**← HTML QUESTIONS →**

**( 30 Marks )**

Q.1 <!DOCTYPE html> is it a tag of html? If not, what is it and why do we use it? (1 marks)

Answer: Doctype html is a document type declaration which gives the information to the browser which html version is coming which ensures that the web page is parsed in the same way by different browsers.

The doctype is not a html tag , it's used to tell the browser what html version to expect

! DOCTYPE html>` is required in the first line of your HTML document. Doctype declaration for HTML5 is not case sensitive and does not require a closing tag.

Q.2 Explain Semantic tags in html? And why do we need it? ( 2 Marks)

Answer:Semantic tags are the tags which are easiest to understand the meaning of a document. Div and span are non-semantic tags that don't tell the meaning of the document . Forms, table elements are the semantic tags which tell the browser as well as the developer about it's meaning. Which means from the form tag we can easily understand that it is the tag for making any type of form.

Semantics tags are needed for ease of development. We don't need to find the meaning of code. It gives the structural information of web page.

Header, section, main , navbar, and aside footer are examples of semantic tags.

Q.3 Differentiate between HTML Tags and Elements? (2 Marks)

Answer: HTML tags are the building block of the html page while the html elements are the whole components which gives the meaning of html tag.

HTML tags are <p> </p> ,<h1></h1>-<h3></h3> etc.

Html elements contain the starting tag,closing tag and the information inside it.

an element refers to a unit of content that has a start tag, content, and an end tag. A tag, on the other hand, is a keyword enclosed in angle brackets that defines an element in HTML.

Q.4 Build Your Resume using HTML only (5 Marks)

Answer 4: **Github link:** https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/tree/main/html%20resume

Q.5 Write Html code so that it looks like the given image Link (5 Marks)

Answer5: **Github link:** https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/tree/main/HTML%202%2C%20image%20que

Q.6 What are some of the advantages of HTML5 over its previous versions? (2 Mark)

Answer: Some of the advantages of HTML5 over its previous versions are:

* HTML5 is easy to use as compared to its previous versions, HTML5 will enable web designers to use cleaner, neater code. We can remove div tags and replace them with semantic HTML5 elements.
* HTML5 is compatible with all web browsers
* It supports client-side databases
* HTML5 has improved performance and security, as it enables storage across numerous browsers.

Q.7 Create a simple Music player using html only (5 Marks)

Answer 7: **Github link: https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/tree/main/Music\_Player**

Q.8 What is the difference between <figure> tag and <img> tag? (1 Marks)

Answer: Image tag (<img>) is used to embed the image in an html document. While figure tag is used to semantically organize the content of an image in the HTML document.

These two tags are not interchangeable . both are different tags which are used according to the need of it.

If we want to give a caption to the image then in <img> tag we have to give a caption in a different tag like p.

While when we want to give caption to the image in the figure tag we use <figcaption>tag in it.

Q.9 What’s the difference between html tag and attribute and give examples of some global attributes? (2 Marks)

Answer: HTML tag is a structural unit of HTML element which makes the content of a web-page. Most of the elements have two tags: start tag and end tag . There are only a few tags like <hr> and <br> which have no closing tag.

While the attribute of the elements give the extra properties of html tag like class , id , style are the example of it.

Global attributes are attributes common to all HTML elements; they can be used on all elements, though they may have no effect on some elements.

Class , id , title, accesskey , style , translate are the examples of global attributes:

Q.10 build Table which looks like the given image Link (5 Marks)

Answer 10: **Github link:**

**← CSS QUESTIONS →**

**( 60 Marks)**

Q.1 Whats Box Model in CSS & Which CSS Properties are part of it ? (2 Marks)

Answer:When laying out a document, the browser's rendering engine represents each element as a rectangular box according to the standard CSS basic box model. CSS determines the size, position, and properties (color, background, border size, etc.) of these boxes.

