Client: Infosys

Prototype/ Virtual DOM/ server-side rendering vs. client-side rendering/ One way data flow/ Props vs. State/ Higher Order Component/ Redux/ Reducer/
Higher Order Component and advantage/ Props vs. State/ Second parameter in setState/ callback/ responsive design media query/ jest/ Webpack

Client: **marathonfund.com**

1. When you click a button how to hide the corresponding content? (you can use Javascript or jQuery)
2. To write a react component? (no requirement)
3. Do you know sqlServer?
4. Introduce yourself? Recent project?
5. Do you have questions to ask them?

Client: Vitusa

1. What is Webpack? How to use it?
2. Have you used react-router? How to use it?
3. Introduce yourself?

Client: Citco

1. Double equal and '==='? difference
2. Null and undefined?
3. Null == 'undefined' ?
4. Null === undefined?
5. Recent project? Detail?
6. AWS?
7. Story point? Jira?
8. Database questions? Cartesian product? What's the draws to add indexes to table?
9. Docker?
10. How to merge in git?
11. Why javascript called javascript?

Apple(Jamie):

1. Introduce yourself, recent project?
2. In css naming convention?
3. How to implement an autocomplete search bar? Debounce vs throttling?
4. How to add accessibility?
5. How to handle a serial async? async and await
6. Biggest challenge that you met?
7. What kind of work you want to find? What do you want to learn in the new job?
8. How to improve the apple assignment that you did?

Apple(Jamie):

1.In css naming convention

Teams have different approaches to writing CSS selectors. Some teams use hyphen delimiters, while others prefer to use the more structured naming convention called BEM.
Generally, there are 3 problems that CSS naming conventions try to solve:
To know what a selector does, just by looking at its name
To have an idea of where a selector can be used, just by looking at it
To know the relationships between class names, just by looking at them

```
.nav--secondary {
  ...
}
.nav__header {
  ...
}
```

That is the BEM naming convention.
B stands for 'Block'
E for element
M for Modifiers

```
.stick-man__head--small {
}

.stick-man__head--big {
}
```

https://medium.freecodecamp.org/css-naming-conventions-that-will-save-you-hours-of-debugging-35cea737d849

2. How to implement an autocomplete search bar? Debounce vs throttling?
What is Auto-complete ?
Whenever you go to google and start typing, a drop-down appears which lists the suggestions. Those suggestions are related to the query and help the user in completing his query.
Auto-complete, or word completion, is a feature in which an application predicts the rest of a word a user is typing
Auto-complete helps users saving time while typing and minimizes chances for spelling mistakes which result to a better customer experience overall, therefore it is essential for Traveloka to have a high quality auto-complete search.
Those suggestions can be clicked or arrowed through and pressing enter will complete the word for the user.
Auto-Completion can be implemented by using any database. In this post, we will use Elasticsearch to build auto-complete functionality.
Auto-complete can be implemented any number of ways in regard to how the suggestions are filtered and presented and for this article we are going to use a fixed list of suggestions passed to our component. As the user types, we will filter the results and only show the suggestions that contain the user’s input anywhere in the suggestion.

3.How It Works

Autocomplete is performed by two components:
Client, which sends requests with a partial query term. This is the part you need to code, and this page shows you how.
Server, which responds with a list of matches for the partial query string. Searchify provides this part. If you're curious about how the server handles data, see API Specification. The client and server exchange data in JSON/ JSONP format. This is handled through jQuery's AJAX support.
What a throttle does is that it triggers predictably after a certain time. Every time. Basically, it's it prevents excessive or repeated calling of another function but doesn't get reset.
Debouncing, unlike throttling, guarantees that a function is only executed a single time, either at the very beginning of a series of calls, or at the very end

4.How to add accessibility?

A search bar should be set up in a form as paired label and input.
Using a submit button to search decreases the number of keystrokes necessary to use the form.
Always include an ARIA role='search' somewhere on the form or fieldset.
ARIA (Assistive Rich Internet Applications), is a spec from the World Wide Web Consortium (W3C) that was created to improve accessibility of web pages and applications by providing extra information to screen readers via HTML attributes.

5.How to handle a serial asynchronous?

asynchronous and await
Asynchronous functions are essentially a cleaner way to work with asynchronous code in JavaScript. In order to understand exactly what these are, and how they work, we first need to understand Promises.

What is Async/Await?

The newest way to write asynchronous code in JavaScript.
It is non blocking (just like promises and callbacks).
Async/Await was created to simplify the process of working with and writing chained promises.
Async functions return a Promise. If the function throws an error, the Promise will be rejected. If the function returns a value, the Promise will be resolved.

6.What kind of work you want to find? What do you want to learn in the new job?

I have used react for two years to do the projects, now I want to learn new technology like Vue or Angular to build applications. And I want to do some backend also.

7.How to improve the apple assignment that you did?

In the assignment, I think I can improve like to improve the performance like to add componentdidMount, and add new reusable Components, like small button. When it grows, it's easier for it to integrate with other components.

setState process is not necessarily synchronous.

Apple: (Amy)

1 HTML5 and previous HTML

2 Difference between Cookie and LocalStorage

3 Anonymous functions and when you use it?

4 How to Create Object in JavaScript?

5 Event bubbling?

6 CSS attributes? What's the difference?

7 Padding and margin?

8 Closure?

9 ES6 in Jquery?

10 What happens after setState in React?

11 Login authentication.

Apple: (vendor)

Closure?

Hoisting?

Event propagation? Event bubbling and event propagation?

Strict mode?

ISO format?

Apple: (client)

Coding

To write the code that can pair the brackets, the paired brackets have the same color.

The link is one method (not good enough)
https://jsfiddle.net/amyzhang/wh83sqr6/

Apple: (client)

1. Recent project and the work flow like jira scrum
2. Unit test general idea not to write a unit test
3. Coding to select between css and javascript
4. Javascript coding

(1) https://jsfiddle.net/p106usw8/

(2) https://jsfiddle.net/amyzhang/qwr4dbfz/

JavaScipt:

For Two weeks, we will have a planning meeting, to talk about what work need to be done and how to finish them, and to split the work into small pieces and choosing the right work items. Every day our team leader will assign works to us, and for the everyday stand up meeting, we will talk about what did we do yesterday, what will we do today, and is these some problem?

Scrum is a framework that helps teams work together. Often thought of as an agile project management framework, Scrum describes a set of meetings, tools, and roles that work in concert to help teams structure and manage their work.

Sprint planning is a collaborative event where the team answers two basic questions: What work can get done in this sprint and how will the chosen work get done?

Choosing the right work items for a sprint is a collaborative effort between the product owner, scrum master, and development team. The product owner discusses the objective that the sprint should achieve and the product backlog items that, upon completion, would achieve the sprint goal.

The team then creates a plan for how they will build the backlog items and get them “Done” before the end of the sprint. The work items chosen and the plan for how to get them done is called the sprint backlog. By the end of sprint planning the team is ready to start work on the sprint backlog, taking items from the backlog, to “In-progress,” and “Done."

During a sprint, the team checks in during the daily scrum, or standup, about how the work is progressing. The goal of this meeting is to surface any blockers and challenges that would impact the teams ability to deliver the sprint goal.

After a sprint, the team demonstrates what they’ve completed during the sprint review. This is your team’s opportunity to showcase their work to stakeholders and teammates before it hits production.

Agile

A user story is the smallest unit of work in an agile framework. It’s an end goal, not a feature, expressed from the software user’s perspective. User stories are a few sentences in simple language that outline the desired outcome.

Stories fit neatly into agile frameworks like scrum and kanban. In scrum, user stories are added to sprints and “burned down” over the duration of the sprint.

Epics are large work items broken down into a set of stories, and multiple epics comprise an initiative.

A story point is a unit of effort required to complete a task / story, relative to other estimates. 2 hours

JavaScript

1.Event Bubbling

Event bubbling and capturing are two ways of event propagation in the HTML DOM API, when an event occurs in an element inside another element, and both elements have registered a handle for that event. The event propagation mode determines in which order the elements receive the event.

With bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.

