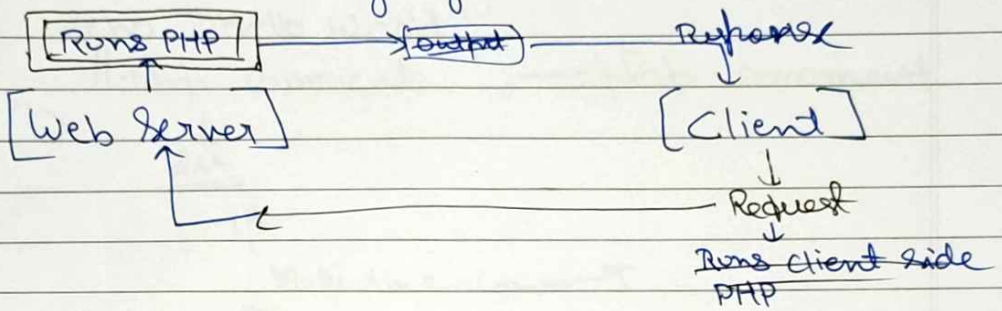
classmate
Date
Page

[PHP] Code With Harry

→ A backend language



JS → 'Show'able logic
PHP → Hidden logic

Request Types → GET
→ POST

XAMPP, Apache + MariaDB + PHP + Pearl

Environment for backend (PHP)

→ In 'htdocs' folder
↳ create your projects
→ create index.php not index.html
→ toca

→ localhost/phpmyadmin

→ localhost/folder name → we name of folder inside htdocs

classmate
Date
Page

index.php

<!-- Simple HTML Coment Code --> → HTML comment

```
<?php
    echo "Hello world";
    //php comment  → php comment
?>
```

also

```
/*
  Multi line comment
*/
```

PHP Code

```
$ variable_name = "value";
echo $variable_name
```

// dynamic lang (no data type) → to be declared

// NOT case sensitive

echo "
" will give newline

//Operators in PHP

Arithmetic operators (+, −, *, /)
Assignment : (=, +=, -=, etc)
Comparison : (==, <=, !=, >=, <, >)
Inc/Decrement : (++, --)
Logical operators (and (&&), or (||), xor, not (!))

var_dump(--) will give data type & value of variable

classmate
Date
Page

Data Types in PHP

1. String
2. Integer
3. Float
4. Boolean
5. Array
6. Object.

Constants

At the start of PHP (Recommended)

```
define ("pi", 3.14);
```

```
echo pi;
```

↳ no $ symbol

we need to do `echo "<br>"` to give line break

* If --- ladder

```
if ( )
{

}
elseif ( )
{

}
```

```
else
{

}
```

classmate Date Page

* Loops

- while
- do while
- for
- for each

```
$a = 03;
while ($a-!=0)
{
    echo $arr[$a];
}
```

```
foreach ($lang as $arr)
{ echo $lang; }
```

⊙ Arrays

```
$arr = array ("Me", "Myself", "AhamBramhasmi");

echo count ($arr);
echo $arr[0];
```

⊙ function in PHP

- built in: var_dump(), strlen()
- user-defined

```
function print5(){
    echo "5";
}
```

we can concat strings using dot (.)

* String functions

- → strlen(...)
- → str_word_count(...)
- → strrev(...)
- → strpos(x, y) (position of word)
  - ↓ returns nothing in string if not found
- → str_replace(x, y, s)

classmate
Date
Page

* Create DB to store form inputs — 1/2/8/2023
  ↳ use phpMyAdmin on XAMPP server

* MySQL operations in PHP

Below is procedural approach not object oriented.

```
$server
$uname
$pwd

$con = mysqli_connect ($server, $username, $pwd)

if (!$con)
  die ("error msg due to". mysqli_connect_error());

$query = " query with $variables "
```

→ Create vars for each input
→ get form data into vars
→

```
$age = $_POST ['age'];
```
(varname)

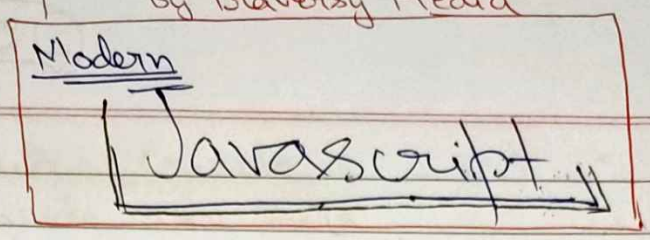
```
if ($con->query($sql) == true)
    echo "success"
else
    echo $con->error
```

```
$con->close();
```

PHP. Finish.

7

→ YT (Modern JavaScript From The Beginning --- By Traversy Media



1/18/10/2023

YT ⇒ "Modern JS from the beginning | The first 12 hours"

* What is JS?

→ high-level interpreted PLang
→ client side as well as server side

* Javascript used for?

→ DOM manipulation
→ Event handling
→ Asynchronous Requests (dynamic content loading)
→ Animations & Effects
→ Data manipulation (sort, filter)
→ Store data on client side (cookies, Local Storage)
→ Single Page ~~Experien~~ Applications (SPAs)
→ Creating APIs & Web Services

for internal JS, we should do it inside body tag.

classmate
Date
Page

* Browser Console

→ Inside Dev Tools
On Chrome → Ctrl ⊕ Shift ⊕ I

Elements → HTML, CSS
Console → JS (testing, debug)
↳ clear() / Ctrl ⊕ L

when we type an obj's name, ↵
it shows attributes and functions

clg shortcut
↳ console.log() ← argument(s)
object, method
(true, 'Hello', 29) ✓

- error() (red)
- warn() (yellow)
- table(obj) (key-value pairs) as table

```
• group('simple');
• log(a);
• log(b);
• groupEnd();
```

Adding css to * console

```
const style = 'padding: 10 px; color: white';
console.log('%cHelloworld', styles);
```

9

Ctrl + coma → settings (VS code)

Ctrl + K + S → Keyboard Shortcuts

* Comments in JS

// Single line comment

/* Multi
Line
comment */

Shortcut: Ctrl + /

Shortcuts

Move current line/selected lines up or down: alt + ↑↓

* → changing multiple instances at once

~~Select all matching → Ctrl + Shift + L~~

~~Select next →~~

Select prev. matching → Ctrl + Shift + J

Select all matching → Ctrl + Shift + K

Select next matching → Ctrl + Shift + L

} custom

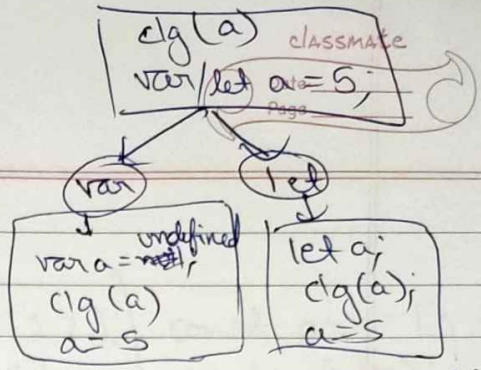
Custom → place cursor at multiple places
↳ hold alt key

Search → Ctrl + F

Toggle Sidebar → Ctrl + B

Toggle Terminal → Ctrl + ~

Open file search bar → Ctrl + Shift + O



* Variables & Constants

* Ways to declare

- var
- let
- const

→ original keyword, not used much anymore ~~hoisting~~
→ ~~enables~~ helps hoisting*
→ can't be directly reassigned. MUST BE INITIALIZED

Variable Name

→ letters, numbers, underscores, dollar sign
→ can't start with number

- allsmall
- ALLCAPS
- camelCase
- PascalCase

Note:

Hoisting is the process in which during compilation of JS code, variables and functions (~~decl~~ declarations only) are moved to the top of the code.

