

HTML	
HTML: <ul style="list-style-type: none"> Widely used language that gives our web page structure HTML documents are described in HTML tags. Makes the text more interactive and dynamic 	Stylesheet: <ul style="list-style-type: none"> Used to build a consistent, transportable, and well-designed style template. We can use this templates on several different webpages
Markup Language: <ul style="list-style-type: none"> Annotating a document in a way that is syntactically distinguishable from the text. 	<!DOCTYPE html> There are many types of html like e.g. HTML 4.01 Strict, HTML 4.01 Transitional, HTML 4.01 Frameset, etc.. So it is used to instruct the web browser about the HTML.
Tags Vs Elements in HTML: <ul style="list-style-type: none"> HTML document contain tags, but not contain the elements. Elements are usually generated after parsing step from these tags. Elements communicate to the browser to render text. Elements are enclosed by <> and form tags. 	Responsive Web Design: Is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices. To make responsive web pages, add the following <meta> element <meta name="viewport" content="width=device-width, initial-scale=1.0">
Physical tag: <ul style="list-style-type: none"> Tell browser how to display the text enclosed in the physical tag. E.g. <big>, Logical tag: <ul style="list-style-type: none"> Tell browser the meaning of the enclosed text. E.g. <important> 	<head> is a container for metadata The following tags describe metadata: <title>, <style>, <meta>, <link>, <script>, and <base>
XHTML: <ul style="list-style-type: none"> Extensive hypertext markup language, a part of family of XML markup language. It is HTML written in XML Stricter than HTML Documents must be documented correctly (well-formed) 	HTML5 Intro: <!DOCTYPE html> is mandatory HTML5 defines 8 new semantic elements All these are block level elements. HTML5 tags: <ul style="list-style-type: none"> Color Date Datetime-local Email Time Url Range Telephone Number Search
Types of lists in html: (unordered), (ordered), (list item)	
"mailto:name@example.com?Subject=" Opens default email program on the visitors computer to compose an email.	
H1 to H6 are different types of headings supported by HTML	
Grouping in HTML: Used to group several HTML controls like input, textarea, selects, etc. <fieldset> element is used for Grouping. It is used to group related elements in a form.	Semantic HTML: Introduces meaning to the webpage rather than just presentation. It is use of html markup to reinforce the semantics or meaning of the content. In semantic html is not used for bold statement. Instead of these we use E.g. <form>, <table>, <article>, <header>, <footer>
Difference between SPAN and DIV: Groups chunk of code. Span is inline and group is block-line.	

Create hyperlink: using anchor tag 	With HTML4 developers used their own id/class names to style elements. With HTML5 this made easier.
<p>HTML tags used to display data in tabular form:</p> <p><table> - defines table</p> <p><tr> - row in a table</p> <p><th> - header cell in a table</p> <p><td> - defines a cell in a table</p> <p><caption> - defines table caption</p> <p><colgroup> - group of 1/more columns in a table for formatting</p> <p><col> - used with <colgroup> element to specify column properties for each column.</p> <p><tbody> - used to group body content in a table.</p> <p><thead> - used to group the header content in a table.</p> <p><tfoot> - used to group the footer content in a table.</p>	<div>HTML Layout Elements:</div> <div> <header> A header element specifies a header for a document or section. Should be used as a container for introductory content. </div> <div> <nav> Defines a set of navigation links </div> <div> <section> A home page could normally split into sections for introduction, content and contact information </div> <div> <article> Specified independent, self-contained content. E.g. forum post, blog post, newspaper article. </div> <div> <aside> Defines some content aside from the content it is placed in (like a sidebar) </div> <div> <footer> Specifies the footer for a document or section. </div> <div> <figure> and <figcaption>: <figure> adds visual explanation to an image. <figcaption> element defines the caption. </div>
<marquee> is used to put the scrolling text on web page.	
<iframe src="" /> is used to display web page inside a webpage	
<p>URL encoding:</p> <ul style="list-style-type: none"> A URL is encoded to convert non-ascii characters into the format that can be used over the internet because a URL is sent over the internet by using ASCII character-set only. Non-ASCII characters are replaced with a % followed by hexadecimal digits. 	
<p>HTML API's:</p> <div> <p>HTML Geolocation:</p> <p>getCurrentPosition() method is used to return the user's position.</p> <p>E.g. navigator.geolocation.getCurrentPosition(showPosition)</p> </div> <div> <p>Drag and Drop:</p> <p>When we grab an object and drag in to a different location.</p> <p>Mark an element draggable </p> <p>What to drag - ondragover</p> <pre>function drag(ev) { ev.dataTransfer.setData("text", ev.target.id) } Do the drop - ondrop function drop(ev) { ev.preventDefault(); var data = ev.dataTransfer.getData("text"); ev.target.appendChild(document.getElementById(data)); }</pre> </div>	<div>HTML Forms:</div> <div> <form> element: Defines a form that is used to collect user input. <input> element is the most important form element. </div>

HTML web storage:

- With web storage, web application can store data locally within the user's browser.
- Before HTML5, application data had to be stored in cookies, included in every server request.
- Web storage is more secure and large amount of the data can be stored locally, without affecting website performance.
- Window.localStorage - stores data with no expiration date
- Window.sessionstorage - stores data for one session

HTML Input Types:

- | | |
|--|--|
| <ul style="list-style-type: none">• Button• Checkbox• Color• Date• Datetime-local• Email• File• Hidden• Image• Month• password | <ul style="list-style-type: none">• Radio• Range• Reset• Search• Submit• Tel• Text• Time• Url• week |
|--|--|

<submit> defines a button for submitting the form data to a form-handler.

<form action="/some.php">: action attribute defines the action to be performed when the form is submitted.

<form action="/some.php" target="_blank">: target attribute specifies if the submitted result will open in a new browser tab, a frame, or in current window

<form action="/some.php" method="get">: method attribute specifies the http method (GET or POST) to be used when submitting the form data.

<select> element defines a dropdown list.

<option> element defines an option that can be selected.

<select name="" size="3"> size attribute to specify the number of visible values.

<select name="" multiple>: multiple attribute to allow the user to select more than one value.

<textarea> multi-line input field.

<button> element defines a clickable button.

<datalist> element specifies a list of predefined options for an <input> element

<output> element represents the result of a calculation.

HTML Input attributes:

- | | |
|---|---|
| <ul style="list-style-type: none">• Value• Readonly• Disabled• Size• Maxlength• required | <ul style="list-style-type: none">• Autocomplete• Autofocus• Form• Formaction• novalidate |
|---|---|

HTML event attributes:

Window events:

onerror, onload, onmessage, onoffline, ononline, onresize, onstorage, onunload, onpagehide, onpageshow

Keyboard events:

onkeydown, onkeypress, onkeyup

Drag events:

ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onscroll

Form events:

onblur, onchange, oncontextmenu, onfocus, oninput, oninvalid, onreset, onsearch, onselect, onsubmit

Mouse events:

onclick, ondblclick, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onwheel

Clipboard events:

oncopy, oncut, onpaste

How do you apply CSS styles to a webpage?

- Inline (<div style="">)
- Style block (<style>..</style>)
- Link to CSS (<link rel="stylesheet" href="style.css">)

How do you apply JS to a webpage?

- Inline (<div onclick="">)
- Script block (<script>..</script>)

- Link to JS (<script src="source.js"/>)

HTTP (Hypertext transfer protocol)

HTTP Status messages:

1xx: Information

100 continue

101 Switching Protocols

103 CheckPoint

2xx: Successful

200 OK

201 Created

202 Accepted

203 Non-Authoritative Information

204 No content

3xx: Redirection

300 Multiple Choices

301 Moved Permanently

302 Found

303 See Other

304 Not Modified

4xx: Client Error

400 Bad request

401 unauthorized

402 payment required

403 forbidden

404 Not found

408 request timeout

5xx: Server error

500 Internal Server Error

501 Not implemented

502 Bad Gateway

503 Service unavailable

504 Gateway timeout

HTTP works as a request response protocol between a client and server

HTTP Methods:

- Get
- Post
- Put
- Head
- Delete
- Patch
- options

Get:

- Used to request data from a specified resource
- Can be cached
- Remain in the browser history
- Can be bookmarked
- Should never be used when dealing with sensitive data.
- Have length restrictions
- Only used to request data

Post:

- Send data to the server to create/update a resource.
- Requests are never cached
- Do not remain in browser history
- Cannot be bookmarked
- No restrictions on data length

Put:

- used to send data to server to create/update a resource
- Put requests are idempotent and post is not.

Head:

- Head is almost identical to get, but without the response body.
- Head request are useful for checking what a get request will return before actually making a get request.

Delete:

- Deletes the specified resource.

Options:

- Describes the communication options for target resource.

