28.02.22

# JAVASCRIPT

.js

Client - Side scripting

→ runs inside browser
→ dynamically typed
→ object oriented

1996
→ by Netscape
(in Navigator browser)

## JAVA

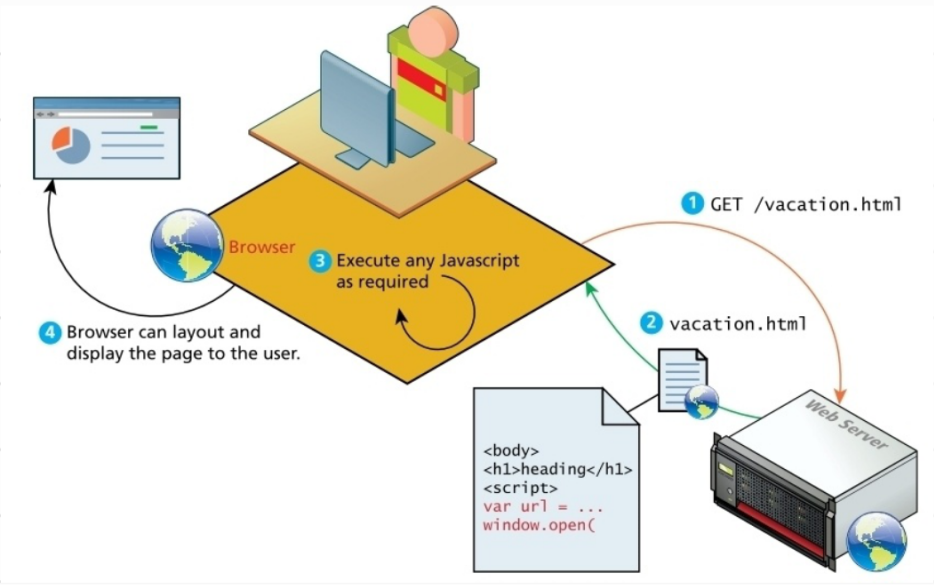
- full - fledged compiled
- OOP
- platform independent
  ↳ JVM

## JAVASCRIPT

- runs inside browser
  ↳ no JVM

JVM → intermediater
(provides platform neutrality)

platform = Hardware + OS




either
.js file
or
<script>
...
</script>

Advantages –

- process offloaded → reduce load on server
- rapid response to user
- JS interact with downloaded HTML in a way server can't

Disadvantages –

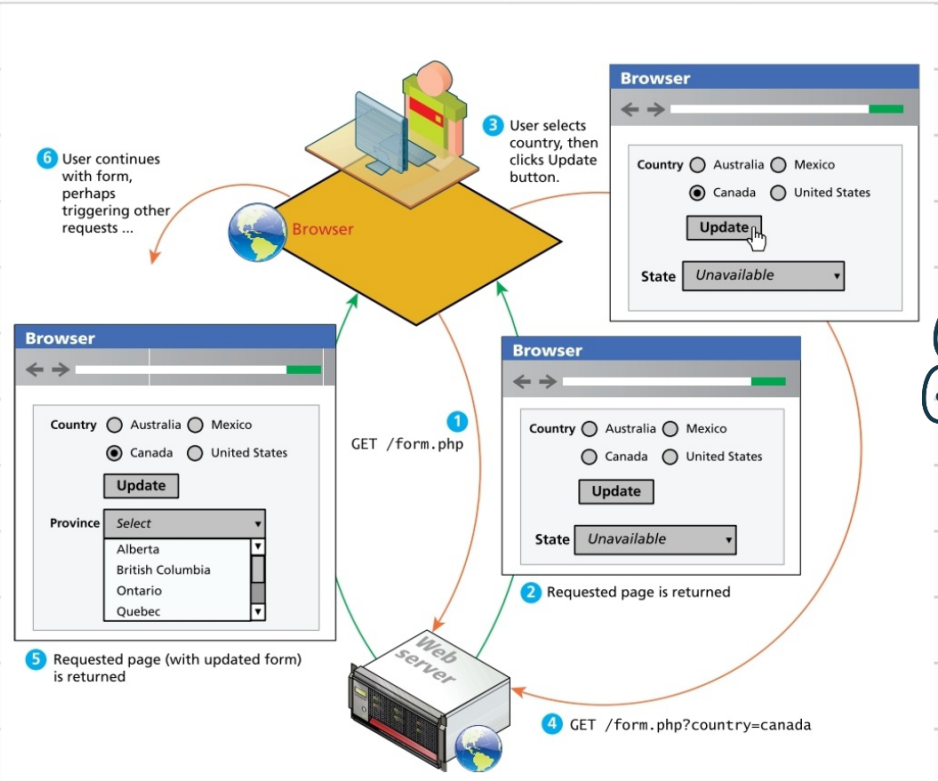
- no guarantee client has JS enabled
- idiosyncrasies b/w browsers & OS
  → difficult to test for all potential client config.
- JS-heavy web apps — complicated to debug & maintain