Ex

c.log(a);
let a=5;
⇓
Error
	c.log(a);
	var a=5;
	⇓
	OK but prints
	undefined
		C.log(f());
		function f()
		{ return 5; }
		⇓
		prints 5

converts code to ⇒

```
var a;
c.log(a);
a=5;
```

Note: (for const)

```
const a = 5;
a = 6;
```
✗ ERROR

```
const arr = [1,2];
arr = [1,2,3]
```
✗ ERROR

```
const arr = [1,2]
arr.push(3);
```
✓ OK

So, basically, you cannot reassign a ~~vari~~ const directly. However, you can change props inside a variable.

PRO TIP

Use const unless you need 'let'.

Also

```
let a, b, c;   ✓
let a=5, b=10, c;   ✓
```

* Data types

PRIMITIVE TYPES

- **String**
  - enclosed in quotes or backticks
- **Number**
  - int as well as floating point
- **Boolean**
  - true or false
- **Null**
  - intentional absence of any value
- **Undefined**
  - when a variable has not yet been defined/assigned.
- **Symbol**
  - Built in obj whose constr. returns a unique symbol.
- **BigInt**
  - for nos. larger than 'Number' type can handle.
  - append n in the end of the number
    i.e, const a = 123n;
  - floating NOT allowed

classmate
Date
Page

• Reference & Types / objects

object literals, arrays, functions

* Dynamic Typing V/S Static Typing Languages

↓	↓
data type isn't needed (to mention)	we HAVE TO/CAN explicitly tell data type for variables

Now,
TypeScript is a superset (more like an extension) of JS, which allows static-typing.

* typeof → ~~tells~~ returns type of the variable

```
c.log(typeof a);
```

Note

•) const a = null;
const b = a;
c.log(typeof b);
⇒ object not 'null' because it's a mistake from developers side.

•) we can explicitly set a variable to undefined.

classmate
Date
Page

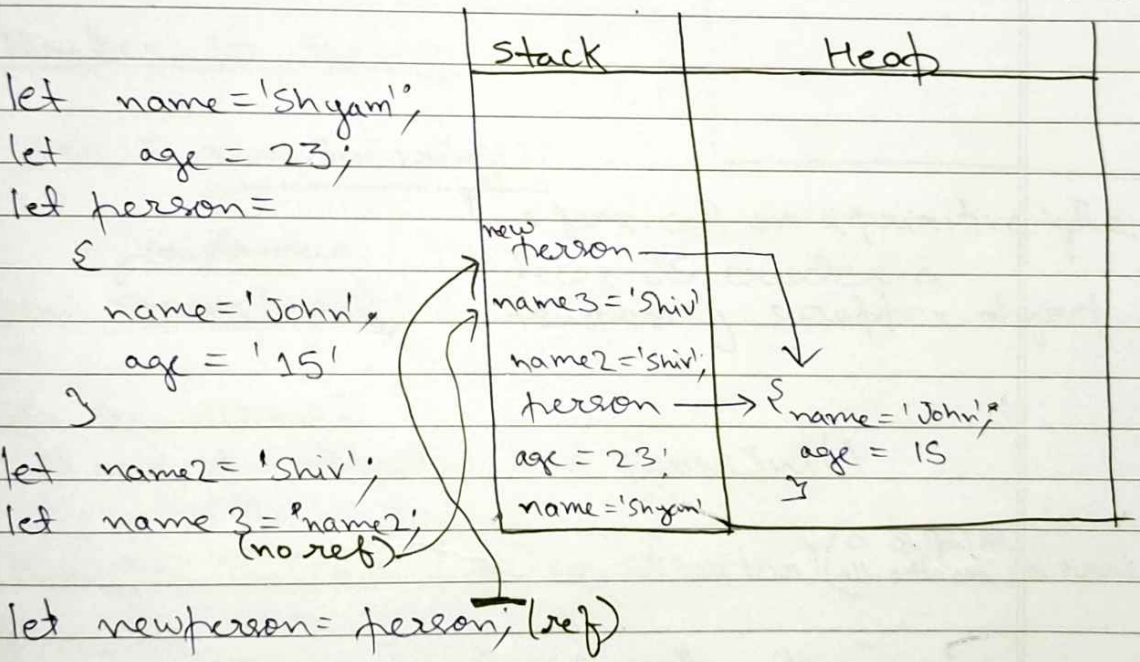
* Reference Types (Arrays | Function | Object)

~~funct~~
const ~~nu~~ numbers = [1,2,3,4];
const output = numbers;
c.log (typeof output);
⇓
['object']

function f1()
{
  c.log('hello');
}
const output = f1;
c.log(typeof output);
'function'
↳ it's still an object (a function object)

no () नहीं तो call no jayega

* How data is stored?



classmate
Date
Page

* Type Conversion (Explicitly) ——— Type Coercion (Implicit)

⊙ String to Number

① num = parseInt(str);

② ~~str = str;~~ (unary operator)
str = +str;

③ num = Number(str); ← constructor

⊙ Number to String

① str = num.toString();
→ func. call on a primitive type bcoz JS creates a temporary wrapper object

② str = String(num); ← constructor

⊙ Str to decimal
↳ use parseFloat() not parseInt()

⊙ Num to Boolean (use constructor) (0 is false, all other is true)

Note NaN is a special value for number type that says, 'हाँ वो तो होना है but tumne jo diya wo number hai hi nhi!'

Eg
```
const str = 'hello'
const num = Number(str);
~~clog(typeof num);~~
clog(num, typeof num);
```
→ NaN 'number'

Global window property

NaN

- when num can't be parsed (Number('string'))
- when result of op. isn't real ($\sqrt{-1}$)
- if one op is NaN
- undefined + 5
- 'foo' / 3

but **null + 5 = 5**

* Operators

⊙ Arithmetic ops

- `+` (also concat op)
- `**` (exponent)
- `++, --` (prefix, postfix)

⊙ Assignment Op.

- `=` value assignment
- `+=` `-=` ... short hands
- `**=`

⊙ comparison ops

- `==` compare values ⇒ `2 == '2'` gives true
- `===` compare value & type
- `!=` `!==` `<` `>` `<=` `>=`

* **Bro tip**: By default, use `===`

classmate
Date
Page

* Type coercion (Implicit conversion)

```
x = 5 + '5';          → 55      (string)
x = 5 + Number('5');    10      (number)
x = 5 * '5';          55555     (string)
  = 5 + null;           5       (number)
```

null → coerced to 0

```
  = 5 + true            6       (number)
```

true → coerced to 1

= 5 + undefined → NaN

If any operand is undefined

* Working with strings → primitive type

```
let str;
const name = 'Shyam'

str = "Hi_" + name;   → Hi_Shyam
```

// Template literals

`Hello, my name is ${name}.`

backticks not double inv. commas

${name} → any JS expression

str.length → 8

Note: This also works ✓

```
const str = new String('Hello world');
```

new String → Wrapper class/object

17

[0 indexed]

* Index Accessing (access values by key)

n = S[1] ⟶ e

JS stores all the functions in prototypes
Here's how to access it.