Constraint validation HTML input attributes:

- disabled
- max
- min
- pattern
- required
- type

Constraint validation CSS pseudo selectors:

- :disabled
- :invalid
- :optional
- :required
- :valid

CSS (Cascading style sheets) web designing language

Advantages of CSS:

- It gives lots of flexibility for setting the properties of an element.
- Easy maintenance
- Allows separation of content of the HTML document from the style and layout of the content basically.
- Loading of pages at a faster pace.
- Compatibility with multiple device.
- Increases the websites adaptability and makes it compatible to future browsers.

Syntax:

Selector { //declaration
Property: value;
}

Selectors:

Element selector: e.g.
p {...}

Id selector: uses id attribute of an html element, e.g.
#para1 {...}

Class selector: selects element with specific class attribute.
.center {...}, p.center {...}

Grouping selectors: e.g.
h1, h2, p {...}

- Backgrounds: Used to define background effects for elements.
- Borders: specify style, width and color of elements border.
- Margins: Used to create space around elements.
- Padding: used to generated space around elements content.
- Height and width to set height and width of an element.
- Box model: is essentially a box that wraps around every html element. It consists of margins, border, padding, content
- Outline: is a line that is drawn around the elements to make the element stand out.

JavaScript

Pros and Cons of functional programming vs object-oriented programming:

OOP pros:

- It's easy to understand the basic concept of objects and easy to interpret the meaning of method calls.

FP pros:

- Using the functional paradigm, programmers avoid any shared state or side-effects, which eliminates bugs caused by multiple functions competing for the same resource.
- Computation that makes use of pure functions is also easy to scale across multiple processors, or across distributed computing clusters without fear of threading resource conflicts, race conditions, etc...

OOP cons:

- OOP typically depends on shared state.
- Object and behaviours are typically tacked together on the same entity, which may be accessed at random by any number of functions with non-deterministic order, which may lead to undesirable behavior such as race conditions.

FP cons:

- Over exploitation of FP features such as point-free style and large compositions can potentially reduce readability because the resulting code is often more abstractly specified, more terse, and less concrete.

- JS is a dynamic computer programming language. It is

Functional programming (also: closures, first class functions, lambdas):

<p>lightweight and most commonly used as part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.</p> <ul style="list-style-type: none">It is an interpreted programming language with object oriented capabilities.JS is a multi-paradigm language, supporting imperative/procedural programming along with OOP and functional programming.JS supports OOP with prototypal inheritance.OOLO (object linking other objects)/ prototypal inheritance		<ul style="list-style-type: none">Functional programming produces programs by composing mathematical functions and avoid shared state and mutable data.Functional programming is the essential concept in JS. Several common functional utilities were added to JS in ES5.	
		Advantages: <ul style="list-style-type: none">Less server interactionImmediate feedback to the visitorsIncreased interactivityRicher interfaces	
Limitations: <ul style="list-style-type: none">Client-side javascript does not allow the reading or writing of files.Client side javascript doesn't have any multithreading or multi-processor capabilities.		What JS can do? <ul style="list-style-type: none">Can change html contentCan change html attribute valuesCan change html stylesCan hide/show html elements	
JS display possibilities: <ul style="list-style-type: none">innerHTMLdocument.write()window.alert()console.log()		JD datatypes: <ul style="list-style-type: none">NumbersStringsObjectsBooleansArraysUndefinedNull -> objectUndefined and null are equal in value but different in type <p>typeof operator is used to check the type of variable</p>	
JS function: <ul style="list-style-type: none">Is a block of code designed to perform a particular task.Defined with function keyword, followed by a name, followed by parenthesesThe code inside the function will execute when something invokes (calls) the function<ul style="list-style-type: none">When an event occursInvoked from JS codeSelf invokedWhy functions?<ul style="list-style-type: none">To reuse code. Define code once and use it multiple times() operator invokes the function.Functions can be used as variable values.Variables added within the JS function, becomes LOCAL to the function.		JS objects: <ul style="list-style-type: none">Creation:<ul style="list-style-type: none">Using object literalVar person = {fn: "krish", ln:"u"};Name:values pairs in JS objects are called properties.Object methods: methods are actions that can be performed on objects.Methods are stored in properties as function definitions.this refer to owner of the function.Do not declare strings, numbers and booleans as objects	
String methods: <ul style="list-style-type: none">LengthindexOflastIndexOfSearchSlice	Number methods: <ul style="list-style-type: none">toStringtoExponential()toFixed()toPrecision()valueOf()	Array methods: <ul style="list-style-type: none">toString()join()pop()push()shift() -> pop first	Math function: <ul style="list-style-type: none">Math.max.apply(null, array)Math.min.apply(null, array)PI

<ul style="list-style-type: none"> ● Substring ● Substr ● Replace ● toUpperCase ● toLowerCase ● Concat ● Trim ● charAt ● charCodeAt ● split 	<p>Number properties:</p> <ul style="list-style-type: none"> ● MIN_VALUE ● MAX_VALUE ● POSITIVE_INFINITY ● NEGATIVE_INFINITY ● NaN 	<ul style="list-style-type: none"> ● Changing element: <ul style="list-style-type: none"> ○ fruits[0] = "Kiwi"; ● Deleting element: <ul style="list-style-type: none"> ○ Delete fruits[0]; ● Splicing an array: <ul style="list-style-type: none"> ○ fruits.splice(2, 0, "L", "K"); ● Concat ● sort() ● reverse() ● forEach() <ul style="list-style-type: none"> ○ value, index, array ● map() <ul style="list-style-type: none"> ○ value, index, array ● filter() <ul style="list-style-type: none"> ○ value, index, array ● reduce() <ul style="list-style-type: none"> ○ total, value, index, array ● reduceRight() ● every() ● some() ● indexOf() ● lastIndexOf() ● find() ● findIndex() 	<ul style="list-style-type: none"> ● round ● pow ● sqrt ● abs ● Ceil ● floor ● sin ● cos ● min ● max ● random <p>Date methods: ISO date: "YYYY-MM-DD"</p> <ul style="list-style-type: none"> ● getFullYear() ● getMonth() ● getDate() ● getHours() ● getMinutes() ● getSeconds() ● getMilliseconds() ● getTime() ● getDay() ● Date.now()
---	---	---	---

<p>== vs ===:</p> <ul style="list-style-type: none"> ● == values are equal ● === value and types are equal 	<p>Regular expression:</p> <ul style="list-style-type: none"> ● Sequence of characters that form a search expression ● Syntax: /pattern/modifiers
<p>JS scopes:</p> <p>There are 2 types of scopes:</p> <ul style="list-style-type: none"> ● Local scope - variable declared within a JS function ● Global scope - variable declared outside JS function <p>Automatically global:</p> <p>Assign value to a variable that has not been declared.</p> <p>In HTML, the global scope is window object. All global variables belong to window object.</p>	<p>Hositing:</p> <p>In JS, a variable can be declared after it has been used. It is JS's default behavior of moving all declarations to the top of the current scope. JS only hoists declarations, not initializations.</p> <p>Strict mode:</p> <ul style="list-style-type: none"> ● "Use strict"; defines that JS code should be executed in "strict mode". ● It makes it easier to write secure JS. ● It changes previously accepted "bad syntax" into real errors. ● Using a variable/object without declaring it is not allowed. ● Deleting a variable/object/function is not allowed. ● Duplicating a parameter name is not allowed ● with and eval are not allowed
<p>"this" keyword refers to the object it belongs to.</p> <ul style="list-style-type: none"> ● In a method, this refers to owners object. ● Alone, this refers to global object. ● In a function, this refers to global object. ● In a function, in strict mode, this is undefined. ● In an event, this refers to element that received the 	