Before → browser plug-ins used (Flash) .swf

Then → Java Applets → <applet>

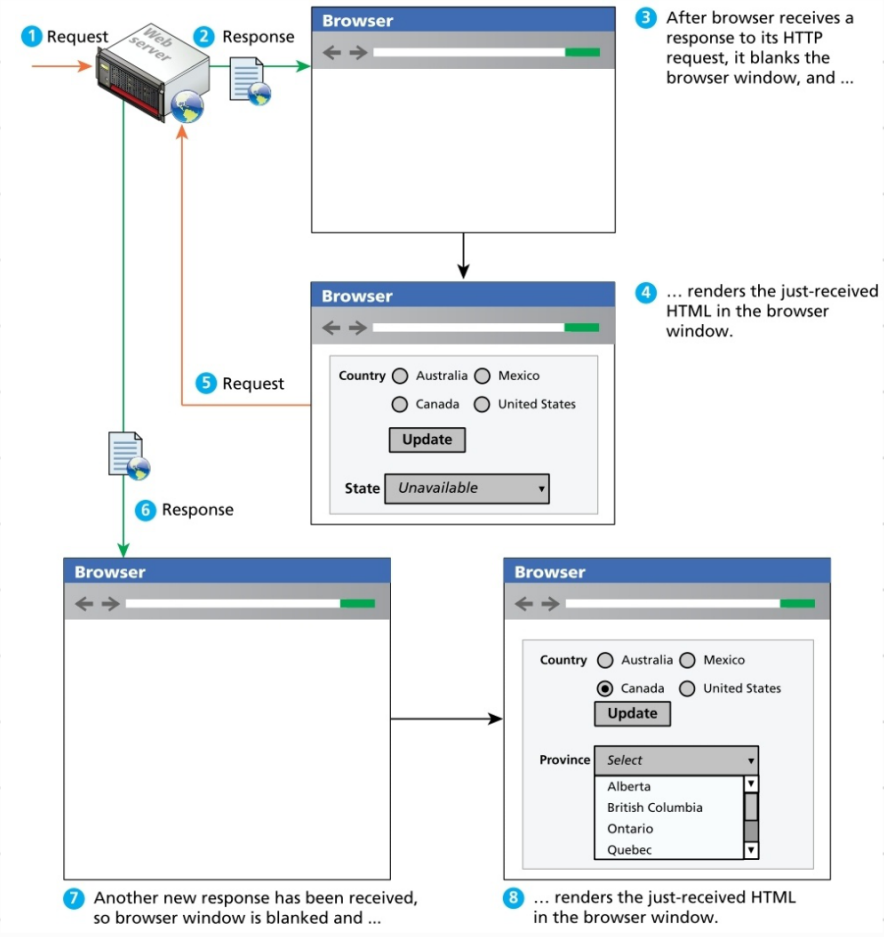
prblm:
converts in BYTE code
∴ didn't know about info. in byte code
→ may contain viruses