n = S.__proto__; ⟹ gives the list of functions

So

```
const S = "Hello";
```

⟹ object

```
0: "H"
1: "e"
2: "l"
3: "l"
4: "o"
```

key value pairs

[[Prototype]] : String
contains all functions
[[PrimitiveValue]] : "Hello"

```
S = "Shyam Everything"
n = S.toUpperCase();
  = S.charAt(0)
  = S.indexOf('H');
```

first

```
  = S.substring(0,4)
```

(3)

from here till end

. slice(-,-) ↳ accepts ove indexing (-n to -1)

classmate
Date
Page

```
S.trim()
 .replace('World', 'John');
 .includes('Hello'); // T/F
 .valueOf(); // returns value of (string) object
 .split(); → returns array of chars (one element)
          (' '); → seperated by space
          ('') → all chars seperate
```

blank / nothing / backticks

* Working with numbers

```
let n;
const num = new Number(5);

n= num.toString(); // converts to string
 = num.toFixed(2); // no. of decimal places
 = num.toPrecision(2); // total no. of digits
 = num.toExponential(2); // gives 5.00e+0
 = num.toLocaleString('en-US'); // still 5, but ukwim
 = num.valueOf(); // value, whis is 5
```

classmate
Date
Page

Number. MAX_VALUE;
. MIN_VALUE;

* The Math Object

console.log(Math); // prints all within the obj

let n;
n = Math.sqrt(9); ⟶ 3

n = Math.abs(-9); ⟶ 9

n = Math.round(4.2); ⟶ 4
. ceil(4.2); ⟶ 5
. floor(4.9); ⟶ 4

n = Math.pow(2,3); ⟶ 8

n = Math.min(2,3); ⟶ 2
(2,3,4); ⟶ 2

. max(2,3); ⟶ 3

n = Math.random(); a random decimal from 0 to 1

classmate
Date
Page

* Dates & Times

```
let d;
d = new Date();
```
→ Fri Oct 28 2022 08:48:44 GMT-0400 (Eastern Daylight Time)

typeof d → object

```
= new Date(2021, 6, 10);
```
(YYYY, M, DD) → Gives July, NOT June
→ Time 00:00:00 (current time zone)

```
(2021, 6, 10, 5, 23, 30);
```
(H, M, S)

```
('2021-07-10')
```
→ July NOT June

```
('2021-07-10T12:30:00');
('07/10/2021 12:30:00');
```

```
= Date.now();
```
→ timestamp
no of milliseconds passed since Jan 1, 1970

```
= d.getTime();
  .valueOf();
```
→ to get timestamp of particular moment in time

```
//create Date obj from timestamp

d = new Date(1234567890123456789);
```

// Date methods

let d = new Date();

- d.getTime()] timestamp in ms
- .valueOf()]
- .getFullYear() → 2023
- .getMonth() → ~~October~~ November 10 (you'll have to do +1 for actual month)
- .getDay() → Day of week
- .getHours()
- .getMinutes()
- .getSeconds()
- .getMilliseconds()

Useful API for Time & Dates

→ Intl

n = Intl.DateTimeFormat('en-US').format(d);
('default') → same as US

Date formatting

OR

n = d.toLocaleString('default',
{
weekday: 'long',
day: 'numeric',
month: 'long',
year: 'numeric'
});

classmate
Date
Page

* Arrays & Objects

→ Special type of 'object' and a data structure in JS that can ~~(same data type)~~ multiple values.

```
const arr = [365, 7, 24];
```
→ Array Literal

```
console.log(arr);
```
(3) [365, 7, 24]
0: 365
1: 7
2: 24
length: 3

// Array Constructor

```
const fruits = new Array('apple', 'banana');
console.log(fruits[0]);
```

→ backticks (template literals)

```
(`My fav. fruit is ${fruits[0]}`);
```
① ·

```
const len = fruits.length;
fruits[1] = 'guava';
```

```
fruits.length = 1;
```
⇒ removes guava

// Adding new item

① `fruits[fruits.length] = 'new-fruit';`

② `fruits.push('new-fruit');`

fruits.pop(); → removes last element

fruits.shift(); → removed from beginning

fruits.unshift('0'); → ~~re~~added to beginning

fruits.reverse(); → ~~IKWY~~ IKWYM

const n;

n = fruits.includes('guava'); → T/F

n = fruits.indexOf('apple'); first index | -1

~~n = ?~~

arr = [1,3,5,7,9];

n = arr.slice(1,4); (4 = end, non-inclusive; 1 = start, inclusive)

n = arr.splice(1,2); (1 = start; 2 = no of elements) ⇒ n = [3,5], arr = [1,7,9]

→ removes them from arr aswell

* ~~Meeting~~ concat, etc. on Arrays

n = [1,2,3];

y = [4,5];

n.push(y); → [1,2,3,[4,5]]

clg(n[3][1]); → 4

classmate
Date
Page

```
z = [x,y]; ⟶ [[1,2,3],[4,5]]
```

Concatenation

```
z = x.concat(y); ⟶ [1,2,3,4,5]
```

* Spread operator (...)

```
z = [...x, ...y]; ⟶ (11)
```

Spreads x
(i.e. replaces this expression with items within x)

// Flatten Array

```
x = [1,2,[3,4],[5,[6,7]]];
clg(x.flat()); → flattens (by one level only)
```

→ [1,2,3,4,5,[6,7]]

// Static Methods on Arrays

```
clg(Array.isArray(x)); // T/F
clg(Array.from('12345')); // ['1','2','3','4','5']
a=1; b=2; c=3;
clg(Array.of(a,b,c)); // [1,2,3]
```

* Object Literals

Key-value pairs
props-values

```
const person
    = {
        name : 'John',
        age  : 30,
        gender: 'Male',
      };
```

(nesting)

```
→ address: {
      PinCode: 400144,
      City: 'Varanasi'
  }
```

⇓ Accessing

person.address.city

```
n = person.name;
  #                  → 'John'
  = person['name'];
```

→ mutable

```
person.name = 'Shyam';

delete person.age;   → removes age property

person.hasChildren = false;   → adds a new prop along with value

person.greet = function()
    { clg("Hey! ${this.name} here!"); }
```

→ person.greet(); → runs the function

→ person.greet; → displays the function

Note

prop name can be a spaced string 'first_name' but can be accessed using bracket notation.

* Object Spread & Methods

```
const todo = {};
todo.id = 1;
```

{} → empty object

⇒ DB = new Object();

```
todo.name = 'Buy Milk';
todo.done = false;
```

```
const a = { p: 1, q: 2 };
const b = { r: 3, s: 4 };

const c = { a, b };
const d = { ...a, ...b };
```

{a: {p:1, q:2}, b: {r:3, s:4}}

{p:1, q:2, r:3, s:4}

spread operator

⇓ another method

```
const c = Object.assign({}, a, b);
```

{} → empty object

a, b → The props of these objects are assigned to

* Most of the time we use array of objects.

