HTML Answers

1. <!DOCTYPE html> is a specific type tag in HTML called Document Type Declaration (DTD), and it is used to specify the browser the version of HTML or XML that the document is written in.
2. Semantic tags in HTML are elements introduced in HTML5 that are used to convey the structure and purpose of the content within a web page. With semantic tags, the readability of the code increases and our webpage becomes more SEO-friendly. Examples of semantic tags are nav, header, main, footer, etc.
3. HTML tags are the individual markup components that define the structure and semantics of the documents whereas, HTML elements consist of the opening and closing tags, along with the content they enclose, and represent the actual components and parts of a webpage.

Example of tags: <p></p>

Example of element: <p>Hello World!</p>

1. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/resume>
2. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/simple%20html>
3. HTML5 introduced several significant improvements and new features over its previous versions. Some of them are listed below:

* Improved semantics with semantic tags
* Support of multimedia with video tag
* Support of canvas
* More input options in the form
* Improved accessibility

1. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/music%20player>
2. The <img> tag is a self-closing tag used to embed an image in an HTML document. It is used to display a single image on a web page.

Example: <imgsrc="image.jpg" alt="An image">

The <figure> tag is used to encapsulate and provide semantic meaning to a self-contained content block, such as an image or multimedia object, along with an optional caption.

1. An HTML tag represents an element in an HTML document. Tags are used to define the structure and content of a web page whereas attributes provide additional information about an HTML element. Attributes are used within the opening tag of an element to specify characteristics or properties of the element.

Some of the global attributes are:

* id
* class
* style
* title
* lang
* data

1. [https://github.com/Zamzam1707/Placement-Assignment/tree/main/table](https://github.com/prateek-budhiraja/placement-assignment_prateek-budhiraja/tree/main/table-html)

CSS Answers

1. The box model is like creating boxes around all the HTML elements. Every block-level element we put on the screen is a box, it has some content, comes with height and width, margin and padding and a border. Properties include height, width, padding, margin and border.
2. CSS provides various types of selectors that allow us to target specific elements or groups of elements on a web page. Different types of include:

* Element selector
* Class selector
* Id selector
* Attribute selector
* Pseudo class selector
* Pseudo element selector

By using CSS selectors, we can select specific elements or group of elements from our webpage and style them using CSS.

1. VW (viewport width) and VH (viewport height) are units of measurement in CSS that allow you to specify sizes relative to the viewport dimensions or to the screen in which our webpage is displayed.

The main difference between VW/VH and PX is that VW/VH are relative units that scale with the viewport size, while PX is an absolute unit that remains fixed regardless of the viewport dimensions.

1. Inline elements only occupy the space necessary to contain their content. They also do not create line breaks before or after themselves.

Block elements start on a new line and occupy the full width available within their parent container. They create line breaks before and after themselves.

Inline-block elements are a hybrid of inline and block elements, allowing them to have width, height, and other box properties while still being able to sit beside other elements horizontally.

1. Difference between box-sizing & content-box

Content Box:

With box-sizing: content-box, the width and height of an element are calculated based on its content box alone.

The width and height specified for an element in CSS represent the dimensions of the content box.

Any padding or border added to the element is added to the specified width and height, increasing the overall size of the element.

Border Box:

With box-sizing: border-box, the width and height of an element are calculated including its content, padding, and border.

The width and height specified for an element in CSS represent the dimensions of the border box.

The padding and border are included within the specified width and height, so they do not increase the overall size of the element.

1. z-index is a CSS property that controls the stacking order of elements on a web page in the z-axis. The z-axis represents the depth or stacking order of elements, with a higher z-index value indicating that an element should appear on top of elements with lower z-index values. The z-index property only applies to elements that have a position value other than static. Proper use of the z-index is essential to control the stacking order of overlapping elements
2. Grid and Flexbox are two CSS layout systems that provide powerful capabilities for creating responsive and flexible layouts. While they both serve similar purposes, they have different approaches and are suited for different use cases.

Flexbox:

Flexbox, short for Flexible Box Layout, is a one-dimensional layout system that focuses on arranging elements in a row or a column. It provides an easy way to create flexible and dynamic layouts, especially for building components like navigation bars, flexible content containers, or vertically aligned elements.

