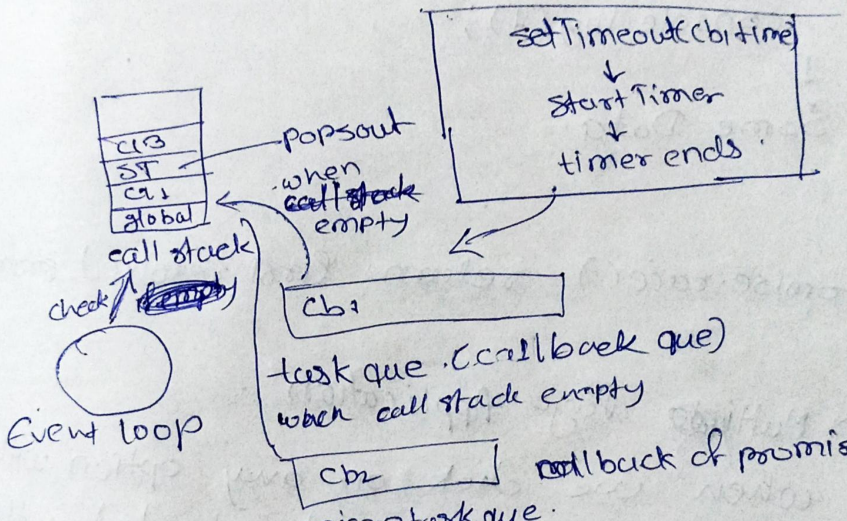
* setTimeout is Browser api function not a language function.

Event Loop :- checks call stack whether empty or not, if empty push callback function into callstack.

```
console.log(1);
setTimeout(function(){
        console.log(2);},0);

console.log(3)
```

=> 1
3
2



microtask que has higher priority the task que.

* Async await keywords.
  → Returns Promise.

```
async function funct1(){
    return "Resolved";
}
funct1()
=> Promise {<fullfilled>: "Resolve"}
funct1().then(data => console.log(data));
=> Resolved
```

* await → ① suspend the function and transition micro task que.
② wait for function to complete
③ Js move to synchronous code.

```
function getData() {
    return Promise.resolve("Some Data");
}
async function abc() {
    const data = await getData();
    console.log(data);
}
abc();
console.log("1");
```

=> 1
Some Data.

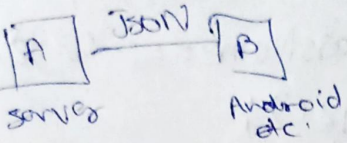
* Promise.race() returns first resolved promise.

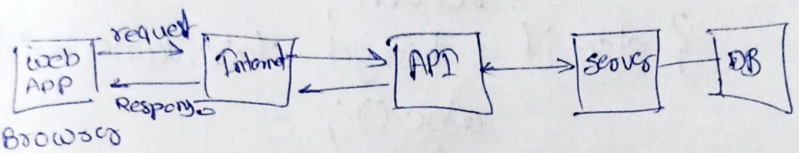
* MPA → Multi Page Application.
when we click on any option with link the whole page get reloaded with another HTML page.

* SPA → Single Page Application.
Only 1 link using JS
- when we click on any option API call happens and only required Data gets provided by server other content does not get refreshed.
- caches gets stored and we can work even internet loots.
- Poor SEO.

* API :- Two Devices are communicating using multiple formats using very minimalistic resources, one of thing is API.

APPLICATION PROGRAMMING Interface.





* Project..

1. What we required.
   - UI
   - functionality
   - Data (types, structure
   - functions
2. Coding.

* Event Delegation,
  - Event delegation is basically a pattern to handle events efficiently.
  - It simplifies intialization, saves memory & improves performance.
  - By using event delegation, we can add an event listener to parent element and call an event on particular target using the '.target' property of the event object.

```
document.addEventListener('click',
                    handleClicks);

function handleClicks(e){
    const target = e.target;
    if (target.className === 'xyz'){
        xyz();
        return;
    } else if (target.className === 'abc'){
        abc();
        return;
    }
}
```

* Module Patterns:-

* IIFE :- Immediately Invoke function Expression.

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

Adv: Do not creates unnecessary global variables & functions.

- Functions & variables defined in IIFE do not conflict with other functions variables even if they have same name.

* Revealing module Pattern.

```
var xyz = (function(){
            return {a:a, fun:fun};
          })()
```

a, fun are available everywhere
xyz.a, xyz.fun