Every box is composed of four parts (or *areas*), defined by their respective edges: the *content edge*, *padding edge*, *border edge*, and *margin edge*.

Q.2 What are the Different Types of Selectors in CSS & what are the advantages of them? (2 Marks)

Answer: CSS Selector are used to select the HTML tag that you want to style. CSS selectors select HTML elements according to its id, class, type, attribute . There are different types of CSS Selectors which give advantage of selecting that tag which we want to focus.

* Individual Selector
* Id Selector
* Class Selector
* Universal Selector
* Group Selector
* Attribute Selector
* Combinator Selector
* Pseudo-Class Selector
* Pseudo-Element Selector

Q.3 What is VW/VH & How its different from PX? (2 Marks)

Answer: PX: Pixels (px) are considered absolute units, although they are relative to the DPI and resolution of the viewing device. But on the device itself, the PX unit is fixed and does not change based on any other element.

VW is Relative to the viewport’s width

VH is Relative to the viewport’s height

While PX is primarily used for font sizing, VW, and VH are mostly used for margins, padding, spacing, and widths/heights.

Q.4 Whats difference between Inline, Inline Block and block ? (3 Marks)

**Answer:** **Block:**The [elements](https://www.javatpoint.com/html-elements) that begin on a new line are known as block elements. A block element takes up the full width of the content.

CSS property: display:block

**Inline:**Inline elements exists a top and bottom margin for these elements

display: inline; /\* the default for span \*/

**Inline Block:**The display value of inline-block works similarly to inline with one exception: the height and width of that element become modifiable.

CSS property: display:inline-block

Block level elements cover up the larger space as compared to the inline elements.

Q.5 How is Border-box different from Content Box? (2 Marks)

**Answer:** Border-box and Content Box are two different values of box-sizing property , both work differently.

Border-Box includes the padding and margin spaces in the given value.

Syntax: box-sizing: border-box;

While the Content box doesn’t include it. It gives the padding and margin outside of the content box.This is the default value of box-sizing

Syntax: box-sizing: content-box;

Q.6 What’s z-index and How does it Function ? (2 Marks)

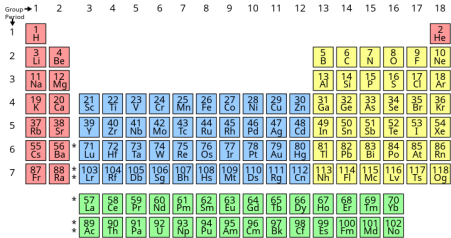
Answer: z-index property **specifies the stack order of an element**. An element with greater stack order is always in front of an element with a lower stack order. In easy words,Z Index ( z-index ) is a CSS property that defines the order of overlapping HTML elements. Elements with a higher index will be placed on top of elements with a lower index. Note: Z index only works on positioned elements ( position:absolute , position:relative , or position:fixed ).

Q.7 What’s Grid & Flex and difference between them? (5 Marks)

Answer: **Flex:** The Flexbox is a flexible box model in CSS which give us flexible layouts, Generally Flexbox is a one dimensional layout model which has flexible layouts with distributes space to control it's alignment and use it efficiently. it is a layout model that provides an easy and clean way to arrange items within a container.

**Grid**: CSS Grid Layout (aka “Grid” or “CSS Grid”), is a two-dimensional grid-based layout system that, compared to any web layout system of the past, completely changes the way we design user interfaces.

A grid layout consists of a parent element, with one or more child elements. grid layout enables an author to align elements into columns and rows. However, many more layouts are either possible or easier with CSS grid than they were with tables. For example, a grid container's child elements could position themselves so they actually overlap and layer, similar to CSS positioned elements.