without JS




1. form
2. country
3. server
4. list

with JS

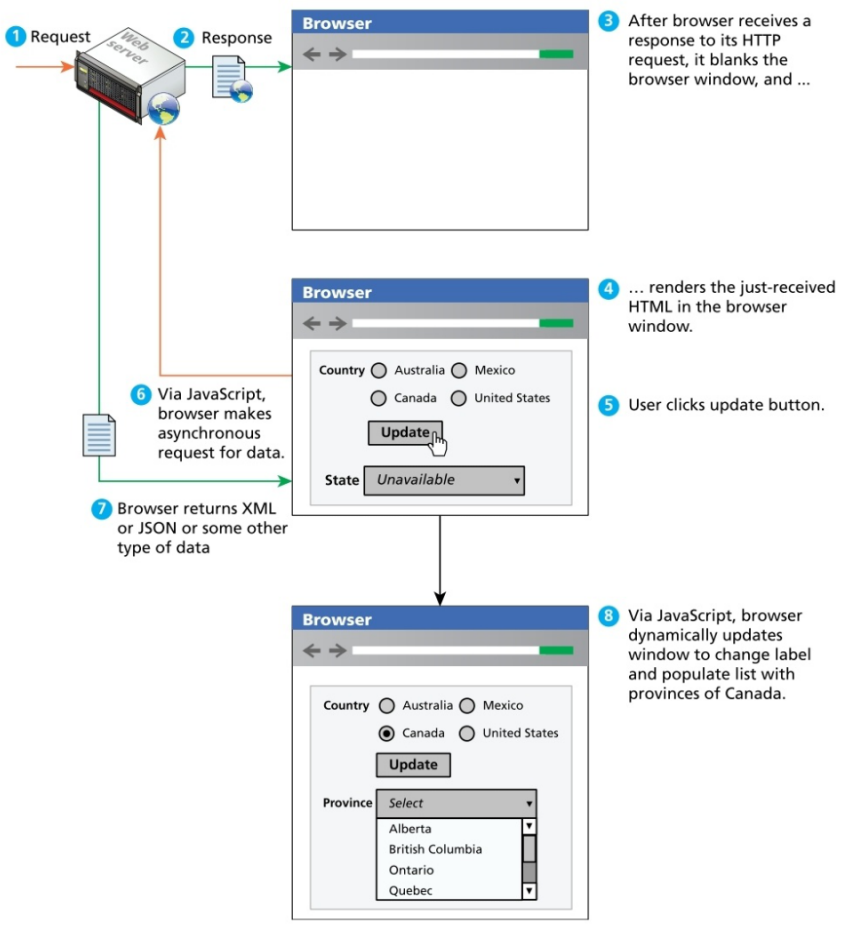



AJAX

Asynchronous JS And XML

↳ asyn. data requests (only data transfer)

# Asynchronous data requests




## Layers

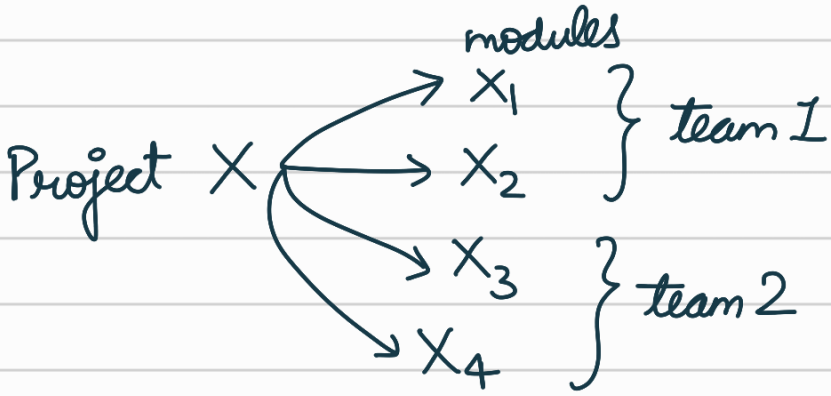
- Presentation: user interface (CSS + HTML)
- Business: real-world entities such as customers, sales
- Data: handle interaction with data sources

