Javascript

```
console.log('Hello World');
```

```
var a = 2, b = 'Harry', 2.30
```

Property Getters & Setters

```
get getname() {
}
```

property

```
console.log(person.getname)
```

this is not function.

```
set setname(n) {
    this.name = n.toUpperCase();
}
```

Constructor (ES5 till)

```
function Student (first, last, age, cls)
{
    this.firstname = first;
    this.lastname = last;
    this.age = age;
    this.class = cls;
}

var Student1 = new Student ("___",
                            "kumar", 25, 5);
```

→ maan lijiye aap constructor me ek property aur add krna chahte ho.

→ Use Prototype –

```
student.prototype.nationality = "Indian";   (✓)
Student.nationality = "Indian";             (X.)
```

Nested Object ;

```
var user = {
    id : 101;
    email:
    personalInfo : {
        name : "abc",
        address : {
            street :
            city :
        }
    }
}
```

Hoisting (to lift, or to pull);

```
x = 7;
console.log(x);
var x = 7;
```

7 (✓)

(Hoist declaration above the code)

```
console.log(x)
x = 7;
var x = 7;
```

(X) error

```
console.log(x)
var x = 7;
```

output – undefined

---

considering let & const –

output – not initialized.

var x → declared

initialized x = undefined (x) → this step does not occur in let & const

(✓) let, const

"use strict"

⇒ JS can use variable without declaration

⇒ So to avoid these things "use strict"

Getter & Setter

get getname ;
set setname ;

DOM (Document Object Model)

DOM is an API

⇒ All html are defined as object.
→ Object can use both Property & Models.

Element selection

by id

```
let elm = Document.getElementById("first");
elm.innerHTML = "Hello";
```

by Class :-

```
let elm = Document.getElementByClassName(" ");
```

by tag Name :-

```
let elm = Document.getElementByTagName("p");
```

mostpowerful selector - Query Selector —

```
div1 = Document.querySelector("p.first");
```

class

```
elm = Document.querySelectorAll(" ");
```

```
for (i=0; i < elm.len; i++)
{
    elm[i].innerHTML = "Hello";
}
```

Traversing Element

selecting parent

```
let elm = document.getElementById('Intro');
let prt = elm.parentElement;
```

selecting one child

```
let elm = document.getElementById(" ");
let1child = elm.firstElementChild;
```

selecting all child

```
let allchildren = elm.children;
```

selecting sibling —

```
let sib = elm.previousElementSiblings.
        - elm.nextElementSiblings;
```

InnerHTML —

```
let elem = Document.getElementById(' ').
elm.innerHTML = " change the content ";
```

InnerText —

innerText ⇒ HTML tags are ineffective.

<H1> — <[H1]

Appending a New HTML Tag

```
let elm = document.getElementById("Intro");
let h1 = document.createElement('h1');
elm.appendchild(h1);
let text = document.createTextNode("This is h1 tag");
h.appendchild(text);
elm.appendchild(h);
```

→ blank

→ method

~~elm.appendchild();~~

```
h1.className = "try Pi";
h.textcontent = "Content added";
```

→ by property

Insert before –

```
list1.insertBefore (item, pos);
```

Remove child element –

```
Parent.removeChild (elm);
```

clone element –

```
let menu = document.getElementById ("menu");
let cloneElm = menu.cloneNode (true);
```

(child + data)

false
sirf parent

ReplaceElement

```
let replace = parent.firstElementChild
              .nextElementSibling.
parent.replaceChild (item, replace);
```

Insert ~~Adj~~ Adjacent HTML –

⇒ It has four position-

```
elm.insertAdjacentElement ('beforebegin', ___);
                          afterbegin
                          beforeend
                          afterend
```

Changing Attribute

```
btn.setAttribute("Name", "form1");
                  class
                  Value
```

How to know attribute

```
let Val1 = btn.getAttribute("Name");
```

Remove

```
btn.removeAttribute();
```

hasattribute :-

```
console.log(btn.hasattribute);
```

Inline style

1. `btn.style.cssText = "background-color-color-blue;";` replace
2. `btn.setAttribute("style", "background-color-color-blue");` replace
3. add `btn.style.color = "Red";`
4. add ~~`btn.setAttribute +=`~~

```
btn.style.cssText += "backgr
                      ";
```

get Attribute

p2.setAttribute('contenteditable', true)

p.setAttribute('ispellcheck', false)

Get Computed css in javascript

```
let btn = document.getElementById("btn");
let css = getComputedStyle(btn);   [tells All styles of btn id]
let css = getComputedStyle(css.color);   [specific prop tells erty]
```

