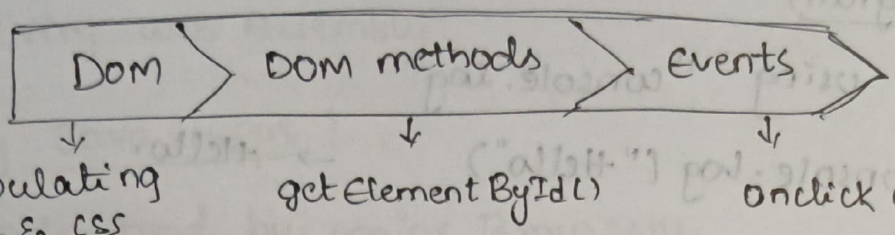
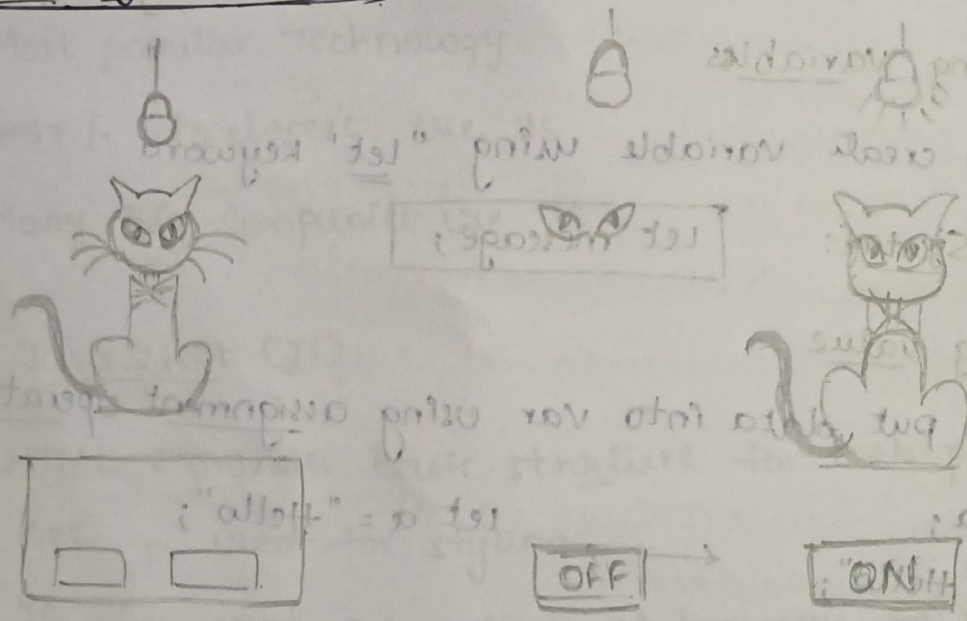
② DOM and Event Fundamentals.



Cat & Light Example

Breaking Down into parts



How do we manipulate HTML & CSS using JS?

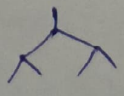
Document Object Model (DOM)

→ DOM is structured representation of content in the document created by Browser

HTML DOM

→ Tree structured representation of HTML doc created by browser.

→ It allows JS to change the document structure, style & content



→ will give html doc to browser, the browser will understand that HTML doc & create a tree structured representation for " ".

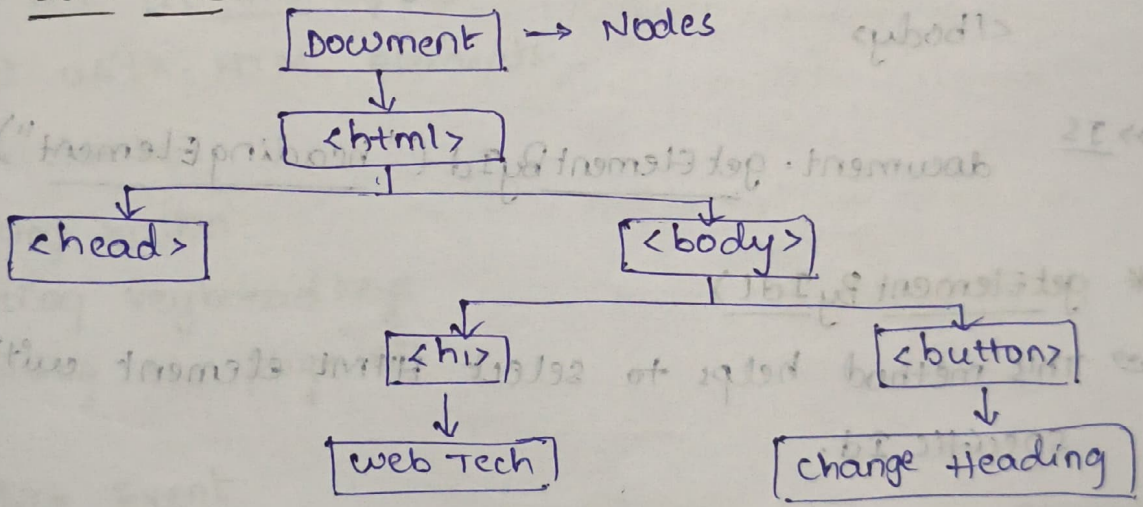
Eg
```
<html>
  <head></head>
  <body>
    <h1> web Tech </h1>
    <button> change Heading </button>
  </body>
</html>
```

O/P

Web Tech

change Heading

HTML DOM tree



→ each html element will be present as a NODE in html DOM tree. (even content also a Node)

→ DOM tree rep. HTML DOC as Nodes.

→ each node is referred to an object

Document object

Document

→ It is entry point of DOM

→ To access any HTML element, you should always starts with accessing the document object first.

Manipulating HTML & CSS

Steps:

1) select Heading element
2) manipulate text content of Heading element
3) manipulate styles of Heading element.

① Selecting Heading element:

→ for this will use "id" attribute

» html

```
<body>
  <h1 id="headingElement"> web Tech </h1>
  <button> change Heading </button>
</body>
```

» JS

```
document.getElementById("headingElement")
```

* getElementById()

→ This method helps to select HTML element with a specific Id.