<p>event.</p> <ul style="list-style-type: none"> Methods like <code>call()</code>, and <code>apply()</code> can refer this to any object. 	
<p><code>call()</code> and <code>apply()</code> are predefined JS methods. They can both be used to call object method with another object as argument.</p> <p>E.g. <code>person1.fullName.call(person2)</code></p>	<p>JS block scope:</p> <ul style="list-style-type: none"> Variables declared with the <code>var</code> keyword cannot have block scope. Variables declared inside a block <code>{}</code> can be accessed from outside the block. Re-declaring a variable using <code>var</code> keyword can impose problems.
<p><code>const</code>:</p> <ul style="list-style-type: none"> Variables declared with <code>const</code> behave like <code>let</code> variables, except they cannot be reassigned. Declaring a variable with <code>const</code> is similar to <code>let</code> when it comes to block scope. JS <code>const</code> variables must be assigned a value when they are declared. It does not define a constant value. It defines a constant reference to a value. Because of this, we cannot change constant primitive values, but we can change the properties of constant objects/elements of constant arrays. Re-declaring a variable with <code>const</code>, in another scope, or in another block, is allowed. 	<p><code>let</code>:</p> <ul style="list-style-type: none"> Variables declared with <code>let</code> keyword can have block scope. Variables declared inside a block can not be accessed from outside the block. Re-declaring a variable using the <code>let</code> keyword can solve the problem arises using <code>var</code> keyword. Can be used in a loop. Similar to <code>var</code> when the variable is declared inside a function. Global variables defined with the <code>let</code> keyword do not belong to the window object. Re-declaring a <code>var</code> variable with <code>let</code>, in the same scope or in the same block, is not allowed. Re-declaring a <code>let</code> variable with <code>let/var</code>, in the same scope, or in the same block is not allowed. Re-declaring a <code>let</code> variable with <code>let</code>, in another scope, or in another block, is allowed. Variables declared with <code>let</code> are not hoisted to the top. Using a <code>let</code> variable before it is declared will result in a <code>ReferenceError</code>.
<p>JS best practices:</p> <ul style="list-style-type: none"> Avoid global variables Always declare local variables Declarations on top Initialize variables when you declare them. Never declare primitives as objects Don't use <code>new</code> object. Beware of automatic type conversions. Use <code>===</code> comparison Use parameter defaults End your switches with defaults. Avoid using <code>eval</code>. Reduce dom access Reduce dom size Avoid unnecessary variables Delay JS loading by placing JS at the end of the page. Avoid using <code>with</code> keyword, it has negative effect on speed. 	<p>ES5:</p> <ul style="list-style-type: none"> Added "strict mode". Added JSON support. Added <code>String.trim()</code> Added <code>Array.isArray()</code> Added Array iteration methods. <ul style="list-style-type: none"> <code>map()</code>, <code>filter()</code>, <code>reduce()</code>, <code>reduceRight()</code>, <code>every()</code>, <code>some()</code>, <code>indexOf()</code>, <code>lastIndexOf()</code>, <code>JSON.parse()</code> <code>JSON.stringify()</code> <code>Date.now()</code> <code>Object.create()</code> to create object Getters and Setters allow you to define Object accessors
<p>ES6:</p> <ul style="list-style-type: none"> Added <code>let</code> and <code>const</code> Added default parameter values. Added <code>Array.find()</code> Added <code>Array.findIndex()</code> Exponential <code>**</code> 	<p>Constraint validation DOM methods:</p> <ul style="list-style-type: none"> <code>checkValidity()</code> <code>setCustomValidity()</code> <p>Constraint validation DOM properties:</p> <ul style="list-style-type: none"> <code>Validity</code> <code>validationMessage</code> <code>willValidate</code>
<p>Why using getters and setters?</p>	<p>Validity properties:</p>

<ul style="list-style-type: none"> ● It gives simpler syntax ● It allows equal syntax for properties and methods ● It can secure better data quality ● It is useful for doing things behind the scenes. <p>Object.defineProperty() method can also be used to add getters and setters.</p>	<ul style="list-style-type: none"> ● customError ● patternMismatch ● rangeOverflow ● rangeUnderflow ● stepMismatch ● tooLong ● typeMismatch ● valueMissing ● valid
<p>Object constructor:</p> <pre>function Person(first, last, age, eye) { this.firstName = first; this.lastName = last; this.age = age; this.eyecolor = eye; }</pre> <p>You cannot add a new property to the object constructor the same way you add a new property to an existing object.</p>	<p>Prototype inheritance:</p> <ul style="list-style-type: none"> ● All JS objects inherit properties and methods from a prototype. ● Object.prototype is on the top of the prototype inheritance chain. ● The JS prototype property allows you to add new properties / methods to object constructors. ● Only modify your own prototypes. Never modify the prototypes of standard JS objects.
<p>Classical Inheritance vs Prototypal inheritance:</p>	
<p>Class Inheritance:</p> <ul style="list-style-type: none"> ● Instances inherit from classes and create sub-class relationships: hierarchical class taxonomies. ● Instances are typically instantiated via constructor functions with the 'new' keyword. ● Classes: create tight coupling or hierarchies/taxonomies 	<p>Prototypal Inheritance:</p> <ul style="list-style-type: none"> ● Instances inherit directly from other objects. ● Instances are typically instantiated via factory functions or 'Object.create()'. ● Instances may be composed from many different objects, allowing for easy selective inheritance. ● In JS, prototypal inheritance is simpler & more flexible than class inheritance.
<p>When is Prototypal inheritance appropriate?</p> <ul style="list-style-type: none"> ● Delegation: Allow an object to delegate behavior to its prototype. ● Composition: your object contains another object, called its prototype. ● Concatenative: <i>mixin</i> provides methods that implement a certain behavior, but we do not use it alone, we use it to add the behavior to other classes. <pre>// mixin let sayHiMixin = { sayHi() { alert(`Hello \${this.name}`); }, sayBye() { alert(`Bye \${this.name}`); } }; // usage: class User { constructor(name) {</pre>	<p>Object composition over class inheritance:</p> <ul style="list-style-type: none"> ● In other words, use can-do, has-a, or uses-a relationships instead of is-a relationships. ● Avoid tight coupling. ● Avoid the gorilla banana problem ● Make code more flexible..

<pre> this.name = name; } } // copy the methods Object.assign(User.prototype, sayHiMixin); </pre> <p>The prototypal inheritance is useful to enable composition, which creates has-a or uses-a or can-do relationships as opposed to the is-a relationship.</p>	
<p>ES5 New Object Methods:</p> <ul style="list-style-type: none"> Object.defineProperty(object, Property, descriptor) defineProperties(object, descriptors) getOwnPropertyDescriptor Keys getPrototypeOf(object) preventExtensions isExtensible seal isSealed freeze isFrozen 	<p>Changing metadata:</p> <ul style="list-style-type: none"> writable: true // value can be changed enumerable: true // property can be enumerated configurable: true // can be reconfigured <p>JS call() method:</p> <ul style="list-style-type: none"> It is a predefined JS method With call an object can use a method belonging to other object. <p>JS apply() method:</p> <ul style="list-style-type: none"> Similar to call method call() method takes arguments separately apply() method takes arguments as an array.
<p>Function definitions:</p> <ul style="list-style-type: none"> Function declarations: function name(params) {...} Function Expressions: var x = function(params) {...} Function() constructor: var myFunc = new Function("a", "b", "return a*b"); Function Hoisting: myFunction(); function myFunction() {...} Self-Invoking functions: (function () {...}) (); Functions can be used as values: function myFunc() {...} var x = myFunc(); Functions are objects: The typeof operator in JS return "function" for functions. But JS functions can be best described as objects. JS functions have both properties and methods. Arguments.length property returns the number of arguments received when a function was invoked. Arrow Functions: 	<p>JS Closures:</p> <p>A counter dilemma:</p> <ul style="list-style-type: none"> Suppose you want to use a variable for counting something, and you want this counter to be available to all functions. You could use a global variable, and a function to increase the counter. There is a problem with the solution above: Any code on the page can change the counter. <p>JS nested functions:</p> <ul style="list-style-type: none"> JS supports nested functions. Nested functions have access to scope "above" them. <p>Self-invoking function: (function () {...}) ();</p> <p>Closure uses nested and self-invoking functions to create a private scope variable, which can only be accessed through exposed functions. E.g.</p> <pre> var add = (function () { var counter = 0; return function () {counter += 1; return counter} })(); add(); </pre> <p>JS promises (ES6):</p> <ul style="list-style-type: none"> A promise is an object that may produce a single value some time in the future: either a resolved value or a

<p>Show short syntax for writing functions.</p> <pre>const x = (x, y) => x * y; // ES6</pre> <p>Using const is safe than val because function expression is always constant value.</p> <ul style="list-style-type: none"> Function arguments are passed by value. Objects are passed by references 	<p>reason why it is not resolved.</p> <ul style="list-style-type: none"> A promise may be in one of the 3 states: fulfilled, rejected, or pending. A promise is an object which can be returned synchronously from an asynchronous function. <ul style="list-style-type: none"> Fulfilled: onFulfilled() will be called (e.g., resolve() was called) Rejected: onRejected() will be called (e.g., reject() was called) Pending: not yet fulfilled or rejected.
<p>Promise Chaining:</p> <p>Because .then() always returns a new promise, it's possible to chain promises with precise control over how and where errors are handled.</p>	
<p>Promise e.g.</p> <pre>Let promise = new Promise(function(resolve, reject){ if(somecondition) { resolve(fromResolve); } else { reject(fromReject); } }); promise.then(function(fromResolve) { // promise is resolved. }).catch(function(fromReject){ // promise is rejected }) promise().then(function(result){ return new promise2(result); }).then(function(result){ return new promise3(result); }).then(function(result){ return new promise4(result); })... Promise.all([promise(), promise2(), promise3(),...]).then(function() { });</pre>	<ul style="list-style-type: none"> Promises following the spec must follow a specified set of rules: <ul style="list-style-type: none"> A promise or “thenable” is an object that supplies a standard-compliant .then() method. A pending promise may transition into a fulfilled or rejected state. A fulfilled or rejected promise is settled, and must not transition into any other state. Once a promise is settled, it must have a value (which may be undefined). That value must not change. .then() method must comply with these rules: <ul style="list-style-type: none"> Both onFulfilled() and onRejected() are optional If the args are not functions, they must be ignored. onFulfilled() will be called after the promise is fulfilled. onRejected() will be called after the promise is rejected. Neither onFulfilled() nor onRejected() may be called more than once. .then() may be called many times on the same promise. .then() must return a new promise. If either of the methods throws an exception e, promise2 must be rejected with e as the reason.
<p>Event Bubbling:</p> <ul style="list-style-type: none"> When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors. Almost all the events bubble from the inner element up through parents like a bubble in the water. The most deeply nested element that caused the event is called target element, accessible as event.target. this(=event.currentTarget) : this - is the current element, the one that has a currently running handler on it. The method to stop event propagation is: event.stopPropagation(). To stop bubbling and prevent handlers on the current element from running, there is a method. After it no 	<p>Capturing:</p> <ul style="list-style-type: none"> There is another phase of event called capturing. It is rarely used in code, but sometimes can be useful. The standard DOM events describe 3 phases of event propagation: <ul style="list-style-type: none"> Capturing phase: <ul style="list-style-type: none"> The event goes down the element. Target phase: <ul style="list-style-type: none"> The event reached the target element. Bubbling Phase: <ul style="list-style-type: none"> The event bubbles up from the element. To catch an event on the capturing phase, we need to set the 3rd argument of addEventListener(event,