Key features of Flexbox:

* One-dimensional: Flexbox operates in a single dimension either horizontally or vertically.
* Flex container and flex items: The parent element becomes a flex container by applying display: flex, and the child elements within it become flex items.
* Flexibility: Flexbox provides flexible resizing and alignment options for the flex items, allowing them to expand, shrink, and adjust their size based on available space.
* Main and cross axis: Flexbox operates along two axes - main axis (horizontal or vertical) and cross axis (perpendicular to the main axis).
* Alignment and ordering: Flexbox offers powerful alignment and ordering properties to control the alignment, spacing, and order of flex items.

Grid:

CSS Grid Layout, commonly known as Grid, is a two-dimensional layout system that enables precise control over both rows and columns. It allows for creating complex grid-based layouts with ease, making it suitable for creating overall page layouts, responsive grids, and grid-based designs.

Key features of Grid:

* Two-dimensional: Grid operates in both rows and columns simultaneously, allowing for complex grid-based layouts.
* Grid container and grid items: The parent element becomes a grid container by applying display: grid, and the child elements within it become grid items.
* Grid tracks and grid cells: Grid defines a grid structure using rows and columns, forming grid tracks, and each cell within the grid is a grid cell.
* Explicit positioning: Grid provides fine-grained control over the placement and sizing of grid items through explicit positioning using grid lines and grid areas.
* Alignment and spanning: Grid offers powerful alignment and spanning capabilities, allowing for easy control over item placement and spanning multiple grid cells.

1. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/periodic%20table>
2. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/layout%20color>
3. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/responsive%20layout>
4. Pseudo-classes are used to select and style elements based on certain states or conditions. They represent a specific state of an element or its relationship to the document or user interaction. Pseudo-classes are denoted by a single colon (:) followed by the name of the pseudo-class.  
   Example:  
   :hover: Selects an element when the user hovers over it.

:active: Selects an element when it is being activated (e.g., clicked).

:focus: Selects an element when it has keyboard focus.

:first-child: Selects the first child element of its parent.

:nth-child(n): Selects elements based on their position in the parent.

JavaScript Answers

1. Hoisting is a behaviour in JavaScript where variable and function declarations are moved to the top of their containing scope during the compilation phase. This means that you can use variables and functions before they are declared in your code.

Hoisting affects variable and function declarations, but not their assignments or initializations. Only the declarations are moved to the top of the scope.

Variable declarations using the var keyword are hoisted to the top of their scope and are implicitly initialized with the value undefined. This means you can use a variable before it is declared, but its value will be undefined until it is assigned a value.

1. In JavaScript, higher-order functions are functions that take other functions as arguments or return functions as their results.

Commonly used higher-order functions in JavaScript:

map(): The map() method creates a new array by calling a provided function on each element of the original array.

Example:

const numbers = [1, 2, 3, 4, 5];

constdoubledNumbers = numbers.map((num) =>num \* 2);

console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]

1. .call(): The .call() method is used to invoke a function immediately and explicitly specify the value of this within that function. It takes arguments individually.

Example:

const person = {

  name: 'John',

  greet: function(message) {

    console.log(`${message}, ${this.name}!`);

  }

};

const person2 = {

  name: 'Jane'

};

person.greet.call(person2, 'Hello'); // Output: "Hello, Jane!"

.apply(): The .apply() method is similar to .call(), but it takes arguments as an array or an array-like object.

Example:

const person = {

  name: 'John',

  greet: function(message) {

    console.log(`${message}, ${this.name}!`);

  }

};

const person2 = {

  name: 'Jane'

};

person.greet.apply(person2, ['Hello']); // Output: "Hello, Jane!"

.bind(): The .bind() method returns a new function with a bound this value, without immediately invoking the function. It allows you to create a function with a specific context that can be invoked later.