Changing CSS class in javascript —

single class

```
let box = document.getElementById("Box");
box.className += " dim";   { it will add dim class property to btn id }
```

if it has multiple class —

```
box.classList(box)
```

adding → box.classList.add("dim");

removing → box.classList.remove("dim");

replacing → box.classList.replace("color","dim"); { color ko replace dim se }

box.classList.contains("dim"); → true/false

box.classList.toggle("dim");

{ agar dim hai to hata do / agar nhi hai to laga do }

Get width and height of element in JavaScript

- (a) offset { border, padding, margin include hota hai; } (b) client { sirf ~~nor~~ padding include hota hai; }

```
let width = box.offsetWidth;
let height = box.offsetHeight;
```
|
```
let width
⇒ box.clientWidth;
```

---

two methods for **DOM Event** — event registering

(a) Inline method

(b) using ~~adding~~ eventlistener methode

(a)

```
<button id="btn" onclick="btnclick()">
  clickme </button>

<script>
    function btnclick() {
        alert("btn was clicked");
    }
</script>
```

(b)

```
<script>
  let btn = document.getElementById("btn");

  btn.addEventListener('click', btnclick);
</script>
```

↓ you can use anonymous function

```
btn.addEventListener('click', function() {
    alert("btn was clicked");
});
```
anonymous function

⇒ You can add multiple event listener on a single element:

```
btn.addEventListener('mouseover', Anonymous fn);
btn.addEventListener('mouseout', An. function);
```

Removing Event Listener –

```
btn.removeEventListener('click', click2);
```
event name (function name)

Page Load event in Javascript

DOMContentLoad – the browser fully loaded HTML and completed building the DOM tree. However it hasn't loaded external resources like stylesheets and images. In this event, you can start selecting DOM nodes or initialize the interface.

Load - The browser fully loaded the HTML and also external resources like images and stylesheets.

Dom Event

① onclick / btn.addEventListener("click", function());

mouseover, mouseOput

Page Load Event —

DOMContentLoad —

Load -

MouseEvent —

oncontextmenu — right clickevent

~~ondou~~ ondblclick —

mousedown —

mouse up —

KeyboardEvent —

window.addeventListener('Keydown', ~~cha~~ checkKey);

function checkKey(event){

console.log(event.key);

}

Scroll Event —

① window.addeventListener('scroll', function(){

});

② wheel

Remove Event Listener –

eventname ↓ function ↓

```
btn.removeEventListener('click', click1)
```

window.addEventListener('DOMContentLoaded', Anonymous fun);

window.addEventListener('load', Anonymous fun);

mouse events in javascript

onclick

oncontextmenu - right click event

ondblclick - on double click.

on mousedown - { jaise button click karenge

onmouseup - { jaise his button chodenge click krke }

Onmouseover -

Onmouseout -

Keyboard events -

(Keydown) / (Keyup)

window.addEventListener('Keydown', checkKey);

function checkkey(event) {

console.log(event.key);

}

↓

tells which button is clicked.

Scroll event in javascript

(scroll, function()); { it does not tell the direction of scrolling (like Vertical up, V down, Horizontal up, Horizontal down }.

window.pageYOffset
→ tells the scroll in pixel.

```
window.addEventListener('wheel', function
(event) {
    if (event.deltaY < 0) {
        console.log("scrolling up...");}
    else if (event.deltaY > 0) {
        console.log("scrolling down...");
    }
});
```