other handlers execute. event.stopImmediatePropagation() <ul style="list-style-type: none"> Bubbling is a convenient, don't stop it without a real need. Sometime event.stopPropagation() creates hidden pitfalls that later may become problems. 	handler) to true.
Event Handlers can access event object properties: <ul style="list-style-type: none"> event.target – the deepest element that originated the event. event.currentTarget (=this) – the current element that handles the event (the one that has the handler on it) event.eventPhase – the current phase (capturing=1, bubbling=3). 	Summary: <ul style="list-style-type: none"> When an event happens – the most nested element where it happens gets labeled as the “target element” (event.target). Then the event first moves from the document root down to the event.target, calling handlers assigned with addEventListener(...., true) on the way. Then the event moves from event.target up to the root, calling handlers assigned using on<event> and addEventListener without the 3rd argument or with the 3rd argument false.

DOM (Document Object Model)

<ul style="list-style-type: none"> HTML DOM object is constructed as tree of objects. JS can change all the HTML elements in the page JS can change all the HTML attributes in the page JS can change all the CSS styles in the page JS can remove existing HTML elements and attributes JS can add new HTML elements and attributes JS can react to all existing HTML events in the page JS can create new HTML events in the page 	DOM method: document.getElementById document.getElementsByTagName document.getElementsByClassName document.querySelectorAll document.createElement document.removeChild document.appendChild document.replaceChild document.write element.setAttribute()
JS Browser BOM <ul style="list-style-type: none"> window.document Global variables are properties of window object. Global functions are methods of window object. window.innerHeight window.innerWidth window.screen window.location window.history window.navigator window.alert window.setTimeout window.setInterval window.document.cookie 	DOM property: element.innerHTML element.attribute Element.style.property Adding event handlers: document.getElementById(id).onclick = function(){...} DOM EventListener: element.addEventListener(event, function, useCapture);

AJAX (Asynchronous Javascript And XML)

<ul style="list-style-type: none"> Read data from a web server. Update a web page without reloading the page. Send data to web server - in the background. 	Ajax just uses a combination of: <ul style="list-style-type: none"> A browser built-in XMLHttpRequest object (to request data from a web server) JS and HTML DOM (to display or use the data)
How AJAX works: <ul style="list-style-type: none"> An event occurs in a web page (the page is loaded, a button is clicked) An XMLHttpRequest object is created by JS. The XMLHttpRequest object sends a request to a web 	<pre>var xhttp = new XMLHttpRequest(); xhttp.onreadystatechange = function() { if (this.readyState == 4 && this.status == 200) { document.getElementById("demo").innerHTML = this.responseText;</pre>

server. <ul style="list-style-type: none">• The server processes the request• The server sends a response back to the web page.• The response is read by JS• Proper action is performed by JS	<pre>} }; xhttp.open("GET", "ajax_info.txt", true); xhttp.send();</pre>
JSON (JS Object Notation)	
<ul style="list-style-type: none">• Syntax for storing and exchanging data.• JSON is text, written with JS object notation.• JSON is lightweight data-interchange format.• JSON.stringify(myObj) // object to text• JSON.parse(myJSON) // text to object	JSON syntax: <ul style="list-style-type: none">• Data is in name/value pairs• Data is separated by commas• Curly braces hold objects• Square brackets hold arrays
JSON data types: <ul style="list-style-type: none">• string• number• object• array• boolean• null	
JQUERY	
JQuery selectors: <ul style="list-style-type: none">• Element by id: \$("#id");• Element by tag name: \$("p");• Element by class name: \$(".class");• Element by css selector: \$("p.class");	JQuery HTML: Set text content: myElem.text("some text"); Get text content: myElem.text(); Set html content: myElem.html("<p>some html</p>"); Get html content: var content = myElem.html();
JQuery CSS: Hiding html element: myElem.hide(); Show html element: myElem.show(); Show html elements: myElem.css("font-size","35px");	
JQuery DOM: Remove an html elem: \$("#id").remove();	Get Parent elem: myElem.parent();
MVC, MVP, MVVM	
These are some of the common patterns to guide programmers towards creating decoupled solutions.	The software behaviors that are common to MVC, MVP and MVVM are: <ul style="list-style-type: none">• Data Layer / Business Logic (Model)• Presentation Layer / UI (View)• Application Logic (Controller, presentation or view model)
Data Layer / Business Logic (Model): This is the behavior which applies the business logic to the application's data. A domain model usually represents the model where the objects are used to mimic the real world entities.	

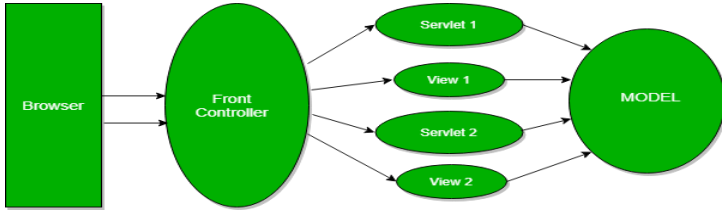
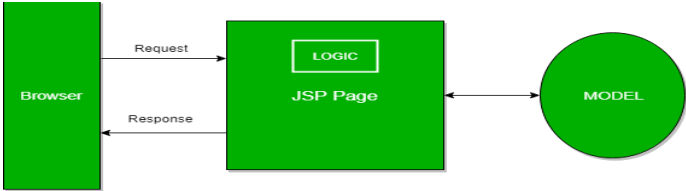
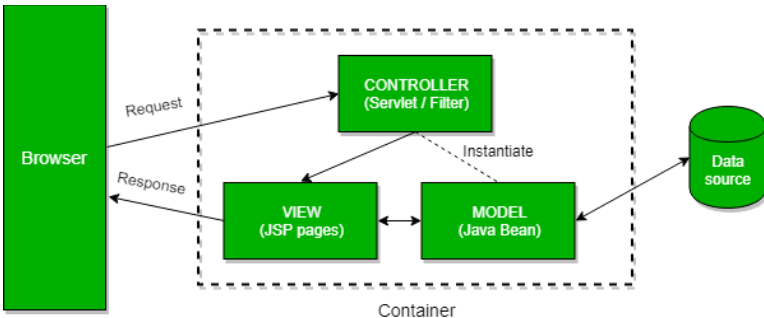
Presentation Layer: View is responsible for the visual presentation of the application. This behavior displays model information to the user.		Application Logic: This behavior holds the logic that implements the interaction between model and the view.
MVC: <ul style="list-style-type: none"> Consists of three layers Model, View and Controller View sits on top of the architecture Controller beside the view Model sits below the controller. So view knows about our controllers and controller know about the Models. Hence, our views have direct access to the Models. May have security and performance costs. Is a compound pattern (A design pattern that combines at least 2 patterns into a solution that solves a recurring or general problem) It uses a front controller pattern that processes web application request through a single controller. Provides better support for TDD. Client side library: backbone.js, angular.js Server-side library: spring MVC In MVC1. JSP page directly leads to JSP page. In MVC2, the links of the JSP pages for the next phase or view would also pass through the controller. 	MVP: <ul style="list-style-type: none"> The role of the controller is replaced by the presenter. Presenter sit at the same level as views, listening to events from both view and the model and mediating the action between them. Presenter and view should have a 1-1 relation, with each view having a reference to the presenter through the interface. Unit testing is easier, as view knows presenter through an interface which can be easily mocked. Client-side library: swing, javaFX Server-side: JSP servlets. 	MVVM: <ul style="list-style-type: none"> Consequently allows us to create view-specific subsets of a model, which can contain state and logic information. Avoiding the need to expose entire model to view. A View model is not required to reference a view. The view can bind to properties on the view model, which in turn expose data contained in the models to the View. E.g. Angular Drawbacks: <ul style="list-style-type: none"> Level of interpretation is needed between ViewModel and View and this can have performance costs.
MVT: Django’s interpretation of MVC, it considers Views to be the controllers and templates to be views.		
<div>SERVLETS</div>		
Difference between web server and application server: Web Server: <ul style="list-style-type: none"> Responsibility is to handle HTTP requests from client browsers and respond with HTML response. A web server understands HTTP language and runs on 	Non-Idempotent: <ul style="list-style-type: none"> A HTTP method is said to be idempotent if it returns the same result every time. HTTP methods GET, PUT, DELETE, HEAD, and OPTIONS are idempotent method and we should implement our application to make sure these methods always return same result. HTTP method 	

