Html Answers:

1. The <!DOCTYPE html> declaration is not an HTML tag but a way to tell the browser that the document is written in HTML5. It helps ensure the page is displayed correctly across different browsers and devices. It's placed at the beginning of the HTML document to specify the version of HTML being used.
2. Semantic tags in HTML are special tags that provide meaning and structure to the content within a web page. They convey the purpose or role of the enclosed content to both browsers and developers. Examples of semantic tags include <header>, <nav>, <article>, <section>, <aside>, <footer>, and more.

Semantic tags in HTML improve accessibility, SEO, and code maintainability. They help screen readers and assistive technologies understand content structure. Search engines benefit from better context understanding. Semantic tags enhance code organisation and readability for developers.

1. HTML Tags are markup characters (enclosed in angle brackets Example. <\_></\_>) used to define elements within an HTML document. They consist of an opening tag, content, and a closing tag.HTML Elements, on the other hand, are complete components made up of HTML tags and the content they enclose. Elements represent structural or semantic parts of a webpage.
2. Github Link:- <https://github.com/AviralSinghPal/PW-assignments/blob/main/answer4.html>
3. Github Link:- <https://github.com/AviralSinghPal/PW-assignments/blob/main/answer5.html>
4. HTML5 brought significant advancements to web development. It introduced semantic elements like <header>, <footer>, and <nav>, improving the structure and search engine optimization of web pages. Native support for multimedia elements such as <video> and <audio> eliminated the need for external plugins and enhanced compatibility and performance. HTML5 also introduced <canvas> for dynamic graphics and animations, along with SVG for scalable vector graphics. The addition of offline and storage capabilities through localStorage and sessionStorage enabled the development of offline web applications. Enhanced form inputs, built-in validation, and improved accessibility features made it easier to create user-friendly and inclusive web experiences. HTML5's mobile support with APIs for geolocation, device orientation, and touch events catered to the growing mobile web. Lastly, HTML5 simplified the DOCTYPE declaration, providing a streamlined approach for declaring the document type.
5. <!DOCTYPE html>

<html>

<head>

<title>Audio Example</title>

</head>

<body>

<audio src="audio-file.mp3" controls></audio>

</body>

</html>

1. <img> tag: Used to embed standalone images within an HTML document.

<figure> tag: Used as a container to group self-contained content, including images,with optional captions.

1. HTML tag are used to create an HTML element. The properties and characteristics for these HTML element are governed by the HTML attributes

Eg: <div class=”abc”>, here div is a tag and class is attribute.

Similarly some global attributes are :- Id, style , title, etc.

1. https://github.com/AviralSinghPal/PW-assignments/blob/main/answer10

CSS Answers:

1. The box model in CSS describes how elements are rendered and how their dimensions are calculated. It consists of four components: content, padding, border, and margin.

The content refers to the actual content of the element, such as text or images. The dimensions of the content area are defined using properties like width and height.

1. CSS provides various types of selectors to target specific HTML elements for styling. The commonly used selectors include type selectors, class selectors, ID selectors, attribute selectors, and pseudo-classes/pseudo-elements.

The advantages of these selectors include flexibility in targeting elements, reusability of styles with class selectors, specificity with ID selectors, and dynamic styling with pseudo-classes/pseudo-elements.

By utilizing the different types of selectors, you can apply styles precisely, create well-structured stylesheets, and enhance the visual appeal and interactivity of your web pages.

1. VW (Viewport Width) and VH (Viewport Height) are units of measurement in CSS that are relative to the size of the viewport, which is the visible area of the web page in the browser window.Whereas, PX represents a fixed number of pixels. When using PX, the size of an element will remain fixed regardless of the viewport size. This can be useful for elements that need to maintain a specific size regardless of the screen size.
2. Inline Elements:

Inline elements do not start on a new line and only occupy the necessary space to contain their content.

They do not allow width and height properties to be applied to them.

Examples of inline elements include <span>, <a>, <strong>, <em>, and <img>.