2.03.22

layers → abstraction

↳ grouping common functionalities & putting in same group

eg




5 software developers in each team

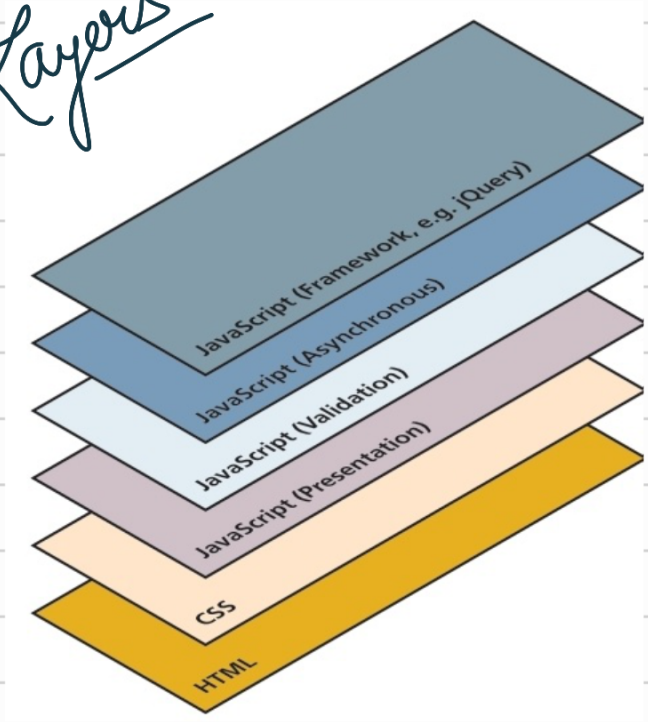
suppose

3 for $X_1$ & 2 for $X_2$

Software modularity → all of dev. are concerned with their parts

Good Code → documentation
→ comment for each line
→ tell fn. purpose, where it'll be used

Layers




change in one library, doesn't require change in others

# Users without JS

- Web Crawlers — client running on behalf of SE to download site
  - ↳ searches for certain keywords
    - ↳ search results
- Browser plug-in — software working with browser, might interfere with JS
- Text-based client — text-based browser
  eg, lynx (Linux OS)
- Visually disabled client — software read contents of web page
  eg, WebIE