<p>HTTP protocol</p> <ul style="list-style-type: none"> Apache Web Server is kind of a web server and then we have specific containers that can execute servlets and JSPs known as servlet container, for example Tomcat. <p>Application Servers:</p> <ul style="list-style-type: none"> Application Servers provide additional features such as Enterprise JavaBeans support, JMS Messaging support, Transaction Management etc. So we can say that Application server is a web server with additional functionalities to help developers with enterprise applications. 	<p>POST is non-idempotent method and we should use post method when implementing something that changes with every request.</p> <p>MIME Type:</p> <ul style="list-style-type: none"> The “Content-Type” response header is known as MIME Type. Server sends MIME type to client to let them know the kind of data it’s sending. It helps client in rendering the data for user. Some of the mostly used mime types are text/html, text/xml, application/xml etc. We can use ServletContext getMimeType() method to get the correct MIME type of the file and use it to set the response content type. It’s very useful in downloading file through servlet from server.
<p>Web Application:</p> <ul style="list-style-type: none"> Web Applications are modules that run on server to provide both static and dynamic content to the client browser. 	<p>Servlet:</p> <ul style="list-style-type: none"> Java Servlet is server side technologies to extend the capability of web servers by providing support for dynamic response and data persistence. The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing our own servlets. All servlets must implement the javax.servlet.Servlet interface, which defines servlet lifecycle methods. When implementing a generic service, we can extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet() and doPost(), for handling HTTP-specific services.
<p>Common tasks performed by Servlet Container:</p> <ul style="list-style-type: none"> Communication Support: Servlet Container provides easy way of communication between web client (Browsers) and the servlets and JSPs. Because of container, we don’t need to build a server socket to listen for any request from web client, parse the request and generate response. All these important and complex tasks are done by container and all we need to focus is on business logic for the applications. Lifecycle and Resource Management: Servlet Container takes care of managing the life cycle of servlet. From the loading of servlets into memory, initializing servlets, invoking servlet methods and to destroy them. Container also provides utility like JNDI for resource pooling and management. Multithreading support: Container creates new thread for every request to the servlet and provide them request and response objects to process. So servlets are not initialized for each request and saves time and memory. JSP support: JSPs doesn’t look like normal java classes but every JSP in the application is compiled by container and converted to Servlet and then container manages them like other servlets. Misc Task: Servlet container manages the resource pool, perform memory optimizations, execute garbage collector, provides security configurations, support for multiple applications, hot deployment and several other tasks behind the scene that makes a developer life easier. 	<p>ServletConfig:</p> <ul style="list-style-type: none"> <code>javax.servlet.ServletConfig</code> is used to pass configuration information to Servlet. Every servlet has its own ServletConfig object and servlet container is responsible for instantiating this object. We can provide servlet init parameters in web.xml file or through use of WebInitParam annotation. We can use getServletConfig() method to get the ServletConfig object of the servlet. <p>ServletContext:</p> <ul style="list-style-type: none"> <code>javax.servlet.ServletContext</code> interface provides access to web application parameters to the servlet. The ServletContext is unique object and available to all the servlets in the web application. When we want some init parameters to be available to multiple or all of the servlets in the web application, we can use ServletContext object and define parameters in web.xml using <context-param> element. We can get the ServletContext object via the <code>getServletContext()</code> method of ServletConfig. Servlet containers may also provide context objects that are unique to a group of servlets and which is tied to a specific portion of the URL path namespace of the host.
<p>Request Dispatcher:</p> <ul style="list-style-type: none"> RequestDispatcher interface is used to forward the request to another resource that can be HTML, JSP or 	

<p>another servlet in same application. We can also use this to include the content of another resource to the response. This interface is used for inter-servlet communication in the same context.</p> <ul style="list-style-type: none"> There are two methods defined in this interface: <ol style="list-style-type: none"> <code>void forward(ServletRequest request, ServletResponse response)</code> – forwards the request from a servlet to another resource (servlet, JSP file, or HTML file) on the server. <code>void include(ServletRequest request, ServletResponse response)</code> – includes the content of a resource (servlet, JSP page, HTML file) in the response. We can get <code>RequestDispatcher</code> in a servlet using <code>ServletContext.getRequestDispatcher(String path)</code> method. The path must begin with a / and is interpreted as relative to the current context root. 	<ul style="list-style-type: none"> <code>ServletContext</code> is enhanced in Servlet Specs 3 to introduce methods through which we can programmatically add Listeners and Filters and Servlet to the application. It also provides some utility methods such as <code>getMimeType()</code>, <code>getResourceAsStream()</code> etc. <p>Differences between <code>ServletConfig</code> and <code>ServletContext</code>:</p> <ul style="list-style-type: none"> <code>ServletConfig</code> is a unique object per servlet whereas <code>ServletContext</code> is a unique object for complete application. <code>ServletConfig</code> is used to provide init parameters to the servlet whereas <code>ServletContext</code> is used to provide application level init parameters that all other servlets can use. We can't set attributes in <code>ServletConfig</code> object whereas we can set attributes in <code>ServletContext</code> that other servlets can use in their implementation.
<p><code>SingleThreadModel</code> interface:</p> <ul style="list-style-type: none"> Provided for thread safety and it guarantees that no two threads will execute concurrently in the servlets service model. 	<p>Servlet attribute and their scope:</p> <ul style="list-style-type: none"> Servlet attributes are used for inter-servlet communication, we can set, get and remove attributes in web application. There are three scopes for servlet attributes – request scope, session scope and application scope. <code>ServletRequest</code>, <code>HttpSession</code> and <code>ServletContext</code> interfaces provide methods to get/set/remove attributes from request, session and application scope respectively. Servlet attributes are different from init parameters defined in <code>web.xml</code> for <code>ServletConfig</code> or <code>ServletContext</code>.
<p>Invoke another servlet in different application:</p> <ul style="list-style-type: none"> Use <code>ServletResponse.sendRedirect()</code> method and provide complete URL of another servlet. That sends a response to client with response code as 301 to forward request to another URL. 	
<p>Servlet lifecycle:</p> <ul style="list-style-type: none"> Servlet class loading Servlet class initialization: Initializes the <code>ServletContext</code> object for the servlet and then invoke its init method by passing servlet config object. Request handling Removal from service 	<p>Servlet life cycle methods:</p> <ul style="list-style-type: none"> <code>Void init(ServletConfig)</code> <code>Void service(ServletRequest, ServletResponse)</code> <code>Void destroy()</code>
	<p>URL Rewriting:</p> <ul style="list-style-type: none"> We can use <code>HttpSession</code> for session management in servlets but it works with cookies and we can disable the cookie in client browser. Servlet API provides support for URL rewriting that we can use to manage session in this case.
<p>Different methods of session management in Servlets?</p> <p>Session is a conversational state between client and server and it consists of multiple request and response between client and server.</p> <p>Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session is passed between server and client in every request and response.</p> <p>Common ways of session management in servlets are:</p>	<p>Cookies:</p> <ul style="list-style-type: none"> Cookies are used a lot in web client-server communication. Cookies are text data sent by server to the client and it gets saved at client local machine. <code>HttpServletRequest</code> <code>getCookies()</code> method is provided to get the array of cookies from request. S <code>HttpServletResponse</code> <code>addCookie(Cookie c)</code> method is provided to attach cookie in response header, there are

