HTML/CSS

>> flex properties

>> sudo classes (active hover etc)

Active hover before after

>> nth child sudo classes

```
p:nth-child(odd) {
  background: red;
}

p:nth-child(even) {
  background: blue;
}
p:nth-child(3n+0) {
  background: red;
}
```

>> block element and inline element and difference

Block elements (takes full width from parent element and start with new line always**)**

<address><article><aside><blockquote><canvas><dd> <dl><dt><fieldset><figcaption><figure><footer><form><h1><h6><header><hr>
<main><nav><noscript>
<p><pre><section><table><tfoot>
<video>

Inline Element

(An inline element does not start on a new line and only takes up as much width as necessary.)

<a><abbr><acronym><b><bdo><big>
<button><cite><code><dfn><i><img><input><kbd><label><map><object><output><q><samp><script><select><small><sub>
<sup><textarea><time><tt><var>

>> transform properties

Transform-origin

Transform-style: preserve-2d / preserve-3d;

Transform: x/y

>> key frame

rule specifies the animation code

```
div {
  width: 100px;
  height: 100px;
  background: red;
  position: relative;
  animation: mymove 5s infinite;
}

@keyframes mymove {
```

```
  from {top: 0px;}
  to {top: 200px;}
}
```

After animation get completed it will back to old position

>> media queries

>> browser friendly queries

>> disply none/ visibility hidden

Display none hides from browser and space is occupied by next

Visibility hidden hides from browser but space not get freed

>> box model (Content padding border margin***)***

```
div {
  width: 300px;
 border: 5px solid green;
 padding: 50px;
margin: 20px; // not get added in width
}
```

What is total width of div ?

o/p : (300 'width' + 50 'pading-left' + 50 'padding-right'+ 5 'border-left' + 5 'border-right') = 410px

---

>> svg/canvas

>> api in html

https://www.w3schools.com/html/html5_geolocation.asp

geolocation

drag/drop

web storage (local storage/ session storage)

>> drag drop

Call preventDefault() to prevent the browser default handling of the data (default is open as link on drop)

Get the dragged data with the dataTransfer.getData() method. This method will return any data that was set to the same type in the setData() method

The dragged data is the id of the dragged element ("drag1")

Append the dragged element into the drop element

>> localstorage example

```
// Store
localStorage.setItem("lastname", "Smith"); or
localStorage.lastname = "Smith";
```

```
  //remove
  localStorage.removeItem("lastname");
```

>> sessionstorage example

```
        // Store
        sessionStorage.setItem("lastname", "Smith");
        // read
        var lastname = sessionStorage.getItem("key");
        // remove
        sessionStorage.removeItem("key");
        //remove all
        sessionStorage.clear();
```

>> HTML media ? how to add source of audio video

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">.
Any text displayed here
</video>

<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Any text displayed here
</audio>
```

The controls attribute adds audio controls, like play, pause, and volume.
The <source> element allows you to specify alternative audio files which the browser may choose from.
The browser will use the first recognized format.
The text between the <audio> and </audio> tags will only be displayed in browsers that do not support the <audio> element.

Plug in: <object> tag or the <embed> tag plug-ins (like Java applets, PDF readers, Flash Players) in web pages.
Youtube: iframe

>> What are Semantic Elements?

A semantic element clearly describes its meaning to both the browser and the developer.
Examples of non-semantic elements: and - Tells nothing about its content.
Examples of semantic elements: <form>, <table>, and <article> - Clearly defines its content.

JS AND REACT

>> Strict in js

"use strict"; Defines that JavaScript code should be executed in "strict mode".

>> Difference between async/await

In summary, async/await is a cleaner syntax to write asynchronous Javascript code. It enhances readability and flow of your code. Things to keep in mind while using async/await :

Async functions return a promise.

Await can only be used inside an async block.

async functions use an implicit Promise to return its result. Even if you don't return a promise explicitly async function makes sure that your code is passed through a promise. ...

await only blocks the code execution within the async function. It only makes sure that next line is executed when the promise resolves.

Every function that returns a promise can be considered as async function .

await is used for calling an async function and waits for it to resolve or reject . await blocks the execution of the code within the async function in which it is located.

await is used for calling an async function and wait for it to resolve or reject . await blocks the execution of the code within the async function in which it is located.

If the output of function2 is dependent on output of function1 then use await .

>> Difference between reduce/filter/map

The filter() method filters the elements from current set while

the find() method searches child elements of selected element. Suppose we want to use filter() method to get all the “<li>” elements having class “findme“.

>> event loop

The **Event Loop** has one simple job — to monitor the Call Stack and the Callback Queue. If the Call Stack is empty, it will take the first event from the queue and will push it to the Call Stack, which effectively runs it. Such an iteration is called a tick in the Event Loop. Each event is just a function callback.

>> call, apply, bind

The bind method creates a copy of the function and sets the this keyword,

while the call and apply methods sets the this keyword and calls the function immediately.

Use .bind() when you want that function to later be called with a certain context, useful in events. Use .call() or .apply() when you want to invoke the function immediately, and modify the context.
apply is similar to call except that it takes an array-like object instead of listing the arguments out one at a time:

- Call invokes the function and allows you to pass in arguments one by one.
- Apply invokes the function and allows you to pass in arguments as an array.
- Bind returns a new function, allowing you to pass in a this array and any number of arguments.

Call