With capturing, the event is first captured by the outermost element and propagated to the inner elements.

```
JavaScript
var logElement = document.getElementById('log');  JavaScript

function log(msg) {
   logElement.innerHTML += ('<p>' + msg + '</p>');
}

function capture() {
    log('capture: ' + this.firstChild.nodeValue.trim());
}

function bubble() {
    log('bubble: ' + this.firstChild.nodeValue.trim());
}

var divs = document.getElementsByTagName('div');
for (var i = 0; i < divs.length; i++) {
    divs[i].addEventListener('click', capture, true);
    divs[i].addEventListener('click', bubble, false);
}

HTML
<div>1
    <div>2
        <div>3
            <div>4
                <div>5</div>
            </div>
        </div>
    </div>
</div>
<section id="log"></section>

CSS
p {
    line-height: 0;
}

div {
    display:inline-block;
    padding: 5px;

    background: #fff;
    border: 1px solid #aaa;
    cursor: pointer;
}

div:hover {
    border: 1px solid #faa;
    background: #fdd;
}
```

2.Promises

To make asynchronous event handling easier, promises were introduced in JavaScript in ES6.

A promise is an object that handles asynchronous data. A promise has three states:

pending : when a promise is created or waiting for data.

fulfilled : the asynchronous operation was handled successfully.

rejected : the asynchronous operation was unsuccessful.

The great thing about promises is that once a promise is fulfilled or rejected, you can chain an additional method to the original promise.

(A promise is an object that wraps an asynchronous operation and notifies when it’s done. This sounds exactly like callbacks, but the important differences are in the usage of Promises. Instead of providing a callback, a promise has its own methods which you call to tell the promise what will happen when it is successful or when it fails. The methods a promise provides are “then(…)” for when a successful result is available and “catch(…)” for when something went wrong.)

The Promise.all(iterable) method returns a single Promise that resolves when all of the promises in the iterable argument have resolved or when the iterable argument contains no promises. It rejects with the reason of the first promise that rejects.

```
var promise1 = Promise.resolve(3);
var promise2 = 42;
var promise3 = new Promise(function(resolve, reject) {
  setTimeout(resolve, 100, 'foo');
});

Promise.all([promise1, promise2, promise3]).then(function(values) {
  console.log(values);
});
// expected output: Array [3, 42, "foo"]
```

The Promise.race(iterable) method returns a promise that resolves or rejects as soon as one of the promises in the iterable resolves or rejects, with the value or reason from that promise.

which promise returns faster, which promise will be returned!!!

```
Promise.race = function(promises) {
    return new Promise(function(resolve, reject) {
        for(let i = 0; i < promises.length; i++) {
            promises[i].then(resolve, reject);
        }
    })
}
var promise1 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 500, 'one');
});

var promise2 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 100, 'two');
});
```

```
Promise.race([promise1, promise2]).then(function(value) {
  console.log(value);
  // Both resolve, but promise2 is faster
});
// expected output: "two"
```

3. Ajax

AJAX stands for Asynchronous JavaScript and XML. The benefits such as updating the web page without reloading the entire page, requesting data from the server and also sending data to the server. It is a new technique to communicate to and from a server/ database without completely refreshing the page. It creates faster, interactive, better web applications with help of CSS, XML, HTML, and JavaScript.

What is XMLHttpRequest object used for?

It exchanges data with a server behind the scenes.

With the XMLHttpRequest object you can update parts of a web page, without reloading the whole page.

You can update a web page without reloading the page, request data from a serer after the page has loaded, receive data from a server after the page has loaded, send data to a server in the background.

List some advantages and disadvantages of using Ajax.

Ajax is a very easy concept if one has a sound knowledge of JavaScript. It uses JavaScript functions to call methods from a web service. It has certain advantages-

Advantages

· Speed- Ajax reduces the server traffic and also the time consumption on the server and client side.
· Ajax is very responsive and fast, data can be transferred at a time.
· XMLHttpRequest plays a significant role in Ajax. It is a special JavaScript object that calls asynchronous HTTP request to the server for transferring data.
· One of the biggest advantages of using Ajax as forms are common elements in the web page. Ajax gives options for validation and much more.
· One doesn't have to completely reload the page.

Explain limitations of Ajax.

1) Browser Integration

The dynamically created page does not register itself with the browser history engine, so triggering the "Back" function of the users' browser might not bring the desired result.

2) Response-time Concerns

Network latency – or the interval between the user request and server response – need to be considered carefully during Ajax development.

3) Search Engine Optimization (SEO)

Websites that use Ajax to load data which should be indexed by search engines must be careful to provide equivalent Sitemaps data at a public, linked URL that the search engine can read, as search engines do not generally execute the JavaScript code required for Ajax functionality.

4) Reliance on JavaScript

Ajax relies on JavaScript, which is often implemented differently by different browsers or versions of a particular browser. Because of this, sites that use JavaScript may need to be tested in multiple browsers to check for compatibility issues.

Differentiate between Synchronous and Asynchronous Ajax requests.

Synchronous Ajax requests: In this, the script stops and waits for the server to reply before continuing. In the web application world, one has to happen after the other, i.e. the interaction between the customer and the server is synchronous. Synchronous is not recommended as it blocks the page until the response is received from the server.

Asynchronous Ajax requests handle the reply as and when it comes and allows the page to continue to be processed. Under Asynchronous, if there is any problem in the request it can be modified and recovered. The request doesn't block the client as the browser is responsive. The user can perform other operations as well.

4.Call Stack

JavaScript is a single-threaded programming language, therefore it can do one thing at a time, which means it has a single Call Stack.

The Call Stack is a data structure which records basically where in the program we are.

Example1:

```
function greeting() {
  // [1] Some codes here
  sayHi();
  // [2] Some codes here
}
function sayHi() {
  return "Hi!";
}

// Invoke the `greeting` function
greeting();

// [3] Some codes here
```

The code above would be executed like this:

1. Ignore all functions, until it reaches the `greeting()` function invocation.
2. Add the `greeting()` function to the call stack list.

Call stack list:
- greeting

3. Execute all lines of code inside the `greeting()` function.

4. Get to the sayHi() function invocation.
5. Add the sayHi() function to the call stack list.

Call stack list:
- greeting
- sayHi

6. Execute all lines of code inside the sayHi() function, until reaches its end.
7. Return execution to the line that invoked sayHi() and continue executing the rest of the greeting() function.
8. Delete the sayHi() function from our call stack list.

Call stack list:
- greeting

9. When everything inside the greeting() function has been executed, return to its invoking line to continue executing the rest of the JS code.
10. Delete the greeting() function from the call stack list.

Call stack list:
EMPTY

Example2:

```
1. console.log('a');
2. setTimeout(() => console.log('b'), 0);
3. console.log('c');
```

What will be displayed?

What happens here?

```
'a', 'c', 'b'
```

1 is pushed to the stack, executed, and popped off from the stack.

browser sees asynchronous code 2, and pushes it to the web api. Web api decides when async code is ready, and adds it to the event queue.

3 is pushed to the stack, executed and popped off from the stack.

event loop sees there is nothing in the stack anymore, and pushes 2 from the event queue onto the stack.

1 -> call stack, executes. 2 -> call stack, web api, event queue. 3 -> call stack, executes. 2 -> event queue, call stack, executes

5 What is closure? When to use closure?

A closure is a function that is defined inside another function and has access to the variable which is declared and defined in parent function scope

The closure has access to the variable in three scopes:

Variable declared in his own scope

Variable declared in parent function scope

Variable declared in the global scope

entries

Example:

```
var globalVar = "abc";

// Parent self invoking function
(function outerFunction (outerArg) { // begin of scope outerFunction
  // Variable declared in outerFunction function scope
  var outerFuncVar = 'x';
  // Closure self-invoking function
  (function innerFunction (innerArg) { // begin of scope innerFunction
    // variable declared in innerFunction function scope
    var innerFuncVar = "y";
    console.log(
      "outerArg = " + outerArg + "\n" +
      "outerFuncVar = " + outerFuncVar + "\n" +
      "innerArg = " + innerArg + "\n" +
      "innerFuncVar = " + innerFuncVar + "\n" +
      "globalVar = " + globalVar);
  // end of scope innerFunction
  })(5); // Pass 5 as parameter
// end of scope outerFunction
})(7); // Pass 7 as parameter
```

Output would be:

Give an example of closure keeping its context.

```
outerArg = 7
outerFuncVar = x
innerArg = 5
innerFuncVar = y
globalVar = abc
const createAdd = () => {
  let counter = 0;
  return () => {
    console.log(counter)
    counter = counter + 1
  }
}

add = createAdd();

add(); //=> 0
add(); //=> 1
add(); //=> 2
```

After createAdd() executed, normally we would expect that the entirety of the inner scope of createAdd() would go away, because we know that the V8 engine employs a Garbage Collector that comes along and frees up memory once it's no longer in use.

Since inner scope of createAdd is no longer in use, it would seem natural that counter should be considered gone.

But the "magic" of closures does not let this happen. That inner scope is in fact still "in use", and thus does not go away.

Who is using it? add function.