Inline-Block Elements:

Inline-block elements are similar to inline elements in that they do not start on a new line.

However, inline-block elements allow width and height properties to be applied to them.

They also respect padding, margin, and line-height properties.

Examples of inline-block elements include <input>, <button>, and <select>.

Block Elements:

Block elements start on a new line and occupy the full available width of their parent container by default.

They allow width, height, padding, margin, and other box-model properties to be applied to them.

Block elements create a visual separation between themselves and surrounding elements, pushing subsequent elements to a new line.

Examples of block elements include <div>, <p>, <h1> to <h6>, <ul>, <li>, and <section>.

1. Content-box (default) calculates the width and height based solely on the content and does not include padding or border. Additional padding and border are added to the specified width and height, resulting in a larger total size.

Border-box calculates the width and height to include the content, padding, and border. The specified width and height represent the total space the element should occupy, and the browser adjusts the content size to fit within these dimensions while accounting for padding and border.

1. z-index is a CSS property that controls the stacking order of positioned elements. It determines the order in which elements are stacked or layered on top of each other in the z-axis (depth) on a web page.

The z-index property accepts integer values, where a higher value represents a higher stacking order. Elements with a higher z-index value will be displayed on top of elements with a lower value.

6. Flexbox is a one-dimensional layout system used for arranging elements in a row or column, while Grid is a two-dimensional layout system used for creating grid structures with rows and columns. Flexbox is best for handling content flow and alignment along a single axis, while Grid is ideal for complex layouts where items need to be placed in specific grid cells.

1. https://github.com/AviralSinghPal/PW-assignments/tree/main/CSS/question7
2. https://github.com/AviralSinghPal/PW-assignments/tree/main/CSS/question8
3. <https://github.com/AviralSinghPal/PW-assignments/tree/main/CSS/question9>
4. <https://github.com/AviralSinghPal/PW-assignments/tree/main/CSS/question10>
5. Gjf
6. pseudo-classes are used to style elements based on states or conditions, such as user interactions or structural relationships. They modify the appearance of existing elements.

Pseudo-elements, on the other hand, create virtual elements that don't exist in the HTML markup. They are used to style specific parts of an element, such as the first letter or line, or to insert content before or after an element.

Both pseudo-classes and pseudo-elements provide additional styling capabilities and enhance the flexibility of CSS selectors.

JavaScript Answers:

1. When we run a javascript code all the variables are scanned and made undefined and all the functions are scanned and made available at the compilation phase this process is called Hoisting in Javascript.

Example for variables :-

console.log(x);

var x = 123;

Output => undefined

Example for functions :-

xyz();

function xyz(){

console.log(“Hello World !“);

}

OUTPUT => Hello World!

1. HOF or higher order functions are the functions which can take a function as an argument/input and return function as output. Various HOF include map, filter, forEach, reduce , etc.

The map function is used to transform each element of an array and return a new array of the same length. It applies a provided function to each element and collects the results in a new array. The original array remains unchanged. The map function returns a new array with the transformed values.

Example:

const arr = [1,2,3,4];

const sq = arr.map((ele)=>(ele\*ele));

console.log(sq); //[ 1, 4, 9, 16 ]

console.log(arr); // [ 1, 2, 3, 4 ]

The forEach function is used to iterate over each element of an array and perform an action for each element. It doesn't return a new array but instead allows you to execute a function for each element.

Example:

let arr2=[];

arr.forEach((ele)=>(arr2.push(ele\*2)));

console.log(arr2); //[ 2, 4, 6, 8 ]

1. call():- takes two arguments: the function to call and the this value. The this value can be an object or a primitive value.

apply():- takes three arguments: the function to call, the this value, and an array of arguments.

bind():- takes two arguments: the function to bind and the this value. The this value can be an object or a primitive value.