* The basic difference between CSS Grid Layout and CSS Flexbox Layout is that flexbox was designed for layout in one dimension - either a row or a column. Grid was designed for two-dimensional layout - rows, and columns at the same time
* Flexbox, gives you more flexibility while working on either element (row or column). HTML markup and CSS will be easy to manage in this type of scenario.****
* GRID gives you more flexibility to move around the blocks irrespective of your HTML markup.
* The Flexbox layout is best suited to application components and small-scale layouts, while the Grid layout is designed for larger-scale layouts that are not linear in design.

Q.8 Difference between absolute and relative and sticky and fixed position explain with example. (5 Marks)

**Answer:**The position CSS property sets how an element is positioned in a document. The top, right, bottom, and left properties determine the final location of positioned elements. The position property specifies the type of positioning method used for an element.

There are five different position values:

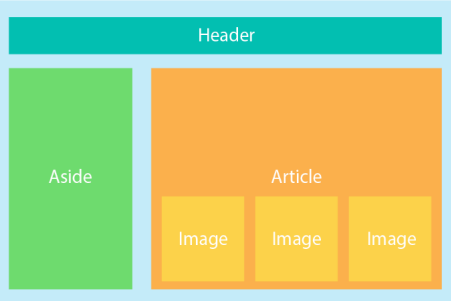
* Static: The element is positioned according to the normal flow of the document. The top, right, bottom, left, and z-index properties have no effect. This is the default value
* Relative: An element with position: relative; is positioned relative to its normal position. Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position.
* Fixed: An element with position: relative; is positioned relative to its normal position. Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position.
* Absolute: An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.
* Sticky: An element with position: sticky; is positioned based on the user's scroll position. A sticky element toggles between relative and fixed, depending on the scroll position.

Q.9 Build Periodic Table as shown in the below image (10 Marks)

**Github link: https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/tree/main/Periodic%20table**

Q. 9 Build given layout using grid or flex see below image for reference . **(5 Marks)**

**Answer: https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/tree/main/flex%20layout**

****

Q.10 Build Responsive Layout both desktop and mobile and Tablet, see below image for reference ? **(10 Marks)**

**Answer: Github link: https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/tree/main/responsive%20layout**

****

Q.11 Build Complete Homepage of Ineuron ( Link ) with responsiveness. **(10 Marks)**

**Answer: Github link: https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/tree/main/Ineuron%20webpage**

Q.12 What are Pseudo class in CSS & How its different From Pseudo Elements? **(2 Marks)**

**Answer: Pseudo-class**: A CSS pseudo-class is a keyword added to a selector that specifies a special state of the selected element(s). For example, the pseudo-class :hover can be used to select a button when a user's pointer hovers over the button and this selected button can then be styled.

**Pseudo Elements:**A CSS pseudo-element is a keyword added to a selector that lets you style a specific part of the selected element(s).

Here we will discuss two elements i.e.

* ::after:
* ::before

::after

The ::after pseudo-element can be used to insert some content after the content of an element.

::before

The ::before pseudo-element can be used to insert some content before the content of an element.

← JavaScript Questions →

**(40 Marks)**

**Q.1** What is Hoisting in Javascript ? **(3 Marks)**

**Answer:** In JavaScript, a variable can be declared after it has been used.

In other words; a variable can be used before it has been declared. Hoisting in JavaScript is a behavior in which a function or a variable can be used before declaration.

For example

**console.log(foo);**

**var foo = 'foo';**

It might surprise you that this code outputs undefined and doesn't fail or throw an error – even though foo gets assigned after we console.log it!

This is because the JavaScript interpreter splits the declaration and assignment of functions and variables: it "hoists" your declarations to the top of their containing scope before execution.

This process is called hoisting, and it allows us to use foo before its declaration in our example above.

**Q.2** What are different higher order functions in JS? What is the difference between .map() and .forEach() ? **(2 Marks)**

**Answer:** In JavaScript, functions are treated as first-class citizens. We can treat functions as values and assign them to another variable, pass them as arguments to another function, or even return them from another function.

This ability of functions to act as first-class functions is what powers higher order functions in JavaScript.