## <noscript>

only displayed to users without ability to load JS

→ prompt users to enable JS
→ also show additional text to SE

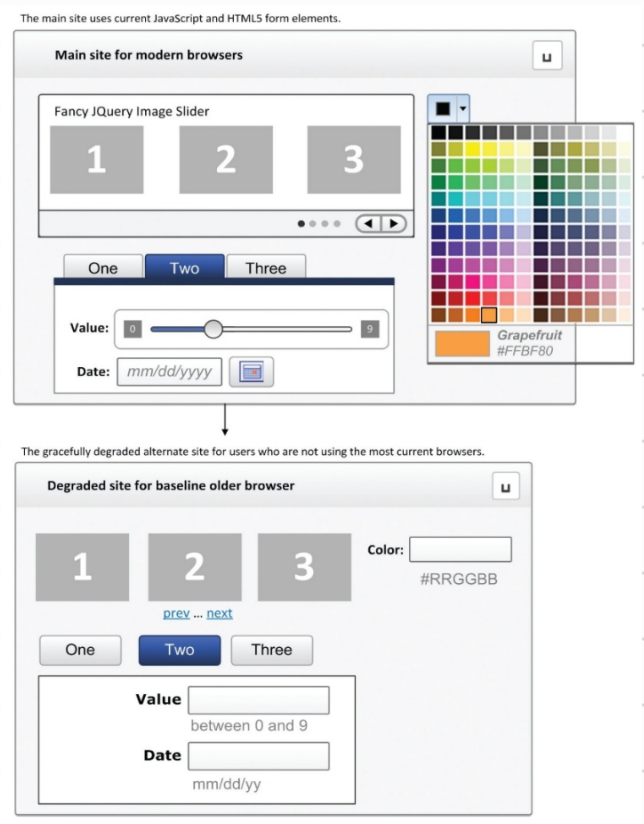
## fail-safe design

↳ user disabled JS, still website runs

JS → may have viruses or trojan horses

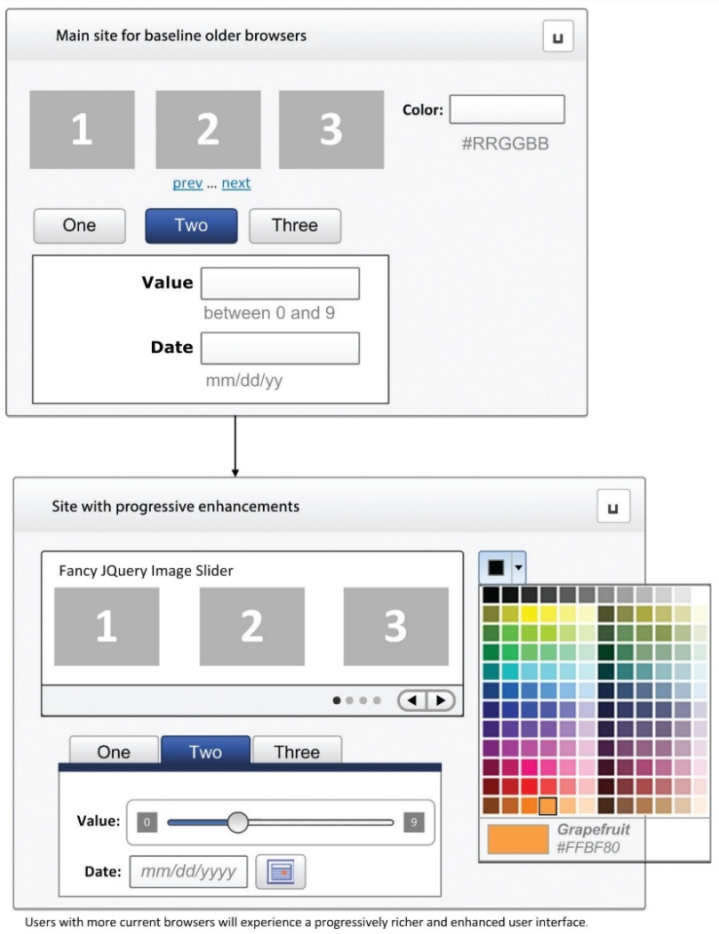
## → Graceful Degradation

if its an old browser or version, then degradation




## → Progressive Enhancement

opp. of graceful deg.



JS → inline (in HTML doc)
→ embedded (<head>)
→ external (separate file)

## INLINE

```
<a href="JavaScript:OpenWindow();"more info</a>
<input type="button" onclick="alert('Are you sure?');" />
```

## EMBEDDED

```
<script type="text/javascript">
/* A JavaScript Comment */
alert ("Hello World!");
</script>
```

## EXTERNAL

```
<head>
  <script type="text/JavaScript" src="greeting.js">
  </script>
</head>
```

# Advanced Technique

→ generate embedded style to reduce request
  ↳ code in ext. file

→ asynchronous load from another (initial) JS file
  ↳ faster initial load
    (later JS keeps loading)

- variables
- assignment
- conditionals
- loops
- arrays
- events
- classes

## Rules of JS

→ type-sensitive

→ scope of var. in blocks not supported

```
{
    int x;
}
```

var. 'x' accessible outside block or loop

→ === data-type equality

→ Null, undefined ⟶ diff. states for var.

→ ; not required (use encouraged)

→ no int type ⟶ only NUMBER

∴ floating-pt. rounding errors prevalent

var

# Variables

→ dynamically typed

```
var x = 10;
var x = "hello";
x = true;
```

} same variable can take diff. values

| Operator | Description | Matches (x=9) |
|---|---|---|
| == | Equals | (x==9) is true<br>(x=="9") is true |
| === | Exactly equals, including type | (x==="9") is false → data type not same<br>(x===9) is true |
| < , > | Less than, greater than | (x<5) is false |
| <= , >= | Less than or equal, greater than or equal | (x<=9) is true |
| != | Not equal | (4!=x) is true |
| !== | Not equal in either value or type | (x!=="9") is true<br>(x!==9) is false |

