**WWW - World Wide Web**

**1.1 World Wide web Introduction**

**Web browser**

**-** Visualize pages on web document.

- Contains link, button, video, image

**internet :**

- hardware - global network of machines , and devices that facilitate interaction

**nodes -** computes

**edges -** connection

**www :**

- software - application on top of internet - using hardware infra structure

- also protocol for communication

- composed of document that are linked to each other through web-browser

**nodes** - documents/content/resources

**edges**- links between documents

**Addresses**

URI - uniform Resource Identifier - unique

URL - uniform resource Locator - type of URI. location on www and protocol

*http://www.example.com/home/inde.html?a=12&b=34*

*http :* protocol

*www.example.com :* host name

*example.com :* domain name

*home/inde.html :* file path

*a=12&b=34 :* query parameter

**Protocol:**

Structure of content of communication between devices

**Contents on www**

**static:** Contents does not change, -who, when-html,css

**dynamic:** content changes - user, event -js

request - receive / send - how display - how generate dynamic content

**1.2 HTTP Client Request, Server Response**

**View content on WWW:**

1980's - text web browsers

1095s - mosaic web browser - images and links between pages

**web browser :**

software displays content and let you navigate

component:

Rendering engine -

decides what to display and how to display on web browser.

presentation, ordering, content

html, css

Java script Engine-

dynamic part - create modify

run code

Web browser works -

http protocol -

web browser communicates - request, transfer documents

client -(laptop, mobile, fridge)

server - server /big computer

connected via connections , devices

client initiates - http request

server - listens - decides response and content and sends http response

client again - decides what to do with the content from server response

http:

plain text, human readable protocol on the web

client server model

request + client info

response + server info



**http request**

verb - Action client hoping to perform

get - requesting resource

head - just header info, not any resource

post -create new content on server

argument - name of the resource requested

protocol - http/1.1

client information + other information

Ex:

Request Verb URI HTTP Version

----- --------------------------- -----------

GET /example/index.html HTTP/1.1 ------------> Request Line

Host: www.edx.org |

User-Agent: Mozilla/4.0 |-----------> Request header

Accept-Language: en-us |

Content-Length: 9 |

-------------> Blank Line Separator

a= 12 & b=34 -------------> Request Message Body

**HTTP Response -**

1st line

Protocol and status code

1XX - Information only

2XX - Success

3XX - Client redirect

4XX - client error

5XX - server error

Most common status codes

200 OK - request succeeded , returning contents requested

404 Not Found - requested resource does not exist

500 Server Error - Error on server side while processing request

2nd line

header information - info about server / info of content

3rd line

blank separator

4th line

response body - content of page requested

Ex:

HTTP Version Status Code

---------- --------

HTTP/1.1 200 OK -----------------------------------> Response Line

Date: Fri, 06 Apr XXXX 09:30:00 GMT |

Server : Apache/1.4 |

Last-Modified: Wed, 04 Apr XXXX |

Connection: close |------------> Response Headers

Content-Type: text/html |

Content-Length: 228 |

--------------> Blank Line Separator

<!DOCTYPE html> <html><Head> --------------> Response Body

**HTML**

**1.3 HTML Basic**

**HTML -**

Structure content / data on WWW

html document - elements -tags - can be nested

plain text - human readable

structure

doesn't decide how to display - web browser decides

**Tags**

<!DOCTYPE html> - version of html

<html> - all contents within

<head>- info of document

<title> title bar

<link> link to other document ex: css -styling

<meta> provide additional information Ex: keyword used by search engine

<script> link to dynamic content Ex: Javascript

<body> - content to be displayed Ex: image, text

**HTTP and HTML -**

browser request for content

client - http request

server - response - content - plain text -html

browser - how to display

Browser - View source Code

html page - right click - view source -html source

**1.4 Important HTML Tags**

<p> : separate paragraph

<h1><h6> : headings

<b> : bold <strong>

<i> : italic <em>

<hr> : horizontal line

<br> : break line

<div> : division. content goes together. useful in styling

<span> : text part. smaller content than div. styling

<!-- comments -->

&nsbp; : single white space

&lt; &gt; : less than / greater than

&amp; : & symbol

&copy; &reg; : copy and registration symbol

**1.5 HTML Attributes**

HTML Attributes -

additional information to tag

name value pair

Ex: <p name="value">

title : title of element. see title when hover over element

<h1 title= "heading">

style: how the content will appear. multiple sub key value pair

<h1 style =" color:red; text-transform:capitalize">

background-color: red, #012345

font-family: verdana, courier

font-size: 12px, 4in, 200%

text-align : center, left, right

id : (#) unique identifier to html element within html tag within html page can be used to formatting