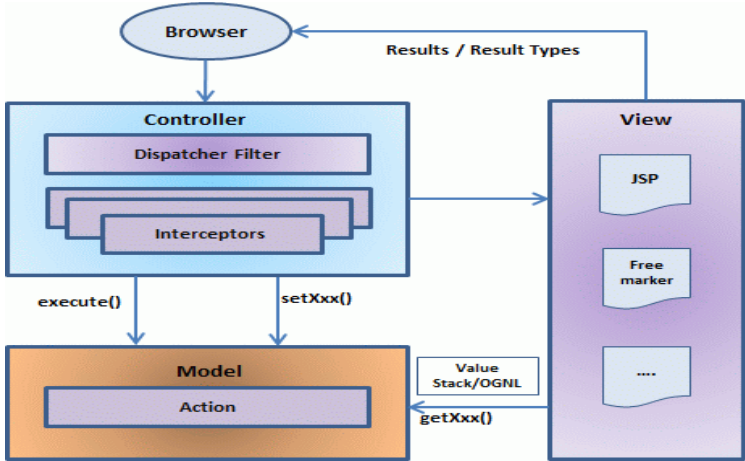
<ul style="list-style-type: none"> • User Authentication • HTML hidden field • Cookies • URL Rewriting • SessionManagementAPI 	<div>no getter methods for cookie.</div> <div>Notify an object in session when session is invalidated or timed-out? <ul style="list-style-type: none"> • Implement javax.servlet.http.HttpSessionBindingListener interface. • The interface defines 2 callback methods - valueBound() and valueUnbound(). These methods can define to implement processing logic when the object is added as attribute to the session and when session is destroyed. </div>
<div>Servlet filters: <ul style="list-style-type: none"> • These are pluggable java components that we can use to intercept and process requests before they are sent to servlets. </div> <div>Some common tasks that we can do with filters are: <ul style="list-style-type: none"> • Logging request parameters to log files. • Authentication and authorization of request for resources. • Formatting of request body or header before sending it to servlet. • Compressing the response data sent to the client. • Alter response by adding some cookies, header information etc </div>	<div>Servlet Listeners: <p>Listener interfaces can be implemented and configured to listen to an event and do certain operations.</p> </div> <div>Handle exceptions thrown by application with another servlet? <pre> <error-page> <error-code>404</error-code> <location>/AppExceptionHandler</location> </error-page> <error-page> <exception-type>javax.servlet.ServletException</exception-type> e> <location>/AppExceptionHandler</location> </error-page> </pre> </div>
<div>Deployment descriptor: <ul style="list-style-type: none"> • Configuration file for the web application and its name is web.xml and it resides in WEB-INF directory. • Servlet container use this file to configure web applications <ul style="list-style-type: none"> ○ Servlets ○ Servlet config params ○ Context init params ○ Filters ○ Listeners ○ Welcome pages ○ Error handlers </div>	<div>How to make sure a servlet is loaded in application startup? <ul style="list-style-type: none"> • Usually servlet container loads a servlet on the first client request but sometimes when the servlet is heavy and takes time to load, we might want to load it on application startup. • We can use load-in-startup element with servler configuration in web.xml file or use WebServlet annotation loadOnStartup variable to tell container to load servlet on system startup. </div>
<div>Different ways of servlet authentication: <ul style="list-style-type: none"> • HTTP Basic AuthN • HTTP Higest AuthN • HTTPS AuthN • Form Based Login </div>	<div>Transport layer security: <ul style="list-style-type: none"> • We can configure our servlet container to use SSL for message communication over the network. • To configure SSL on Tomcat, we need a digital certificate that can be created using Java keytool for development environment. </div>
<div>SPRING MVC</div>	
<div>Why Spring MVC? <ul style="list-style-type: none"> • Spring MVC implements a clear separation of concerns that allows us to develop and unit test our application easily. </div>	<div>@Autowired: <ul style="list-style-type: none"> • Can be used with fields or methods for injecting a bean by type. </div>

<ul style="list-style-type: none"> ● The concepts like: <ul style="list-style-type: none"> ○ Dispatcher servlet ○ Controllers ○ View resolver ○ Views, models ○ ModelAndView ○ Model and Session Attributes <p>Are completely independent of each other</p> <ul style="list-style-type: none"> ● It's based on interfaces (with provided implementation classes), and we can configure every part of the framework by using custom interfaces. ● Another important thing is that we aren't tied to a specific view technology (for example, JSP), but we have the option to choose from the ones we like the most. ● Also, we don't use Spring MVC only in web applications development but in the creation of RESTful web services as well. 	<p>@ModelAttribute:</p> <ul style="list-style-type: none"> ● binds a method parameter or a method return value to a named model attribute and then exposes it to a web view. <p>Difference Between @Controller and @RestController:</p> <ul style="list-style-type: none"> ● The main difference between the <i>@Controller</i> and <i>@RestController</i> annotations is that the <i>@ResponseBody</i> annotation is automatically included in the <i>@RestController</i>. ● This means that we don't need to annotate our handler methods with the <i>@ResponseBody</i>. ● We need to do this in a <i>@Controller</i> class if we want to write response type directly to the HTTP response body.
<p>@RequestBody and @ResponseBody:</p> <ul style="list-style-type: none"> ● The <i>@RequestBody</i> annotation, used as a handler method parameter, binds the HTTP Request body to a transfer or a domain object. Spring automatically deserializes incoming HTTP Request to the Java object using Http Message Converters. ● When we use the <i>@ResponseBody</i> annotation on a handler method in the Spring MVC controller, it indicates that we'll write the return type of the method directly to the HTTP response body. We'll not put it in a <i>Model</i>, and Spring won't interpret as a view name. 	<p>Model, ModelMap and ModelAndView:</p> <ul style="list-style-type: none"> ● The <i>Model</i> interface defines a holder for model attributes. ● The <i>ModelMap</i> has a similar purpose, with the ability to pass a collection of values. It then treats those values as if they were within a <i>Map</i>. ● We should note that in <i>Model (ModelMap)</i> we can only store data. We put data in and return a view name. ● On the other hand, with the <i>ModelAndView</i>, we return the object itself. We set all the required information, like the data and the view name, in the object we're returning.
<p>SessionAttributes and SessionAttribute:</p> <ul style="list-style-type: none"> ● <i>@SessionAttributes</i> annotation is used for storing the model attribute in the user's session. ● If we want to retrieve the existing attribute from a session that is managed globally, we'll use <i>@SessionAttribute</i> annotation as a method parameter 	<p>@EnableWebMVC:</p> <ul style="list-style-type: none"> ● Purpose is to enable Spring MVC via Java configuration. ● This annotation imports Spring MVC Configuration from <i>WebMvcConfigurationSupport</i>. ● It enables support for <i>@Controller</i>-annotated classes that use <i>@RequestMapping</i> to map incoming requests to a handler method.
<p>ViewResolver:</p> <ul style="list-style-type: none"> ● enables an application to render models in the browser – without tying the implementation to a specific view technology – by mapping view names to actual views. 	<p>BindingResult:</p> <ul style="list-style-type: none"> ● <i>BindingResult</i> is an interface from <i>org.springframework.validation</i> package that represents binding results. ● We can use it to detect and report errors in the submitted form. ● It's easy to invoke — we just need to ensure that we put it as a parameter right after the form object we're validating. The optional <i>Model</i> parameter should come after the <i>BindingResult</i> <p>@PostMapping("/user") Public String submitForm(@Valid Form form, BindingResult result, Model model) {...}</p>
<p>Form Backing Object:</p> <ul style="list-style-type: none"> ● The form backing object or a Command Object is just a POJO that collects data from the form we're submitting. 	
<p>@Qualifier:</p> <ul style="list-style-type: none"> ● It is used simultaneously with the <i>@Autowired</i> annotation to avoid confusion when multiple instances of a bean type are present. 	

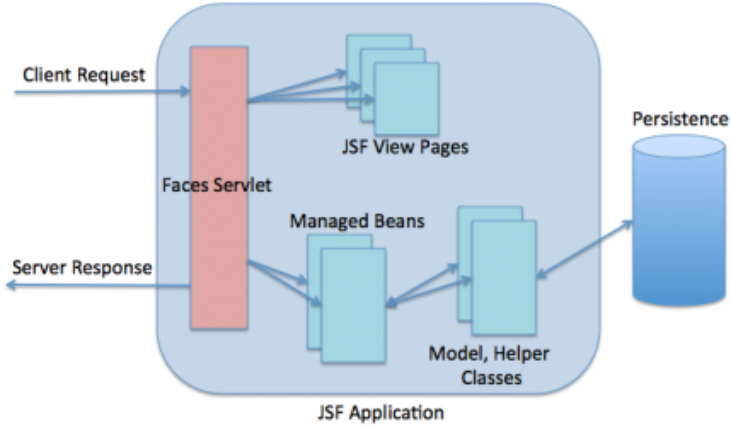
<p>@Required:</p> <p>The @Required annotation is used on setter methods, and it indicates that the bean property that has this annotation must be populated at configuration time. Otherwise, the Spring container will throw a BeanInitializationException exception.</p>	<p>Describe Front controller pattern:</p> <ul style="list-style-type: none"> • In the Front controller pattern, all requests will first go to the front controller instead of the servlet. • It will make sure responses are ready and will send them back to the browser. • The front controller will identify the servlet that should handle the request first. • Then it will get data back from the servlet, it will decide which view to render and finally send the rendered view back as response. 
<p>Model1 and Model2 architectures:</p> <ul style="list-style-type: none"> • In model 1, a request comes to a servlet or jsp where it gets handled. The servlet or the JSP processes the request, handles business logic, retrieves and validates data and generates the response. It isn't convenient for large scale web applications.  <ul style="list-style-type: none"> • Model 2 is based on MVC design pattern. • It separates view from the logic that manipulates the content. • The model represents the dynamic data structure of an application. It is responsible for data and business logic manipulation. • The view is in charge of displaying data. • While the controller serves as an interface b/w the 2. • In Model 2, a request is passed to the controller, which handles the required logic in order to get the right content that should be displayed. • The controller then puts the content back into the request, typically as a java bean or a POJO. • It also decides which view should render the content and finally passes the request to it. 	<p>Difference between @Controller, @Component, @Repository, and @Service:</p> <p>@Component is a generic stereotype for any Spring-managed component. @Repository, @Service, and @Controller are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.</p> <ul style="list-style-type: none"> • @Controller – indicates that the class serves the role of a controller, and detects @RequestMapping annotations within the class • @Service – indicates that the class holds business logic and calls methods in the repository layer • @Repository – indicates that the class defines a data repository; its job is to catch platform-specific exceptions and re-throw them as one of Spring's unified unchecked exceptions
<p>Multipart Resolver:</p> <p>This interface is used for uploading files.</p>	<p>Dispatcher Servlet:</p> <ul style="list-style-type: none"> • in the Front Controller design pattern, a single controller is responsible for directing incoming HttpRequests to all of an application's other controllers and handlers. • Spring's DispatcherServlet implements this pattern and is, therefore, responsible for correctly coordinating the HttpRequests to the right handlers.
<p>Spring MVC Interceptors:</p> <ul style="list-style-type: none"> • allow us to intercept a client request and process it at three places – before handling, after handling, or after completion (when the view is rendered) of a request. • The interceptor can be used for cross-cutting concerns and to avoid repetitive handler code like logging, changing globally used parameters in Spring model, etc. 	<p>ContextLoaderListener:</p> <ul style="list-style-type: none"> • ContextLoaderListener starts up and shuts down Spring's root WebApplicationContext. It ties the lifecycle of ApplicationContext to the lifecycle of the ServletContext. We can use it to define shared beans working across different Spring contexts.
<p>@InitBinder</p>	<p>Controller Advice:</p>