A function which takes another function as an argument or returns a function is known as a higher order function.

JavaScript provides us with some inbuilt higher order functions like **map() , filter() , reduce()** and so on.

The difference between .map() and .forEach() is:

* The map() method returns a new array, whereas the forEach() method does not return a new array.
* The map() method is used to transform the elements of an array, whereas the forEach() method is used to loop through the elements of an array.
* The map() method can be used with other array methods, such as the filter() method, whereas the forEach() method cannot be used with other array methods.
* **Example of map():**

**let arr= [1,2,3,4,5,6];**

**let result= arr.map(value=>{**

**return value\*2**

**})**

**console.log( result);**

**//result: 2, 4,6,8,10,12 //here it generate new array**

* **Example of forEach()**

**let arr= [1,2,3,4,5,6];**

**let result= arr.forEach(value=>{**

**return value\*2**

**})**

**console.log( result);**

**//result is undefined because forEach has no effect of return.**

**Q.3** What is the difference between .call() .apply() and .bind()? explain with an example **(2 Marks)**

**Answer: Call():** call() is a function which help in changing the context of invoking function.means by just writing one function we can change the result output of the function which invoke it or call it. Let’s learn this with an example:

| **let name= {**  **firstname: "Akshay",**  **lastname: "kumar",**    **}**  **let Printfullname= function(){**  **console.log(this.firstname + " " + this.lastname);**  **}**  **Printfullname.call(name);**  **let name2= {**  **firstname: "Rakhshanda",**  **lastname: "begum",**    **}**  **Printfullname.call(name2);**  **//Akshay kumar**  **//Rakhshanda begum** |
| --- |

In this example we create one function and have two objects , and we use call method to invoke the output of this objects.

**apply():**apply() method is same as the call() method the basic difference between this and call method is that it uses array as an argument list.

| **let name= {**  **firstname: "Akshay",**  **lastname: "kumar",**    **}**  **let Printfullname= function(hometown, state){**  **console.log(this.firstname + " " + this.lastname + " from " + hometown + " ,"+ state );**  **}**  **Printfullname.call(name , "Mumabai", "Maharashtra"); //call**  **let name2= {**  **firstname: "Rakhshanda",**  **lastname: "begum",**    **}**  **Printfullname.apply(name2, ["Bhopal", "MP"]);// apply()**  **//Output:**  **Akshay kumar from Mumabai ,Maharashtra**  **Rakhshanda begum from Bhopal ,MP** |
| --- |

**bind():** Bind is a function that helps you create another function that you can execute later with the new context of this that is provided.

| **let name= {**  **firstname: "Akshay",**  **lastname: "kumar",**    **}**  **let Printfullname= function(hometown, state){**  **console.log(this.firstname + " " + this.lastname + " from " + hometown + " ,"+ state );**  **}**  **let name2= {**  **firstname: "Rakhshanda",**  **lastname: "begum",**    **}**  **let FunctionforLater= Printfullname.bind(name2, "Mumbai", "Maharashtra");**  **FunctionforLater(); //it will show the output**  **//Rakhshanda begum from Mumabai ,Maharashtra** |
| --- |

**Q.4** Explain Event bubbling and Event Capturing in JavaScript **(2 Marks)**

**Answer:** **Event bubbling** as a sequence of calling the event handlers when one element is nested in another element, and both the elements have registered listeners for the same event. So beginning from the deepest element to its parents covering all its ancestors on the way to bottom to top, calling is performed.

**Event Capturing:** Event Capturing is opposite to event bubbling, where in event capturing, an event moves from the outermost element to the target. Otherwise, in case of event bubbling, the event movement begins from the target to the outermost element in the file. Event Capturing is performed before event bubbling but capturing is used very rarely because event bubbling is sufficient to handle the event flow.