class : (.) subgroup of elements

html - style tag

**1.6 CSS - Style - CSS**

- http response : html

- html looks for css

- inline / same html file / separate file

- separate file - client makes another request for css -http response

- html apply css

inline css:

<h1 style="color:red:>heading1</h1>

advantage : easy to use when want something just once

disadvantage: mixing content html and css

same html :

<style>

h1 { -----> CSS selector

color: red; -----> property: value pair

}

</style>

Advantage: separate content in head, easy to use

Disadvantage : head might get big, html specific

External CSS:

separate file - link within the head

.html

<head>

<link rel="stylesheet", type="text/css", href = "movie-styles.css" />

</head>

.css

h1 {

color:red

}

Advantage : separate content and html , can use css file for multiple pages

disadvantage : need to manage files

css selectors:

external selector :

h1 { color:red; }

all <h1>

class selector :

.address { ... }

<div class="address"> ... </div>

id selector:

#section1 { ... }

<p id="section1"> ... </p>

1.7 Other HTML tags

Lists:

ordered : <ol> <li>

unordered : <ul><li>

Style :

<ul style="list-style-type:circle"> <li></li> </ul>

<ol type="A> <li> </li> </ol>

Nested List : <ul> <li> </li>

<li> </li>

<ol>

<li> </li>

<li> </li>

</ol>

<li> </li>

</ul>

Forms:

<input type="value">

text

email

password

date

range : slider - min, max

radio : value

checkbox : value

color

file

submit

reset

button

hidden : hidden input file

image : image as submit button

month : Month and year control

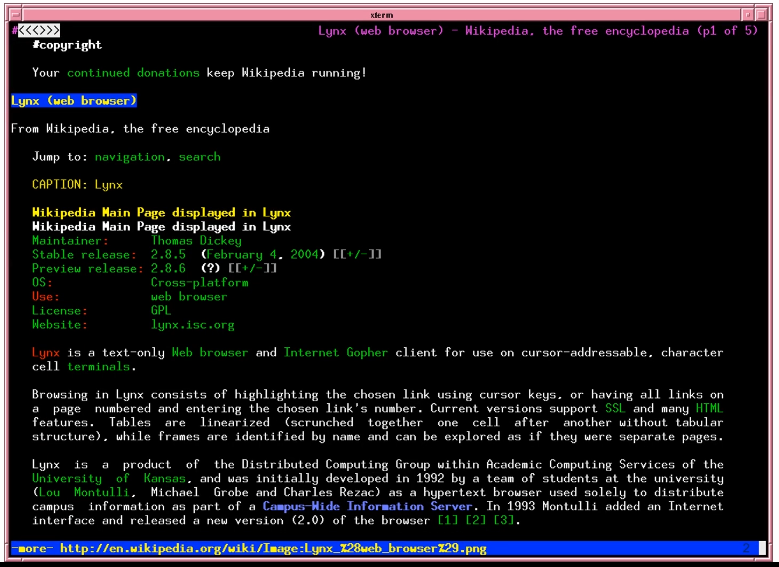
time : time control

number

tel : phone number

url

**1.8 Links, tables and images in HTML**



<img> :

not in the html page

link to image

attribute : alt, src, height, width, title

src - absolute path, relative path

links:

surf one page to another page

<a>

href : location (absolute, relative)

target : link should be open in new window/tag

text : clickable link

</a>

bookmarks: link within webpage

id - attribute

< ahref=#health>

tables:

<table> border=1

<tr> <th>

<tr> <td>

border

padding

text-align

border-spacing

rowspan : span rows

colspan

1.9 Using Bootstrap

IBM

users - different devices

responsive web designing (RWD)- smaller device - fills space on webpage accordingly

different CSS for different devices

bootstrap - automatic web response

- grid system

bootstrap

html - head - link tag, js lib, meta tag,

body - apply bootstrap - container class

grid system - content - organized in rows -

horizontal group of 1 to 12 columns

JavaScript

**2.1 Developing in JavaScript**

HTML : structure of contents

CSS : specify appearance

JS: dynamic content of webpage

3 ways to develop JavaScript:

1. <script> : embed in html page

<link> : link to external js file

2. REPL : Read-Evaluate-Print-Loop

Chrome console - write js and see output

3. JS file : Js file. execute in standalone environment - Node.js

JS interact with HTML

framework for client side JS

framework for server side JS

Course :

JS 1.7, Chrome 58.0, Mac OSX 10.11.5

**2.2 Primitive Data Types**

Variable declaration

var age=22;

var name='Uday';

var isMale=True;

Variable View

document.write(age);

console.log(age);

alert(age);

REPL > age

Changing Variable type:

var id = 33.2;

id = 'secret'

5 primitive types:

Number - float, int

Number('tree' ) : NaN

Number.Max\_Value()

arithmetic operations

String

'Uday' "Uday's" , '\n' '\t' '\\'

Strings are immutable

.concat() , toUpperCase(), toLowerCase()

name.length

'Hello, '+name;

Boolean

Truthy : true, 'cow', 'false' 5 ..

Falsy : false, null, undefined, 0, NaN, '' ..

Null

no value, empty

Undefined - not seen before

no value assigned yet, just initialization

**2.3 JavaScript array and object**

**Array**

collections of data types

array of same value, different value

view element of array

var myArray = ['cars' , 12, false];

var age = myArray[1]

myArray.length

myArray[4] : undefined

myArray[-9] : undefined

Add element to array

myArray[4] = 'panda'

myArray [-5] = 'elephant'

console.log(myArray)

//(5) ['cars' , 12, false, undefined X 1, "panda", -5: " elephant "]

var myArray = ['car' , 'bike'];

myArray.push('scooter') ; ://car, bike, scooter

myArray.unshift('scooter'); ://train, car, bike, scooter

Vehicle =myArray.pop() ://train, car, bike

Vehicle =myArray.shift() : // car, bike

**Object:**

collections of var, values

key-value pair within {}

variables are called property

accessed by .property

var person = {

name: 'John Doe',

age: 25,

isMale: true,

personality: ['patient', 'loyal', 'happy'],

company: [name: 'edX', id:2984]

}

console.log(person.age); //25

console.log(person['company'].id //2984

Modifying object:

person.isMarried: True;

**2.4 JavaScript Control Structure**

**Conditional statement**

car a =

var b =

var max;

if(a>b){

max = a;

}

else{

max=b;

}

console.log(max);

**comparison operator**

== 1='1' //true

=== equal to and same type

!=

!== not equal to or different type

>

>=

<

<=

**Logical Operator**

||

&&

!

**Comparing Number to String**

compare operation, JavaScript converts String to number

5 < '20' //true

'zebra'>'giraffe // true //alphabetical

**compare objects**

var cooper = {age:11}

var flanders = {age:11}

if(cooper == flanders) //false

if(cooper ===flanders) //false

alias then true

var myDog ==cooper;

if(myDog==cooper) //true

**Loops**

var n =

var fact = 1;

for (var i; i <= n ;i++) {

fact \*= i ;

}

var i=1;

while(i<=n) {

fact \*= i;

i++;

}

var i=1;

do{

fact \*=i;

i++;

}

while(i <=n);

**2.5 JavaScript Functions**

function fact(n){

var product =1;

for (var i=1; i<=n; i++) {

product \*= i;

}

return product;

}

var x =

var f = fact(x);

console.log(f);

//apply array to function

var nums=[4,8,12,2];

function print\_func(n) {

console.log(n);

}

nums.forEach(print\_func);

function isEven(n) {

return n%2 == 0;

}

nums.every(isEven); //true

nums.some(isEven);

function square(n){

return n\*n;

}

var sq = nums.map(square); // [16, 64, 144, 4]

//returns new array

**Pass by value**

- primitive arguments are **passed by value**

- function cannot change them

function tryToChange(x){

x=4;