.What is Colusure

A "closure" is an expression (typically a function) that can have free variables together with an environment that binds those variables (that "closes" the expression).

Private variables can be made possible with closures. The nested function can access the variable in the parent scope. This is called a JavaScript closure. It makes it possible for a function to have private variables. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

6 What is JSON? For what is used for?

JSON (JavaScript Object Notation) is a data storage and communication format based on key-value pair of JavaScript object literals. It is a lightweight text-based open standard designed for human-readable data interchange which is derived from the JavaScript programming language for representing simple data structures and associative arrays, called objects.

In JSON

· all property names are surrounded by double quotes.
· values are restricted to simple data: no function calls, variables, comments, or computations.

JSON is used for communication between javascript and serverside technologies.

6.1 How to convert Javascript objects into JSON?

JSON.stringify(value); is used to convert Javascript objects into JSON.

Example Usage:`var obj={website:"Onlineinterviewquestions"}; JSON.stringify(obj); // '{"website":"Onlineinterviewquestions"}'`

6.2 List types Natively supported by JSON?

JSON supports Objects, Arrays, Primitives (strings, numbers, boolean values (true/false), null) data types.

6.3 How do you decode a JSON string?

Use JSON.parse method to decode a JSON string into a Javascript object.

```
var json='{"website":"Onlineinterviewquestions"}' JSON.parse(json); //
{website:"Onlineinterviewquestions"}
```

7 What are design patterns in javascript?

The design patterns in question include the following:

Module

Prototype

observer

Singleton

https://scotch.io/bar-talk/4-javascript-design-patterns-you-should-know#undefined

8 How to tell an array is an array not an object?

Array.isArray()   ---- true

Object.prototype.toString.call()   ----“[object Array]”;

9 Array.double?

Array.prototype.double = function() { return this.map(item => item*2); }

9 Currying

Currying is the process of taking a function with multiple arguments and turning it into a sequence of functions each with only a single argument.

Function Currying, also known as partial function application, is the use of a function (that accept one or more arguments) that returns a new function with some of the arguments already set. The function that is returned has access to the stored arguments and variables of the outer function.

This works because JS supports closures.

```
const notCurry = (x, y, z) => x + y + z; // a regular function

const curry = x => y => z => x + y + z; // a curry function
```

10 ES6

There were so many great features added to help JavaScript developers that include:

· new keywords like let and const to declare variables,
· new function syntax using Arrow functions,
· creation of Classes,
· parameters with default values,
· promises for asynchronous actions,
· and many more!

1 Rest Parameter

Extended Parameter Handling

Aggregation of remaining arguments into single parameter of variadic functions.

```
ECMAScript 6 — syntactic sugar: reduced | traditional
function f (x, y, ...a) {
    return (x + y) * a.length
}
f(1, 2, "hello", true, 7) === 9//true
ES5
function f (x, y) {
    var a = Array.prototype.slice.call(arguments, 2);
    console.log(a); //["Hello", true, 7]

    return (x + y) * a.length;
};
f(1, 2, "hello", true, 7) === 9; //true
```

2 Spread Operator

Spreading of elements of an iterable collection (like an array or even a string) into both literal elements and individual function parameters.

```
var params = [ "hello", true, 7 ]
var other = [ 1, 2, ...params ] // [ 1, 2, "hello", true, 7 ]

function f (x, y, ...a) {
    return (x + y) * a.length
}
f(1, 2, ...params) === 9

var str = "foo"
var chars = [ ...str ] // [ "f", "o", "o" ]
```

· Destructurig

Destructuring simply implies breaking down a complex structure into simpler parts. In JavaScript, this complex structure is usually an object or an array. With the destructuring syntax, you can extract smaller fragments from arrays and objects. Destructuring syntax can be used for variable declaration or variable assignment. You can also handle nested structures by using nested destructuring syntax.

```
const student = {
    name: 'John Doe',
    age: 16,
    scores: {
        maths: 74,
        english: 63,
        science: 85
    }
};

function displaySummary({ name, scores: { maths = 0, english = 0, science = 0 } }) {
    console.log('Hello, ' + name);
    console.log('Your Maths score is ' + maths);
    console.log('Your English score is ' + english);
    console.log('Your Science score is ' + science);
const student = {
    firstname: 'Glad',
    lastname: 'Chinda',
    country: 'Nigeria'
};

// Object Destructuring
const { firstname, lastname, country } = student;

console.log(firstname, lastname, country); // Glad Chinda Nigeria
```

3 Why use arrow function?

In React, we usually use arrow function as the event handler. We are using arrow function so that we do not need to bind this keyword in the constructor any more.

What is the disadvantage of using arrow function? Arrow function can not be used as a constructor because it doesn't have prototype property.

11. Hoisting

Conceptually, for example, a strict definition of hoisting suggests that variable and function declarations are physically moved to the top of your code, but this is not in fact what happens. Instead, the variable and function declarations are put into memory during the compile phase, but stay exactly where you typed them in your code.

variable declaration are hoisted to the top of the function (if defined in a function), or the top of the global context, if outside a function.

In JS, a variable can be declared after it has been used. Initialization are not hoisted.

```
console.log(num); // Returns undefined
var num;
num = 6;
```

If you declare the variable after it is used, but initialize it beforehand, it will return the value:

```
num = 6;
console.log(num); // returns 6
var num;
```

Function hoisting only occurs for function declarations, not function expressions.

```
    definitionHoisted();// Outputs: "Definition hoisted!"

    definitionNotHoisted();// TypeError: undefined is not a function

    function definitionHoisted() {

    console.log("Definition hoisted!");

}

    var definitionNotHoisted = function () {

    console.log("Definition not hoisted!");

};

        funcName();// ReferenceError: funcName is not defined

        varName();// TypeError: undefined is not a function

        var varName = function funcName() {

            console.log("Definition not hoisted!");

};
```

12 call, bind, apply

(1)

call and apply invoke the function immediately .

bind returns a new function with this preset.

call - comma (takes in comma-separated arguments),

apply - array (takes in array of arguments),

bind - a bit different (returns a new function instead. comma-separated arguments).

```
call(self, a, b)
apply(self, [a, b])
```

We use the bind () method primarily to call a function with the this value set explicitly. In other words, bind () allows us to easily set which specific object will be bound to this when a function or method is invoked.

call and apply, both takes the value of this as first parameter. However, call takes a collection of arguments after first parameter whereas apply use an array of arguments as second parameter.

The apply and call methods are almost identical when setting the this value except that you pass the function parameters to apply () as an array, while you have to list the parameters individually to pass them to the call () method.

Borrowing Functions with Apply and Call (A Must Know)

The most common use for the Apply and Call methods in JavaScript is probably to borrow functions. We can borrow functions with the Apply and Call methods just as we did with the bind method, but in a more versatile manner.

Borrowing Array Methods

Arrays come with a number of useful methods for iterating and modifying arrays, but unfortunately, Objects do not have as many native methods. Nonetheless, since an Object can be expressed in a manner similar to an Array (known as an array-like object), and most important, because all of the Array methods are generic (except toString and toLocaleString), we can borrow Array methods and use them on objects that are array-like.

An array-like object is an object that has its keys defined as non-negative integers. It is best to specifically add a length property on the object that has the length of the object, since the a length property does not exist on objects it does on Arrays.

I should note (for clarity, especially for new JavaScript developers) that in the following examples, when we call Array.prototype, we are reaching into the Array object and on its prototype (where all its methods are defined for inheritance). And it is from there—the source—that we are borrowing the Array methods. Hence the use of code like Array.prototype.slice—the slice method that is defined on the Array prototype.