② Manipulating Text content

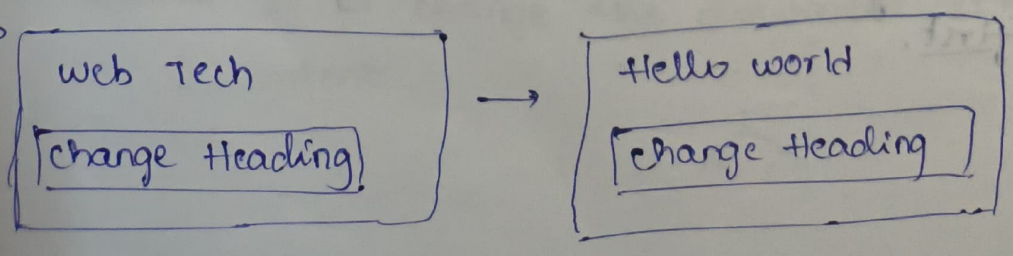
* textContent property

→ To manipulate text within html element we use textContent property.

» JS

```
document.getElementById("headingElement").textContent =
                                        "Hello world";
```

O/P



⑤ Manipulating styles

* style property

→ style property is used to get / set specific styles of HTML element using different CSS properties

>> JS.

document.getElementById("headingElement").style.color="red";

How to manipulate HTML & CSS Based on User Actions?

Events

→ Events are actions by which the User & Browser interact with HTML elements.

Actions can be:

↳ clicking button

↳ pressing keyboard keys

↳ scrolling the page etc.

* onclick Event

→ onclick event occur when user clicks on HTML Element

onclick Attribute

>> HTML

```
<body>
  <h1 id="headingElement"> web Tech </h1>
  <button onclick="manipulateStyles()"> change Heading </button>
</body>
```

(event / funct_name)

Defining Function

(Py)

```
def funct_name():
```

(JS)

```
function funct_name() {
}
```

>> JS

```
function manipulatiStyles() {
    Console.log("Hi everyone")
}
```

→ whenever we click the button, this function will get executed & print Hi everyone.

User Actions

Manipulating styles

>> JS

```
function manipulatingStyles() {
  document.getElementById("headingElement").textContent
                                = "Hello world";
  document.getElementById("headingElement").style.color =
                                "blue";
}
```

— × —

Cat & Light Example

Observations

↳ switch status changes
↳ Bulb goes on and off
↳ Cat become visible & invisible.
↳ switch (button) bg color changes

Steps

1) make switches listen to user actions
2) make bulb Go on and off
3) make cat visible & invisible
4) updati switch status

5) change bg color of switch

① making switches listen to user Actions

>> HTML

```
- - -
- - -
<div class="switch-board">
  <h1 class="switch-status"> switch on </h1>
  <button class="off-switch" onclick="switchOff()"> Off
  </button>
  <button class="on-switch" onclick="switchOn()"> ON
  </button>
</div>
```

→ we've to define these functions in JS.

② make bulb go on & off

→ first uniquely identify html element for bulb.
→ specifying id.

>> HTML

```
<div class="dark-background text-center">
  <div>
    <img src="__" class="bulb-image" id="bulbImage"/>
  </div>
  =
</div>
```

>> JS

```
document.getElementById("bulbImage").src="__";
```

③ make cat visible & invisible

>> HTML

```
<div>
  <img src="__" class="cat-image" id="catImage" />
</div>
```

>> JS

document.getElementById("catImage").src = "___";

④ updata switch status

→ By specifying id.

>> HTML

```
<div class="switch-board">
    <h1 class="switch-status" id="switchstatus">switched on
    </h1>
</div>
```

→ changing text content.

>> JS.

```
document.getElementById("switchStatus").textCont =
                                    "switched Off";
```

⑤ changing bg colors of btn

→ we need to change colors for both buttons.

→ Add ids to both buttons

→ then change bg color of btn

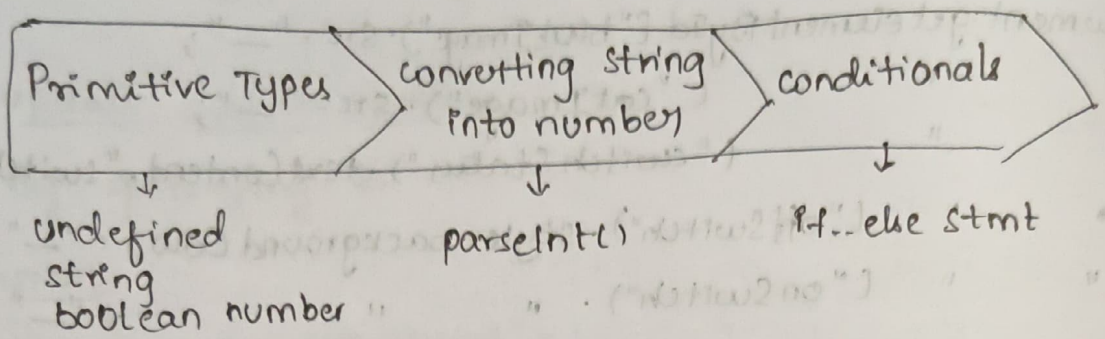
>> HTML

```
<button class="off-switch" onclick="switchOff()"
                    id="offSwitch">OFF</button>
<button class="on-switch" onclick="switchOn()"
                    id="onSwitch">ON</button>
```

>> JS.

```
function switchOff() {
document.getElementById("onswitch").style.backgroundColor="___";
    "       "       "  ("offSwitch"). "    "       "___";
}
```

③ Primitive Types & conditionals



Counter Application

Counter

Decrease Reset Increase

what kind of other values can we assign to variables?

* values → Primitive Types
  → Reference Types.

* Primitive Types
  - ↳ Number
  - ↳ Boolean
  - ↳ string
  - ↳ Undefined etc.

① Number

→ Python: Based on fractional part, no: are of integer and float data type. To find type → type()

→ JS: All numbers are of Number type.
To find type of value → typeof()

```
>> a = 900        (Py)
   b = 7.6
print (type(a))  (int)  →
print (type(b))  (float)
```

```
>> let a = 900;          (JS)
   let b = 9.6;
console.log (typeof (a));  (number)
Console.log (typeof (b));  (number)
```

JS

② Boolean

→ Python: Boolean val are True / False. true ✓ True ✗

→ JS: Boolean val are true / false false ✓ False ✗

>> JS
```
let isApproved = false;
console.log (typeof (isApproved))   (boolean)
console.log (typeof (true))         (boolean)
```

③ string

→ string is stream of characters

→ should be enclosed with

- single quotes - ' '
- Double " " " "
- Backticks - ` `

④ Undefined

→ If value is not assigned to variable, then it takes as undefined as its val.

→ In JS, undefined refers to val not being assigned

>> JS
```
let age;
console.log (age);
console.log (typeof (age))
```

O/P
undefined.
undefined.

Counter Application

Steps

1) make Buttons listen to user Actions
2) Updat counter val based on button click.
3) Reset Counter val
4) change color of no: based on counter val.