Events on form in Javascript :-

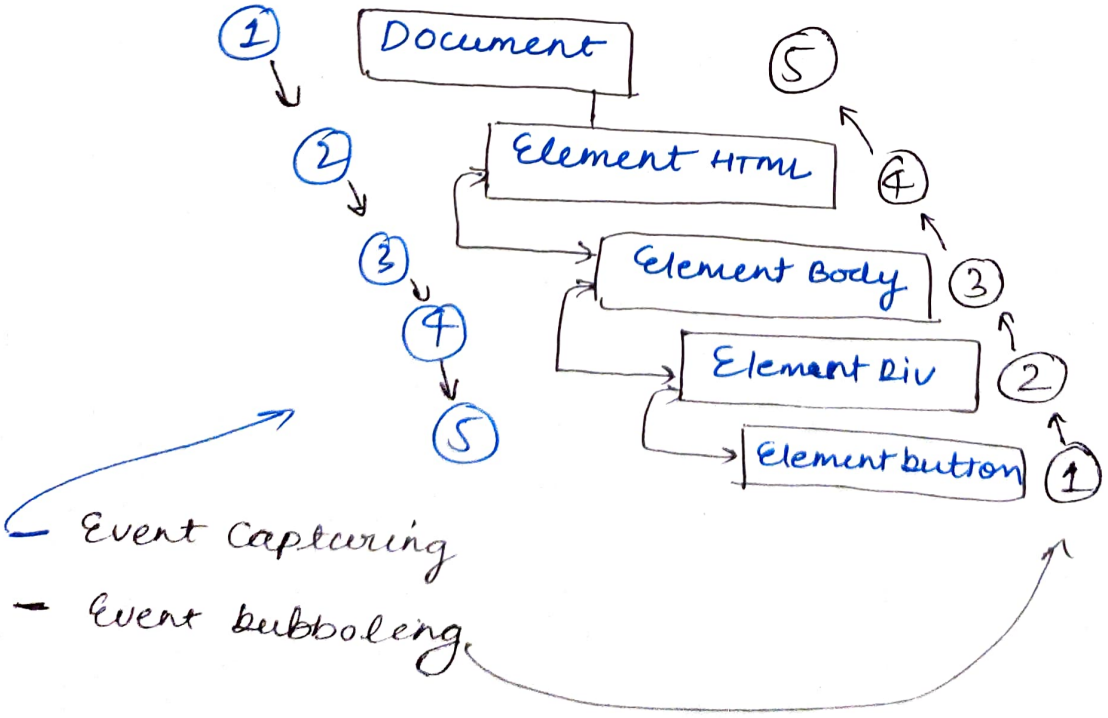

```
Let x = document.getElementById("my
                                input")
x.addEventListener('focus', myFocus);
x.addEventListener('blur', myBlur);
function myFocus() { x.style.background = "yellow";
                   }
function myBlur() { x.style.background = "cerise";
                  }
```

```
x.addEventListener('change', function() {
    console.log(this.value);
});
```
{ when the box value change & and we blur }

```
x.addEventListener('input', function() {
    console.log(this.value);
});
```
{ whenever any change in value of box area happens }

Event Bubbling & Event Capturing -

Event Bubbling - In the event bubbling model, an event starts at the most specific element and then flows upward toward the least specific element.



Event capturing model

⇒ In the event capturing model, an event starts at the least specific element and flows downwards towards the most specific element.

```
b.addEventListener('click', btnClicked);   // by default bubbling
b.addEventListener('click', btnClicked, true);   // capturing
```

⇒ how to stop propogation →

```
fn btnClicked(event) {
    event.stopPropogation();
}
```

Prevent Default

`e.preventDefault();` prevents default property

BOM Browser Object model

⇒ The window object represents a window in Browser.

⇒ All global objects, function and varicebles with the var keyword automatically become member of the window object.

⇒ Global Variables are the properties of window object.

⇒ Global functions are the methods of the window object.

Window Object —

innerHeight, InnerWidth — only display area of content.

outerHeight, OuterWidth — browser area.

(window.innerHeight);

```
let btn1 = document.getElementById("btn1");
let url = "https://www.google.com";
let features = "height=500, width=500px";
btn1.addEventListener('click', function() {
let win = window.open(url, 'google', features);
});
```

open a new window with given url.

Name

```
btn2.addEventListener('click', function() {
    win = window.open("https://www.yahoo.com", 'google');
});
```

url

window name

change the url of the open window of given name.

```
btn3.addEventListener('click', function() {
    win.close();
});
```

Time Out (once) and Time Interval (Repeat)

```
let x =
setTimeOut(myFunction, 5000);

function myFunction() {
    console.log("done...");
}
```

5000 = 5 sec

after 5 sec output

```
clearTimeOut(x);
```

~~SetTimeout~~ setInterval -

```
var t1 =
    setInterval (fun, 3000);

fn fun() { console.log("please subscribe---");
}
```

↑ after every 3 sec
repeat

Stopping it ↓

```
let btn1 = document.getElementById("btn");
btn1.addEventListener ('click', function()
{   clearInterval (t1);
});
```

Location Object (sub object of window) -

```
console.log(location.href);
            (location.pathname);
            (location.protocol);
```

redirecting ↓

① window.location = "https://google.com";
② location.href = " " ";
③ location.assign (" https://google.com");
④ ——— . replace ———

windows have history ←

back button nhi milegu

⑤ location.reload();

Navigator Object in JS —

navigator.appName
- appVersion
- appCodeName
- cookieEnabled
- userAgent
- platform

} properties

- javaEnabled(); % method

Screen Object — screen.width } % screen size
.height }

.colorDepth % 32/24 bit
{ matlab 24/32 different color dikha sakta hai }

- Orientation { landscape.

ES6

⇒ till ES5 - var
in ES6 - let, const

Var is property of window object

```
var count = 2;
console.log(window.count);
```
↓
2

```
let count = 2;
console.log(window.count);
```
undefined

⇒ var can be redeclared but let can not be.