&& (and)
|| (or)
! (not)

```
var hourOfDay;   // var to hold hour of day, set it later...
var greeting;   // var to hold the greeting message.
if (hourOfDay > 4 && hourOfDay < 12){
   // if statement with condition
   greeting =  "Good Morning";
}
else if (hourOfDay >= 12 && hourOfDay < 20){
   // optional else if
   greeting =  "Good Afternoon";
}
else{ // optional else branch
   greeting = "Good Evening";
}
```

# Loops

```
var i=0;
while(i < 10){
 //do something with i
 i++;
}
```

```
for (var i = 0; i < 10; i++){
  //do something with i
}
```

3.03.22

# Functions

↳ no return-type, no parameter type

```
function power(x,y){
  var pow=1;
  for (var i=0;i<y;i++){
     pow = pow*x;
  }
  return pow;
}
```

alert ()
pop-up

alert ("Good Morning");

console.log ("Put messages here");
↳ O/P to a log

# Exceptions

```
try {
  nonexistantfunction("hello");
}
catch(err) {
  alert("An exception was caught:" + err);
}
```

JS engine — error
↳ exception

exceptions interrupt regular, sequential execution of prgm
+ stop JS engine

∴ try-catch, prevent disruption

user-defined:

```
try {
   var x = -1;
   if (x<0)
      throw "smallerthan0Error";
}
catch(err){
   alert (err + "was thrown");
}
```

4.03.22

# JS Objects

→ not support inheritance & polymorphism
→ no classes per se

∴ not fully-fledged object oriented

Objects → constructors
→ properties
→ methods

## Constructors

```
var Obj = new ObjName (p1, p2, ...., pn);
```

```
class x;
x obj1;
```

object reference

object → variables
∴ memory needed

```
x obj2 = new x ();
```

assigns space to object from memory

object → require memory
class → not " "
→ Objects are real, classes are virtual

(formal) `var greet = new String ("Hello");`

or

(shortcut constructor) `var greet = ("Hello");`

## Properties

```
alert (someObject.property);
```

## Methods

```
someObject.doSomething();
```

## Objects

- → Array
- → Boolean
- → Date
- → Math
- → String
- → Dom objects

Document object model

# Array → resized dynamically

- var greet = new Array();
- var greet = new Array("Hi", "Hello");
- var greet = ["Hi", "Hello"];
- greet[0] = "Hi";
  greet[1] = "Hello";

```
alert(greet[0]);

for(var i=0; i<greet.length; i++) {
    alert(greet[i]);
}
```

greet.push("Hey"); → add item

pop() → remove item

concat()
slice()
join()
reverse()
shift()
sort()

# Math

max(), min(), pow(), sqrt(), exp(),

sin(), cos(), arctan()

constants ⟶ PI, E, SQRT()

```
Math.PI;
Math.sqrt(2);
Math.random();
```

⟶ random no° b/w 0 & 1

max(a, b, c).

# String

Concatenation

(long form) `var str = greet.concat("Hey");`

(operator) `var str = greet + "Hey";`

`charAt()` → access a single char.
`indexOf()` → search for one index

`split()`
`search()`
`match()`

Date → to convert in string, use toString()

```
var d = new Date();
alert ("Today is " + d.toString());
```