① making Buttons listen to User Actions:

→ Define function for each action.

>> JS ↳ function onIcrement()
↳ " onDecrement()
↳ " onReset()

→ for listening to user action, we define onclick Attr where we make funct() listen to user Action.

>> HTML

```
<div>
  <button class="button" onclick="onDecrement()">DECREASE
                                                </button>
    "     "     "   onclick="onReset()">RESET  "
    "     "     "   onclick="onIncrement()">INCREASE
                                                </button>
</div>
```

* Event Listeners

→ functions which are defined in JS are called event listeners.

→ Bcz, based on events, they are triggered, then action is performed.

② updati counter value

* Incrementing counter value

→ we need to change value for this specific html element where there is counter value.

→ specifying id.

>> HTML

```
<div> ...
  <h1 class="counter.heading"> counter </h1>
  <p id="counterValue" class="counter-value"> 0 </p>
...
</div>
```

Accessing Counter Element in JS.

>> JS

```
let counterElement = document.getElementById("counterValue")
```

Accessing counter element value

>> JS

```
let counterElement = document.getElementById("counterValue");
function onIncrement() {
    let prevCounterValue = counterElement.textContent;
    let updatedValue = prevCounterValue + 1;
    console.log(updatedValue);
}
```

O/P 01

→ Inside counterElement, text is present which is counterValue
→ we'll access that context & that val is stored in prevCounterValue. variable.
→ next updati val by 1 & stored in var.
→ print " value. & O/P will be 01

* prevCounterValue & updatedValue types are "strings"
* we are adding '+1' to string ("0"+"1") → 01
So that we are not getting expected output.

How can we convert String into Number? '0' +1 → 1
0+1 → 1

(*) parseInt()

→ parseInt() function accept string & convert it into Integer.

>> JS

```
let updatedValue = parseInt(prevCounterValue) + 1;
```

updating counter value

>> JS

```
function onIncrement() {
    =
    counterElement.textContent = updatvalue;
}
```

* Decrementing counter value

Same as Incrementting but "-1"

>> JS.

```
function onDecrement() {
    let prevCounterValue = counterElement.textContent;
    let updatidvalue = parseInt(prevcountervalue) -1;
    counterElement.textContent = updatidvalue;
}
```

O/P

(3) Resct counter value

→ just updati text content with '0'

>> JS.

```
function onReset() {
    updatid value = 0
    counterElement.textContent = updatidvalue;
}
```

④ change color of NO: Based value (Neg = Red; +ve = green)

✱ conditional statements

→ conditional statements allows you to execute block of code only when specific condition is true.

If-else statement

Python	JS

if condA:
  ==
elif condB:
  ==
else:
  ==
	if (condA) {
	  ==
	}
	else if (condB) {
	  ==
	}
	else {
	  ==
	}

→ Chang color of counterElement acordingly.

Eg counterElement.style.color = "red";

— x —

JS code

» let counterElement = document.getElementById("counter value");

function onDecrement() {

let prevCounterValue = counterElement.textContent;

let updatidvalue = parseInt(prevCountervalue) - 1;

counterElement.textContent = updatidvalue.

if (updatidvalue > 0) {

counterElement.style.color = "green";

}

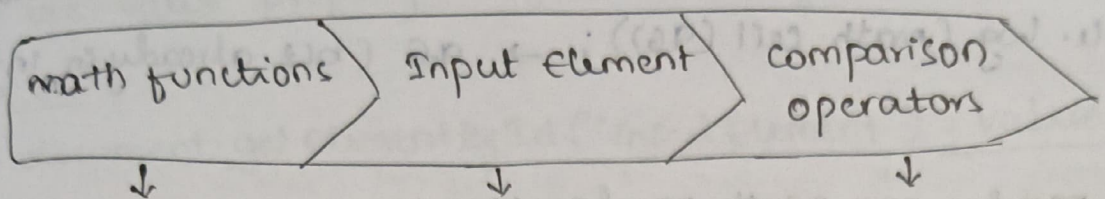
else if (updatidvalue < 0) {

counterElement.style.color = "red";

}

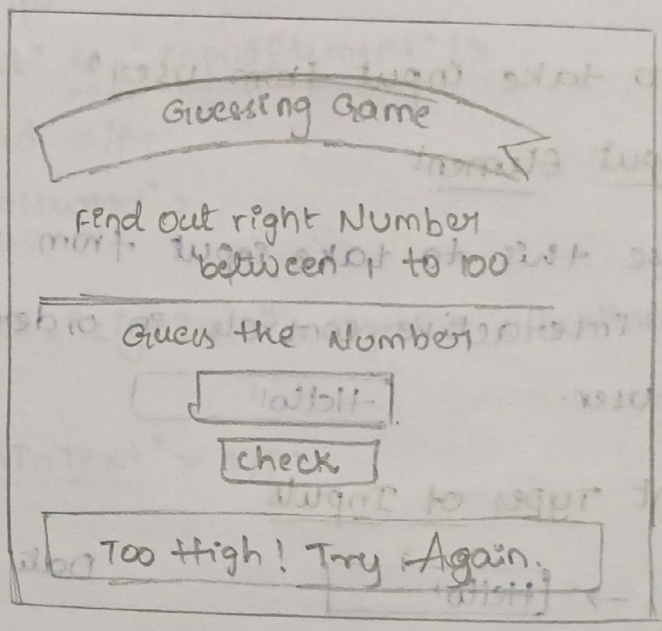
```
    else {
      counterElement.style.color = "black";
    }
}

function onIncrement() {
    let prevCounterValue = counterElement.textContent;
    let updatedValue = parseInt(prevCounterValue) + 1;
    counterElement.textContent = updatedValue;
    if (updatedValue > 0) {
      counterElement.style.color = "green";
    }
    else if (updatedValue < 0) {
      counterElement.style.color = "red";
    }
    else {
      counterElement.style.color = "black";
    }
}

function onReset() {
    let updatedValue = 0;
    counterElement.textContent = updatedValue;
    counterElement.style.color = "black";
}
```

④ Input Element & Math Functions



- Math.ceil()
- Math.random()
- text input
- password input
- Loose Equal to (==)
- Strict Equal to (===)

Guessing Game



How to generate random no: in JS?

✱ **Math.random()**

→ return random no: (float value) in range 0 to less than 1.

0 <= random no: < 1

Eg console.log(Math.random()); O/P 0.5327065551

Random no: from 1 to 100

» Math.random() * 100

→ generate blw 1 to 100. but random no: should be int not float.

✱ **Math.ceil()**

→ rounds no: up to next largest integer.