Let’s create an array-like object and borrow some array methods to operate on the our array-like object. Keep in mind the array-like object is a real object, it is not an array at all:

```
// An array-like object: note the non-negative integers used as keys
var anArrayLikeObj = {0:"Martin", 1:78, 2:67, 3:["Letta", "Marieta", "Pauline"], length:4 };
```

Now, if wish to use any of the common Array methods on our object, we can:

```
// Make a quick copy and save the results in a real array:
// First parameter sets the "this" value
var newArray = Array.prototype.slice.call (anArrayLikeObj, 0);

console.log (newArray); // ["Martin", 78, 67, Array[3]]

// Search for "Martin" in the array-like object
console.log (Array.prototype.indexOf.call (anArrayLikeObj, "Martin") === -1 ? false : true);
// true

// Try using an Array method without the call () or apply ()
```

```
console.log (anArrayLikeObj.indexOf ("Martin") === -1 ? false : true); // Error: Object has
no method 'indexOf'

// Reverse the object:
console.log (Array.prototype.reverse.call (anArrayLikeObj));
// {0: Array[3], 1: 67, 2: 78, 3: "Martin", length: 4}//the anArrayLikeObj was reversed
itself.

// Sweet. We can pop too:
console.log (Array.prototype.pop.call (anArrayLikeObj));
console.log (anArrayLikeObj); // {0: Array[3], 1: 67, 2: 78, length: 3}

// What about push?
console.log (Array.prototype.push.call (anArrayLikeObj, "Jackie", "Jennifer"));// 5
=== //the difference between call and apply in syntax
console.log (Array.prototype.push.apply (anArrayLikeObj, ["Jackie", "Jennifer"]));// 5

console.log (anArrayLikeObj); // {0: Array[3], 1: 67, 2: 78, 3: "Jackie", "Jennifer", length:
5}
```

Borrowing String Methods with Apply and Call

Like the preceding example, we can also use apply () and call () to borrow String methods. Since Strings are immutable, only the non-manipulative arrays work on them, so you cannot use reverse, pop and the like.

Borrow Other Methods and Functions

Since we are borrowing, lets go all in and borrow from our own custom methods and functions, not just from Array and String:###

```
var gameController = {
    scores  :[20, 34, 55, 46, 77],
    avgScore:null,
    players :[
        {name:"Tommy", playerID:987, age:23},
        {name:"Pau", playerID:87, age:33}
    ]
}

var appController = {
    scores  :[900, 845, 809, 950],
    avgScore:null,
    avg     :function () {

        var sumOfScores = this.scores.reduce (function (prev, cur, index, array) {
            return prev + cur;
        });

        this.avgScore = sumOfScores / this.scores.length;
    }
}

// Note that we are using the apply () method, so the 2nd argument has to be an array
appController.avg.apply (gameController);
console.log (gameController.avgScore); // 46.4

// appController.avgScore is still null; it was not updated, only gameController.avgScore was
updated
console.log (appController.avgScore); // null
```

Sure, it is just as easy, even recommended, to borrow our own custom methods and functions. The gameController object borrows the appController object's avg () method. The "this" value defined in the avg () method will be set to the first parameter—the gameController object.

With the bind () method, we can explicitly set the this value for invoking methods on objects, we can borrow and copy methods, and assign methods to variable to be executed as functions. And as outlined in the Currying Tip earlier, you can use bind for currying.

Bind () Allows us to Borrow Methods

```
 // Here we have a cars object that does not have a method to print its data to the console
var cars = {
    data:[
        {name:"Honda Accord", age:14},
        {name:"Tesla Model S", age:2}
    ]

}

// We can borrow the showData () method from the user object we defined in the last example.
// Here we bind the user.showData method to the cars object we just created.
cars.showData = user.showData.bind (cars);
cars.showData (); // Honda Accord 14
```

One problem with this example is that we are adding a new method (showData) on the cars object and we might not want to do that just to borrow a method because the cars object might already have a property or method name showData. We don't want to overwrite it accidentally. As we will see in our discussion of Apply and Call below, it is best to borrow a method using either the Apply or Call method.

13 Javascript anonymous functions

Anonymous functions are functions that are dynamically declared at runtime. They're called anonymous functions because they aren't given a name in the same way as normal functions.

Anonymous functions are declared using the function operator instead of the function declaration. You can use the function operator to create a new function wherever it's valid to put an expression. For example you could declare a new function as a parameter to a function call or to assign a property of another object.

An anonymous function is a function that was declared without any named identifier to refer to it. As such, an anonymous function is usually not accessible after its initial creation. The most commom use for anonymous functions are as arguments to other functions, or as a closure.

There are quite a few different ways to create a new function in javascript, but the two most common are using a function declaration or using the function operator.

This function is created using a function declaration:

```
function destroyDeathStar() {
    alert("Stay on target, stay on target");
}
```

This function is created using the function operator:(this is function expression)

```
var destroyDeathStar = function() {
    alert("Stay on target, stay on target");
}
destroyDeathStar();
```

The biggest difference between using a function declaration or a function operator is when the function is created. Functions made with a function declaration are moved to the top of the code and created before the rest of the function is run. Functions made with the function operator are created at runtime where they are in the javascript code.

14 Prototype

When a function is created in JavaScript, JavaScript engine adds a prototype property to the function. This prototype property is an object (called as prototype object) has a constructor property by default. constructor property points back to the function on which prototype object is a property.

Prototype object of the constructor function is shared among all the objects created using the constructor function.

As prototype object is an object, we can attach properties and methods to the prototype object. Thus, enabling all the objects created using the constructor function to share those properties and methods.

Problem with constructor: Every object has its own instance of the function

Problem with the prototype: Modifying a property using one object reflects the other object also

```
//Define the object specific properties inside the constructor
function Human(name, age){
    this.name = name,
    this.age = age,
    this.friends = ["Jadeja", "Vijay"]
}
//Define the shared properties and methods using the prototype
Human.prototype.sayName = function(){
    console.log(this.name);
}
//Create two objects using the Human constructor function
```

```
var person1 = new Human("Virat", "Kohli");
var person2 = new Human("Sachin", "Tendulkar");

//Lets check if person1 and person2 have points to the same instance of the sayName function
console.log(person1.sayName === person2.sayName) // true

//Let's modify friends property and check
person1.friends.push("Amit");

console.log(person1.friends)// Output: "Jadeja, Vijay, Amit"
console.log(person2.frinds)//Output: "Jadeja, Vijay"
```

Prototype in JS

Every JS object has a prototype. The prototype is also an object. The prototype property allows you to add new properties and methods to existing object. All JavaScript objects inherit their properties and methods from their prototype called Object.prototype. The standard way of creating a prototype is through the constructor function.

In JavaScript you first create an object, then you can add objects or create new objects from it.

```
Example:
    //Define a functional object to hold persons in Js
    var Person = function(name){
        this.name = name;
    };

    //Add dynamically to the already defined object a new getter
    Person.prototype.getName = function(){
        return this.name;
    }

    //Create a new object of type Person
    var john = new Person("John");

    alert(john.getName());
    // John

    //If now I modify Person object, also john gets the updates
    Person.prototype.sayMyName = function(){
        alert('Hello, my name is '+ this.getName());
    }

    john.sayMyName();
    // Hello, my name is John
```

15 Inheritance

JavaScript does not have classes unlike other languages. It uses the concept of prototypes and prototype chaining for inheritance.

Prototype Chaining

Prototype chaining means an objects dunder proto or proto will point to another object instead of pointing to the constructor function prototype. If the other object's dunder proto or proto property points to another object it will results into chain. This is prototype chaining.

```
//SuperType constructor function
function SuperType(){
    this.name = "Virat"
}

//SuperType prototype
SuperType.prototype.getSuperName = function(){
    return this.name
}

//SubType prototype function
function SubType(){
    this.age = 26
}

//Inherit the properties from SuperType
SubType.prototype = new SuperType();

//Add new property to SubType prototype
SubType.prototype.getSubAge = function(){
    return this.age;
}

//Create a SubType object
var subTypeObj = new SubType();
console.log(subTypeObj.name); //Output: Virat
console.log(subTypeObj.age); //Output: 26
console.log(subTypeObj.getSuperName()); //Output: Virat
console.log(subTypeObj.getSubAge()); //Output: 26
```

16 how do you delete item from array

Method1:

```
.splice()
arrayList.splice([startIndex], deleteCount);
var months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
// inserts at 1st index position
console.log(months);
```

```
// expected output: Array ['Jan', 'Feb', 'March', 'April', 'June']

months.splice(4, 1, 'May');
// replaces 1 element at 4th index
console.log(months);
// expected output: Array ['Jan', 'Feb', 'March', 'April', 'May']
```

Method2: .filter()

Method3: `var trees = ["redwood", "bay", "cedar", "oak", "maple"]; delete trees[3];// ["redwood", "bay", "cedar", empty, "maple"]`

Answer

17 Data storage

1) cookie: server & browser can both use but the size is small. (it can only store limited content)

2) sessionStorage:

· a method of window;
· can only be used in browser

· can only exist in one tab, after close the tab, it does not exist

3) localStorage:
· a method of window
· an only be used in browser
·
· it will exist for long time until you delete it

LocalStorage and SessionStorage can use up to **10MB** of storage but the number is actually the sum of both.

18 querySelector

The Document method querySelector() returns the first Element within the document that matches the specified selector, or group of selectors. If no matches are found, null is returned.

```
element = document.querySelector(selectors);
```

19 What is a deep copy VS shallow object copy?

A shallow copy will clone the top-level object, but nested object are shared between the original and the clone. That's it: an object reference is copied, not cloned. So if the original object contains nested object, then the clone will not be a distinct entity.