Example:

const person = {

  name: 'John',

  greet: function(message) {

    console.log(`${message}, ${this.name}!`);

  }

};

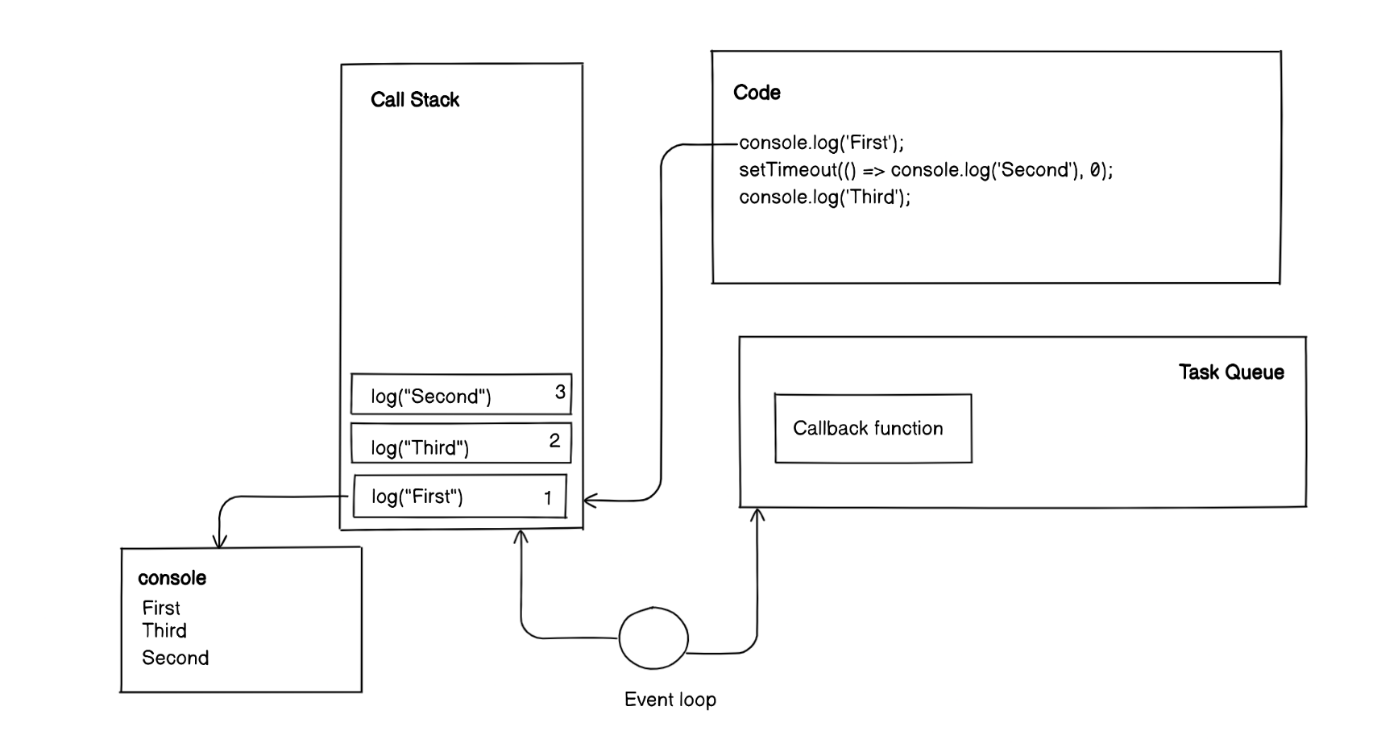
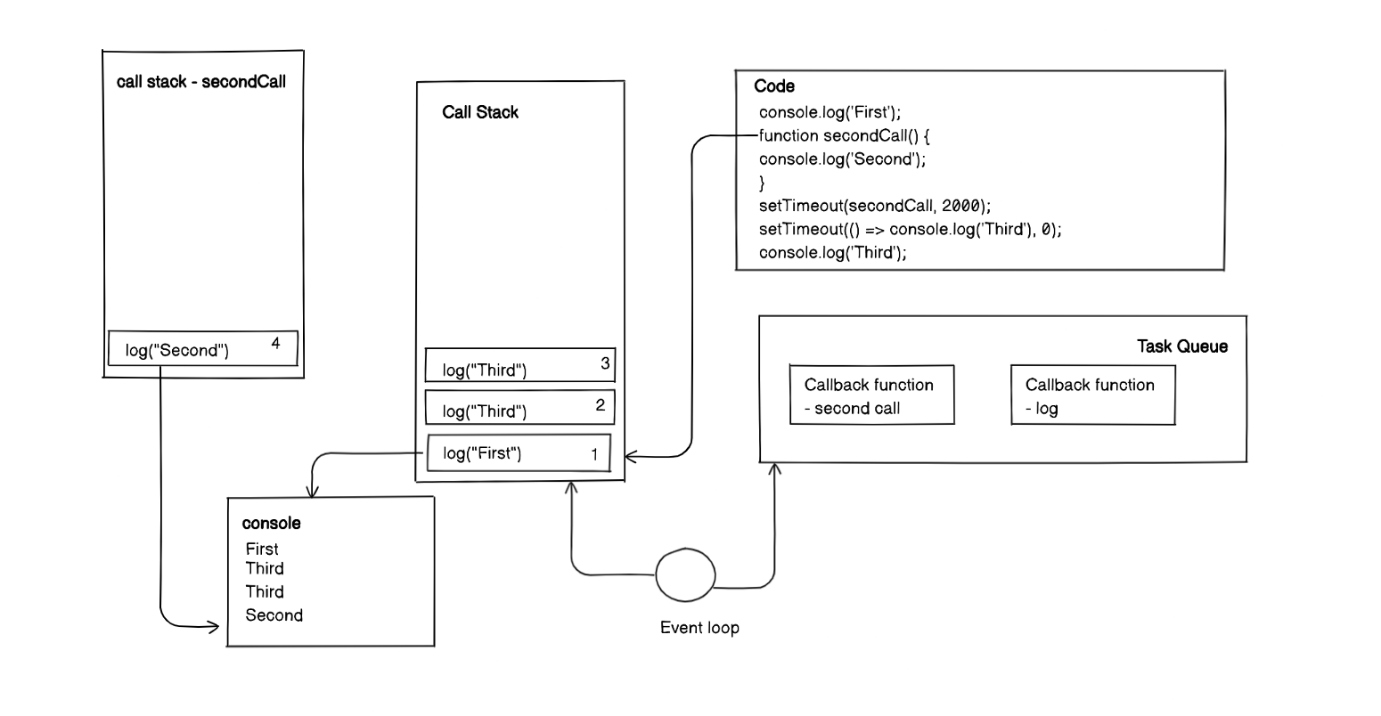
const person2 = {