```
var person1 = {firstName: 'Jon', lastName: 'Kuperman'};
var person2 = {firstName: 'Kelly', lastName: 'King'};

function say(greeting) {
   console.log(greeting + ' ' + this.firstName + ' ' + this.lastName);
}

say.call(person1, 'Hello'); // Hello Jon Kuperman
say.call(person2, 'Hello'); // Hello Kelly King
```

Apply

```
var person1 = {firstName: 'Jon', lastName: 'Kuperman'};
var person2 = {firstName: 'Kelly', lastName: 'King'};

function say(greeting) {
   console.log(greeting + ' ' + this.firstName + ' ' + this.lastName);
}

say.apply(person1, ['Hello']); // Hello Jon Kuperman // pass in arguments as an array
say.apply(person2, ['Hello']); // Hello Kelly King
```

Bind

```
var person1 = {firstName: 'Jon', lastName: 'Kuperman'};
var person2 = {firstName: 'Kelly', lastName: 'King'};

function say() {
   console.log('Hello ' + this.firstName + ' ' + this.lastName);
}

var sayHelloJon = say.bind(person1);
var sayHelloKelly = say.bind(person2);

sayHelloJon(); // Hello Jon Kuperman
sayHelloKelly(); // Hello Kelly King
```

>> map, foreach

>> filter, find, findindex

findIndex

for empty array returns -1

```
var ages = [3, 10, 18, 20];
function checkAdult(age) {
 return age >= 20;
}
console.log(ages.findIndex(checkAdult));
```

o/p: 3 // [0, 1,2, 3] 20 is at 3rd index

Filter

```
var ages = [3, 10, 18, 20];
function checkAdult(age) {
            return age >= 10;
}
console.log(ages.filter(checkAdult));
```

o/p: 10,18,20 // returns all element

---

find

find() returns the value of that array element (and does not check the remaining values)

Otherwise it returns undefined

find() does not change the original array.

```
var ages = [3, 10, 18, 20];
function checkAdult(age) {
 return age >= 20;
}
console.log(ages.find(checkAdult));
```

o/p: 20 // 20 is at 3rd index

>> promise

>> Write program for digital clock

>> event bubbling and event capturing

>> debouncing and throttling (flipkart)

>>call by value call by reference

>> setTimeout and setInterval

>> lifecycle methods

Constructor
getDerivedStateFromProps() // execute in both phase state/prop get changed or not
shouldComponentUpdate // after state change
componentWillUpdate
render // both
componentDidUpdate // after state change
componentDidMount // at first/ initial render

>> getDerivedstatromprops

>> setState() in componentWillMount()

.componentWillMount() is invoked immediately before mounting occurs. It is called before render(), therefore setting state in this method will not trigger a re-render. Avoid introducing any side-effects or subscriptions in this method.

.setState is asynchronous and componentWillMount not good for asynchronous call as it not get re-rendered

---

>> controlled and uncontrolled component

In a controlled component, form data is handled by a React component.
A Controlled Component **takes its current value through props and notifies changes through callbacks**.
A parent component "controls" it by handling the callback and managing its own state and passing the new values as props to the controlled component.

The alternative is uncontrolled components, where form data is handled by the DOM itself.
To write an uncontrolled component, instead of writing an event handler for every state update, you can use a ref to get form values from the DOM.

>> main features of ES6

- arrow function,
  - No scope for this keyword
  - No return method
  - Arguments objects
  - Best for callback method or reduce map
- Destructuring
- spread operator
- classes

>> What replace componentDidMount in functional component?

useEffect replace componentDidMount and componentDidUpdate

```
import React, { useState, useEffect } from 'react';
function Example() {
  const [count, setCount] = useState(0);
  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
```