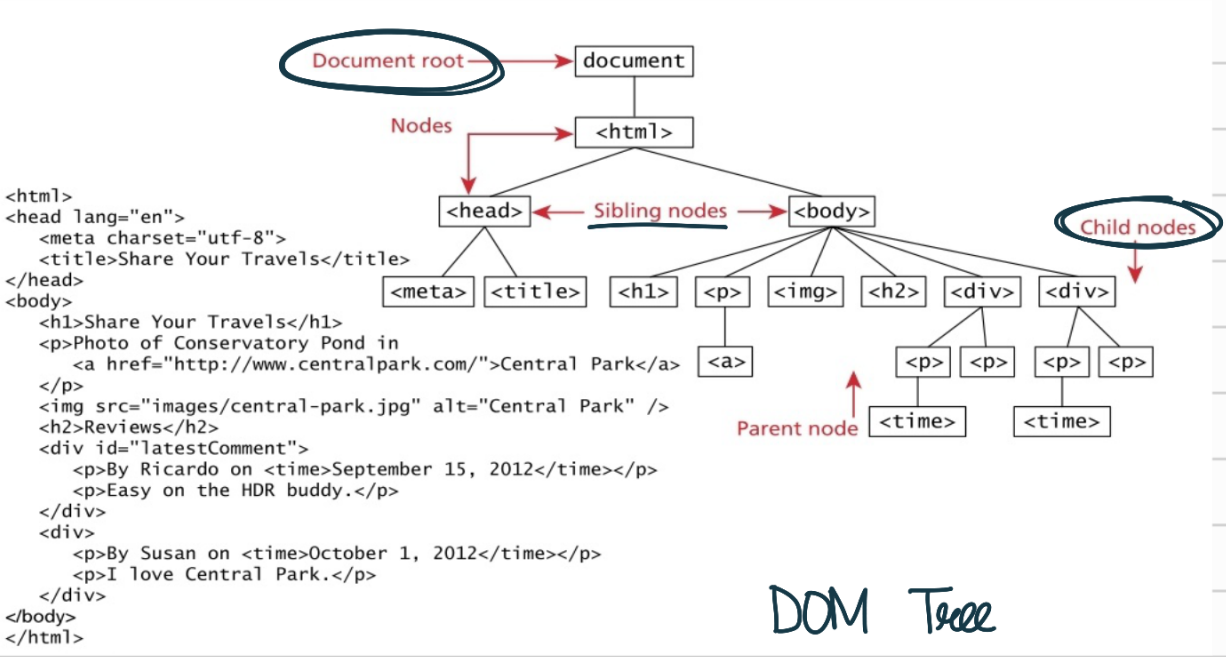
O/P Today is Mon Mar 07 2022 00:39:54 GMT+0530 (India Standard Time)

7.03.22

# DOM — Doc. Object Model

↳ platform & lang-neutral interface

that allow prgrms & scripts
↳ dynamically access & update
content, structure & style of doc




DOM Tree

In DOM each element within HTML doc
↳ Node

- element node
- text node
- attribute node

attribute node

<a href="#">School</a>

element node

| Property | Description |
|---|---|
| attributes | Collection of node attributes |
| childNodes | A NodeList of child nodes for this node |
| firstChild | First child node of this node |
| lastChild | Last child of this node |
| nextSibling | Next sibling node for this node |
| nodeName | Name of the node |
| nodeType | Type of the node |
| nodeValue | Value of the node |
| parentNode | Parent node for this node |
| previousSibling | Previous sibling node for this node. |

DOM doc. object → root JS object

↳ properties, methods
→ globally accessible as document

eg, var a = document.doctype.html;

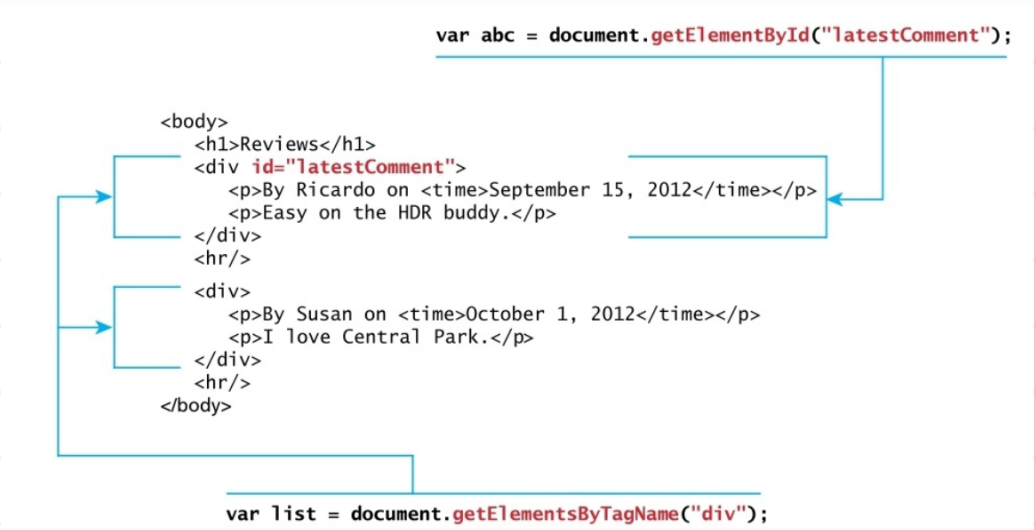
var b = document.inputEncode;
page encoding