>> JS

```
console.log(Math.ceil(95.9066...));  → 96. (next int val)
console.log(Math.ceil(96));  → 96 (bcz already in int)
```

>> JS

```
let randomno = Math.random() * 100;
Console.log(randomno)                    41.906342537
console.log(Math.ceil(randomno));        42.
```

How to take input from user?

HTML input Element

→ We use this to take input from user.

→ Create interactive controls in order to accept data from user. [Hello!!]

Different Types of Inputs

Text → [Hello!]

Date → [19/2/24]

password → [.....]

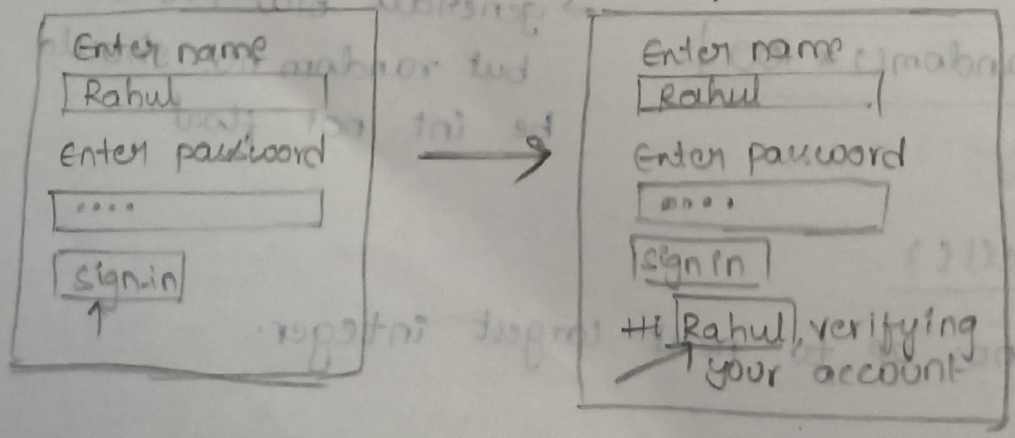
Radio → ○ Male ◉ Female

checkbox → ☐ car ☐ bike

>> HTML

```
<body>
    <p> Enter name </p>
    <input type = "text" />   # take i/p from user
</body>
```

How to access values entered by user?



Value Property

→ use value property to get value of HTML Input element

» JS

```
document.getElementById("inputElement").value;
```

Sign-In

» HTML

```
<body>
  <p> Enter name </p>
  <input type="text" id="inputElement" />
  <p> Enter password </p>
  <input type = "password">
  <div>
    <button onclick = "signIn()"> Sign In </button>
  </div>
  <p id = "signInSignInText" > </p>
</body>
```

» JS

```
let inputElement = document.getElementById("inputElement");
let signInTextElement = document.getElementById("signInText");
function signIn() {
    let inputValue = inputElement.value;
# update text
    let verifyText = "Hi " + inputValue + ", verifying your Account";
    signInTextElement.textContent = verifyText;
}
```

O/P

←

Guessing Game

Steps

↳ Generate Random no:
↳ Access user entered value
↳ compare with Random no: & display result
↳ Handle Invalid user i/p.

① Generate random no:

>> JS

```
let gameResult = document.getElementById("gameResult");
let userInput = "                    ";
let randomNum = Math.ceil(math.random() * 100);
console.log(randomNum);
function checkGuess() {

}
```

② Accessing user entered val

>> JS

```
function checkGuess() {                    → should be int. so conv to int.
    let gussedNum = parseInt(userInput.value);
}
```

③ User Guess > Random Number

>> JS

```
if (gussedNum > randomNum) {
    gameResult.textContent = "Too High! Try Again.";
    gameResult.style.~~color~~backgroundColor = "—";
}
```

§ Random no: 13
entered no: 50

o/p Too High! Try Again

User Guess < Random Number

>> JS

```
else if (guessedNum < randomNum) {
    gameResult.textContent = "Too Low! Try Again";
    gameResult.style.backgroundColor = "—";
}
```

User Guess == Random Number

>> JS

```
else if (guessedNum === RandomNum) {
    gameResult.textContent = "Congrats!! You got it";
    "  .style.backgroundcolor = "green";
}
```

(4) Handle Invalid User Input

>> JS

```
else {
    gameResult.textContent = "Provide valid Input";
    "  .style.backgroundColor = "—";
}
}
```

(*) Loose equal to (==)

→ compare two values for equality but doesn't compare types of values.

→ Eg

```
console.log(2 == '2')
```

O/P true

Strict equal to (===)

→ compare two values for equality including types of values.

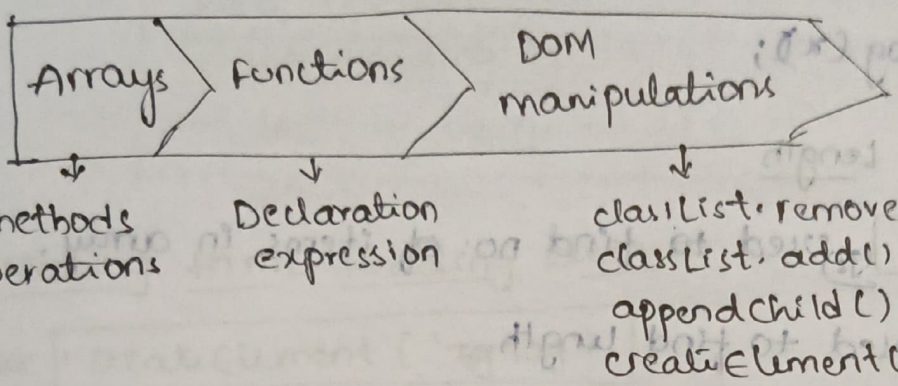
→ Eg

```
console.log(2 === '2')
```

O/P false.

Module-2 Arrays and objects

① Arrays & More DOM manipulations



* Data structures
→ Allow us to store & organize data efficiently.
→ allow us to easily access & perform operations on data.

JS Built-in Data structures —
- → Arrays
- → objects
- → maps
- → sets

① Arrays

→ Array (JS) holds ordered sequence of items.
→ List (py) holds ordered " "

creating array

>>JS
```
let x = [5, "one", 7.6];
console.log(x);
```
O/P [5, "one", 7.6]

Accessing array items

>>JS
```
let x = [5, "six", 2, 7.6];
console.log(x[0])
Console.log(x[3])
```
O/P 5
7.6

Modify Array Item

>> JS
```
let x = [5, "one", 7.2, 4];
x[2] = 9.7;
console.log(x);
```
o/p [5, "one", 9.7, 4]

Finding Array Length

→ `array.length` (JS) used to find no: of items in array.