⇒ Default Parameter in JS -

```
function talk(msg = "Hi")   // Default parameter
{
    (msg);
}
talk("Hello");   → Hellow
talk(_);   Hi
```
empty

⇒ Rest parameter

```
function fun(... args) {
    let sum = 0;
    for(i=0; i< args.length; i++)
        sum += args[i];
    return sum;
}
```

sum(2, 3); — 5

sum (2, 3, 4); — 9

sum (); — 0

Spread Operator –

let array1 = [1, 2, 3, 4];

let array2 = [5, 6, 7, 8, ... array1]

array1 → 1, 2, 3, 4

array2 → 5, 6, 7, 8, 1, 2, 3, 4

for...of loop

```
let score = [80,39,70,54];
for ( let x of score) {
    console.log(x);
}
```

80
39
70
54

Template literals –

→ Before ES 6, ' ', " " are used for string But in ES 6 (` `) backtick can be used.

→ as write with same spaces and lines as in compiler.

let var = "~~String~~" "Abhay - - - Singh" → Abhay Singh

let var = `Abhay - - - Singh` → Abhay - - - Singh

2nd use -

```
let str = "Abhay";
console.log("My name is ", + str + ".");
console.log(`My name is ${str}.`);
```
} same output

Array Destructuring -

```
let book = ["Advance js", 200, 150];
```

Old -

```
let name = book[0];
let page = book[1];
let price = book[2];
```

New - [ES6] Both are same.

```
let [name, page, price] = book;
```

also if extra variable is there corresponding to which there is no variable / value in array then it is defined as undefined.

Eg

```
let [name, page, price, Publication] = book;
```
undefined.

```
let book = ["Advance Js",, 200, 150]
                          ↑ empty
```

undefined

```
let [name, page, ~~publication~~ price, pubelication] = book;
```

```
let [name, page=500, price, pubelication] = book;
```

default value - 500 if it is undefined

```
let book= ["Advance Js", 100, 200, ["Techsum", 2021]];
let [name, page, price, [pubelication, year]] = book;
```

-

→ use of this destructuring -

```
function book() {
    return ["Advance Js", 200];
}
let [book_title, Price] = book();
```

Object Destructuring

```
let book = {
    name : "Advance js";
    page : 200,
    price : 150
};
```

Old -

```
name = book.name;
```

New - name should be exactly same as in object.

```
let {name, page, price} = book;
```

how can change name ?

```
let {name: title, page, price = 300} = book;
```

(name changed → title; default value → 300)

⇒ default value - It work only when there is no property/value associated with it.

nested objects

```
let book = {
    name = "Advance js";
    page = 200;
    publication: {
        pub-name: "techgun";
        year = 2024;
    }
};
```

let {name: title, page, publication: {pub-name, year}} = book;

Modules

file1.js — let name = "Abhay";

file2.js — function greet(name) {
alert(name); }

file3.js — greet(name);

script - file1 ✓
script - file2 (✓)
script - file3

⇒ It will not give error because variable is declared first, then function is declared in second and it is called in third.

Problem - script - file2
script - file3 (X)
script - file1

⇒ It will give error to solve this problem, module is introduced.

user.js –

```
export let name = "Abhay";
       let age = 27;
export function code () {
    console.log("coding...");
function cook () {
    console.log("cooking...");
}
```

account.js

```
~~export~~ let account_no = 432154;
    let account_type = "saving";

export function withdraw() {
    console.log("Amount deducted");
}
(1) export function deposit () {
    console.log("Balance updated...");
}
```

or (2) — two ways of exporting

```
export {withdraw, deposit};
```

app.js

two ways of importing

```
import { name } from 'user.js';
import { code } from 'user.js';
```
— (1)

```
import { withdraw, deposit } from 'account.js'
```
↓ (2)

```
withdraw();
deposit();
```

also -

⇒ we can rename import function as

```
import { withdraw as wd, deposit }
        from 'path';
```

— what if multiple files have same function and we have to import every function in a singe file.

- Eg - createWithdraw() → user.js
createWithdraw() → account.js

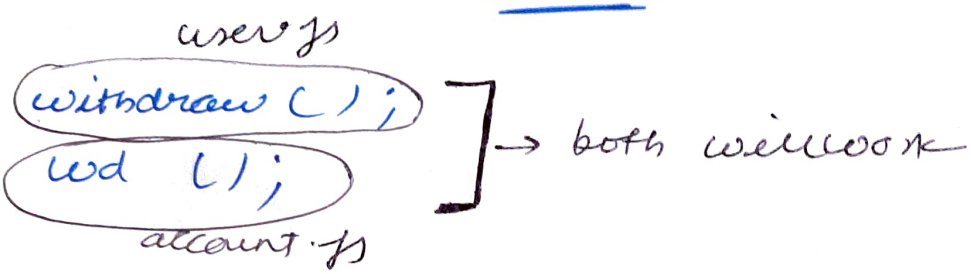
```
import {withdraw} from user.js
import {withdraw} from account.js

withdraw();  (X)
```

error with redeclaration message.