Examples:-

function foo() {

console.log(this.name);

}

var obj = { name: "Bar" };

foo.call(obj); // Prints "Bar"

—--------------------------------

function foo() {

console.log(this.name, arguments[0]);

}

var obj = { name: "Bar" };

foo.apply(obj, ["Baz"]); // Prints "Bar Baz"

—--------------------------------

function foo() {

console.log(this.name);

}

var boundFoo = foo.bind(obj);

boundFoo(); // Prints "Bar"

1. When an event occurs in an element, it is first handled by the element's parent. Then, the event is propagated down the DOM tree to the element's child, grandchild, and so on, until it reaches the target element , this process is known as Event capturing.

Event bubbling is the default way that events propagate. When an event occurs in an element, it is first handled by that element. Then, the event is propagated up the DOM tree to the element's parent, grandparent, and so on, until it reaches the top of the tree.

Event bubbling is the most common way to handle events. It is the simplest to use, and it is the way that most users expect events to work.

Event capturing is less common, but it can be useful in some cases. For example, you might use event capturing to prevent an event from reaching the target element. You can also use event capturing to add additional functionality to an event handler.

Consider HTML:-

<div id="outer">

<div id="inner">

<button id="btn">Click Me</button>

</div>

</div>

*Example for Event Bubbling->*

const btn = document.getElementById('btn');

const inner = document.getElementById('inner');

const outer = document.getElementById('outer');

btn.addEventListener('click', () => {

console.log('Button clicked!');

});

inner.addEventListener('click', () => {

console.log('Inner div clicked!');

});

outer.addEventListener('click', () => {

console.log('Outer div clicked!');

});

Output:-

Button clicked!

Inner div clicked!

Outer div clicked!

Example of Event Capturing (for the same HTML):-

const btn = document.getElementById('btn');

const inner = document.getElementById('inner');

const outer = document.getElementById('outer');

btn.addEventListener('click', () => {

console.log('Button clicked!');

}, false); // The 'false' parameter indicates event bubbling (default behavior).

inner.addEventListener('click', () => {

console.log('Inner div clicked!');

}, false);

outer.addEventListener('click', () => {

console.log('Outer div clicked!');

}, true); // The 'true' parameter indicates event capturing.

Output:-

Outer div clicked!

Inner div clicked!

Button clicked!

1. Function Currying is a technique used in function to accept multiple arguments using a sequence of function while accepting one argument.

Example:-

function calculateVolume(length) {

return function(width) {

return function(height) {

return length \* width \* height;

};

};

}

const volume = calculateVolume(5)(3)(2);

console.log(volume); // Output: 30

2. A promise helps in achieving asynchronous behaviour in javascript. In a promise an operation is either completed successfully or it fails. It help in making the output of the operation more predictable. There can be 3 states of promise Pending, Fulfilled, Rejected.

Promises have two important methods: .then() and .catch(). The .then() method is used to handle the fulfilment of a promise, while the .catch() method is used to handle any errors or rejections.

Example using an API :-

function fetchData() {

return new Promise((resolve, reject) => {

fetch('https://api.example.com/data')

.then(response => {

if (response.ok) {

return response.json();

} else {

throw new Error('Error fetching data');

}

})

.then(data => resolve(data))

.catch(error => reject(error));

});

}

// Using the promise

fetchData()

.then(data => {

console.log(data); // Output: Fetched data from API

})

.catch(error => {

console.log(error); // Output: Error fetching data

});

1. In JavaScript, the use of the "this" keyword can vary depending on the context in which it is used. Your statement is partially correct, but there are a few additional cases where "this" is used:

Inside an object method: When "this" is used inside a method of an object, it refers to the object itself. For example:

const obj = {

name: "John",

greet: function() {

console.log("Hello, " + this.name);

}

};

obj.greet(); // Output: Hello, John

In this case, "this" refers to the "obj" object.

Inside a constructor function: When "this" is used inside a constructor function, it refers to the newly created instance of the object. For example:

function Person(name) {

this.name = name;

}

const john = new Person("John");

console.log(john.name); // Output: John

Here, "this" refers to the newly created "john" object.