Methods on objects

`x = Object.keys(todo);` → array of all keys

`x = todo.length;` → doesn't work (undefined)

~~`x = keys.`~~

`x = Object.keys(todo).length;` ✓ no of keys

`x = Object.values(todo);` → array of values

`x = Object.entries(todo);` →
```
[
  ['id', 1],
  ['name', 'Brush']
]
```
// array of arrays

`x = todo.hasOwnProperty('name');` → true

* Destructuring & Naming

```
const id = 1;
const name = 'Shyam';
```

```
const Student
{
   id: id,
   name: name
}
```

⟹

```
const Student
{
   id, name
}
```

No need of redundant coding

* Destructuring

```
const todo
{
  user:
    {
      name: 'Shyam'
    },
  id: 1
}
```

```
const idOut = todo.id;
```

OR

① `const {id} = todo;`
pulling out id from todo

② `const {id, name} = todo;` — even deeper levels can be accessed directly

`clg( id, name);`
because basically line 1 is equiv. to -
```
id = todo.id;
name = todo.user.name.
```

③
```
const {user} = todo;
```
⇓ {name:'Shyam'}

④
```
const { user: {name}} = todo;
clg(name);
```
pull out name from user

⑤
```
const { user: userName } = todo;
clg( userName);
```
rename user to userName, for your code ONLY not in the object

⟷ `userName = todo.user;`

* Destructuring arrays

```
const n= [1,2,3,4];
const [f,s]=n;
```
f → ①, s → ②

```
const [f,s, ...rest] = n;
```
f → ①, s → ②
...rest: has to be last element; rest is just a variable name
remaining array ⇓ [3,4]

classmate
Date
Page

* JSON Intro → in a way, array of objects

JavaScript Object Notation

→ Earlier, XML was std. for sending and recieving data to/from servers.
→ Now, JSON is the new standard.

Eg. api.github.com/users/yourusername → gives your details

returns list of first 20 users of GitHub, in JSON format

Different from Object literals
→ double quotes around keys
→ double quotes around strings

* converting objects to JSON & vice-versa

```
const post = {
  id: 1,
  name: 'Post One'
};
// object to JSON(string)
const str = JSON.stringify(post);
// string(JSON) to object
const obj = JSON.parse(str);
```

[] bracket will be added if multiple objects are there

```
{"id": 1,
 "name":
  "Post One"
}
```

This whole text is a String, NOT object, NOT array

classmate Date Page

* Functions, Scope & Execution Content

5/1/2023

* Creating function

```
function f_name(___)  → defining/declaring a f(n)
{
   //_____
}
f_name(___);          → invoking a f(n)
```

Note

Parameter vs arguements

parameters

```
function add (a,b)
{
   return a+b;
}
```

```
add(5,10);
```

arguements (a for आर्ग्युमेंट)

If it is in script.js file won't be printed in console

→ Function may/may not return a value.
→ Blank returns are allowed (to exit the function)

* Parameters & Arguements

```
function f(p)
{
   clg(p);
}
```

no parameter passed

```
f();
```

p is of type undefined → undefind ~~is registered~~

Sol^n: Give default value

```
if(!p)
{ p="Hi"
}
```

modern

```
function f(p="Hi")
{
   clg(p);
}
```

classmate
Date
Page

* Using Rest operator in functions

```
function prints(...list)
{
    clg(list);
}
```

Rest op^r

⇒ all the params will be in list, as an array

```
for (const num of list)
{
    clg(num + "\t");
}
```

loop using iterator

* objects as params

```
function details(user)
{
    clg(`The user ${user.name}
        lives in ${user.address}`);
}
```

Note

```
window.alert("Hi!");
alert("Hi!");
```

both are fine as window is to topmost object. Hence, its props are accessible without mentioning

```
details({name:'Saran', address:'Nala-Sopara'});
```

an object

* Global & Function Scope

→ Global - outside all scopes, accessible from anywhere

→ any var/const can be accessed in the current block and inner ones (w.r.t where it is declared.

classmate
Date
Page

Variable Shadowing

```
~~int~~ const a = 5;          -> This is shadowed by inner a.
function f()
{
    const a = 100;
    clg(a);                   -> prints 100
}
```

Note

① `Var` is not block scoped.

```
if(true)
~~function f()~~
{
    let a = 5;
    const b = 10;
    var c = 500;
}
```

```
clg(a); -> error (let)
clg(b); -> error (const)
clg(c); -> fine! (var)
```

as it's not block-scoped, means, it's not restricted to its block

but it is function scoped.

meaning scoping works on `var` but only in functions, and not in any other blocks such as if, loops, etc.

② a global `var` is added to the window object.

* Nested Scopes

Error (b is not defined) ← b cannot be accessed out of its scope

```
if (true)
{ let a=5
    if(true)
    { let b=6;
      clg(a+b);          -> works fine! (can access parent element)
    }
  clg(a+b);
}
```

=> applicable for other block as well

classmate
Date
Page

* Declaration VS Expression

function f(a) {
  clg(a);
}
	const f = function(a)
	{
	  clg(a);
	};
	• not supported (can't use before declaring)

• hoisting supported	

* Arrow Functions

fat arrow

```
const add = (a,b) => {
  return a+b;
}
```

```
// implicit return
const subtract = (a,b) => a-b;
```
→ automatically understood that return करना है।

```
// can leave off () if
   • single param
   • single line to run
const double = a => a*=2;
```

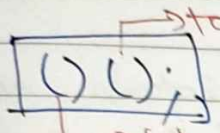
```
// returning an object.
const showObj = () => ({name:'Brad'});
```
→ add braces to avoid syntax confusion

```
// callback f(n) — f(n) calling f(n).
  const arr = [1,2,3];
arr.forEach( function(n) {
      clg(n);
});
```
→ (=>) can be omitted

classmate Date Page

* Immediately Invoked function Expressions (IIFE)

//declare & invoke simultaneously.
→ to avoid global scope pollution.




```
function () {
    clg("Hi!");
};
```

```
function hi() {
  clg("Hi");
  hi();
};
hi();
```

→ can apply recursion → (if infinite, browser's console crashes)
but can't be called exclusively.

```
//parametrized.
( function (a)
  { clg (a); }
) (5);
```
↳ passing the params

//5/12/2023

* function challenges.

1) Temprature converter
2) Min Max of Array

*** Note**

```
function minMax() {
  const arr
    = [1,2,3];
  const min
    = Math.min(...arr);
  const max
    = Math.max(...arr);
  return
    {min, max};
}
```
↓
returns an object
{ min : 1,
  max : 3,
}
↓
these names are added implicitly

classmate
Date
Page

* Execution Context

- When a JS code is run, a special environment is created (by browser or runtime engine such as Node) to handle transformation & execution of code.

- It contains the currently running code & everything that aids in execution.

There is a global execution content. And then for each function call there is a function execution content.

Memory	Execution (code)
name: 'John' age: '5' fn: {...} Variable Environment (variables and functions as key value pairs)	Thread of Execution line by line execution.

Note

JavaScript is **single-threaded** & execution is a single thread/process.

Synchronous

line by line execution

(however, JS does provide asynchronous properties)

* Phases of execution content

- **Memory creation Phase**
  - → global obj created (browser → window, Node → global)
  - → this obj created, ref to global obj.
  - → memory heap set up for variables & f(n) ref.
  - → stores them (both) in global content, sets them to undefined

- **Execution phase**
  - → executes code line by line
  - → when a f(n) call occurs, new content is created and all of the steps happen for it (from this obj onwards)
  - → values of vars and implementations of f(n) are allocated memory in this that

Example

```
let x = 100;
let y = 50;
function f(x,y){
   let sum = x1+x2;
   return sum;
}
let sum = f(x,y);
```

Global EC

Memory Creation
1) x, y created, assigned 'undefined'.
2) f(x) created, code saved
3) sum is allocated memory

Execution Phase
1) Place 100 in x
2) Place 50 in y
3) Skip f(x) as it is not a f(x) call
4) f(x) EC is created
5) 150 stored in sum

f(x) EC

Mem. Creation
1) x, y, sum created
~~2) calculation~~

Execution
1) calculation
2) 150 stored in sum
3) sum is returned to global EC and f(x) EC is done with

(150)

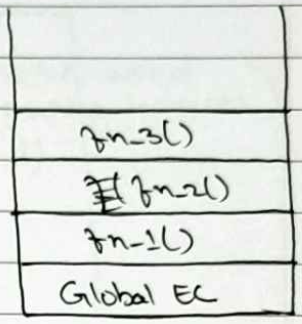
* Debugging in browser

- go to sources tab in inspection
- click on the JS file
- click on any line to add breakpoint
- Reload the page.

classmate
Date
Page

* The call Stack

- Stacks of f(n) to be executed
- Manages Execution contexts
- Follows LIFO



call Stack

Example

```
function f()
 { clg('f'); }
function g()
 { clg('g'); }

f();          → push f
              → pop f
g();          → push g
              → pop g
```

```
function f()
 { g(); }
function g()
 { clg("Hi"); }

f();          → push f
              → push g
              → print (Hi)
              → pop g
              → pop f
```

7/12/2023

* Logic & control flow

```
if(x>y)
{ clg("x is greater than y"); }
else if()
{ ||——— }
 else
{ ||——— }
```

not compulsory

`x==y` `x!=y`
compares x & y (only value)

`x===y` `x!==y`
compares x & y, and their data type.