Solution -

```
import {withdraw} from user.js
import {withdraw} from account.js
        as wd
```



Qu what if we have to import many (100) variables / function from another file? it is useless to call / import them individually.

⇒ So we have to change method for only importing ~~and exporting~~ -

```
import * as usr from 'path';
```

withdraw(); (X)
usr.withdraw(); (✓)

⇒ When you import something from any other file, you have capability to use it but not to export it.

⇒ to export it you have to write individually to export.

```
export {circle} from path
export {triangle} from path
export {square} from path
```

Object Oriented Programming in Javascript -

⇒ DRY {Don't repeat yourself}

Imp ⇒ four pillars of object oriented Programming.

Three important words to understand :-

1. Object
2. Class
3. Inheritence

Object

⇒ In this world everything is a object (living and non living things)

Every object has 2 things -

1. Feature or Properties
2. Action performed by or performed on it.

Procedural VS Object Oriented

⇒ In OOP concept we group function and variable in block called class.

This is not the exact syntax:

1. Encapsulation :-

⇒ Encapsulation means wrapping data and member function (method) together into a single unit i.e. class.

2. Abstraction :-

Abstraction is the proces of showing only essential features of an entity/object to the outside world and hide the other irrelevant info.

3. Inheritence

Inheritence allow a class (subclass) to acquire the properties and behaviour of another class (super class).

⇒ It helps to reuse, customize and enhance the existing code. So it help to write code accurately and reduce the development time

4. Polymorphism —

So ~~polyphorn~~ polymorphism means "many forms"

A subclass can define its own unique behaviour and still share same functionalities or behaviour of its parent / base class.

```
class square(){
  area();
}

var S1 = new square();
S1 -> area();
```

```
class circle(){
   area();
}

var C1 = new circle();
C1 -> area();
```

Object Oriented Programming before ES6 -

⇒ There was no class like things before ES6.
How did we used class like thing, object, inheritence?

Creating object -

let persons = { }; - object - (1) 1st method

let person1 = {
firstname: 'Abhay',
'last_name' / last_name : 'singh',
age : 28 (no comma) (x)
};

person1.firstname

person2.lastname - [lastname : 'singh']

person1[lastname] - ['last name' : 'singh']

creating method -

{
fullname: function () {
console.log (this.firstname + " " + this.lastname);
}
};

Calling - person1.fullname();

- Creating a property outside object -

```
Person1.name = "Abhay Singh"; - property
```

```
Person1.sayHi = function () {

function.          console.log("Hi");
                   };
```

creating object method - (2nd)

```
let person1 = new Object();
```

⇒ Suppose we have to create n person with same property, then we have to create it individually since till ES5 we don't have classes like things

⇒ So to avoid creating individually we used classes like things called constructor.

How to create constructor function -

⇒ try to write first letter of constructor capital [standard way]

```
function Person(first, last, a) {
        this.firstname = first;
        this.lastname = last;
        this.age = a;
}
```

creating object using constructor-

```
let person1 = new Person("Abhay", "Singh", 20);
let person2 = new Person("Abhi", "Regh", 21);
```

adding method -

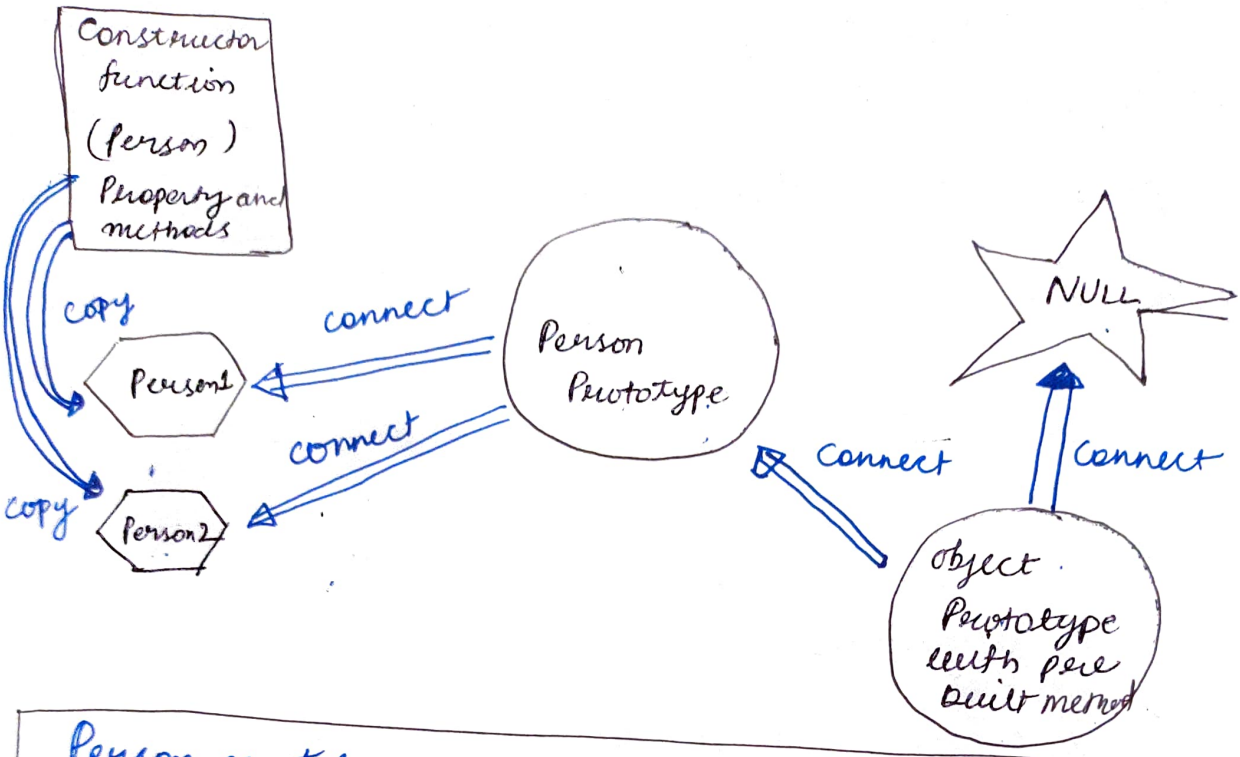
```
this.sayHi() {
    alert();
};
this.changeAge(newage) {
    this.age = newAge;
};
person1.changeAge = (23);
```