};

var y=11;

tryToChange(y);

console.log(y); //still 11

**Passed by reference:**

object arguments are passed by reference.

function changeMe(obj){

obj.age++;

};

var p = {age:30};

ChangeMe(p);

console.log(p.age); //now 31

**Functions are object**

JS functions are objects

functions can have properties

we can refer variable to them

var add = function(a,b) {

return a+b;

};

console.log(add(3,5));

JavaScript functions can also be declared and used in object

var johnDoe = {

name: 'John Doe',

age: '32',

greeting: function() {

return 'Hello! Nice to Meet you!';

}

}

console.log(johnDoe.greeting());

**Object Prototype**

prototype : basic set of properties from which it can inherit

\_\_proto\_\_ property: object which has proto

root prototype property : Object.prototype

object to inherit properties of other objects

**Create Prototype:**

like class definition

this - current object

new - create new object

function Person(name,age) {

this.name = name;

this.age = age;

this.greeting = function(){

return 'Hello! '+this.name;

}

}

var johnDoe = new Person('John Doe', 32);

johnDoe.greeting();

var janeDoe = new Person('Jane Doe', 28);

johnDoe.greeting();

**Extending Prototype:**

function Student(name, age, school){

***this.\_\_proto\_\_ = new Person(name,age);***

this.school = school;

}

var sarahBrown = new Student('Sarah Brown', 17, 'PennX');

sarahBrown.greeting();

sarahBrown instanceof Person; //true

**Prototype properties**

var Person = function(name, age, occupation){

this.name =name;

this.age=age;

this.occupation=occupation;

}

***Person.prototype.planet = 'Earth';***

***Person.prototype.introduction = function() {***

***return 'I am a '+this.occupation;***

***}***

var johnDoe = new Person('John Doe', 32, 'Denstist');

johnDoe.planet; //earth

johnDoe.introduction // I am a Dentist

**2.6 JavaScript Regular Expression**

**JavaScript Strings**

Character sequence

Strings are immutable

Strings are object and has its own functions

var name ='toucan';

name.length; //6

name.charAt(3); //'c' //index-0 based

name[3]; //'c'

var animal = 'cat';

animal[0] = 'r';

console.log(animal); //still 'cat'

animal.toUpperCase(); //CAT

console.log(animal); //'cat'

var msg = ' Hello Everyone ';

msg = msg.trim();

var my = 'cat.concat('mouse'); //catmouse

msg.startsWith('Hello'); //true

msg.startsWith('HELLO'); //false

msg.endsWith(''Everyone'); //true

msg.includes('JavaScript'); //false

index of contained substring

var title = 'the title of my book';

var start = title.search('title'); //4

start = title.search('banana'); //-1

**regEx:**

pattern of characters

string search function

var status = 'I am working VERY hard';

status.search(/VERY/); //13

status.search(/very/); //-1

status.search(/very/i); //13

/script/.test('javascript is so much fun!'); //true

var numbers '5 8 2 5 7 6';

numbers.search(/[012]/); //4 (2)

/[012]/.test(numbers); //true

var password = 'passwprd4real';

password.search(/[a-z]/); //0 //0-char

password.search(/\d/); //8 //4-digit

using Ranges

var code = 'abc123d4e5';

code.search(/[0-9][a-z][0-9]/); //5 //3d4

not in range [^ ] not

var chars = 'abc123B456';

chars.search(/[^0-9a-z/); //6 //B

Quantifiers

optional single occurrence-0 or 1

/[a-z][0-9]?[a-z]/.test('a1b'); //true

/[a-z][0-9]?[a-z]/.test('abc'); //true

/[a-z][0-9]?[a-z]/.test('a123b'); //false

multiple occurrences - 0 or more

/[a-z][0-9]\*[a-z]/.test('a123b'); //true

starts with and ends with matches

/^[a-z][0-9]/.test('a1b'); //true //^[ ]--starts with

/^[a-z][0-9]/.test('ab12');