→ `len()` (py) used to find length

>> JS
```
let x = [1,2,7,6];
let leng = x.length;
console.log(leng);
```
o/p 4

How to add / remove items in an array?

* * Array Methods

① push() `push(val)`

→ add new item to end of array.

Eg
```
let x = [5, "one", 7.2, 8.6];
x.push(90);
console.log(x);
```
→ o/p [5, "one", 7.2, 8.6, 90]

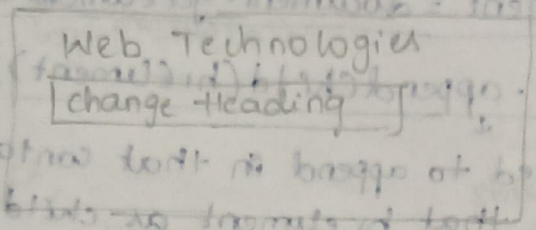
② pop() `pop(val)`

→ remove last item of array & return that item

Eg
```
let x = [5, "six", 2, 8.2];
let last = x.pop();
console.log(x);
console.log(last);
```
→ o/p [5, "six", 8]
8.2.

⑧ DOM Manipulations

How to creati HTML element using JS ?



1) Creating HTML Heading element

doc . createElement ('Tag')

>> JS

```
let hiElement = document.createElement('h1');
hiElement.textContent = "Web Technologies";
console.log(hiElement);
```

O/P `<h1> Web Technologies </h1>`

2) Appending to DOC. Body obj

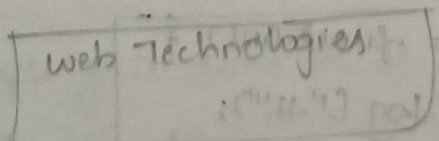
`body.appendChild(__);`

- Inside which we coant that element
- child for that body (which we need to append)

>> JS

```
let hiElement = document.createElement('h1');
hiElement.textContent = "Web Technologies";
document.body.appendchild(hiElement);
```

O/P



(or)

3) Appending to existing container element

>> HTML : Add id

```
<div id = "myContainer"> </div>
```

>> JS

```
let hiElement = document.createElement('hi');
hiElement.textContent = "Web Technologies";
let containerElement = document.getElementById('myContain
containerElement.appendChild(hiElement);
```

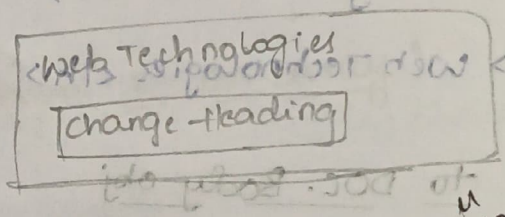
Accessing cont. element

we need to append in that container, we'll have that hi element as child for cont. element

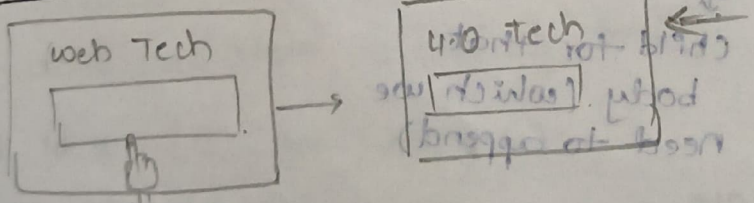
4) creating & appending Button Element

>> JS

```
let btnElement = document.createElement('button');
btnElement.textContent = "chang Heading";
containerElement.appendChild(btnElement);
```



How to manipulate HTML heading based on User Actions



* JS Functions

→ There is another syntax for creating function which is called function Expression.

syntax

```
let showMsg = function() {
    console.log("Hi");
};
showMsg()
```

Call function by using var name.

Function Declaration

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

Function Expression

```
let showMsg = function () {
    console.log("Hello");
};
showMsg();
```

5) Adding event listeners Dynamically

>>JS

≡

```
let btnElement = document.createElement('button');
btnElement.textContent = "change Heading";
btnElement.onclick = function() {
    hiElement.textContent = "4.0 Technologies";
};
```

6) Adding Heading styles Based on User Actions"

>>JS

```
btnElement.onclick = function() {
    hiElement.textContent = "4.0 Tech";
    hiElement.style.color = "blue";
    hiElement.style.fontSize = "40px";
    hiElement.style.fontFamily = "caveat";
    hiElement.style.textDecoration = "underline";
}
```

* Here styling is done in JS, but we'll actually write styling in CSS. SO,

We can provide class Names dynamically in JS

* classList.add()

>>JS =

```
btnElement.onclick = function() {
    hiElement.textContent = "4.0 Tech";
```

hiElement.classList.add("heading");
↳ class name

* Instead of writing in JS, we provide cls name & the style can be written in css.

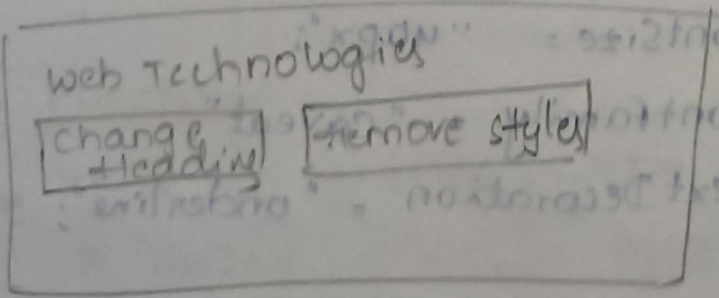
>> CSS

```
.heading {
    color : blue;
    font-size : 40px;
    font-family : "caveat";
    text-decoration : underline;
```

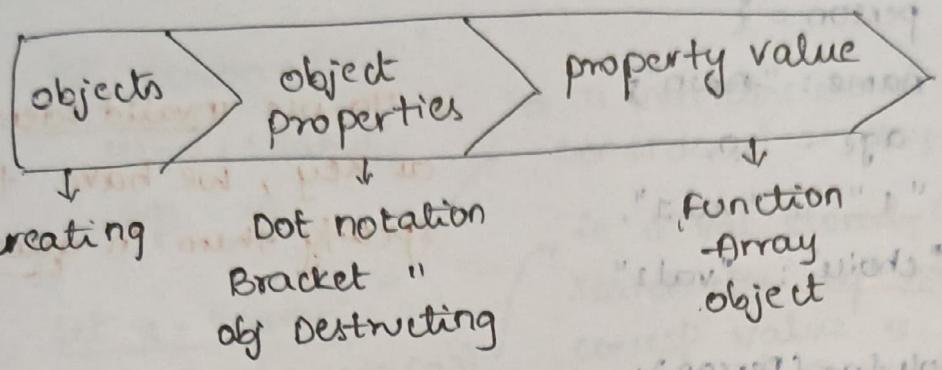
* classList.remove()

>> JS

```
let removeStylesBtnElement = document.createElement("button");
removeStylesBtnElement.textContent = "Remove Styles";
removeStylesBtnElement.onclick = function () {
    hiElement.classList.remove("heading");
};
containerElement.appendChild(removeStylesBtnElement);
```



② Objects



① objects

→ object is collection of properties

→ property is an association between name (key) & value

eg: person has name, age, city etc.
These are properties of person.

key	value
name	Rahul
age	20
city	Hyd.