**Q.5** What is function currying with example? **(2 Marks)**

**Answer:** Currying in JavaScript transforms a function with multiple arguments into a nested series of functions, each taking a single argument. Currying helps you avoid passing the same variable multiple times, and it helps you create a higher order function.

| **let multiply= function(x){**  **return function(y){**  **console.log(x\*y);**  **}**  **}**  **let multiplyone= multiply(2,5);**  **multiplyone(4);**  **let multiplytwo= multiply(10);**  **multiplytwo(4); //Output : 8, 40** |
| --- |

**Q.6** Explain execution context diagram of following code snippets, use white board to draw execution context diagram **(4 Marks)**

**Code Snippet 1**

**console.log('First');**

**setTimeout(() => console.log('Second'), 0);**

**console.log('Third');**

**Code Snippet 2**

**console.log('First');**

**function secondCall() {**

**console.log('Second');**

**}**

**setTimeout(secondCall, 2000);**

**setTimeout(() => console.log('Third'), 0);**

**console.log('Third');**

**Answer:**

**Q.7** What are promises? What are the different states of a promise? Support your answer with an example where you need to create your own promise. **(2 Marks )**

**Answer:** Promises in Javascript is a special object that links the producing code and consuming code together.

**Producing Code:** "Producing code" is code that can take some time

**Consuming Code:**"Consuming code" is code that must wait for the result

The **promise** constructor takes only one argument which is a callback function · The callback function takes two arguments, resolve and reject.These functions are defined in a nested then() method.

To specify what callbacks to call, you use the following two functions:

* resolve(value): This indicates that the asynchronous task was successful. This will call the fulfillment callback in the then() handler.
* reject(error): This indicates an error while trying to run the asynchronous task. This will call the rejection callback in the then() handler.

There are three states of Promises:

* *pending*: initial state, neither fulfilled nor rejected.
* *fulfilled*: meaning that the operation was completed successfully.
* *rejected*: meaning that the operation failed.

| **let myPromise = new Promise(function(myResolve, myReject) {**  **// "Producing Code" (May take some time)**  **myResolve(); // when successful**  **myReject(); // when error**  **});**  **// "Consuming Code" (Must wait for a fulfilled Promise)**  **myPromise.then(**  **function(value) { /\* code if successful \*/ },**  **function(error) { /\* code if some error \*/ }**  **);** |
| --- |

Example:

| **<!DOCTYPE html>**  **<html>**  **<body>**  **<h2>JavaScript Promise</h2>**  **<p id="even"></p>**  **<script>**  **function myDisplayer(check) {**  **document.getElementById("even").innerHTML = check;**  **}**  **let myPromise = new Promise(function(myResolve, myReject) {**  **let x = 4;**  **// some code (try to change x to 5)**  **if (x%2==0) {**  **myResolve("Number is even");**  **} else {**  **myReject("Number is not even");**  **}**  **});**  **myPromise.then(**  **function(value) {myDisplayer(value);},**  **function(error) {myDisplayer(error);}**  **);**  **</script>**  **</body>**  **</html>**  **//Output: Number is even** |
| --- |

**Q.8** What is ‘this’ keyword in JavaScript? explain with an example & create **(4 Marks)**

**Answer:** In JavaScript, the this keyword always refers to an object. The thing about it is that the object it refers to will vary depending on how and where this is being called.

And there's a few different ways in which you can use the this keyword, so let's see the most common cases and how it behaves in each of them.

this is not a variable – it's a keyword, so its value can't be changed or reassigned.

* If we call this by itself means not with function, object etc. then it will return global window object.
* If we call this in object method we will see that it is not longer refer to the object itself. This is also used to access other properties and methods from the same object:
* When using [**strict-mode**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode), calling this within a function will return undefined.
* When using this in an event listener, this will refer to the DOM element that fired the event.
* this keyword is also used in call(), apply() and bind() which we have discussed above .
* In a function, this refers to the global object.