  name: 'Jane'

};

const greetPerson2 = person.greet.bind(person2);

greetPerson2('Hello'); // Output: "Hello, Jane!"

1. Event bubbling and event capturing are two different phases of the event propagation process in JavaScript. When an event occurs on an element nested within other elements, the event can propagate through the DOM tree, triggering event handlers on different elements. Understanding event bubbling and event capturing can help in handling events effectively.
2. Function currying is a technique in JavaScript where a function with multiple arguments is transformed into a sequence of functions, each taking a single argument. It allows you to create new functions based on an existing function by partially applying the arguments.
3.   
   
4. Promises are a JavaScript feature introduced to handle asynchronous operations and make it easier to write asynchronous code in a more readable and manageable way. A promise represents the eventual completion (or failure) of an asynchronous operation and allows you to handle the result of that operation asynchronously. Promises have three different states:

* Pending: The initial state of a promise. It means that the asynchronous operation is still in progress and hasn't been fulfilled or rejected yet.
* Fulfilled: The state of a promise when the asynchronous operation is successfully completed. It means that the promise has resolved with a value.
* Rejected: The state of a promise when the asynchronous operation encounters an error or fails. It means that the promise has been rejected with a reason or an error.

Example:

function fetchData() {

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

    // Simulating an asynchronous operation

    setTimeout(() => {

      const data = 'Some data';

      if (data) {

        resolve(data); // Resolve the promise with the data

      } else {

 reject('Error: Data not found'); // Reject the promise with an error message

      }

    }, 2000);

  });

}

// Using the custom promise

fetchData()

  .then((data) => {

    console.log('Data:', data);

  })

  .catch((error) => {

    console.log('Error:', error);

  });

1. In JavaScript, this keyword refers to the object on which a function is being executed or the object that owns the currently executing code. The value of this depends on how a function is called and it can be different in different contexts.  
   Example:

 const person = {

 name: 'John',

 age: 30,

 greet: function() {

    console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old.`);

  }

};

// Method invocation

person.greet(); // Output: Hello, my name is John and I'm 30 years old.

1. JS event loop is a mechanism with which we can achieve concurrent programming in single level programming language which is javascript. Event loop keeps track of callstack as well the task queue, whenever it finds the task in task queue, it waits for the callback to be empty and when it finds it emply, it transfers the task from task queue to callback stack for execution.
2. Debouncing is a technique used in JavaScript to limit the frequency of a function's execution. It is particularly useful in scenarios where a function is triggered rapidly, such as during user interactions like scroll events, input events, or resize events. By debouncing a function, we ensure that it is only executed after a certain period of inactivity, reducing unnecessary function calls and improving performance.  
   <https://github.com/Zamzam1707/Placement-Assignment/tree/main/debouncing>
3. n JavaScript, a closure is a combination of a function and the lexical environment within which that function was declared. It allows a function to access variables from its outer (enclosing) scope even after the outer function has finished executing. In simpler terms, a closure gives a function the ability to remember and access variables from the scope in which it was defined.
4. !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

React Answers

1. React is a JavaScript library for building user interfaces. It offers a component-based approach, efficient rendering through a virtual DOM, and promotes code reusability. Its advantages include fast performance, a large community, and support for server-side rendering and mobile app development with React Native.
2. The Virtual DOM is a concept in React where a lightweight copy of the actual DOM (Document Object Model) is maintained. When there are updates to the state or props of components, React compares the virtual DOM with the real DOM and efficiently updates only the necessary parts.

Advantages of the Virtual DOM include increased performance by minimizing the number of actual DOM manipulations, making React faster and more efficient. It also simplifies development by providing a declarative programming model and allowing developers to work with components as building blocks. The Virtual DOM enables easier state management, enhances code reusability, and improves the overall user experience with smooth and responsive UI updates.

1. The lifecycle of React components refers to the different phases a component goes through, including mounting, updating, and unmounting. These phases provide hooks to execute code at specific points, allowing for initialization, rendering, handling updates, and cleanup operations, enhancing component functionality and behavior.
2. Functional components are simpler and rely on JavaScript functions to define the component. They don't have internal state or lifecycle methods. Class components, on the other hand, are defined as JavaScript classes and can have internal state, lifecycle methods, and more advanced features, making them suitable for complex components.
3. Hooks are functions introduced in React 16.8 that allow developers to use state and other React features in functional components. Hooks enable functional components to have local state, handle side effects, and access lifecycle methods. Hooks are specific to functional components and cannot be used in class components.
4. Lifecycle methods in React are special methods that are automatically called at different stages of a component's life. These methods include componentDidMount, componentDidUpdate, and componentWillUnmount, among others. They provide hooks to execute code at specific points, allowing for initialization, fetching data, handling updates, and cleanup operations.

Advantages of lifecycle methods include the ability to control component behavior and perform tasks at the right time. They facilitate data fetching and updating, handling external dependencies, managing animations, and integrating with third-party libraries. Lifecycle methods provide control over component rendering and enable efficient resource allocation, enhancing performance and ensuring a smooth user experience. However, with the introduction of hooks, some lifecycle methods are being replaced by more flexible and concise alternatives.

1. The useState hook in React is a built-in function that allows functional components to have state. It returns a state variable and a function to update that variable. By using the useState hook, developers can easily manage and update component state without using class components or lifecycle methods.

Advantages of the useState hook include simplifying state management, reducing boilerplate code, and improving component readability. It enables developers to handle state within functional components, promoting a more functional programming style. The useState hook also helps to avoid common pitfalls of shared state and enables easy testing and debugging. It is a key tool in building scalable and maintainable React applications.