1. Call Stack: The call stack is a data structure that keeps track of function calls in a program. Whenever a function is called, a new frame (also known as an execution context) is created and pushed onto the top of the call stack. This frame contains information about the function call, such as the function's arguments and local variables. The function's execution starts, and any further function calls within that function are added on top of the stack. When a function completes its execution, its frame is popped off the stack, and the control returns to the previous function in the stack.

Callback Queue: The callback queue, also known as the task queue or message queue, is a queue data structure that holds callbacks or events. In JavaScript, callbacks are commonly used for asynchronous operations such as AJAX requests, timers, and event handlers. When an asynchronous operation completes or an event occurs, the corresponding callback is placed in the callback queue.

Event Loop: The event loop is a fundamental part of JavaScript's concurrency model. Its main job is to continuously monitor the call stack and the callback queue. If the call stack is empty, the event loop takes the first callback from the callback queue and pushes it onto the call stack, allowing its execution to begin. This process is repeated in a continuous loop, ensuring that callbacks are executed in a sequential and non-blocking manner.

1. Debouncing is used to limit the number of function executions. In some cases a function could be called very frequently (scrolling or typing) in such cases debouncing is useful.

Example: - <https://github.com/AviralSinghPal/PW-assignments/blob/main/js/Debouncing/index.html>

1. Closure is a concept in Javascript where a inner function can access the variable that are accessible to outer function .It essentially "closes over" the variables, creating a scope chain that persists even when the outer function has completed.

Example:-

function createCounter() {

let count = 0;

return function() {

return ++count;

};

}

const counter = createCounter();

console.log(counter()); // Output: 1

console.log(counter()); // Output: 2

console.log(counter()); // Output: 3

Closure can be very useful when we try pass one function to other function(that accept only function as parameter).

Example:-

Function clickhandler(‘color’){

return function(){ //here this function has the access to color variable

document.body.style.backgroundcolor=`${color}`

}

}

Now this function can be passed to other function that take only function as parameter such as onClick().

Other than this closure are widely used in react also.

1. HTML:- <https://github.com/AviralSinghPal/PW-assignments/blob/main/js/index.html>

JS:- <https://github.com/AviralSinghPal/PW-assignments/blob/main/js/app.js>

React Answers:

1. REACT.js is a javascript library for implementing the frontend of the websites.

It follows a component based rendering architecture. We can implement a large scale project by efficiently breaking it down to simple components.

Advantages of using React is that ➖

* There are various components which can be reused very easily and efficiently.
* Virtual DOM helps in quickly rendering the nodes in which changes were made.
* There are many lifecycle methods which help to hook different lifestyle components. It helps in monitoring the unused variable and updating the state.

1. Virtual DOM is like a copy of actual DOM, in react we can efficiently update the virtual DOM in such a way that the component in which changes are made is the only node that is being rerendered which helps us in better User experience and is light on the browser also.
2. React provide various lifecycle methods which allow us to hook into the lifecycle of the component such as update of component and on this basis we can perform such action like initiation , rendering or removal .It helps in monitoring the unused variable and updating the state.
3. Functional Components:

* Defined as JavaScript functions.
* Do not have their own internal state (prior to React Hooks).
* Do not have lifecycle methods (prior to React Hooks).
* Simpler and shorter code structure.
* Better performance.
* Modern and streamlined development experience.

Class Components:

* Defined as ES6 classes that extend React.Component.
* Have their own internal state managed by setState.
* Have lifecycle methods like componentDidMount, componentDidUpdate, etc.
* More complex code structure.
* Slightly lower performance compared to functional components.
* Still supported and necessary in certain scenarios.

1. Hooks are to track state and lifecycle of a component in React. There are various Hooks.

example : useState => it helps in managing the state of a functional component .

useEffect=> it allows functional components to perform side effects, such as fetching data, subscribing to events, or manipulating the DOM.

1. React provide various lifecycle methods which allow us to hook into the lifecycle of the component such as update of component and on this basis we can perform such action like initiation , rendering or removal .It helps in monitoring the unused variable and updating the state.
2. The useState hook in React is a function that allows functional components to manage state. It takes an initial state value as an argument and returns an array containing the current state value and a function to update that state.