**Q.9** Explain event loop, Call Stack, Callback queue and Micro Task queue in Your Words **(2 Marks)**

**Answer: Call Stack:** Javascript is single threaded programming language that’s means there is only one main stack where all the lines getting pushed and popped out. Let’s learn with better points:

* When the JavaScript engine invokes a function, it adds it to the stack, and the execution starts.
* If the currently executed function calls another function, the engine adds the second function to the stack and starts executing it.
* Once it finishes executing the second function, the engine takes it out from the stack.
* The control goes back to resume the execution of the first function from the point it left it last time.
* Once the execution of the first function is over, the engine takes it out of the stack.
* Continue the same way until there is nothing to put into the stack.

The function execution stack is also known as the **Call Stack.**

**Event loop:** Let’s learn the event loop with an example for the better understanding.

| **console.log(“line 1”);**  **setTimeout(function(){**  **console.log(“Line 2”);**  **},2 );**  **console.log(“Line 3” );** |
| --- |

* Let’s learn what happen , in this example it is quite predictable that first line will get execute.
* For the line 2 it will wait for 2 seconds by that time it will execute line 3.
* And when 2 seconds complete it will execute line2.
* So the output is line1, line3, line2.
* Now let’s understand what happen in call stack.
* First , Line 1 will go in call stack and it will execute in no time.
* Then LIne2 will go in call stack and as it has timeout of 2 seconds , so the API will take care of it , It holds the timeout and it will wait for 2 seconds.
* by the time Line 3 will execute.
* Now when 2 second is over then it will pushed to the **callback queue** or the task queue , now there is something called **Event loop** which keep check on task queue.
* The moment it sees the task in call stack which is queued there that get pushes the task in call stack.
* The event loop takes cares of all the queues and keeps on pushing the task in the call stack ,the moment this process goes to call stack it will executed immediately.

**Micro Task queue:** A microtask is a short function which is executed after the function or program which created it exits and only if the JavaScript execution stack is empty, but before returning control to the event loop being used by the user agent to drive the script's execution environment

**Q.10** Explain Debouncing and Create a project where you are using Debouncing **(5 Marks)**

**Answer:** Debouncing is a programming technique that assures that time-consuming activities do not trigger the web page's performance decreases. In other words, the Debounce methods do not run when invoked. Instead, they wait a predetermined period of time until executing. When we call the same process again, the previous process is canceled, and the timer is reset.

Let's say that we want to show suggestions for a search query, but only after a visitor has finished typing it.

Or we want to save changes on a form, but only when the user is not actively working on those changes, as every "save" costs us a database trip.

**Q.11** Explain Closures and Use cases of Closures **(2 Marks)**

**Answer:** Closures are functions that have access to the variables that are present in their scope chain even if the outer function ceases to exist.

To understand this in more detail, let's understand what a scope chain is. Scope chain refers to the fact that parent scope does not have access to the variables inside its children's scope, but the children's scope does have access to the variables present in its parent scopes.

Closures do not have access to their outer function’s args parameter.

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives you access to an outer function’s scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

To use a closure, define a function inside another function and expose it. To expose a function, return it or pass it to another function.The inner function will have access to the variables in the outer function scope, even after the outer function has returned.

In JavaScript, closures are the primary mechanism used to enable data privacy. When you use closures for data privacy, the enclosed variables are only in scope within the containing (outer) function.

**Q.12** Create a Blog web app using JavaScript **(10 Marks)**

- Fetch data from https://jsonplaceholder.typicode.com/posts and show it to ui - User can also add new blog

- Add Delete functionality also

Answer: **Github link: https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/tree/main/blogapp%20in%20js**

← React Questions →

**(50 Marks)**

Q.1 What’s React and What are the advantages of it? **(1 Marks)**

**Answer:** React. js is a lightweight JavaScript Library used for building user interfaces, which can also be applied for creating reusable User Interface components. React. js is a declarative, efficient, and flexible JavaScript library that helps develop fast and user-friendly web applications.