A deep copy will recursively clone every objects it encounters. The clone and the original object will not share anything, so the clone will be a fully distinct entity.

20. Singleton:

It limits the number of instances of a particular object to just one. It is implemented as an immediate anonymous function, the Module pattern is JavaScript's manifestation of it. Ensure all object access the single instance

```
    var Singleton = (function(){
        var instance;

        function createInstance(){
            var object = new Object("I am instance");
            return object;
        };

        return { getInstance: function(){
            if(!instance){
                instance = createInstance();
            }
            return instance;
        }};
    })();
    function run(){

    var instance1 = Singleton.getInstance();
    var instance2 = Singleton.getInstance();

    alert("Same instance? " + (instance1 === instance2));
    }
```

21. Strict Mode?

strict mode delete some JavaScript silent errors by changing them to throw erros.

strict mode code can sometimes be made to run fast than identical code that is not strict mode.

strict mode is declared by adding "use strict;" to the beginning of the JS file or JS function.

declared at the beginning of a JS file, it has glogbal scope, declared in a function it has local scope.

22.Difference between == and ===

==: compare value

===: compare value and type

23. Optimize JavaScript code:

· Use local function variables instead of global variables

Global variables lives in a highly-populated namespace, browser must distinguish between global variables and properties of objects that are in the current context

· Avoid adding short strings to long strings

It needs to duplicate the long string

· Avoid pitfalls with closures

Closure are a powerful and useful feature of JavaScript, however, they have several drawbacks, including: they are the most common source of memory leaks; creating a closure is significantly slower than creating an inner function without a closure, and much slower than reusing a static function.

· Use hoisting carefully

24. Typeof null and typeof undefined:

undefined means the variable has been declared, but no value has been assigned to the variable.

Looking up non-existent properties in an object:

var test = {};

console.log(test.prop);

// undefined

null means the variable has been assigned a value as a representation of no value. ‘null’ is an assignment value. This can be assigned to a variable as a representation of no value:

```
var a = null;

console.log(a); //shows null

console.log(typeof a); //shows object
```

25. Use a object literal:

A easy way to create JS object, you both define and create an object in one statement.

An object literal is a list of name: valule pairs inside curly braces.

· use the new keyword:

```
var person = new Object();
person.firstname = "John";
```

· use an object constructor

If a object type is needed, the standard way to create it is to use an object constructor function:

```
function person(first, last){
    this.first = first;
    this.last = last;
}
var girl = new person("a","b");
```

26 What is JavaScript Scope?

In JavaScript, scope is the set of variables, objects, and functions you have access to.

local scope: can only be accessed within the function

global scope: all scripts and functions on web page can access it

27 The lifetime of JavaScript variables

Starts when the variable declared

Local variable ends when the function is completed.

Global variables are deleted when you close the page.

28 What is HTML DOM?

HTML DOM is standard for how to get, change, add, or delete HTML elements. With the HTML DOM, javascript can access and interact with all the elements of an HTML document.

29 addEventListener() method

It attaches an event handler to the specified element. It can assign different function to a specified element.

30 How to define and call a function within an object?

An object method is a function definition, stored as a property value

31 how to get value of checkbox

```
<input class="messageCheckbox" type="checkbox" value="3" name="mailId[]">

<input class="messageCheckbox" type="checkbox" value="1" name="mailId[]">
```

For modern browsers:

```
var checkedValue = document.querySelector('.messageCheckbox:checked').value;
```

By using jQuery:

```
var checkedValue = $('.messageCheckbox:checked').val();
Checkbox: <input type="checkbox" id="myCheck" value="myvalue">

var x = document.getElementById("myCheck").value;
```

32.Json and Jsonp difference

JSON is a language designed for lightweight data exchange in an format that is easy for both humans and computers to understand.

Unfortunately, as much as JSON appears to be the best thing since sliced bread as far as web developers are concerned, the technologies used by JavaScript to send and receive JSON messages (XMLHttpRequest, and Microsoft’s family of ActiveXObject‘s) are restricted by the same origin policy which is enforced in all major browsers; i.e. you cannot make a request to a resource using a different protocol, domain or port to which the current page is using.

However, it was discovered that the JSON language we know and love could work on a technology which is not restricted by the same origin policy with only a small amount of changes required. The combination of this technology, along with the adapted form of JSON was branded JSONP.

33.what is the type of variable without a value:

undefined

34 Set a variable to empty:

set it as null

35.Difference of settimeout and setinterval

Settimeout calls a function or evaluates an expression after a specified number of milliseconds.

SetInterval calls a function or evaluates an expression at time intervals in milliseconds. and will continue calling the function until clearInterval is called, or the window is closed.

36.Difference between overloading and overriding

Overloading deals with the notion of having two or more methods in the same class with the same name but different arguments.

Overriding means having two methods with the same argument, but different implementation. One of them would exist in the Parent class while another will be the derived class.

37.Explain this keyword

In JavaScript, this will always refer to the owner of the function we are executing or as an object method

38.What are browser detection and feature detection

Browser detection and feature detection are two most common ways to test whether the technologies you use in your application can apply to the current browser or not. Browser detection mainly check to match the browser type strings, while the feature detectors are consist of small scripts to test whether the feature you need is supported by the current browser or not(this is not about the version or the browser type)

Browser Detection: It is one approach to cover browser differences. The most common approach is to use JavaScript to query the user-agent header:

```
<script type="text/javascript">browser
    if(navigator.userAgnet.indexof("MSIE")>0){
        //run custom code for internet explorer
    }
</script>
```

Usually use it when dealing with the legacy applications.

Disadvantages:

(1). Should track which features the browser supports in detail, one wrong assumption could break the site.

(2). Does not consider the version of the browser, it is not future-proof.

Feature Detection:

Better approach to handle differences in web browsers in web pages

Before using a feature that you know has different implementations in the various browsers, you can run a small set of tests where you look for the availability of a specific object, method, property, or behavior.

In most cases, this can be done by trying to create a new instance of the feature in question. If that instantiation returns something other than null, then executing browser knows the feature. If not, you can test to see if there's a workaround or priority legacy implementation of the feature available.

Two important recommendations for feature detection:

(1). test for standards, as a browser often supports the newer standard as well as the legacy work around

(2). only target related features in a single check , to minimize assumptions about a browser's capability

39. JS frameworks:

angularjs knockout.js node.js backbone

40. How to catch unhandled JavaScript errors?

Assign the window.onerror event to event handler.

```
    <img src="image.gif" onerror="myFunction()>

    <script>
        function myFunction(){
            alert('The image could not be loaded.');
        }
    </script>
```

41. When does the window.onerror event fire?

In a nutshell, the event is raised when either:

1) there is an uncaught exception

2) a compile time error occurs

Uncaught exceptions: throw some messages, call something undefined, security exception, compile error

42. What is immediate function

Immediate functions execute as soon as JavaScript encounters them. Creating an immediate function: add the open/close parentheses after the closing curly bracket, and then wrap the entire function in parenthese. Self invoke function

43. How to reload page with JavaScript

location.reload(forceGet);

forceGet: true-page must reloaded from the server; false(default)-page reloaded from the cache

44. Minification

In JS, it means removing all unnecessary characters, such as spaces, new lines, comments without affecting the functionality of the source code.

45. document.write()

The write method writes HTML expressions or JS code to a document.

46. Namespace

It is used for grouping the desired functions, variables, under a unique name. It is a name that has been attached to the desired functions, objects and properties. This improves modularity in the coding and enables code reuse.

In JavaScript a namespace is just another object containing methods, properties and objects.

JS has a big design flaw, where it is very easy to create global variables that have the potential to interact in negative ways. The practice of namespacing is usually to create an object literal encapsulating your own functions and variables, so as not to collide with those created by other libraries:

```
var MyApplication = {
    var1: val1,
    myfun: function(){

    },
    ...
};
```

Instead of calling myfun() globally, it would always be called as: MyApplication.myfun(). It is then less likely for our method or function to collide with other libraries.

47. JS testing:

Unit testing: a software testing method by which individual part of source code get tested.

End to end testing: test whether the flow of an application from start to finish is behaving as expected, the purpose of performing it is to identify system dependencies and to ensure that the data integrity is maintained between various systems components and systems.

The entire application is tested for critical functionalities such as communication with the other systems, interfaces, database, network and other applications.

Jasmine is a behavior-driven development framework for testing JS code. It does not depend on any other JS frameworks. It does not require a DOM, it has a clean, obvious syntax so that you can easily write tests.