Shorthand Syntax

```
if (x>y)
    x+=y, ← multiple stmts seperated by comma( no curly braces)
  clg (`x = ${x}` and y = ${yy}`);
else
    x-=y,
    clg (`The else part`);
```

Logical operators

`a>b && a%2==0` — Logical AND

`a>b || a%2==0` — Logical OR

`(!a>b)` — NOT

`a & b` — bitwise AND

`a | b` — bitwise OR

* Switch statement

- Syntax same as java (old one)
- ~~cases~~ cases allow ranges: (in exceptional scenes)

- switch (——) a var. or an expression

```
switch(true){
  case x<12:
    //—
    break
    ⋮
}
```

- unique labels required

Q Calculator using Switch case

classmate Date Page

* Truthy & Falsy values

"any to boolean"

false | 0 | "" | '' | null | undefined

NaN all these are falsy values

everything else is true. (truthy)

'0' (zero in a string)
'false' (false as string)
[] (empty array)
{} (empty obj.)
fn(){} is empty

→ truthy values

11/12/2023

* Loose Equality

false == 0 | null == 0 → all these return true

"" == 0 | null == undefined

That's why === is better

* Logical Operators

(bool_exp && bool_exp)
(bool_exp || bool_exp)

① && returns first falsy value. If none, returns last value.

~~for~~ const x = 5 && 20;
gives 20

Real life example

```
const posts = [...];
posts.length > 0
  && clg(posts);
```

|| returns first truthy value, or else last value.

```
const b = 0 || null ||undefined;
```
↳ b is undefined

* Nullish Coalescing operator (??)

```
10 ?? 20 → 10        0 ?? 30 → 0
null ?? 20 → 20      undefined ?? 50 → 50
```

returns right operand if left one is null or undefined, else returns left operand.

* Logical assignment operators

①
```
if (!a)
  a = 10;
```
→ `a = a || 10;` → `a ||= 10;`

②
```
if (a)
  a = 20;
```
→ `a = a && 20;` → `a &&= 20`

③
```
if (a === null || a === undefined)
  a = 5;
```
→ `a = a ?? 5`

↓

`a ??= 5;`

classmate
Date
Page

* Ternary Operator

```
age >= 18 ? console.log("ELIGIBLE") : clg('NOT ELIGIBLE');
```

→ condition ? If_true_stmt : If_false_stmt.

~~both the statements will be logged~~

```
const redirect
    = isAuthorized ?
      (alert('Logged In'), '/dashboard') :
      (alert('Not logged in'), '/login');
```

redirect will be set to /dashboard or /login

* Shorthand (if there is no else)

```
isAuth ? clg('Logged In!') : null
```

⇓

```
isAuth && clg('Logged In');
```

* Loops

for loop

(use let, const will give error)

```
for (const i = 1; i < 10; i++)
  clg('Number: ${i}');
```

* break & continue same as Java.

classmate
Date
Page

* while loop — same as java
* do-while loop — same as java

* For ---Of Loop (foreach जैसा has : की जगह of)

modern way to loop through iterating items.

```
const list = [1,2,3];
for (const item of list)
  clg(item);
```

Works for
- list (array)
- object
- String (letters)
- maps

```
const map = new Map();
map.set ('a', 1);
map.set ('b', 2);

for (const [key, value] of map)
  clg (key, value);
```

output

a	1
b	2

* For ---In Loop (It returns keys)

to loop through literals of an object

```
const colors = {
    c1: 'Red',
    c2: 'Green',
    c3: 'Blue'
  };

for (const label in colors)
  clg ( label, colors[label]);
```

→ If you run this loop on an array, it will give keys (0,1,2, etc. ---)

classmate
Date
Page

* Higher Order Array methods → uses a callback function

* Array.forEach() (doesn't return anything)

```
arr.forEach(
    (elem) => {
        clg(elem);
    });
```

```
arr.forEach(
    (elem, inden, arr) =>
        clg(inden, '-', elem);
```

can pass inden & whole array aswell

* Array.filter() (returns an array)

```
const evenNums
    = nums.filter(
        function (num){
            return num%2===0;
        });
```

shorter

```
const evenNums = nums.filter(num =>num%2===0);
```

* Arrays.map() except that it can return anything (even modify data)

```
const doubles
    = nums.map((num) => num *2);
```

```
const minPerson
    = Person.map((p) => {name : p.name,
                         age : p.age});
```

Note: We can chain map methods.

```
const DoubleOfSquares=
    nums.map((n) => n*n)
        .map((n) => 2*n);
```

Similarly, we can chain multiple methods to get desired result.

(Eg) Square of even nos.

```
const evenSquares =
  nums.filter( n => n%2===0)
      .map ( n => n*n);
```

* Arrays.reduce() to reduce an array to a single element, based on some calculation

```
const cart = [1, 23, 57];

const total = cart.reduce
  ((acc, curr) => acc+curr), 0);
```

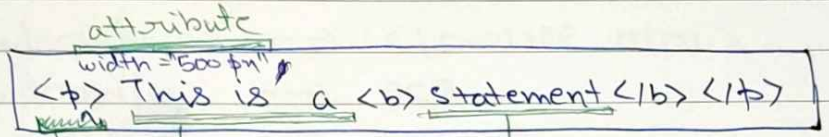
→ takes two params
- a function
- initial value(0)