It is actively maintained by Meta and a community of skilled developers and companies.

**Some advantages of React are:**

* It is composable.
* It is declarative.
* It saves time.
* Write once, and learn anywhere.
* It is simple.
* SEO friendly.
* Fast, efficient, and easy to learn.
* It is backed by a strong community.

Q.2 What's Virtual Dom in React & What are the advantages of it? **(2Marks)**

**Answer:** Virtual DOM in React is a memory replica of actual DOM.React uses the virtual DOM as a strategy to compute minimal DOM operations when re-rendering the UI. It is not in rivalry with or faster than the real DOM.

The virtual DOM provides a mechanism that abstracts manual DOM manipulations away from the developer, helping us to write more predictable code. It does so by comparing two render trees to determine exactly what has changed, only updating what is necessary on the actual DOM.

Manipulating DOM is slow, but manipulating Virtual DOM is fast as nothing gets drawn on the screen. So each time there is a change in the state of our application, the virtual DOM gets updated first instead of the real DOM.

Q.3 Explain LifeCycle of React Components? **(1 Marks)**

**Answer:** In ReactJS, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component's lifecycle. These lifecycle methods are not very complicated and are called at various points during a component's life. The lifecycle of the component is divided into four phases. They are:

* Initial Phase(It is the **birth** phase of the lifecycle of a ReactJS component.)
* Mounting Phase(In this phase, the instance of a component is created and inserted into the DOM. )
* Updating Phase(It is the next phase of the lifecycle of a react component. Here, we get new **Props** and change **State)**
* Unmounting Phase(It is the final phase of the react component lifecycle. It is called when a component instance is **destroyed** and **unmounted** from the DOM)

Q.4 Whats the difference between between Functional Components and Class Components? **(1 Marks)**

| **Functional Components** | **Class Components** |
| --- | --- |
| 1. Functional Components are js pure functions which accept props as an statement and render the response in JSX 2. render() method is not available here. 3. The code in this component is shorter. 4. Constructor are not used in this components. 5. Functional components only use props as a arguments | 1 The Class Componets require the developer to extend from the react. To render the response it needs to create the components as well as the render function  2. render() method is available here.  3. The code in this component is longer.  4. Constructor are used to store the state.  5. In this we can use both props as arguments and set the state also. |

Q.5 What are the hooks in React & Can we use Hooks in Class Components? **(2 Marks)**

**Answer:** Functional components were stateless and they were used for simple components. When hooks were released, the main motivation was having functional components with the same capabilities as class components. There are different type of in-built hooks which we can use in out components or we can customize out hook too. Few hooks are:

* useState.
* useEffect
* useContext
* useRef

You can't use a hook directly in a class component, but you can use a hook in a wrapped function component with a render prop to achieve this.

Q.6 What are the LifeCycle methods and the advantages of it? **(2 Marks)**

**Answer: There are mainly 4 stages of lifecycle.**

* **Initial Phase:**

It is the **birth** phase of the lifecycle of a ReactJS component.

* **Mounting Phase:**

In this phase, the instance of a component is created and inserted into the DOM. React supports three mounting lifecycle methods for component classes: **componentWillMount(), render(), and componentDidMount(). componentWillMount()** will be called first followed by the **render()** method and finally the **componentDidMount()** method.

* **Updating Phase**:

It is the next phase of the lifecycle of a react component. Here, we get new Props and change StateWhen a component updates, **shouldComponentUpdate()** gets called after **componentWillReceiveProps()**, but still before the rendering begins. It automatically receives two arguments: **nextProps and nextState.**

**shouldComponentUpdate()** should return either true or false. The best way to use this method is to have it return false *only under certain conditions*. If those conditions are met, then your component will not update.