| Method | Description |
|---|---|
| createAttribute() | Creates an attribute node |
| createElement() | Creates an element node |
| createTextNode() | Creates a text node |
| getElementById(id) | Returns the element node whose id attribute matches the passed id parameter |
| getElementsByTagName(name) | Returns a NodeList of elements whose tag name matches the passed name parameter |

element node

only one element at a time

↳ multiple tags together




| Property | Description |
|---|---|
| **className** | The current value for the `class` attribute of this HTML element. |
| **id** | The current value for the `id` of this element. |
| **innerHTML** | Represents all the things inside of the tags. This can be read or written to and is the primary way in which we update particular  elements using JavaScript. |
| **style** | The `style` attribute of an element. We can read and modify this property. |
| **tagName** | The tag name for the element. |

# JS Events

or action that can be detected by JS

specified in HTML markup with hooks to JS

listener (frameworks)


Click for pop-up

problem

HTML & JS together

no use of layers

not separate content (html) from behaviour (JS)

```
var greetingBox = document.getElementById('example1');
greetingBox.onclick = alert('Good Morning');
```

**LISTING 6.10** The "old" style of registering a listener.

```
var greetingBox = document.getElementById('example1');
greetingBox.addEventListener('click', alert('Good Morning'));
greetingBox.addEventListener('mouseOut', alert('Goodbye'));

// IE 8
greetingBox.attachEvent('click', alert('Good Morning'));
```

**LISTING 6.11** The "new" DOM2 approach to registering listeners.

```
function displayTheDate() {
   var d = new Date();
   alert ("You clicked this on "+ d.toString());
}
var element = document.getElementById('example1');
element.onclick = displayTheDate;

// or using the other approach
element.addEventListener('click',displayTheDate);
```

object → convert to String

event

action

**LISTING 6.12** Listening to an event with a function

```
var element = document.getElementById('example1');
element.onclick = function() {
   var d = new Date();
   alert ("You clicked this on " + d.toString());
};
```

**LISTING 6.13** Listening to an event with an anonymous function

all DOM event objects
↳ event handlers

event object

```
function someHandler (e) {
}
```

# Event Object

- Bubbles — event → true → event handler
  ↳ false → bubble to parent & trigger event handler
- Cancelable — boolean value
  ↳ whether or not event can be cancelled

- preventDefault → stop cancelable default action

```
function submitButtonClicked(e) {
  if (e.cancelable){
    e. preventDefault();
  }
}
```

## Event Types

→ mouse events

→ keyboard "

→ form "

→ frame "

## Mouse Event

| Event | Description |
|---|---|
| **onclick** | The mouse was clicked on an element |
| **ondblclick** | The mouse was double clicked on an element |
| **onmousedown** | The mouse was pressed down over an element |
| **onmouseup** | The mouse was released over an element |
| **onmouseover** | The mouse was moved (not clicked) over an element |
| **onmouseout** | The mouse was moved off of an element |
| **onmousemove** | The mouse was moved while over an element |

object → method → event

```
document.getElementById("keyExample").onkeydown = function
myFunction(e){
   var keyPressed=e.keyCode;      //get the raw key code
   var character=String.fromCharCode(keyPressed); //convert to string
   alert("Key " + character + " was pressed");
}
```

**LISTING 6.15** Listener that hears and alerts keypresses

| Event | Description |
| --- | --- |
| **onkeydown** | The user is pressing a key (this happens first) |
| **onkeypress** | The user presses a key (this happens after onkeydown) |
| **onkeyup** | The user releases a key that was down (this happens last) |

Keyboard Event