<p>A method annotated with <code>@InitBinder</code> is used to customize a request parameter, URI template, and backing/command objects. We define it in a controller and it helps in controlling the request. In this method, we register and configure our custom <code>PropertyEditors</code>, a formatter, and validators.</p>	<ul style="list-style-type: none"> ● The <code>@ControllerAdvice</code> annotation allows us to write global code applicable to a wide range of controllers. We can tie the range of controllers to a chosen package or a specific annotation. ● By default, <code>@ControllerAdvice</code> applies to the classes annotated with <code>@Controller</code> (or <code>@RestController</code>). ● If we want to restrict applicable classes to a package, we should add the name of the package to the annotation. ● It's also possible to use multiple packages, but this time we need to use an array instead of the <code>String</code>. ● It's noteworthy to remember that we should use it along with <code>@ExceptionHandler</code>. This combination will enable us to configure a global and more specific error handling mechanism without the need to implement it every time for every controller class.
<p><code>@ExceptionHandler</code>:</p> <ul style="list-style-type: none"> ● The <code>@ExceptionHandler</code> annotation allows us to define a method that will handle the exceptions. We may use the annotation independently, but it's a far better option to use it together with the <code>@ControllerAdvice</code>. Thus, we can set up a global error handling mechanism. In this way, we don't need to write the code for the exception handling within every controller. <p>Exception handling in web applications:</p> <ul style="list-style-type: none"> ● There are 3 options for exceptions handling in Spring MVC: <ul style="list-style-type: none"> ○ Per exception ○ Per controller ○ Globally 	
<p style="text-align: center;">STRUTS</p>	
<p>Struts framework comprised of the following components:</p> <ul style="list-style-type: none"> ● Action classes ● Interceptors ● Result pages, JSP or Freemarker templates. ● Valuestack, OGNL and Tag libraries. 	<p>Role of handlers in MVC:</p> <ul style="list-style-type: none"> ● Transfer the requests to appropriate models. ● Handlers use mapping information from configuration files for request transfer.
<p>Flow of requests in struts based application:</p> <ul style="list-style-type: none"> ● User interacts with view by clicking any link or by submitting form. ● Upon user's interaction, the request is passed towards controller. ● Controller is responsible for parsing the request to appropriate action. ● Action is responsible for calling a function in Model which has all business logic implemented. ● Response from the model layer is received back by the action which then passes it towards the view where user is able to see the response. 	<p>Role of action class in struts?</p> <p>In struts, action class acts as controller and performs the following key tasks:</p> <ul style="list-style-type: none"> ● After receiving user request, it processes the users request. ● Uses appropriate model and pulls data from the model. ● Selects proper view to show the response to the user. <p>“Struts-config.xml” file is the configuration file used by controller to get mapping info to decide which action to use for routing of users action.</p>
<p>Interceptors in struts2:</p> <ul style="list-style-type: none"> ● Interceptors are backbone of struts2 framework. ● Responsible for most of the processing done by framework <ul style="list-style-type: none"> ○ Passing request params to action classes ○ Validation ○ I18n support,etc ● One of the best example for Chain of responsibility pattern <p>Struts2 interceptors are singleton classes and a new thread is</p>	<p>Different ways to create action classes in struts2:</p> <ul style="list-style-type: none"> ● By implementing action interface ● Using struts2 <code>@Action</code> annotation ● By extending <code>ActionSupport</code> class ● Any normal class with <code>execute()</code> method returning string can be configured as Action class <p>Struts2 action classes are thread safe because an object is instantiated for every request to handle it.</p> <p>Default suffix for struts action URI was “.do”. We can change the</p>

<p>created to handle the request, so its not thread safe.</p> <p>Lifecycle: init(), destroy() and intercept()</p>	<p>suffix by defining struts.action.extension constant value in our configuration file.</p>
<p>ValueStack:</p> <p>ValueStack is the storage area where the application data is stored by Struts2 for processing the client requests. The data is stored in ActionContext objects that use ThreadLocal to have values specific to the particular request thread.</p>	<p>Object-Graph Navigation Language is a powerful expression language that is used to manipulate data stored on valuestack. Both interceptors and result pages can access data stored on valuestack using OGNL</p>
<p>Useful annotations introduced in Struts2:</p> <ul style="list-style-type: none"> • @Action • @Actions • @Namespace and @Namespaces • @Result • @ResultPath 	<p>Namespace:</p> <p>Struts2 namespace configuration allows us to create modules easily. We can use namespace to separate our action classes based on their functionality.</p>
<p>execAndWait interceptor:</p> <p>Used for long running action classes. We can use this interceptor to return an intermediate response page to the client and once the processing is finished, final response is returned to the client. This interceptor is defined in struts-default package.</p>	<p>Token interceptor:</p> <p>One of the major problem with web application is the double form submission. If not taken care, double form submission could result in charging double amount to customer or updating database values twice. We can use token interceptor to solve the double form submission problem. This interceptor is defined in struts-default package but it's not part of any interceptor stack, so we need to include it manually in our action classes.</p>
<p>Struts2 tags:</p> <p>Struts2 provides a lot of custom tags that we can use in result pages to create views for client request. These tags are broadly divided into three categories- Data tags, Control tags and UI tags.</p> <p><%@ taglib uri="/struts-tags" prefix="s"/></p> <p>Data tags: property, set, push, bean, action, include, i18n and text tag</p> <p>Control tags: if-elseif-else, iterator, append, merge, sort, subset and generator tag</p> <p>UI tags: o generate HTML markup language, binding HTML form data to action classes properties, type conversion, validation and i18n support. Some of the important UI tags are form, textfield, password, textarea, checkbox, select, radio and submit tag.</p>	<p>Best practices:</p> <ul style="list-style-type: none"> • Always try to extend struts-default package while creating your package to avoid code redundancy in configuring interceptors. • For common tasks across the application, such as logging request params, try to use interceptors. • Always keep action classes java bean properties in a separate bean for code reuse and implement ModelDriven interface. • If you have custom interceptor that you will use in multiple actions, create interceptor stack for that and then use it. • Try to divide your application in different modules with namespace configuration based on functional areas. • Try to use Struts2 tags in result pages for code clarify, if needed create your own type converters. • Use development mode for faster development, however make sure production code doesn't run in dev mode. • Use Struts2 i18n support for resource bundles and to support localization. • Struts2 provides a lot of places where you can have resource bundles but try to keep one global resource bundle and one for action class to avoid confusion. • struts-default package configures all the interceptors and creates different interceptor stacks. Try to use only what is needed, for example if you don't have localization requirement, you can avoid i18n interceptor.
<p>Exception handling:</p> <p>Struts2 exception handling can be done at global package level as well action level.</p>	