* **Unmounting Phase**:

It is the final phase of the react component lifecycle. It is called when a component instance is **destroyed** and **unmounted** from the DOM. React supports one unmounting lifecycle method, **componentWillUnmount,** which will be called right before a component is removed from the DOM. **componentWillUnmount()** is used to do any necessary cleanup (canceling any timers or intervals, for example) before the component disappears.

Q.7 What’s useState Hook & Advantages of it? **(2 Marks)**

**Answer:** useState is React Hook that allows you to add state to a [functional component](https://blog.logrocket.com/fundamentals-functional-programming-react/). It returns an array with two values: the current state and a function to update it. The Hook takes an initial state value as an argument and returns an updated state value whenever the setter function is called. It can be used like this:

const [state, setState] = useState(initialValue);

Here, the initialValue is the value you want to start with, and state is the current state value that can be used in your component. The setState function can be used to update the state, triggering a re-render of your component.

**Advantage of useState hook:** In React, useState can store any type of value, whereas the state in a class component is limited to being an object. This includes primitive data types like string, number, and Boolean, as well as complex data types such as array, object, and function. It can even cover custom data types like class instances.

Basically, anything that can be stored in a JavaScript variable can be stored in a state managed by useState.

Q.8 Explain useEffect & Advantages of it **(2 Marks)**

**Answer:** The **useEffect** Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

**useEffect** accepts two arguments. The second argument is optional.

**useEffect(<function>, <dependency>)**

useEffect inside the component lets us access the count state variable (or any props) right from the effect.

Q.9 Explain Context Api and create a minor project on it **(5 Marks)**

- Create a dashboard and with a button on clicking on that change theme to dark and light

**Answer:** Managing state is an essential part of developing applications in React. A common way to manage state is by passing props. Passing props means sending data from one component to another. It's a good way to make sure that data gets to the right place in a React application.

But it can be annoying to pass props when you have to send the same data to lots of components or when components are far away from each other. This can make an application slower and harder to work with.

Fortunately, React provides a built-in feature known as the context API that helps “teleport” data to the components that need it without passing props.

For saving our valuable time, we make context for particular tasks and react with the component with context. This concept is called context api.

**GIthub Link:** https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/blob/main/Theme%20Changer%20in%20react

Q.10 Explain useReducer and Its advantages. **(2 Marks)**

**Answer:** The useReducer Hook is similar to the useState Hook.

It allows for custom state logic.

If you find yourself keeping track of multiple pieces of state that rely on complex logic, useReducer may be useful.

The useReducer Hook accepts two arguments.

**useReducer(<reducer> , <initialState>)**

The reducer function contains your custom state logic and the initialStatecan be a simple value but generally will contain an object.

The useReducer Hook returns the current stateand a dispatchmethod.

useReducer is suitable for managing complex state. It allows you to define a separate action object and switch statement to handle different state updates.

Q.11 build a Todo Web App Using React and useReducer Hook**. (5Marks) Answer: Github Link:** https://github.com/rakii8458/FSJS2.0-Pre-Placement-Assignment/tree/main/todo-app

Q.12 Build A simple counter app using React **(4 Marks)**

Q.13 Build Calculator Using React Only **(4 Marks)**

Q.14 Build a Tic Tac Toe Game using Class Component of React **(5 Marks)**

Q.15 Explain Prop Drilling & How can we avoid it? **(2 Marks)**

**Answer:** In React, passing props is a fundamental concept that enables a parent component to share data with its child components as well as other components within an application.

passing props can be an effective way to share data between different parts of your application. But passing props down a chain of multiple components to reach a specific component can make your code overly cumbersome.

to pass data down to the component "Child X", we need to pass it down through all the intermediate components, even if those components don't actually use the data themselves. This is what is referred to as **"prop drilling."**

We can avoid prop drilling by simply using context api, which we have discussed above.

Q.16 Create a task manager where user can create tasks and see his task **(10 Marks)**

- Redirect him to task dashboard section after login

- Use https://reqres.in/ api to authenticate user and redirect him to task manager dashboard where he can see his task and create