↳ function takes two params
- accumulator
- current element

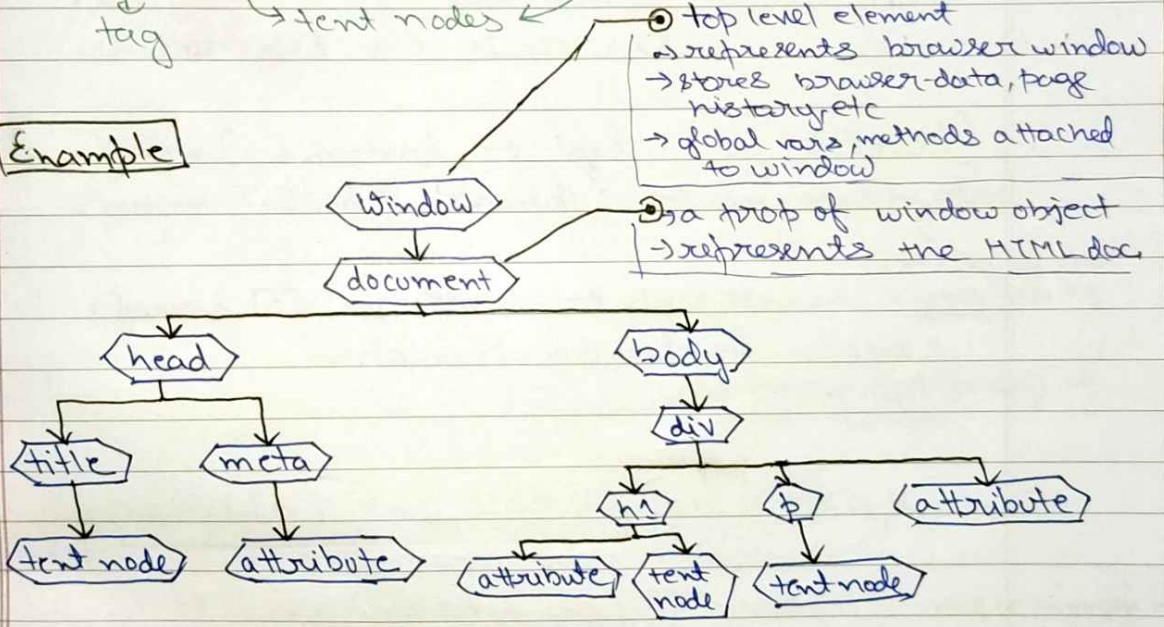
Some Assignments on Array methods

* Document Object Model (DOM)

- Programming interface for web/~~HTML~~ HTML elements
- Structure that we can interact with via JS (or other languages as well) which compile to JavaScript (typescript, coffeescript)
- includes tags, attributes, text nodes, etc.
- Represented as a tree structure.



Example



Note

console.dir(document)	console.log(document)
• shows the whole structure of the object passed.	• used for general logging and printing information
• in a represented as interactive list.	• human-readable representation

* Properties of Document object

[] document.all; → (deprecated) all of the page

[]
- documentElement → head, body, etc.
- head
- body
- head.children → children of head (title, meta)

- doctype → '<!DOCTYPE HTML>
- domain → 127.---
- URL → full URL
- characterSet → char set

- forms[0].method → 'get' (by default)
- forms[0].id = 'New-id'; → can modify too

- forms[0].className → class Name (a string of class names sep by spaces)
  - classList → list of classes
    ↳ a DOMTokenList (array) of class names

.images[0].src → images DOMTokenList

↳ These access methods aren't recommended.

Note

Soln

Array.from(document.forms).forEach((f) => clg(f));

↳ DOESN'T WORK bcoz this isn't an array

classmate
Date
Page

* DOMSelectors (Single Element)

☆ document.getElementById('app-title');

```
//getting attributes
.getElementById('_').getAttribute('class');  ✓
```

.class ; ✗

.className ; ✓

```
//setting attributes
.id = 'new-id' ; → adds the attribute if not present
.setAttributes('class','class-1');
```

// Accessing element content

① title.innerText = "Hey Guyzzzz";
② .textContent = "Do you know what";
③ .innerHTML = " thisss iszz?";

// Styles

```
title.style.color = 'red';
.padding = '10px';
.backgroundColor = 'red';
```

camel case

textContent	innerText
includes scripts & styles	includes scripts & styles
removes & retains spaces	normalizes trailing spaces/leading spaces
more widely supported	—
more performant	—

classmate
Date
Page

//Query Selector ES6 update (works somewhat like css selectors, but first element only)

```
document.querySelector('h1');  → first h1 element

('#my-id');  → id
('.my-class');  → class

('input[type="text"]');  → attribute sel.

('li:nth-child(2)');  → pseudo selectors
//Applicable on any object (not just document)
const li = document.querySelector('ul');
clg(li.querySelector('li:nth-child(2)');
```

//4/1/2023

Note

Use query selector all the time instead of specific methods such as getElementById, getElementsByClassName.

Query Selector All

for conciseness (document)

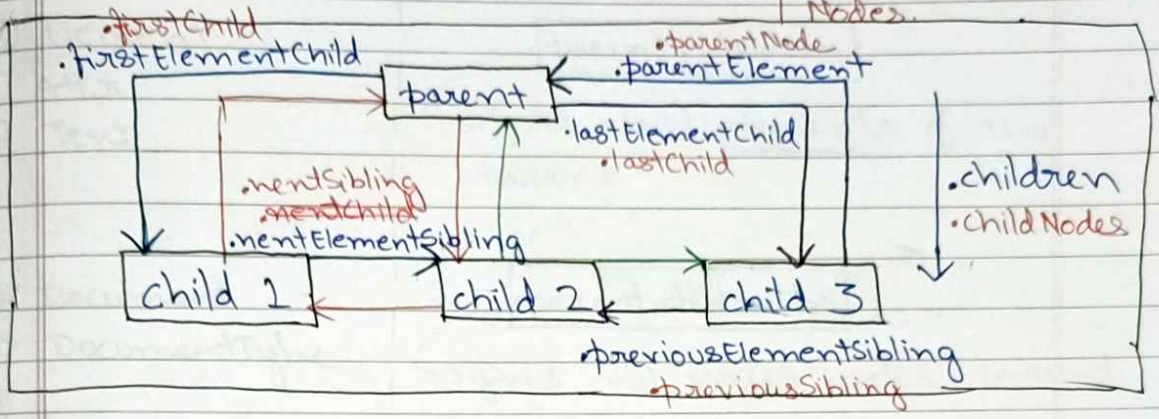
```
const # list = dcm.querySelectorAll('li');

list.forEach( (item) => { clg(item) });

document.getElementsByClassName('red-bg');
```

forEach won't work on it as this (HTMLCollection) isn't an (array).

classmate
Date
Page

* Working with DOM Elements

Words written in Red are applicable for all kind of Nodes.



const parent = dcm.querySelector('.parent');

op = parent.children; → collection of childs

= parent.nodeName; → name of element (ul, li, h1, etc.)

= parent.className; → class Name

= parent.firstElementChild; → first child

*

= parent.children[0].parentElement; → parent itself.

const c2 = ~~dcm~~ parent.children*[1];

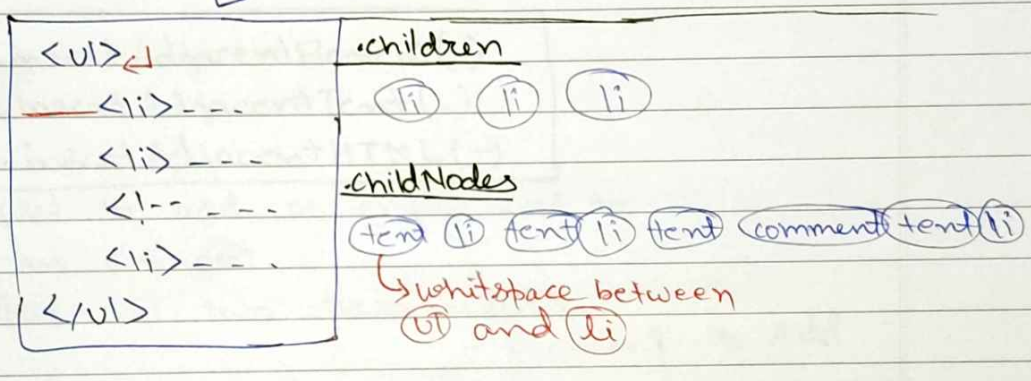
op = c2.parentElement.children; → all siblings of c2, including c2 itself.

* DOM Node Types

1. Element
2. Attr
3. Text
9. Document
10. DocumentType
11. ...
12. ...

parent.children
→ gives child elements of the parent

parent.childNodes
→ gives all nodes under parent element.