A test suite begins with a call to the global Jasmine function named describe with two parameters: a string and a function. The string is a name or title for a spec suite. The function is a block of code that implements the suite.

48. What are JavaScript types:

Number, string, boolean, function, object, null, undefined

49. What is the use of isNaN function

it returns true if the argument is not a number, otherwise false.

50. Explain how can you submit a form using JavaScript

document.form[0].submit();

51. Disable the enter key on HTML form:

```
$(document).keypress(
    function(event){
        if(event.which == '13') {
            event.preventDefault();
        }
});
//or this way
$('input').on('keydown', function(event) {
   var x = event.which;
   if (x === 13) {
       event.preventDefault();
   }
});
```

52.Difference between JavaScript Framework and Library

Library like jQuery gives us a neat toolkit to help with implementing some functions.

Frameworks provides us with an entire project architecture to follow in order to build a well structured javascript project.

53.Speed up web:

minificaiton,

less downloading,

ajax,

css in the front,

javascript on the end,

54. Tagged Template Literals

The way the tag template works is you simply make a function, and you take the name of the function that you want to run against the string, and you just put the name right in front of the template:

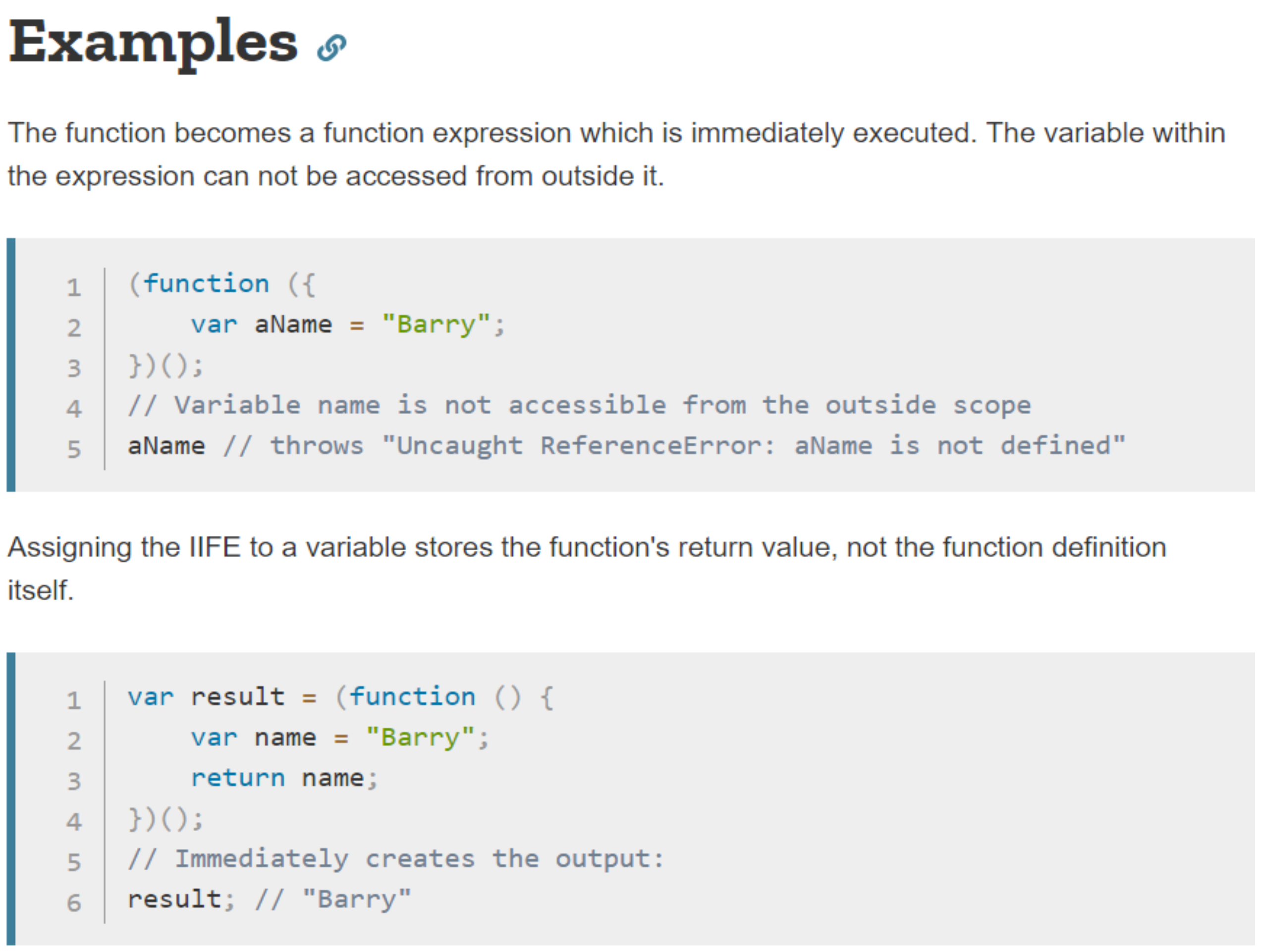
```
function highlight() {
    // do something
}

const name = 'Snickers';
const age = '100';
const sentence = highlight`My dog's name is ${name} and he is ${age} years old`;
console.log(sentence)
```

This tags it with a function name. What will happen is the browser will run this function, it will provide it with all the information about the string, all the pieces that the person typed, as well as the variables that get passed in. Whatever we return from this function is what sentence actually is going to be.

55. IIFE (Immediately Invoked Function Expression)

An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined. It is a design pattern which is also known as a Self-Executing Anonymous Function and contains two major parts. The first is the anonymous function with lexical scope enclosed within the Grouping Operator (). This prevents accessing variables within the IIFE idiom as well as polluting the global scope.



Examples

The function becomes a function expression which is immediately executed. The variable within the expression can not be accessed from outside it.

```javascript
(function ({
    var aName = "Barry";
})();
// Variable name is not accessible from the outside scope
aName // throws "Uncaught ReferenceError: aName is not defined"
```

Assigning the IIFE to a variable stores the function's return value, not the function definition itself.

```javascript
var result = (function () {
    var name = "Barry";
    return name;
})();
// Immediately creates the output:
result; // "Barry"
```

General React

1.Differentiate bentween Real DOM and Virtual DOM.

2.What is React?

- React is a front-end JavaScript library developed by Facebook in 2011.
- It follows the component based approach which helps in building reusable UI components.
- It is used for developing complex and interactive web and mobile UI.
- Even though it was open-sourced only in 2015, it has one of the largest communities supporting it.

3.What are the features of React?

- It uses the virtual DOM instead of the real DOM.
- It uses JSX.
- It follows one way data flow.

4.List some of the major advantages of React.

- It increases the application's performance
- It can be conveniently used on the client as well as server side
- Because of JSX, code's readability increases
- React is easy to intergate with other frameworks like Angular etc
- Using React, writing UI test cases become extremely easy

5.What are the limitations of React?

- React is just a library, not a full-blown framework
- Its library is very large and takes time to understand
- It can be little difficult for novice programmers to understand
- Coding gets complex as it uses inline templating and JSX

6.What is JSX?

JSX is shorthand for JavaScript XML. This is a type of file by React which utilizes the expressiveness of JavaScript along with HTML like template syntax. This makes the HTML file easy to understand. This file makes applications robust and boosts its performance.
Deference between JSX and HTML:

1) className vs. class
In JSX, class is a Javascript reserved key word, so use className to replace it.
2) reference JS variable with curly braces.

Below is an example of JSX:

```
render(){
   return(
       <div>
         <h1> Hello World from Edureka!!</h1>
       </div>
   );
}
```

7.What do you understand by Virtual DOM? Explain its working.

Here's what happens when you try to update the DOM in React:

- The entire virtual DOM gets updated.
- The virtual DOM gets compared to (real DOM) what it looked like before you updated it. React figures out which objects have changed.
- The changed objects, and the changed objects only, get updated on the real DOM.
- Changes on the real DOM cause the screen to change.
- A virtual DOM is a lightweight JavaScript object which originally is just the copy of the real DOM. It is a node tree that lists the elements, their attributes and content as Object and their properties. React render function

creates a node tree out of the React components. It then updates their tree in response to the mutations in the data model which is caused by various actions done by the user or by the system.

The Virtual DOM (VDOM) is an in-memory representation of Real DOM. The representation of a UI is kept in memory and synced with the "real" DOM. It's a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called reconciliation.

8.Why can't browsers read JSX?