```
  </div>
 );
}
```

>> how to validate props

>> slice splice

>> difference between constructor, getDerivedState, getInitialState

- **getInitialState:**
  - only invoked when the component is first created.
  - initial state is passed as an argument to useState; useState([]) in functional component

getInitialState is used with React.createClass in ES5 and
constructor is used with React.Component in ES6.

>> virtual DOM

>> setState is asynchronous why?

This is because setState alters the state and causes rerendering.
This can be an expensive operation and making it synchronous might leave the browser unresponsive.
Thus the setState calls are asynchronous as well as batched for better UI experience and performance.

>> Is react synchronous?

Short answer to your question is - NO, react doesn't have sync method setState .

>>difference between functional component and class component

>> Pure component

A Pure component can replace a component that only has a render function. Instead of making a full-blown component just to render some content to the screen, we can create a pure one instead. Pure components are the simplest, fastest components we can write.

>>hooks

React Hooks are functions that let us hook into the React state and lifecycle features from function components. By this, we mean that hooks allow us to easily manipulate the state of our functional component without needing to convert them into class components.

React to correctly preserve the state of Hooks between multiple useState and useEffect calls.

Why hooks?

it reduces the complexity of state management

“Hooks are a new addition to React in version 16.8 that allows you use state and other React features, like lifecycle methods, without writing a class.” ... Hooks let you always use functions instead of having to constantly switch between functions, classes, higher-order components, and render props.

>>life cycle hooks

>> redux

>> prop drlling / context api

>> React memo

React. memo is a higher order component. ... If your function component renders the same result given the same props, you can wrap it in a call to React. memo for a performance boost in some cases by memoizing the result. This means that React will skip rendering the component, and reuse the last rendered result.

What does useCallback/useMemo do in React? As said in docs, useCallback Returns a memoized callback. Pass an inline callback and an array of inputs. useCallback will return a memoized version of the callback that only changes if one of the inputs has changed.

What is the difference between createElement and cloneElement?

createElement is what JSX gets compiled to and is what React uses to create React Elements (object representations of some UI). cloneElement is used to clone an element and pass it new props. They nailed the naming on these two

---

>> debounce and throttle

The main difference between throttling and debouncing is that
throttling executes the function at a regular interval,
while debouncing executes the function only after some cooling period.
Debouncing and throttling are not something provided by JavaScript itself.

Let’s consider an example of a search bar on a website.

Every time you type something in the search bar, it makes an API call to fetch the data from the server on the basis of the letters typed in the search bar.

Programs

>> program for palindrome (eye === eye)

```
var x = 'eye';
var y = x.split('').reverse().join('');
if(x === y)
console.log('This is a palindrome value');
o/p 121
```

```
console.log("One");
setTimeout(() => { console.log("Two"); }, 500);
console.log("Three");
```

output:
One
Three
Two

---let

```
myFirstPromise = new Promise((resolve, reject) => {
   setTimeout(function(){
      resolve("Success!"); // fulfilled
   }, 4000);
});

myFirstPromise.then((successMessage) => {
   console.log("Yay! " + successMessage);
});
```

o/p
Yay! Success!

---***>> reverse each other word of string***

```
function ReverseString(str) {
   const revArray = [];
  wordArray = str.split(' ');
   const length = wordArray.length;
   for(let i = length; i >= 0; i--) {
         if(i%2 != 0 && wordArray[i] !== undefined ){
         reverseString = (wordArray[i].split("").reverse().join(""));
```

```
            revArray.push(reverseString);
            }
            else{
            revArray.push(wordArray[i]);
            }
    }
    return revArray.reverse().join(' '); // riversing array and join with spaces
}
document.write(ReverseString("I really dont like revesing strings!")) ;
```

o/p
I yllaer dont ekil revesing !sgnirts

>> max three integer sum

```
function maxTriSum(array_values) {
 var my_array=array_values;
    document.getElementById("my_array").innerHTML = my_array;
            var new_array = [];
            var max = my_array.reduce(function(a, b) {
             for(var i in my_array) {
                        let nnn = (Math.max(a, b));
                                new_array.push(nnn);
                        }
                        var uniqueArray1 = new Set(new_array);
                        const abc1 = [...uniqueArray1];

                        let sum = abc1.flat().reduce((p, c) => p + c, 0);
                        document.getElementById("result").innerHTML = sum;
    return Math.max(a, b);
            });
}
 maxTriSum([-7,12,-7,29,-5.0,-7,0,0,29]);
```

o/p
41

```
let x = this;
console.log(x);
```
//o/p in console window object with parameter
>window{}

```
console.log(x);
let x = 10; o/p: reference error
```

```
console.log(x);
var  x = 10; o/p: undefined
```

```
const x = [1,2,3];
console.log(x.indexOf(1000)); //o/p: -1
```

```
const x = [1,2,3];
x[-1] =  60;
console.log(x[x.indexOf(1000)]); o/p: 60
```

```
const x = [1,2,3];
x[-1] =  030;
console.log(x[x.indexOf(1000)]); o/p: 24
```

```
const x = [1,2,3];
x[4] =  30;
console.log(x[x.indexOf(1000)]); o/p: [1,2,3,,30]
```