* Creating & Appending Elements

11/1/2024

```
const div = document.createElement('div');

div.className = 'my-elem';
div.id = 'myelem';

div.setAttribute('title', 'this comes upon hover');
```

```
div.innerText = 'Hello Duniya';
```

OR

```
const text = document.createTextNode('Hello World');
div.appendChild(text);
```

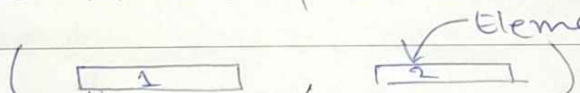
Q NotesApp

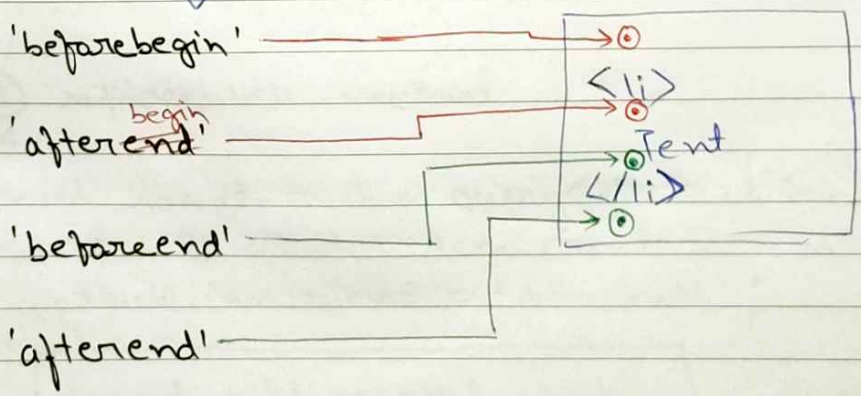
//6/1/2023

* Few OtherMethods to add Nodes

- .insertAdjacentElement(-)
- .insertAdjacentText(-)
- .insertAdjacentHTML(-)

- used to add an entity wrt to an element
- takes in two parameters





Example

```
const li = dcm.querySelector('li');
const item = dcm.createElement('li');
item.innerText = 'New Item';

li.insertAdjacentElement('beforebegin', item);
```

* insertBefore method

```
ul = dcm.querySelector('ul');
li_3 = dcm.querySelector('li:nth-child(3)');

ul.insertBefore(item, li_3);
```

ul → called on parent
item → Element to add
li_3 → before this item

Note:- There is no inbuilt insertAfter method

(Q) Custom insertAfter Method
11/8/1/2024

* replacing Elements

(M1) replaceWith method

```
const target = dcm.querySelector('h1');
const attack = dcm.createElement('h2');
attack.innerText = 'Attacked!';

target.replaceWith(attack);
```

classmate
Date
Page

(M2) outerHTML

```
target.outerHTML = '<h2> Attacked </h2>';
```

↳ includes the element's HTML as well

⊙ Changing all ~~the~~ siblings at once

```
list.forEach(
    (item, inden) =>
        item.innerHTML = `Item ${inden+1}`
);
```

- You can put any parameter names
- first parameter presents iterator
- second one represents inden no.
  (in case of unordered data structures, NOT APPLICABLE)
- if only one parameter → iterator
- if more than two → remaining param(s) won't be automatically populated (undefined)

⊙ ~~M~~ Based on condition

→ run the same forEach loop and in the loop, write

```
item.innerHTML
    = (inden === 1) ?
        'Second item' : 'Other Items';
```

↳ ternary operator

⊙ replaceChild Method

```
parentElem.replaceChild(newChildElem, oldChildElem);
```

* Removing Elements

```
element.remove();
```

OR

```
parentElem.removeChild(childElem);
```

* Adding & Removing Style & Classes

item.className → string of class names seperated by spaces
item.classList → DOMTokenList of class names

```
item.className = 'new-class';
```
→ all classes will be removed and this new class will be added

```
item.classList.add('new-class');
```
add if ~~present~~ absent

```
item.classList.remove('old-class');
```
remove if present

```
item.classList.toggle('new-class');
```
add or remove whichever possible

```
item.classList.replace('old', 'new');
```

classmate
Date
Page

⊙ change style

```
item.style.color = '#000';
item.style.lineHeight = '12px';
```
↳ camelcase for multiple words

* Events

actions/occurences happening in system/program which system can tell the program about, and program can then respond to it.

Eg
click, typing, hovering, submit, close window, dragging element, resizing element

* Event Listeners

(M1) Inline Event Listeners

In element, write an attribute 'onclick'

```
<h1 onclick="run()">Hey Guyyyy!</h1>
```

In JS, write the method run

```
function run()
{
    clg('The function Ran! (Running Emoji)');
}
```

(M2) .onclick

```
clearBtn.onclick = function () { clg('clear out!'); };
```

(M3) using addEventListener()

```
clearBtn.addEventListener
    ('click', () => clg('Btn Clicked'));
```

multiple event listeners can be added in this and M1, not in M2.

Note:-

```
Btn.addEventListener('click', () => clg('Clicked!'));
Btn.addEventListener('click', () => alert('OKKay?!'));
```

Order matters here, as `alert` is a synchronous function/blocking code.

When you want to have a function and an event listener to do the same thing, pass that function in the event listener.

- You can also run the event from within the script.

```
clearBtn.click();
```

classmate
Date
Page

⊙ setTimeout method

asynchronous function to ~~remove~~ hold a script.

```
setTimeout ( cg('finally, 9 got printed!'), 5000);
```

millisec(s)

* Comparing different ways to clear a list

```
① itemList.innerHTML = '';

② items.forEach ((item) => item.remove
        ());

//Most performant way
③ while(itemList.firstChild){
    itemList.removeChild (itemList.firstChild);
  }
```

- Baingan
- Bhupendra Jogi
- Kacha Badaam

Clear All

* Mouse Events

combinations =>

- click
- dblclick (double click)
- contextmenu (right click)
- mousedown
- mouse up
- wheel
- mouseover (hover)
- mouseout (hover off)
- dragstart
- drag (keeps running)
- dragend

Note

for any event, the default action is not removed when you add an event listener. This can be issue in situations such as a right click where you want to show a custom content menu.

To fix that, you can create an event listener to remove the default action.

```
profilePic.addEventListener('contextmenu', (event)=>
    event.preventDefault());
```

OR using inline event handler-

```
<button onclick='f(event)'> Click Me </button>
<script>
  function f(event){
    event.preventDefault();
  }
</script>
```

* Event Object

10/1/2024

```
logo.addEventListener('click', (e) => {
});
```

we're writing here

console.log(e); → PointerEvent object
e.target; → elem that triggered the event
e.currentTarget; elem that event is attached to
target & currentTarget different when listener is on some parrentElement, such as body.

Epoch - a predefined point in time used as reference.
Eg:- UNIX Epoch (00:00:00 UTC on Jan 1, 1970)

Note

```
logo.addEventListener('click', f1);
function f1(e)
{ clg(e); }
```

Event object passed automatically to f1.

This works!

e.type; // type of event (click, mouseover, etc.)

e.timestamp; // a number (represents webpage's time in milliseconds, wrt an epoch time)

e.clientX; left to right position
e.clientY; top to bottom position
} of mouseclick relative to window

e.offsetX;
e.offsetY;
// relative to ~~win~~ element
Unit: pixels

e.pageX; // wrt page
e.pageY;

e.screenX;
e.screenY;
// entire monitor

There are specific no MouseEvent, or other cases, its values can be unpredictable

* preventDefault() method

Examples

```
link.addEventListener('click', fakeSubmit);
function fakeSubmit(e)
{ e.preventDefault();
  link.outerHTML = '<h1> Form Submitted.
    Thank You. </h1>';
}
```

→ prevents default action of click ~~for~~ from running

In arrow function, bracket around parameter isn't required if there is only one parameter.