Browsers can only read JavaScript objects but JSX is not a regular JavaScript object. Thus to enable a browser to read JSX, first, we need to transform JSX file into a JavaScript object using JSX transformers e.g. Babel and then pass it to the browser.

9.How different is React's ES6 syntax when compared to ES5?

Syntax has changed from ES5 to Es6 in following aspects:

1.<u>require vs import</u>

```
// ES5
var React = require('react');

// ES6
import React from 'react';
```

2.<u>export vs exports</u>

```
module.exports = Component;

// ES6
export default Component;
```

3.<u>components and function</u>

```
// ES5
var MyComponent = React.createClass({
   render: function() {
      return <h3>Hello Edureka!</h3>;
   }
});

// ES6
class MyComponent extends React.Component {
    render() {
        return <h3>Hello Edureka!</h3>;
    }
}
```

4.props

```
// ES5
var App = React.createClass({
    propTypes: { name: React.PropTypes.string },
    render: function() {
        return <h3>Hello, {this.props.name}!</h3>;
    }
});

// ES6
class App extends React.Component {
    render() {
        return <h3>Hello, {this.props.name}!</h3>;
    }
}
```

5.state

```
// ES5
var App = React.createClass({
    getInitialState: function() {
        return { name: 'world' };
    },
    render: function() {
        return <h3>Hello, {this.state.name}!</h3>;
    }
});

// ES6
class App extends React.Component {
    constructor() {
        super();
        this.state = { name: 'world' };
    }
```

```
  render() {
    return <h3>Hello, {this.state.name}!</h3>;
  }
}
```

10.How is React different from Angular?

React:

- React is a library and has only the View layer
- React handles rendering on the server side
- React uses JSX that looks like HTML in JS which can be confusing
- React Native, which is a React type to build mobile applications are faster and more stable
- In React, data flows only in one way and hence debugging is easy

Angular:

- Angular is a framework and has complete MVC functionality
- AngularJS renders only on the client side but Angular 2 and above renders on the server side
- Angular follows the template approach for HTML, which makes code shorter and easy to understand
- Ionic, Angular's mobile native app is relatively less stable and slower
- In Angular, data flows both way i.e it has two-way data binding between children and parent and hence debugging is often difficult

11. What do you understand from "In react, everything is a component."

Components are the building blocks of a React application's UI. These components split up the entire UI into small independent and reusable pieces. Then it renders each of these components independent of each other without affecting the rest of the UI.

12. Explain the purpose of render() in React.

Each React component must have a render() mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as form, group, div etc. This function must be kept pure i.e.,it must return the same result each time it is invoked.

13. How can you embed two or more components into one?

```
class MyComponent extends React.Component{
    render(){
        return(
            <div>
                <h1>Hello</h1>
                <Header/>
            </div>
        );
    }
}
class Header extends React.Component{
    render(){
        return <h1>Header Component</h1>
   };
}
ReactDOM.render(
    <MyComponent/>, document.getElementById('content')
);
```

14.What is Props?

Props are inputs to components. They are single values or objects containing a set of values that are passed to components on creation using a naming convention similar to HTML-tag attributes. They are data passed down from a parent component to a child component.

The primary purpose of props in React is to provide following component functionality:

- Pass custom data to your component.
- Trigger state changes.
- Use via this.props.reactProp inside component's render() method.

For example, let us create an element with reactProp property:

```
<Element reactProp={'1'} />
```

This reactProp (or whatever you came up with) name then becomes a property attached to React's native props object which originally already exists on all components created using React library.

```
props.reactProp
```

Props is the shorthand for Properties in React. They are read-only components which must be kept pure i.e. immutable. They are always passed down from the parent to the child components throughout the application. A child component never send a prop back to the parent component. This help in maintaining the one-way data flow and are generally used to render the dynamically generated data.

15.What is a state in React and how is it used?

States are the hearts of React components. State of a component is an object that holds some information that may change over the lifetime of the component. We should always try to make our state as simple as possible and minimize the number of stateful components. Let's create an user component with message state. States are the source of data and must be kept as simple as possible. Basically, states are the objects which determines components rendering and behaviors. They are mutable unlike the props and create dynamic and interactive components. They are accessed via this.state().

16.Difference between states and props

Both props and state are plain JavaScript objects. While both of them hold information that influences the output of render, they are different in their functionality with respect to component. Props get passed to the component similar to function parameters whereas state is managed within the component similar to variables declared within a function.

Why should not we update the state directly?

If you try to update state directly then it won't re-render the component.
//Wrong
this.state.message = 'Hello world'
Instead use setState() method. It schedules an update to a component's state object. When state changes, the component responds by re-rendering.
//Correct
this.setState({ message: 'Hello World' })
Note: You can directly assign to the state object either in constructor or using latest javascript's class field declaration syntax.

17. Explain the lifecycle methods of React components in details.

- componentWillMount() – Executed just before rendering takes place both on the client as well as server-side.
- componentDidMount() – Executed on the client side only after the first render.
- componentWillReceiveProps() – Invoked as soon as the props are received from the parent class and before another render is called.
- shouldComponentUpdate() – Returns true or false value based on certain conditions. If you want your component to update, return true else return false. By default, it returns true.
- componentWillUpdate() – Called just before rendering takes place in the DOM.
- componentDidUpdate() – Called immediately after rendering takes place. Don't call setState() in componentDidUpdate(), it will cause infinite loop: componentDidUpdate()~setState()~render()~componentDidUpdate()
- componentWillUnmount() – Called after the component is unmounted from the DOM. It is used to clear up the memory spaces.

18. What are synthetic events in React?

Synthetic events are the objects which act as a cross-browser wrapper around the browser's native event. They combine the behavior of different browsers into one API. This is done to make sure that the events show consistent properties across different browsers.

```
import React, { Component } from 'react';

class ShowAlert extends Component {

  showAlert() {
    alert("Im an alert");
  }

  render() {
    return (
      <button onClick={this.showAlert}>show alert </button>
    );
  }
}
```

```
export default ShowAlert;
```

19. What do you understand by refs in React?

Refs is the shorthand for References in React. It is an attribute which helps to store a reference to a particular React element or component, which will be returned by the components render configuration function. It is used to return references to a particular element or component returned by render(). They come in handy when we need DOM measurements or to add methods to the components.
The callback function receives the React component instance or HTML DOM element as its argument, which can be stored and accessed elsewhere.
First we defined the ref using a callback function and storing it in this.inputDemo as follows:

```
class ReferenceDemo extends React.Component{
     display() {
         const name = this.inputDemo.value;
         document.getElementById('disp').innerHTML = name;
     }
render() {
    return(
         <div>
            Name: <input type="text" ref={input => this.inputDemo = input} />
            <button name="Click" onClick={this.display}>Click</button>
            <h2>Hello <span id="disp"></span> !!!</h2>
        </div>
    );
   }
 }
```

20. List some of the cases when you should use Refs.

Following are the cases when refs should be used:

- When you need to manage focus, select text or media playback
- To trigger imperative animations
- Integrate with third-party DOM libraries

21. What are Higher Order Components(HOC)?

Higher Order Component is an advanced way of reusing the component logic. Basically, it’s a pattern that is derived from React’s compositional nature. HOC are custom components which wrap another component within it. They can accept any dynamically provided child component but they won’t modify or copy any behavior from their input components. You can say that HOC are ‘pure’ components.

A higher-order component in React is a pattern used to share common functionality between components without repeating code. A higher-order component is actually not a component though, it is a function. A HOC function takes a component as an argument and returns a component. It transforms a component into another component and adds additional data or functionality.

Two HOC’s implementations that you may be familiar with in the React ecosystem are connect from Redux and withRouter from React Router. The connect function from Redux is used to give components access to the global state in the Redux store, and it passes these values to the component as props. The withRouter function injects the router information and functionality into the component, enabling the developer access or change the route.

22. What can you do with HOC?（看看就行. 行吧）

HOC can be used for many tasks like:

- Code reuse, logic and bootstrap abstraction
- Render High jacking
- State abstraction and manipulation
- Props manipulation

23 What are Pure Components?

Pure components are the simplest and fastest components which can be written. They can replace any component which only has a render(). These components enhance the simplicity of the code and performance of the application.

React.PureComponent is exactly the same as React.Component except that it handles the shouldComponentUpdate()method for you. When props or state changes, PureComponent will do a shallow comparison on both props and state. Component on the other hand won't compare current props and state to next out of the box. Thus, the component will re-render by default whenever shouldComponentUpdate is called.