* creating an object

→ we can add properties into { } as key:value pairs.

eg:
```
let person = {
    name: "Rahul";
    age : 28
};
console.log(person);
```

O/P
object :{name: "Rahul", age:28}

→ keys are identifiers.
valid identifier should follow below rules:
* contain alphanumeric char _ & $
* cannot start with number.

name ✓
$name ✓
_name ✓
name12 ✓
12name ×
name 12 ×

Using invalid identifiers as Property key

>> JS

```
let person = {
    name : "Arun",
    age : 20,
    "1" : "val1",
"my choice" : "val2"
};
console.log (person);
```

* To use invalid identifier as key, we have to specify them in quotes

* Accessing object properties

- Dot Notation (.)
- Bracket Notation []

> Dot Notation

```
let x = {
    name = "Arun";
};
console.log (x.name);
```

↑ only if valid identifier

Bracket Notation

```
let x = {
    name = "ram"
};
console.log (x['name']);
```

x use dot notation when key is valid identifier.

```
console.log (x.1);  x
console.log (x.choice); x
```

→ If it is invalid identifier, use bracket notation

>>

```
console.log (x["1"]);  ✓
Console.log (x["my choice"]);  ✓
```

Accessing Non-existent properties

* we are trying to access particular key that does not exist in object, we get o/p as "undefined" in both dot notation & bracket notation

* Variable as a Key

>> JS

```
let person = {
    name : "Rahul",
    age : 28,
};
let a = "name";
console.log (person[a])
console.log (person.a)
```

'a' is not considered as var instead considered as key.

in 'a' val stored is "name" search for key 'name' & give corresp value. ie, Rahul. considered as variable

search for 'a' key in obj as there is no key 'a' it returns undefined. considered as key

O/P
Rahul
undefined.

—x—

* Object Destructuring

→ To unpack properties from object we use object destructu - ri (assigning val to variables)

→ variable name should match with key of object

>> JS

```
let person = {
    name : "Rahul",
    age : 28
};
let {name} = person;
console.log (name);
console.log (person.name);
```

O/P
Rahul
Rahul.

val of "name" ie. Rahul is stored in var "name" from "person" obj

* var name should match key

>> JS

```
let {name, age} = person;
console.log (name);
console.log (age);
console.log (gender);
```

O/P
Rahul
28.
→ undefined.

—x—

⊛ Modifying objects

Dot Notation

```
let person = {
    name: "Rahul",
    age : 20
};
person.name = "arun";
console.log (person.name);
```

o/p arun

Bracket Notation

```
let person = {
    name: "Rahul",
    age : 28
};
person['name'] = "arun";
console.log (person['name']);
```

o/p arun.

* Adding object property

Dot

```
let person = {
    name: "Rahul",
    age : 20
};
person.gender = "male";
console.log (person)
```

Bracket

```
let person = {
    name: "Rahul",
    age : 20
};
person['gender'] = "male";
console.log (person);
```

⊛ Property value

→ value of object property can be
- function, Array, obj. etc

function as value

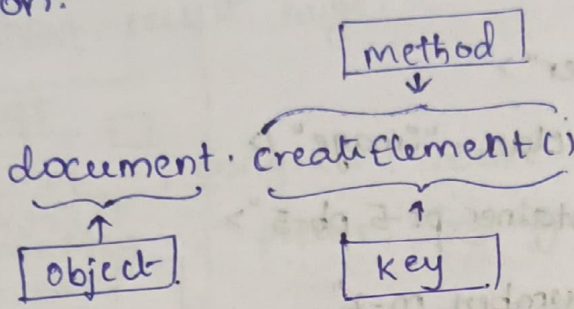
```
let person = {
    name: "Rahul",
    age : 20,
    run: function(){
        console.log ("start Running")
    }
};
person.run();
```

o/p Start Running

key.

* methods

→ A javascript method is property containing function definition.



Arrays as value

```
>>JS    let person = {
           name: "rahul",
           age: 28,
           habits: ["playing", "sleping"]
       };
       console.log (person.habits);  → ["playing", "sleping"]
       "    "  (person.habits[0]);  → playing
       "    "  (person["habits"][1]);  → sleping.
```

O/P

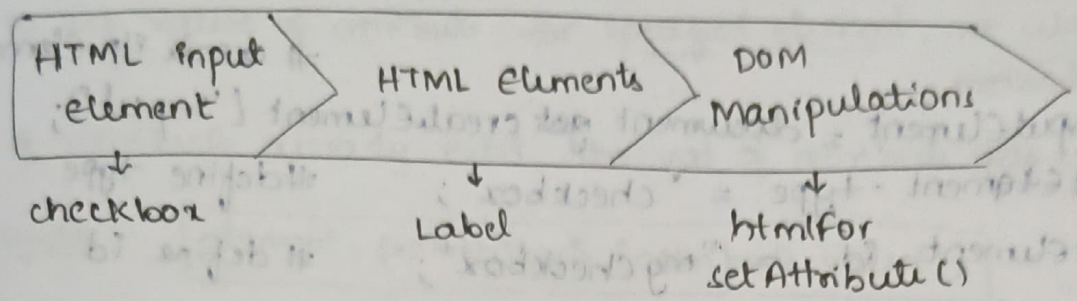
Object as value

```
>>JS    let x = {
           name: "rahul",
           age: 28,
           car: {
               name: "audi",
               model: "A6",
               color: "white",
           }
       };
       console.log (x.car.name);
       "    "  (x.car["color"]);
```