Advantages are as follows:-

* Functional Approach: useState aligns with the functional programming paradigm, promoting immutable data and pure functions to update state.
* Declarative Updates: State is updated using the provided update function, ensuring correct state updates and triggering component re-rendering.
* Multiple State Variables: Multiple independent state variables can be declared and managed separately within a single component using useState.

1. The useEffect hook is a built-in hook in React that allows functional components to perform side effects, such as data fetching, subscriptions, or manipulating the DOM. It is used to handle lifecycle-related operations within functional components.

Advantage are as follows:-

* Handles Side Effects: useEffect provides a clean and concise way to handle side effects, such as data fetching, API calls, DOM manipulation, and subscriptions, within functional components.
* Dependency Tracking: By specifying dependencies as the second argument to useEffect, you can control when the effect should be re-run. This helps optimise performance and avoids unnecessary re-renders.
* Encourages Modularity and Separation of Concerns: useEffect promotes a modular approach by allowing you to separate different side effects into individual effect functions. This improves code organisation, readability, and maintainability.

12. <https://github.com/AviralSinghPal/PW-assignments/tree/main/React/question12>

13. <https://github.com/AviralSinghPal/PW-assignments/tree/main/React/question13>

15. Prop drilling refers to the process of passing props through multiple layers of components, even if some intermediate components do not actually need those props. It can lead to complex and less maintainable code. To avoid prop drilling, you can use techniques like Context API, Redux, or React Router to manage and share data across components without the need for explicit prop passing. These approaches promote cleaner code and make it easier to maintain and scale your application.

Express Answers:

1. <https://github.com/AviralSinghPal/PW-assignments/blob/main/Express/question1/app.js>
2. <https://github.com/AviralSinghPal/PW-assignments/blob/main/Express/question2/app.js>
3. <https://github.com/AviralSinghPal/PW-assignments/blob/main/Express/question3/app.js>
4. Authentication is the process to identify if the user’s credentials are correct or not and accordingly give them access to any resource or functionality.

Authorization is granting or denying the access request for a specific resource or functionality based on Authenticated user’s privileges.

1. EJS (Embedded JavaScript):

EJS is a templating engine that allows you to embed JavaScript code within HTML templates. It is commonly used in web development frameworks like Express.js. EJS modules are not used for module management like CommonJS but serve as a way to dynamically generate HTML content by combining static markup and dynamic JavaScript logic. EJS templates are parsed and executed on the server-side before being sent to the client. Whereas in CommonJS, it is a module system used in Node.js and is primarily designed for server-side JavaScript applications. It follows a synchronous approach to module loading, where modules are loaded and executed synchronously.

1. JWT is JSON Web Token With JWT, you can achieve stateless authentication, where the server does not need to store session data for each user.. It is a compact, URL-safe means of representing claims between two parties. JWTs are commonly used for authentication and authorization purposes in web applications.

<https://github.com/AviralSinghPal/PW-assignments/tree/main/Express/question6>

1. We should always encrypt the password before store it in database . Storing the password in the database in a clear text format is a very bad practice . To securely store passwords in Node.js, you can use the bcrypt library, which provides a convenient way to hash passwords with an added level of security. Here's an example of how to store a password using bcrypt in Node.js:

const bcrypt = require('bcrypt');

// User password to be stored

const password = 'myPassword123';

// Generate a salt

const saltRounds = 10;

const salt = bcrypt.genSaltSync(saltRounds);

// Hash the password with the generated salt

const hashedPassword = bcrypt.hashSync(password, salt);

1. .In Node.js, the event loop is a fundamental part of its architecture and plays a crucial role in handling asynchronous operations. The event loop is responsible for managing and executing the different tasks in a non-blocking manner, allowing Node.js to handle concurrent requests efficiently.
2. https://github.com/AviralSinghPal/PW-assignments/tree/main/Express/question9