1. The useEffect hook in React is used to perform side effects in functional components. It allows you to perform actions such as data fetching, subscriptions, or manually changing the DOM after the component has rendered. The useEffect hook takes a callback function as its first argument and an optional array of dependencies as its second argument.

Advantages of the useEffect hook include providing a unified way to handle side effects within functional components, simplifying the separation of concerns by keeping the side effect logic in one place. It allows you to manage subscriptions and clean them up when the component unmounts. The useEffect hook also provides a clear syntax for specifying dependencies and controlling when the effect should run. It enhances code readability and maintainability, making it easier to reason about the component's behavior.

1. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/context%20api>
2. The useReducer hook in React is a state management hook that offers an alternative approach to managing complex state logic within functional components. It takes a reducer function and an initial state as arguments and returns the current state and a dispatch function to trigger state updates.

Advantages of the useReducer hook include providing a more structured way to manage state that involves multiple actions and transitions. It helps to avoid prop drilling and simplifies the management of shared state. The useReducer hook promotes code modularity, reusability, and testability by separating the state transitions into a reducer function. It also integrates well with other React hooks, allowing for a more flexible and scalable approach to state management in React applications.

1. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/todo>
2. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/counter>
3. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/calculator>
4. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/Tic%20Tac%20Toe>
5. Prop drilling is a term used in React to describe the process of passing props through multiple layers of components that do not actually need the props themselves. This can result in unnecessary complexity and cluttered code.

To avoid prop drilling, you can use techniques like component composition, where components closer to the data source pass the necessary props to child components directly, or using React context to provide a central state accessible by all components without the need for prop drilling.

1. [https://github.com/Zamzam1707/Placement-Assignment/tree/main/task%20manager](https://github.com/prateek-budhiraja/placement-assignment_prateek-budhiraja/tree/main/task-manager)

Express Answers

1. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/simple%20get%20post>
2. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/middleware-post>
3. <https://github.com/Zamzam1707/Placement-Assignment/tree/main/blog%20api>
4. Authentication is the process of verifying the identity of a user or entity. It ensures that the user is who they claim to be. Authentication typically involves presenting credentials, such as a username and password, to prove identity. The goal of authentication is to establish trust and grant access to specific resources or functionalities within a system. Successful authentication results in the issuance of an authentication token or session, which is used to identify the user in subsequent requests. Whereas, Authorization, on the other hand, is the process of granting or denying access rights and permissions to authenticated users. Once a user's identity is authenticated, authorization determines what actions or resources the user is allowed to access or perform within the system. Authorization is based on a user's roles, permissions, or other attributes associated with their identity. It ensures that users can only access the resources or perform actions that they have been explicitly granted permission for.
5. CommonJS is a module system designed for server-side JavaScript environments, primarily used in Node.js. It provides a way to define and import modules using the require() function and export modules using the module.exports object. CommonJS modules use synchronous loading, meaning that the modules are loaded and executed synchronously, blocking the execution until the module is fully loaded. This module system is well-suited for server-side applications and environments where synchronous loading is acceptable.  
   EJS is a templating language that allows for dynamic HTML generation. It is typically used in web development to generate dynamic HTML content based on data. EJS templates are processed on the server-side to generate the final HTML, which is then sent to the client. EJS provides features like embedding JavaScript code within HTML templates using <% %> tags and rendering dynamic data using <%= %> tags. EJS does not have its own module system; it can be used in conjunction with any JavaScript module system, including CommonJS or ES modules.

1. We should hash our password before storing it in the database. Hashing is a one-way process that converts the password into a fixed-length string of characters. To make it more secure, we can add salt to our password and then we can hash it. So that it’ll be even more difficult for hackers to decrypt the passwords.
2. The event loop is a fundamental part of the Node.js runtime environment. It is responsible for handling and executing asynchronous operations in a single-threaded, non-blocking manner. The event loop enables Node.js to efficiently handle a large number of concurrent connections and I/O operations without blocking the execution of other code.  
   The compiler creates an event loop which keeps track of the call stack and the task queue, it keeps on checking both task queue and call stack and if there’s a task in the task queue and the call stack is empty, it pushes the operation from task queue

to call stack.