⇒ prototype is most confusing than other language.

Prototype and Prototype inheritence-

```
let person2 = {
    name: "Abhay"
};
console.log(person2.hasOwnProperty('age')); false
console.log(person2.hasOwnProperty('name')); True
```

by default property of prototype object.

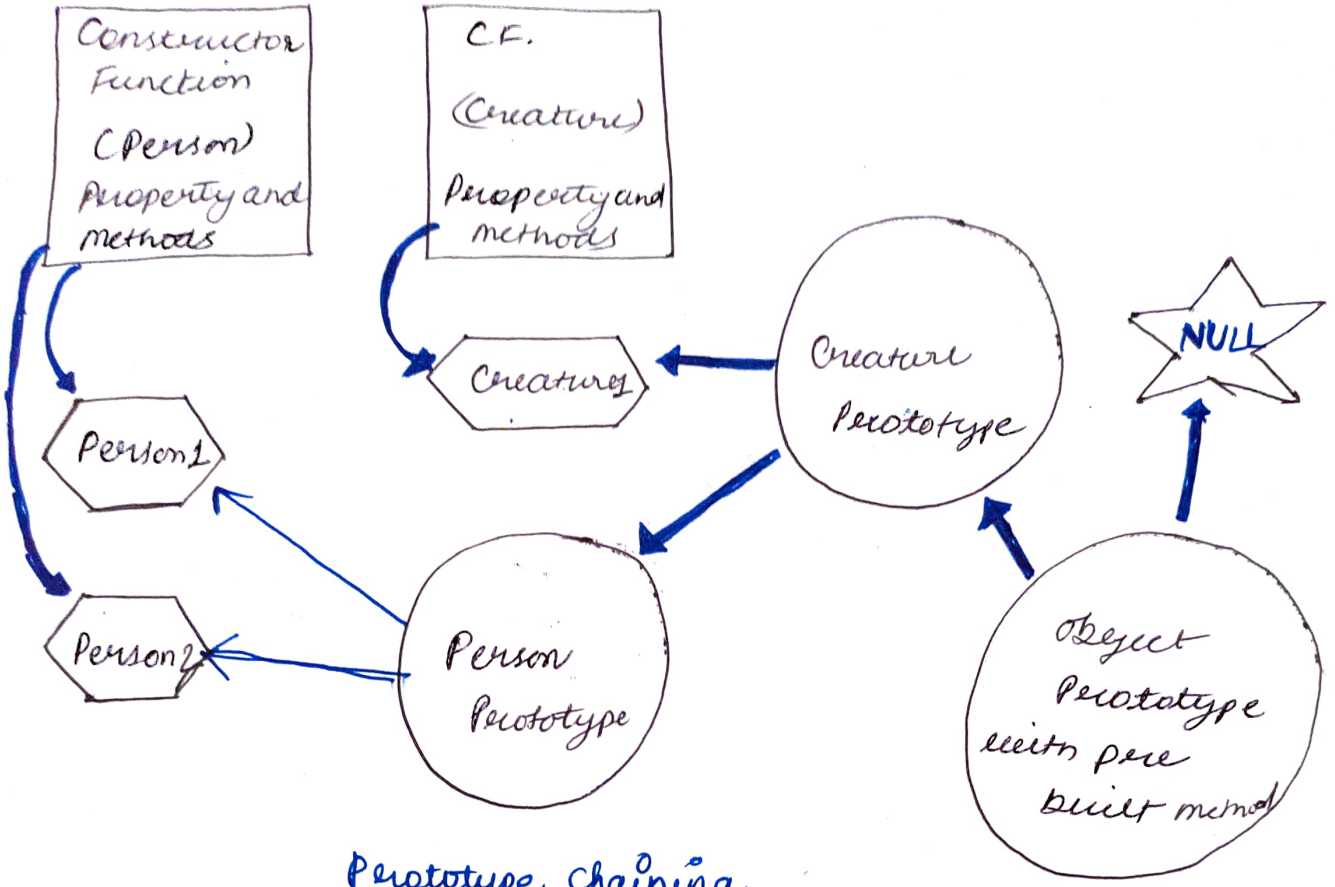



```
Person.prototype.fullName = function () {
    console.log(this.firstName + " " + this.lastName
})
```