/^[a-z][a-z]$/.test('123abc'); //true //$ -ends with

/^[a-z][a-z]$/.test('123abc456'); //false

exact match /^..$/

/^[a-z][0- 9][a-z]$/.test('a1b') ; //true

/^[a-z][0- 9][a-z]$/.test('a1b2c'); //false

/^[a-z][0- 9a-z]\*[a-z]$/.test('a1b2c'); //true

**2.7 intro to the DOM**

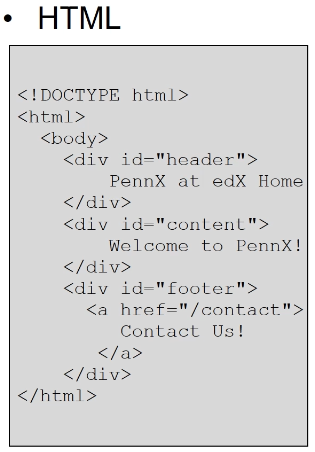
DOM

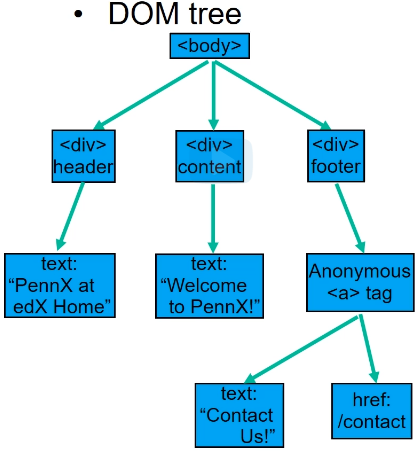
- document object Model

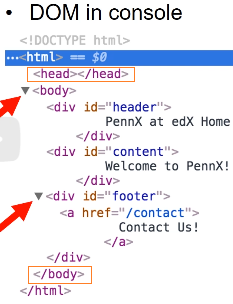
- structured tree representation of a web page

- HTML of every web page is turned into a DOM representation by browser.

- HTML Specifies Structure of content - DOM allows us to access the structure



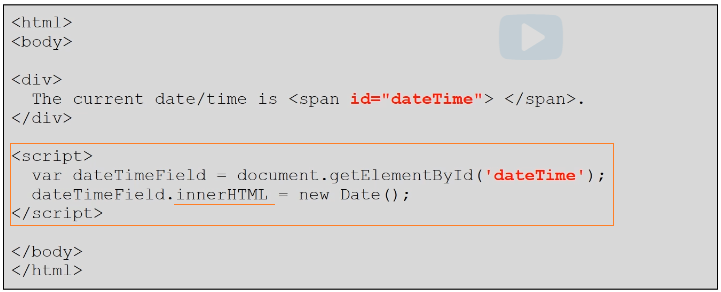




- global root element - document -> html

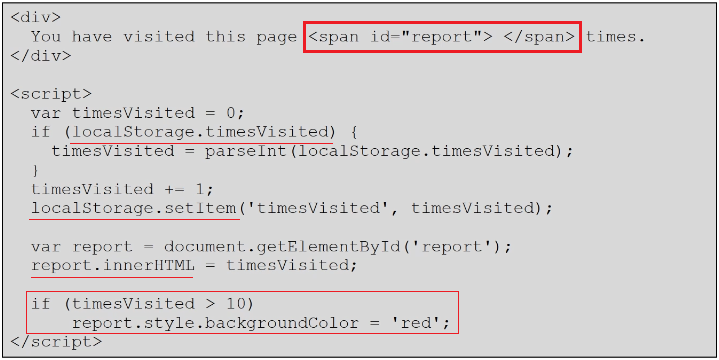
- elements un DOM tree can be retrieved and manipulated

JS to manipulate HTML



Store data across multiple page request

- global variable - localStorage



**2.7 Object as JSON**

-javascript object notification

-storing object in local storage

-textual representation of JS

-object can be store as String.

-exchange data between programs

-key-value pair

Storing JSON

- objects in between page request

-JS object converted to JSON

JSON.Stringify(myObject)

-String to JS Object

JSON.parse(jsonString)