Note :-

events such as drag are continous. They run on repeat while youire dragging, and faster if you drag faster.

It can be useful in making an app involving a canvas, which shows live cursor position in the status bar.

To avoid continous running, you can use alternatives such as dragstart, dragend.

Q A simple Drawing App (SimpleDrawingApp)

* Keyboard Events & Key properties

- Keypress → runs once, even if you hold the key
- keyup → when key is released.
- keydown → when key is held down (continously fires as long as held)

- e.key → K, L, O, (, -, ↑ (space, enter non-character keys not included) on 'keypress' but included on 'keyup' & 'keydown'
- e.keyCode → numbers assigned to keys
- e.code → KeyF, Digit1, Space

Key press	output key	output keyCode	output code (value attached to key not it's value)
a	a	97	KeyA
Shift a	A	65	KeyA
1	1	49	Digit1
Shift ! 1	!	33	Digit1
Space	_	32	Space
Enter	Enter	13	Enter

Keypress works for character keys
Keyup & keydown works for all keys

e.repeat ; → true or false (true when holding down the key)

But

```
tentBox.addEventListener('keydown', (e) =>
{
    if(e.repeat)
      clg('Noice!');
}
```

This will run continously & 'Noice' will be printed on repeat as long as a key is held.

```
e.shiftKey;
e.ctrlKey;
e.altKey;
```
T/F

Ex
```
if(e.ctrlKey && e.shiftKey
    && e.key=='t')
  clg('Opening previously
       closed tab');
```

⑥ Keycode MiniProject (KeyPressInfo)
(displays key Info when you press a key.)

11/1/2024

* Input Events

⊙ You can use keydown but it won't work for non-text inputs (checklist, sliders, etc.)

⊙ input data can be accessed by ~~getting~~ getting value of the target

e.target.value

```
inputElem.addEventListener('input', (event) =>
{
  // your code here
});
```

clg(event.target.value);

↳ This gives the whole value in the input and not just the current key pressed, unlike the case with keydown event.

- for select lists, value parameter selected right now is returned.
- change event can also be used here.
- for checkbones, its checked not value

event.target.checked → true or false

* focus & blur events
  - when input is active (clicked upon)
  - when input is inactive (clicked out of)

* Form Submission & FormData object

- Usually, submit passes data to a backend server using some method. (GET, POST, ...)
- Here, we'll take control of the submit process.

```
form.addEventListener('submit', (e) => {
    e.preventDefault();
    //your code here
});
```

Even if the form has no `action` attribute, it'll submit data to same page (page reload)

// getting values

→ one way is to use event.target.----

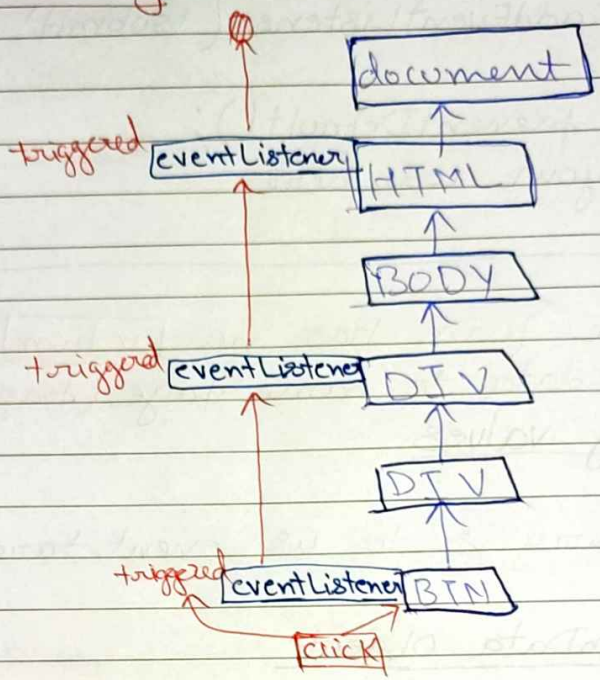
→ FormData Object

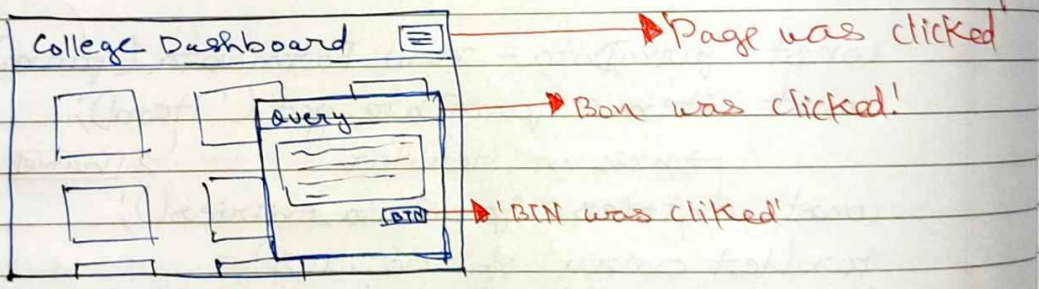
```
const formData = new FormData(form);   // form element
const item = formData.get('item');     // name attribute
const entries = formData.entries();    // gives an iterator
for (let entry of entries)
{
    clg(entry);   ⟶ ['item', 'potato']
}
```

array of `name` and `value`

classmate
Date
Page

* Event Bubbling





stopPropagation method

```
btn.addEventListener('click', (e) => {

    alert('btn was clicked');
    e.stopPropagation();
});
```

↳ stops bubbling right there

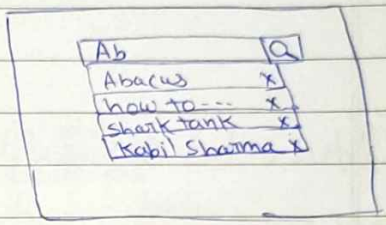
- There is also stopImmediatePropagation().

classmate Date Page

* Event Delegation & Multiple Events

Say on Google search, you are shown a short list of recent searches, and each search history has an [X] to delete that history.

Now, we want to add event listener to all the [X]s.



One way is to use [forEach].

(M2) → adding event listener on parent

```
list.addEventListener('click', (e) =>
{
    if(e.target.tagName === 'BUTTON')
        e.target.parentElement.remove();
}
```

→ and checking if click was actually on [X] button

* Page Loading & Window Events

Note

If script tage is in [head], javascript runs before page load.

Solution

```
window.onload = () =>
{
    // ------
}
```

```
window.addEventListener
('load', (e) =>
{
    //---
})
```

classmate
Date
Page

load event	DOMContentLoaded event
waits for loading of all resources (DOM, images, icons, videos, etc.)	as soon as DOM is loaded.

⊙ <script src="script.mjs" defer></script>

defer → waits for page load.

Means that JS will be fetched asynchronously while HTML parsing continues, but will be executed only after the document is fully parsed.

⊙ 'resize' event
and log window.innerWidth, window.innerHeight

⊙ 'scroll' event (or window.onScroll)

window.scrollX window.scrollY
no of pixels scrolled (from left/top)

⊙ 'focus' & 'blur' events
Example
→ Online Exams ('no tab switching')
→ saving memory when web page not being viewed.

This was Part 1.
@EVERYTHINGSHYAM
all small

11:53:59 Video Finisshh!