⇒

```
Person.prototype.age = "25";
```

⇒ Prototype is same for every object hence only method is recommended to put in ~~another~~ prototype.



Prototype Chaining

```
function Creature(ls) {
        this.lifespan = ls;
}

Creature.prototype.breath = function() {
        console.log("breathing");

let creature1 = new Creature(100);

function Person(first, last, a) {
        this.firstName = first;
        this.lastName = last;
        this.age = a;
}
```

```
Person.prototype.__proto__ = Object.create
                              (Creature.prototype);
```

```
let person1 = new Person ("Abhay", "Singh", 21);
let person2 = newPerson( "Abhi", "Raghu", 21);

  console.log(person1);
  console.log(person2.breathe());
```

ES6 - class and object

```
class person {

}

let person1 = new person();
```

→ constructor is automatically called whenever object is created.

```
class person {                    // creating an object
    constructor(n,a){                with given values.
        this.name = n;
        this.age = a;
    }
}

let person1 = new person("Abhay", 21);
```

→ In class, method is already contained in prototype.

Static

```
static method -

    static hello() {
        console.log("hello...");
    }
    static person = "something";
```

person1.hello(); (X) object se call nhi hota hai

Person.hello(); (✓) class se call hota hai

⇒ Static method and static property prototype se connect nhi hota hai. isliye object se call bhi nhi hota hai.

Inheritence in Javascript :-

```
class emp {
    constructor() {
        console.log("emp constructor called");
    }
}

class manager extends emp {

}

let mng1 = new manager();
```

emp constructor called → mng1 → emp
                                ↓
                              manager

Calling parent class from child class :-

```
class emp {
    constructor(n) {
        this.name = n;
    }
    msg() {
        console.log("Hi");
    }
}

class manager extends emp {
    constructor(p, d) {
        super(p);
        this.department = d;
    }
}

let mrg1 = new manager("Vishavajeet",
                       "Web development");

console.log(mrg1.msg());
```

How can we call function of parent clan in child clan?

```
class manager extends emp{
    constructor(p,d) {
        super(p);  → p
        this.department = d;
    }

info() {
    super.msg();  → method
    console.log(this.name + " is
                  property
manager of department " + this.departme
    nt);
    }
}
```

—

⇒

```
msg() {
    console.log("method of child class");
    }

this.msg();  → child clan

super.msg();  → parent clan
```

Multiple Inheritence –

```
class admin extends manager {
    }

let admin1 = new admin("Abhay", "web
                        development");
console.log(admin1);
```

Public & Private in js:–

~~Public~~ Private

```
class emp {
  #name = " ";   using #
  constructor(n){
    this.#name = n;
  }
  getName(){                    private  #getName(){
    console.log(this.#name);              console.log(this.#name
    }
}

let emp1 = new emp("Abhay");
console.log(emp1.name); (X) prible
console.log(emp1.getname());
```

what if we want to add property of an object to any class ?

what if we want to introduce property to an already extended class ?

// mixin

mixin – mixin

```
let usefulmethod = {
    sayHi(){
        console.log("Hi---");
    },
    sayBye(){
        console.log("Bye...");
    }
};

class user{
  constructor(){
    this.name = "Vishwajeet";
  }
}

class admin extends user {
}

Object.assign(admin.prototype, usefull methods);

let admin1 = new admin();

console.log(usr1);
```

Arrow Function

```
let sum = (a, b) => {
        return (a+b);
    }
```

] multiple line.

```
let sum = (a, b) => (a+b);
```

```
let double = (a) => 2*a;
```

(a) ↑ single parameter

```
let random = ( ) => Math.random( );
```

—

```
document.addEventListener('click', function(){
    console.log("clicked..");
});
```

```
document.addEventListener('click', ()
    => console.log("clicked"));
```