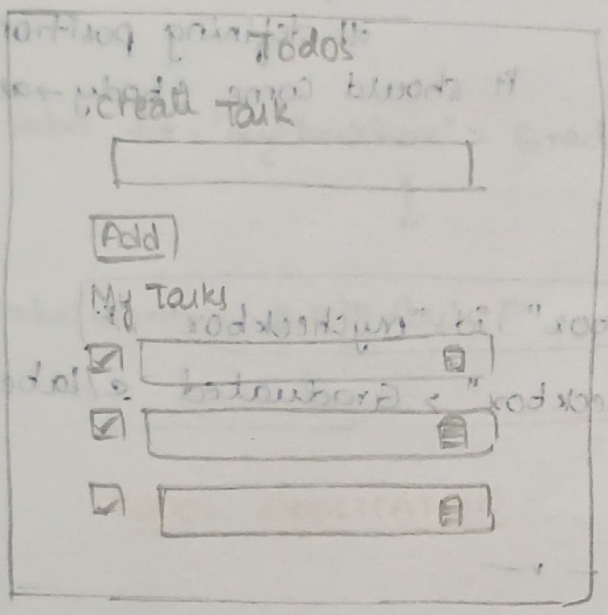
O/P = audi
white

Module-3 Todos Application

① Todos Appl. Introduction

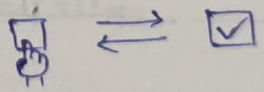


* Todos Application:



How to add A checkBox statically using HTML ?

```
<body>
    <input type="checkbox" />
</body>
```



Adding label to checkbox

graduated ⇄ graduated

```
<body>
    <input type="checkbox" id="mycheckbox" />
    <label for="mycheckbox"> Graduated </label>
</body>
```

* adv of adding label text is, if we click on labeled text the checkbox will be selected.

* we're writing for="mycheckbox". this says that this label belong to that checkbox.

* same id val) should be given to "for" attribute

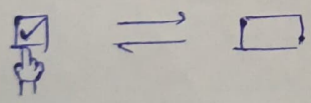
How to creati checkbox Dynamically using "JS"

>> JS.

```
let inputElement = document.getCreateElement ('input');   # creati ilp.Element
inputElement.type = "checkbox";                           #define type
inputElement.id = "myCheckbox";                           # define id
document.body.appendChild (inputElement);
                                  # defining position (where it should come (body → as child))
```

>> HTML

```
<input type="checkbox" id="mycheckbox" />
<label for="myCheckbox"> Graduated </label>
```



— x —

>> HTML

```
<input type="checkbox" id="mycheckbox" />
<label for="mycheckbox"> Graduatid </label>
```

>> JS.

```
let labelElement = document.createElement ("label");
labelElement.htmlFor = "mycheckbox";
labelElement.textContent = "Graduatid";
document.body.appendChild (labelElement);
```

O/P ☐ Graduatid ⇄ ☑ Graduatid

* DOM Manipulations

setAttribute()

→ To set value of attribute for specified element, we use this method.

→ If attribute already exist, the val of attribute gets updated

Syntax `element.setAttribute(attribute, value);`

small

>> HTML

```
<label for="mycheckbox"> Graduated </label>
```

↕

>> JS

```
labelElement.setAttribute("for", "myCheckbox");
```

TODOS APPLICATION

Steps

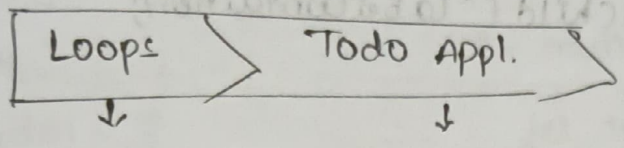
→ create single Todo item
→ create multiple Todo items
→ Take User Input & create Todos Dynamically.
→ Add delete todo item functionality

① create single Todo item (statically)

>> HTML

```
<ul class="todo-items-container" id="todoItemsContainer">
  <li class="todo-item-container d-flex flex-row">
    <input type="checkbox" id="checkboxInput" class="checkbox-input" />
    <div class="d-flex flex-row label-container">
      <label for="checkboxInput" class="checkbox-label">
        Learn HTML
```

② Todos Application part 1

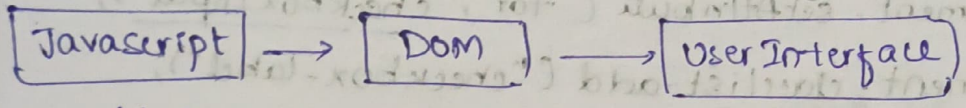


for...of loop

Adding multiple Todos dynamically.

Creat Todo item (Dynamically) using JS

Application Flow



Var, obj, Array etc

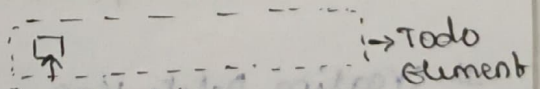
createElement(), textContent etc

→ with Data (JS) we perform DOM, with this UI created.

>> JS.

creating Todo Element

```
let todoItemsContainer = doc.getElement
                         ById("todoItemsContainer");
let checkboxId = "checkbox" + uniqueNo;
let todoElement = document.createElement("li");
todoElement.classList.add("todo-item-container", "d-flex",
                          "flex-row");
todoItemsContainer.appendChild(todoElement);
```



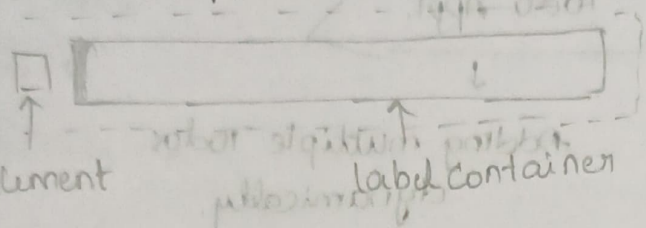
creating checkbox

```
let inputElement = document.createElement("input");
inputElement.type = "checkbox";        → checkboxId.
inputElement.id = "checkboxInput";
inputElement.classList.add("checkbox-input");
todoElement.appendChild(inputElement);
```

creating label container

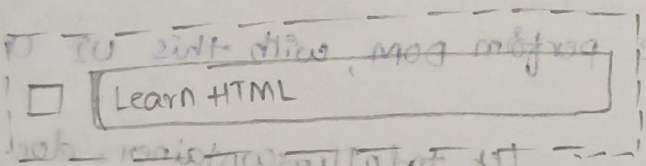
```
let labelElementContainer = doc.createElement("div");
```

```
labelContainer.classList.add("label-container", "d-flex",
                                                "flex-row");
todoElement.appendChild(labelContainer);
```



create label Element (label Text)

```
let labelElement = doc.createElement("label");
labelElement.setAttribute("for", "checkboxInput");
labelElement.classList.add("checkbox-label");
labelElement.textContent = "Learn HTML";
labelContainer.appendChild(labelElement);   # position
```