All value must be a -

string,

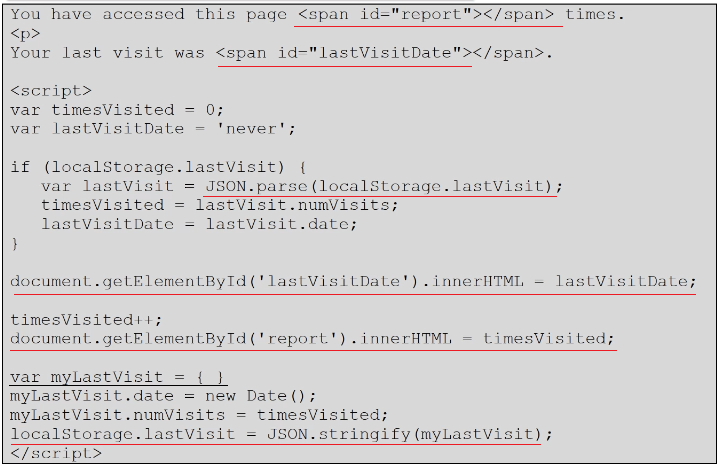
number,

array,

boolean,

null,

another JSON object



**2.8 DOM Event**

- Event driven programming.

- program behavior based on Event

- call back function

- asynchronous programming

- synchronous prog - wait for user input .keep checking

**2.9 Introduction to jQuery**

- Simplifies JS usage on web apps

- Easy with DOM

- Cross browser support

- Animation and effects

- download jQuery.com

- add downloaded .js file to HTML page

<script src="jQueryFile.js"> </script>

- select DOM using $

$("\*") all elements

$(this) current element

$("div") all <div> element

$(".title") class="title"

$("#name") id ="name"

-Manipulation

$(selector).action(args...)

$("#name").html("hello");

$("#name").append(" world!");

$("#name").addClass("greeting");

$("#name").hide();

$("#name").show();

- Add event listener

$(selector).event(callback)

callback

-name of event

-name of function

**2.10 jQuery event handling**

select elements of DOM

Advance Selector:

$(someNodes).find(selector)

$ selectors can be chained

- $("div.book") : -div with class="book"

-$("div, .book"):- all divs and all class="book"

-$("p:hidden"):- element p with property hidden

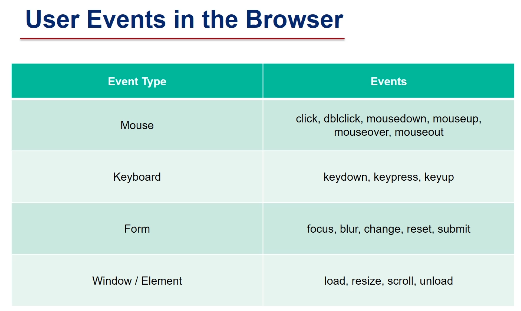
-$ these selectors are css selectors







2.11 jQuery Event handling 2



3.1 Introduction to React

- jQuery library to manipulate DOM

React:

- build user interfaces

- webpage is composed of components that have a lifecycle and state change, which affects how they are rendered.

- introduced - virtual DOM which allows add/Remove elements

- Facebook

3 Features

1. Modularity

- organize module that can work together

2. Lifecycle Maintenance

- modifying component based on its state,

- add event listeners

- simplify conditional component render

3. JSX

- embed html into javascript

Components

- core of react

- make nodes inside VirtualDom

- each component include and maintain state that changes with events

- components maintain their state and application can be configured to respond to state of individual component.

Virtual DOM

- structural html elements

- Virtual DOM: selectively renders-re-renders subtree of nodes based on state changes

- Virtual DOM does least amount of DOM Manipulation to update components

Normal DOM - node updated - browser updates all nodes

Virtual DOM - node updated - React figures out which part of DOM need to change, only changes nodes that are affected.

-diff : identify node that is changes

-reconciliation : update nodes that are affected by the change

re-render only node affected by change

Developers steps with React:

1. within HTML - allocate position on page in which desired react component will be rendered eg: div