Callback function

⇒ passing function as parameter

```
function sayHello (){
    console.log("Hello");
}

function.add (num1, num2, callback){
    console.log( num1, num2) ;
    callback();
}

let a=10;
let b = 20;

add (9,b ; sayHello)
```

```
30
Hello
```

map function

⇒ let arr = [2, 3, 9, 5, 6];

```
let arr1= arr.map (function(val) {
        return val * 3;
});
```

arr1 = {6, 9, 12, 15, 18};

can be done same using arrow function -

```
let arr1 = arr.map((val) => val * 3);
```

Filter Function -

```
let arr1 = arr.filter(val => val > 10);
```

Cookies

```
document.cookie = "item=milk";
document.cookie = "loc=India";
```

} expires = session end

parameters (like - expire, domain)

```
document.cookie = "item=milk; expires=Sat,
                   20 mar 2022 12:00:00 UTC";
```

How to change cookies?

```
document.cookie = "item=wine, eggs";
```

How to read?

```
let x = document.cookie;
alert(x);
```

How to delete -

give it a expired date as expiry date.

Web Storage (HTML5)

- Local Storage
- Session Storage

	Cookies	Local Storage	Session Storage
Capacity	4kb	10mb	5mb
Browsers	HTML4&5	5	5
Accessible from	Any window	Any wind	Same Tab
Expires	Manually set	Never	on tab close
Storage Location	Browser and Server	Browser only	Brown only
Sent with requests.	Yes	No	No

```
localStorage.setItem('FirstName', 'Abhay');
localStorage.setItem('Location', 'Delhi');

console.log(local.storage.getItem('Locasim'));

localStorage.removeItem('Location');

sessionStorage.setItem('FirstName', 'Abhay')
```

All Same -

JSON (Javascript Object Notation)

⇒ commonly used for API and config files

⇒ JSON syntax is derived from Javascript object Notation syntax:

Data is in name/value pairs.
Data is seprated by commas.
curly braces hold objects.
Square bracket hold arrays

```
{
  "name": "Abhay Singh",
  "age" : 25,
  "lang" : ["C", "C++", "Java", "php",
            "Python"]
}
```

⇒ JSON is language independent.
(code for reading and generating JSON exists in many programming languages).

⇒ Earlier XML was used instead of JSON.

Then why JSON -

⇒ JSON is faster in terms of data parsing.

⇒ JSON is cleaner than XML.

⇒ JSON is light weight than XML.

⇒ Easy to write and read.

JSON File → data.json

```
{ "name" : "vishwajeet",
  "age" : 25,
  "is_student" : null,
  "P_lang" : ["C", "C++", "Java", "python"]
  "address" : {
        "city : "delhi" :,
        "state" : "new delhi",
        "pincode" : (1011)
  }
}
```

array of object

```
{ "students" : [
    { "name" : "Abhay",
      "age" : 20},
    { "name : "Adarsh",
      "age" : 20 },
  { "
  } }
```

Parse JSON [data from server]

```
let data = ` data`
```

```
let dobject = JSON.parse(data);
console.log( dobject ['name'] );
```

How to create JSON string ? [data to server]

```
let student = {
    name : "vishwajeet",
    age : 25,
    city : delhi
    }
```

```
let data = JSON.stringify(student);
```

Pyramid of doom :-

When we have callback inside callbacks, the code gets difficult to manage.

```
loadscript() {
        loadscript ...
                loadscript
                        loadscript
```

⇒ As callback become more nested, the code becomes deeper and increasingly more difficult to manage especially if we have real time code instead of ....

This is sometimes called "callback hell" or "pyramid of doom";

Promises, Asyn, Await — 1 hour

Promise - The solution to the callback hell is promises. A promise is a "promise of code execution". The code either executes or fails in both the cases the subscriber will be notified.

The syntax of a promise looks like this:

```
let promise = new Promise( function (resolve,
                reject) {
// code
});
```

Resolve and Reject are two callbacks provided by javascript itself. They are called like this -

resolve(value) → if the job is finished successfully

reject (error) → if the job fails.

The promise object returned by the new Promise constructor has properties. —

State - Initially pending, then changes to either "fulfilled" when resolve is called or "rejected" when reject is called.

Result - Initially undefined, then changes to value if resolved or error when rejected.

Consumers: then & catch

The consuming code can recieve the final result of a promise through then & catch.

The most fundamental one is then

```
promises.then( function(result) { /* handle */ },
               function(error) { /* handle error */ })
```

If we are interested only in successful completion we can provide only one function argument to .then():

```
let promise = new Promise(resolve => {
    setTimeout(() => resolve("done"), 1000);
});

promise.then(alert);
```

If we are interested only in errors, we can use null as the first argument:

then(null, f) or we can use catch!

promise.then(catch)

promise.catch(alert)

Promise chaining