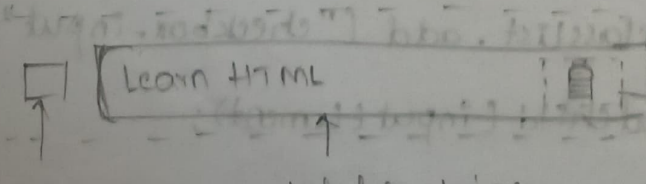


creating Delete Icon container

```
let deleteIconContainer = doc.createElement("div");
deleteIconContainer.classList.add("delete-icon-container");
labelContainer.appendChild(deleteIconContainer);
```

creating Delete Icon

```
let deleteIcon = doc.createElement('i');
deleteIcon.classList.add("far", "fa-trash-alt", 'delete-icon');
deleteIconContainer.appendChild(deleteIcon);
```



we've done :- DOM → UI

creating Multiple Todo Items

→ creating Todo object

for now,

```
let todo1 = {
    text : "learn HTML"
}
...
```

for multiple,

```
let todo1 = {
    text : "learn HTML"
}
let todo2 = {
    text : "learn CSS"
}
let todo3 = {
    text : "learn JS"
}
```

It is not proper way of writing, Instead we create a list of objects.

→ creating list of Todo objects

>>JS

```
let todoItemsContainer = doc- get -- ("todoItems container");
let todoList = [
    {
        text : "Learn HTML", uniqueNO: 1
    },
    {
        text : "Learn CSS", uniqueNO: 2
    },
    {
        text : "Learn JS", uniqueNO: 3
    },
];
```

→ creating Reusable function

we need to write all of these creations for each todo Item, instead write all the code inside function. It gets repeated & reused. whenever we want to create item, function is called

```
function createAppendTodo() {

}
```

→ we are creating Todo item only with "Learn HTML" but actually text is different from one to one.

→ So that info is maintained in "todoList".

→ Now pass this object to function createAppend as argument

» function createAppend (todo) {

≡

labelElement.textContent = todo.text;

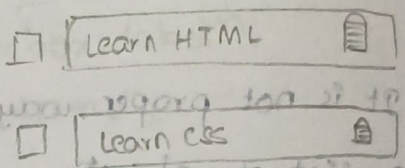
≡

};

#calling funct

createAppend (todoList[0]);

createAppend (todoList[1]);



Introduction to Loops

→ Loops allow us to execute a block of code several times

↳ for...of loop (for values)

↳ for...in Loop (for keys / indexes)

↳ for loop

↳ while loop ...many..

① for...of loop

python

```
for each_item in seq:
    ≡
```

JS

```
for (let each_item of iterable) {
    ≡
}
```

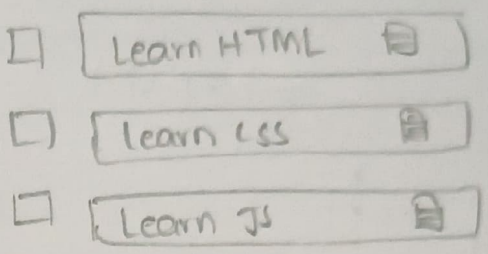
```
x = [1,2,3,4];
for i in x:
    print(i)
```

```
let x = [1,2,3,4];
for (let i of x) {
    console.log(i);
}
```

At last,

» for (let todo of todoList) {
    createAppend(todo);
}

o/p



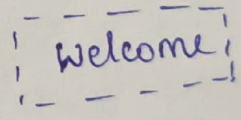
CSS Box properties

Border
- Border-width
- border-style (req)
- border-color.

Instead,

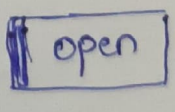
border: border-width borderstyle border-color

```
.button {
    border: 2px dashed green;
}
```



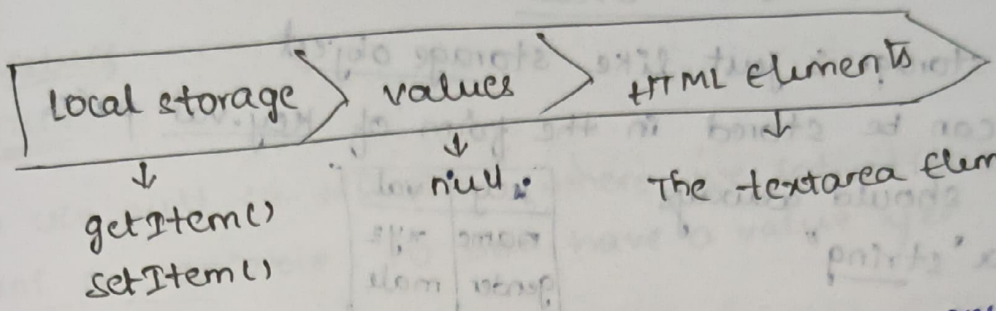
To specify sides,
- border-top
- border-bottom
- border-right
- border-left

```
.button {
    border-left: 5px solid green;
}
```



Module-4 Todos Application 2

① Todos Application part3



→ If we reload / refresh page, the entire page got reset (as before) Initial pg will display. Entire changes will be lost

* Execution Context

→ environment in which JS code runs is called execution context.

→ execution context contain all variables, obj & functions.

var
objects
functions

On reloading,

→ execution context is destroyed & recreated (run again) whenever we reload an Application.

How to persist todo items even on reload ?

* Storage Mechanisms

① client-side Data storage

→ storing data on client (user's m/c)

② server-side Data storage

→ storing Data on server using some kind of DB.

Client-side Data storage Mechanisms

1) Local storage
2) session storage
3) cookies
4) IndexedDB

⊙ local storage

→ It allows web applications to store data locally within the user's Browsers

→ Local storage is just like storage object

→ Data can be stored in the form of key-val pairs

→ value should always be a "string"

key	val
name	xyz
gender	male
city	vzg

→ To access & work with local storage, we have methods:

- setItem()
- getItem()
- clear()
- removeItem()

setItem()

syntax `localStorage.setItem("key", "value");`

>>JS
```
localStorage.setItem("name", "Rahul");
    "      "     ("gender", "male");
```

To check whether stored properly or not,

- go run JS
- right side o/p space click & go to inspect
- go to Application
- left side there will be storages, click on local storage
- then we can see website URL

getItem()

syntax `localStorage.getItem("key");`

>>
```
let name = localStorage.getItem("name");
let gender =    "       "    ("gender");
console.log(name)          o/p Rahul
  "   "  (gender)              male.
```