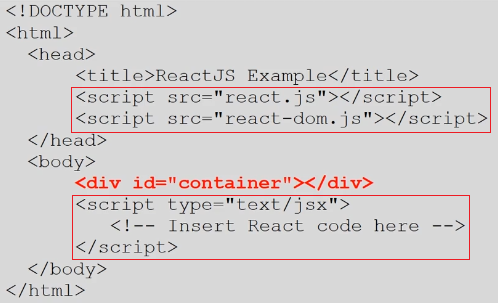
2. create react component in JS

- establish an initial state

- define any event that could change the component state over lifecycle

- render function for the HTML

3.Drop component into position allocated in Step1

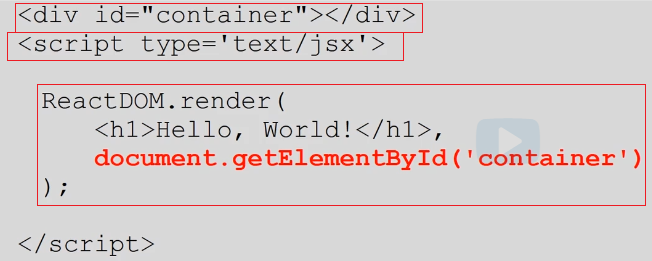


JSX

- JS XML Syntax Transform

- HTML tag within JS

- Converts text(HTML) to React code



3.2 React Component

- modular reusable code

- drop into html

- generate content dynamically based on state and use events

- js object -based off React.Component prototype

- 4 parts

1. properties

2. event-based state variable

3. call back function/ event handlers

4. render function

- Virtual DOM manages each component

- and calls render function

ES6

- recent version of js

- allows a class instead of single object

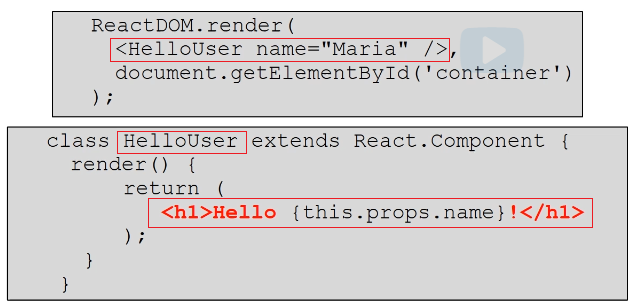
React component Attribute

1. Properties

- attributes and values when component is created

- should never be modified after initialization

- access properties through this.props



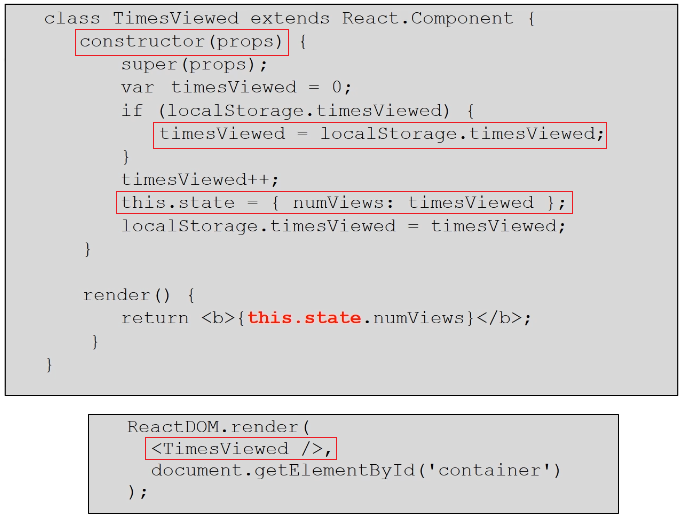
2.State

- attribute and values that represent current state of the component

- can be modified during component lifecycle

- should be initialized in the constructor

- access state by this.state



both can be used while rendering the component

properties -initialize

state -change during

Component Lifecycle

- React VirtualDOM invokes callback functions on component during lifecycle

- 3 stages

1. creating/mounting

2.updating

3. destroying / unmounting

-at each point react invoke optional functions can be written to control the component

Call Back functions

I. Mounting

- Component is created or added to VirtualDOM

1. constructor: creates component, initializes states based on properties

2. componentWillMount: invoked before component is added to VirtualDOM

3. componentDidMount: involved after component has been added to VirtualDOM and has been rendered

II. Updating:

- Called components prop or state is changing and component is re-rendered

1. componentWillReceiveProps: invoked before receiving new props. Eg: when its parent components are re-rendered

2. shouldComponentUpdate: Can be used to determine whether to re-render

3. componentWillUpdate: invoked before re-rendering after changing the state

4. componentDidUpdate: invoked after being re-rendered.

III. Unmounting:

- called when a component is being removed from the virtualDOM

1. componentWillUnmount: Invoked before component is removed from VirtualDOM and destroyed