<div>struts.xml:</div> <div><package name="user" namespace="/" extends="struts-default"></div> <div><global-results></div> <div><result name="exception">/exception.jsp</result></div> <div><result name="r_exception">/r_exception.jsp</result></div> <div><result name="error">/error.jsp</result></div> <div></global-results></div> <div><global-exception-mappings></div> <div><exception-mapping exception="java.lang.Exception"</div> <div>result="exception"></exception-mapping></div> <div><exception-mapping exception="java.lang.Error"</div> <div>result="error"></exception-mapping></div> <div><exception-mapping exception="java.lang.RuntimeException"</div> <div>result="r_exception"></exception-mapping></div> <div></global-exception-mappings></div> <div><action name="myaction" class="com.journaldev.struts2.exception.MyAction"></div> <div></action></div> <div><action name="myspecialaction"</div> <div>class="com.journaldev.struts2.exception.MySpecialAction"></div> <div><exception-mapping exception="java.io.IOException" result="login"></div> <div></exception-mapping></div> <div><result name="login">/error.jsp</result></div> <div></action></div> <div></package></div>	<div><p>The diagram illustrates the Struts2 architecture. At the top, a 'Browser' oval sends requests to a 'Controller' box. The 'Controller' box contains a 'Dispatcher Filter' box, which in turn contains three stacked 'Interceptors' boxes. An arrow labeled 'execute()' points from the 'Controller' to a 'Model' box. The 'Model' box contains an 'Action' box. An arrow labeled 'setXxx()' points from the 'Controller' to the 'Action' box. An arrow labeled 'getXxx()' points from the 'Action' to a 'Value Stack/OGNL' box. The 'Value Stack/OGNL' box is connected to a 'View' box. The 'View' box contains 'JSP', 'Free marker', and '....' components. An arrow labeled 'Results / Result Types' points from the 'View' back to the 'Browser'.</p></div>
<div>Struts-config.xml:</div> <div><struts-config></div> <div><!-- ===== Form Bean Definitions ===== --></div> <div><form-beans></div> <div><form-bean name = "login" type = "test.struts.LoginForm" /></div> <div></form-beans></div> <div><!-- ===== Global Forward Definitions ===== --></div> <div><global-forwards></div> <div></global-forwards></div> <div><!-- ===== Action Mapping Definitions ===== --></div> <div><action-mappings></div> <div><action</div> <div>path = "/login"</div> <div>type = "test.struts.LoginAction" ></div> <div><forward name = "valid" path = "/jsp/MainMenu.jsp" /></div> <div><forward name = "invalid" path = "/jsp/LoginView.jsp" /></div> <div></action></div> <div></action-mappings></div> <div><!-- ===== Controller Definitions ===== --></div> <div><controller contentType = "text/html;charset = UTF-8"</div> <div>debug = "3" maxFileSize = "1.618M" locale = "true" nocache = "true"/></div> <div></struts-config></div>	<div>Struts.properties file:</div> <div>### When set to true, Struts will act much more friendly for developers</div> <div>struts.devMode = true</div> <div>### Enables reloading of internationalization files</div> <div>struts.i18n.reload = true</div> <div>### Enables reloading of XML configuration files</div> <div>struts.configuration.xml.reload = true</div> <div>### Sets the port that the server is run on</div> <div>struts.url.http.port = 8080</div>
JSF (JavaServer Pages)	
<div><ul style="list-style-type: none">Is a front end framework which makes the creation of user interface components easier by reusing the UI components.</div>	<div>Managed bean:</div> <div>A managed bean is a java class registered to JSF which makes interaction between the UI and the business class possible. Can be created by @MangedBean annotation</div>
<div>Managed property:</div> <div>@ManagedProeprty annotation enables us to inject a managed bean into another managed bean.</div>	<div>@ApplicationScoped indicates that the bean is valid as long as the web application is valid.</div> <div>@ViewScoped indicates that the bean is alive as long as the user</div>

<p>Advantages of JSF:</p> <ul style="list-style-type: none"> ● Clean separation between presentation and business logic. ● Manages UI state across multiple server requests. ● Implementation of custom components. ● Easier flow of data between the components. ● JSF specs that helps custom implementations such as PrimeFaces 	<p>interacts with the same JSF view page</p> <p>@SessionScoped: indicates that the bean is valid as long as HTTP session is alive</p> <p>@CustomScoped: indicates that the bean lives as long as the beans entry in the custom map which is created for this scope lives.</p> <p>@RequestScoped: indicate that the bean lives as long as the HTTP request-response lives.</p>
<p>Backing Bean:</p> <p>A JSF application includes one or more backing beans, each of which is a type of managed bean that can be associated with the components used in a particular page.</p> <p>Functions performed:</p> <ul style="list-style-type: none"> ● Validating a component's data ● Handling an event fired by a component ● Performs processing to determine the next page to which the application must navigate 	<p>Standard JSF tag libraries:</p> <ul style="list-style-type: none"> ● JSF core tag library ● JSF html library <pre><html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html" xmlns:ui="http://java.sun.com/jsf/facelets" xmlns:c="http://java.sun.com/jsf/core"></pre>
<p>Differences between backing bean and managed bean:</p> <p>Backing Beans should be defined in request scope whereas managed bean can be defined in request,session or application scopes.Backing bean is referenced by a form whereas a managed bean is registered with JSF and created automatically when needed.</p>	
<p>Resource bundling:</p> <p>The phenomenon of storing the UI label, date, status messages and other UI textual elements in a separate file instead of hardcoding these in a page is called resource bundling.</p>	<p>Different types of page navigation supported by JSF:</p> <ul style="list-style-type: none"> ● Implicit navigation ● Navigation through managed bean ● Navigation through faces-config.xml ● Forward versus redirect navigation ● Conditional navigation
<p>JSF lifecycle:</p> <ul style="list-style-type: none"> ● Restore view phase ● Apply request values phase ● Process validations phase ● Update model values phase ● Invoke application phase ● Render response phase 	<p>Event:</p> <p>An event is defined as a signal triggered base upon the actions such as click of button, hyperlink, changing the input value, etc.</p> <p>Different types of JSF events:</p> <ul style="list-style-type: none"> ● Action Events: ● Value change events: ● Phase events
<p>Listener class:</p> <p>A class which is associated with an event is called a listener class.</p>	<p>Facelets tag:</p> <p>JSF provides a special set of tags that gives the flexibility to manage common tags/parts in one place for more than one application. These tags allow us to create a common layout that can be used across applications.</p> <p>Important facelets tags are:</p> <pre><ui:component> tag <ui:composition> tag <ui:decorate> tag <ui:define> tag <ui:fragment> tag <ui:include> tag</pre>
<p>Type of validations in JSF:</p> <ul style="list-style-type: none"> ● Declarative validations: the validations that are fired using JSF standard validators or Bean validators fall under declarative type. ● Imperative validation: The standard validation messages would not be sufficient in all cases and sometimes require complex validations that are declared by the user overriding the standard validations. 	
<p>JSF Expression language supports:</p> <ul style="list-style-type: none"> ● Immediate value expressions: rendered as soon as page 	<p>Different implementations of JSF API:</p> <ul style="list-style-type: none"> ● ADF Faces: Oracle's implementation for the JSF

<p>is displayed initially \${}</p> <ul style="list-style-type: none"> Deferred value expressions: evaluated during the lifecycle phase whenever it is request by the user #{} Value expression and method expression 	<p>standard.</p> <ul style="list-style-type: none"> Reference Implementation (RI): by Sun Microsystems. Apache MyFaces: open source JavaServer Faces (JSF) implementation. Primefaces: JSF components with Ajax framework.
<p>JSF is developed based on the Model View Controller(MVC) pattern. The Model View Controller separates the business logic from presentation.</p>	<p>The JSF application contains</p> <ul style="list-style-type: none"> UI components represented as stateful objects on the server Server-side helper classes Validators, event handlers, and navigation handlers Application configuration resource file for configuring application resources JavaBeans components as models containing application-specific functionality and data A custom tag library for representing event handlers and validators A custom tag library for rendering UI components
 <p>The diagram illustrates the JSF Application architecture. A 'Client Request' enters from the left, passing through a 'Faces Servlet' (represented by a red vertical bar). The request then goes to 'JSF View Pages' (represented by a stack of blue squares). These pages interact with 'Managed Beans' (represented by a stack of blue squares). The Managed Beans interact with 'Model, Helper Classes' (represented by a stack of blue squares). These classes interact with a 'Persistence' layer (represented by a blue cylinder). Finally, a 'Server Response' is sent back from the Faces Servlet to the client. The entire process is labeled 'JSF Application' at the bottom.</p>	<p>Required configuration for JSF framework:</p> <p>Web.xml: details of deployment.</p> <p>Faces-config.xml: allows to configure the application, managed beans, converters, validators and navigation.</p>
<p>Component rendering model:</p> <p>JSF component architecture is designed such that the functionality of components is defined by component classes. Whereas the actualities of rendering itself can be defined by a separate renderer called as component rendering model.</p>	<p>Render Kit:</p> <p>How component classes map to component tags that are appropriate for a client.</p>
<p>View Object:</p> <p>A view object is a model object used specifically in the presentation tier but defined outside it. It contains the data that must be displayed in the view layer and the logic to validate user input, handle events, and interact with the business-logic tier.</p>	<p>Bean scope:</p> <p>Bean scope is the mechanism of binding the beans and other objects to be available in the different components of a web application.</p>
<p>Best practices:</p> <ul style="list-style-type: none"> Avoid using JSF components for static value. Short component Id Avoid component bindings Facelets for dynamic includes 	<p>Difference between Spring MVC and JSF:</p> <p>Spring MVC is request-response based framework whereas JSF is component based framework.</p>

Note: Information gathered in this document has been collected from various sources on Internet.