24 What is Lifting State Up in React?

When several components need to share the same changing data then it is recommended to lift the shared state up to their closest common ancestor. That means if two child components share the same data from its parent, then move the state to parent instead of maintaining local state in both of the child components.

```
class Calculator extends React.Component {
  constructor(props) {
    super(props);
    this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
    this.handleFahrenheitChange = this.handleFahrenheitChange.bind(this);
    this.state = {temperature: '', scale: 'c'};
  }

  handleCelsiusChange(temperature) {
    this.setState({scale: 'c', temperature});
  }

  handleFahrenheitChange(temperature) {
    this.setState({scale: 'f', temperature});
  }

  render() {
    const scale = this.state.scale;
    const temperature = this.state.temperature;
    const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) : temperature;
    const fahrenheit = scale === 'c' ? tryConvert(temperature, toFahrenheit) : temperature;
    return (
      <div>
        <TemperatureInput
          scale="c"
          temperature={celsius}
          onTemperatureChange={this.handleCelsiusChange} />
        <TemperatureInput
          scale="f"
          temperature={fahrenheit}
          onTemperatureChange={this.handleFahrenheitChange} />
        <BoilingVerdict
          celsius={parseFloat(celsius)} />
      </div>
    );
  }
}
```

```
class TemperatureInput extends React.Component {
  constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    this.props.onTemperatureChange(e.target.value);
  }

  render() {
    const temperature = this.props.temperature;
    const scale = this.props.scale;
    return (
      <fieldset>
        <legend>Enter temperature in {scaleNames[scale]}:</legend>
        <input value={temperature}
               onChange={this.handleChange} />
      </fieldset>
    );
  }
}
```

25 What is reconciliation?

When a component's props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called reconciliation.

26 What is the purpose of using super constructor with props argument?

A child class constructor cannot make use of this reference until super() method has been called. The same applies for ES6 sub-classes as well. The main reason of passing props parameter to super() call is to access this.props in your child constructors.

Passing props:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props)

    console.log(this.props) // prints { name: 'John', age: 42 }
  }
}
```

Not passing props:

```
class MyComponent extends React.Component {
  constructor(props) {
    super()

    console.log(this.props) // prints undefined

    // but props parameter is still available
    console.log(props) // prints { name: 'John', age: 42 }
  }

  render() {
    // no difference outside constructor
    console.log(this.props) // prints { name: 'John', age: 42 }
  }
}
```

The above code snippets reveals that this.props is different only within the constructor. It would be the same outside the constructor.

27 What would be the common mistake of function being called every time the component renders?

You need to make sure that function is not being called while passing the function as a parameter.

```
render() {
  // Wrong: handleClick is called instead of passed as a reference!
  return <button onClick={this.handleClick()}>{'Click Me'}</button>
}
```

Instead, pass the function itself without parenthesis:

```
render() {
  // Correct: handleClick is passed as a reference!
  return <button onClick={this.handleClick}>{'Click Me'}</button>
}
```

28 What are fragments?

It's common pattern in React which is used for a component to return multiple elements. Fragments let you group a list of children without adding extra nodes to the DOM.

```
render() {
  return (
    <React.Fragment>
      <ChildA />
      <ChildB />
      <ChildC />
    </React.Fragment>
  )
}
```

There is also a shorter syntax, but it's not supported in many tools:

```
render() {
  return (
    <>
      <ChildA />
      <ChildB />
      <ChildC />
    </>
  )
}
```

29 Why fragments are better than container divs?

- Fragments are a bit faster and use less memory by not creating an extra DOM node. This only has a real benefit on very large and deep trees.
- Some CSS mechanisms like Flexbox and CSS Grid have a special parent-child relationships, and adding divs in the middle makes it hard to keep the desired layout.
- The DOM Inspector is less cluttered.

30 What is the impact of indexes as keys?

- Keys should be stable, predictable, and unique so that React can keep track of elements.
- Component instances are updated and reused based on their key. If the key is an index, moving an item changes it. As a result, component state for things like uncontrolled inputs can get mixed up and updated in unexpected ways.
- In the below code snippet each element's key will be based on ordering, rather than tied to the data that is being represented. This limits the optimizations that React can do.

```
{todos.map((todo, index) =>
  <Todo
    {...todo}
    key={index}
  />
)}
```

If you use element data for unique key, assuming todo.id is unique to this list and stable, React would be able to reorder elements without needing to reevaluate them as much.

```
{todos.map((todo) =>
  <Todo {...todo}
    key={todo.id} />
)}
```

31 Is it good to use setState() in componentWillMount() method?

First, we don't use componentWillMount() anymore. Instead, we use getDerivedStateFromProps().
It is recommended to avoid async initialization in componentWillMount() lifecycle method. 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. We need to make sure async calls for component initialization happened in componentDidMount() instead of componentWillMount().

```
componentDidMount() {
  axios.get(`api/todos`)
    .then((result) => {
      this.setState({
        messages: [...result.data]
      })
    })
}
```

32 What will happen if you use props in initial state?

If the props on the component are changed without the component being refreshed, the new prop value will never be displayed because the constructor function will never update the current state of the component. The initialization of state from props only runs when the component is first created.

The below component won't display the updated input value:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props)

    this.state = {
      records: [],
      inputValue: this.props.inputValue
    };
  }

  render() {
    return <div>{this.state.inputValue}</div>
  }
}
```

Using props inside render method will update the value:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props)

    this.state = {
      record: []
    }
  }

  render() {
    return <div>{this.props.inputValue}</div>
  }
}
```

33 Why we need to be careful when spreading props on DOM elements?

When we spread props we run into the risk of adding unknown HTML attributes, which is a bad practice. Instead we can use prop destructuring with ...rest operator, so it will add only required props. For example,

```
const ComponentA = () =>
  <ComponentB isDisplay={true} className={'componentStyle'} />

const ComponentB = ({ isDisplay, ...domProps }) =>
  <div {...domProps}>{'ComponentB'}</div>
```

34 How you implement Server Side Rendering or SSR?

React is already equipped to handle rendering on Node servers. A special version of the DOM renderer is available, which follows the same pattern as on the client side.

```
import ReactDOMServer from 'react-dom/server'
import App from './App'
```

```
ReactDOMServer.renderToString(<App />)
```

This method will output the regular HTML as a string, which can be then placed inside a page body as part of the server response. On the client side, React detects the pre-rendered content and seamlessly picks up where it left off.

35 What is the purpose of getDerivedStateFromProps() lifecycle method?

The new static getDerivedStateFromProps() lifecycle method is invoked after a component is instantiated as well as before it is re-rendered. It can return an object to update state, or null to indicate that the new props do not require any state updates.

```
class MyComponent extends React.Component {
  static getDerivedStateFromProps(props, state) {
    // ...
  }
}
```

This lifecycle method along with componentDidUpdate() covers all the use cases of componentWillReceiveProps().

36 What is React proptype array with shape?

If you want to pass an array of objects to a component with a particular shape then use React.PropTypes.shape() as an argument to React.PropTypes.arrayOf().

```
ReactComponent.propTypes = {
  arrayWithShape: React.PropTypes.arrayOf(React.PropTypes.shape({
    color: React.PropTypes.string.isRequired,
    fontSize: React.PropTypes.number.isRequired
  })).isRequired
}
```

37 How conditionally apply class attributes?

You shouldn't use curly braces inside quotes because it is going to be evaluated as a string.

```
<div className="btn-panel {this.props.visible ? 'show' : 'hidden'}">
```

Instead you need to move curly braces outside (don't forget to include spaces between class names):

```
<div className={'btn-panel ' + (this.props.visible ? 'show' : 'hidden')}>
```

Template strings will also work:

```
<div className={`btn-panel ${this.props.visible ? 'show' : 'hidden'}`}>
```

38 What is the difference between React and ReactDOM?

The react package contains React.createElement(), React.Component, React.Children, and other helpers related to elements and component classes. You can think of these as the isomorphic or universal helpers that you need to build components. The react-dom package contains ReactDOM.render(), and in react-dom/server we have server-side rendering support with ReactDOMServer.renderToString() and ReactDOMServer.renderToStaticMarkup().

39 How to pretty print JSON with React?

We can use pre tag so that the formatting of the JSON.stringify() is retained:

```
const data = { name: 'John', age: 42 }
```

```
class User extends React.Component {
  render() {
    return (
      <pre>
        {JSON.stringify(data, null, 2)}
      </pre>
    )
  }
}

React.render(<User />, document.getElementById('container'))
```

40 What are the possible ways of updating objects in state?

- Calling setState() with an object to merge with state:

1) Using Object.assign() to create a copy of the object:

```
const user = Object.assign({}, this.state.user, { age: 42 })
this.setState({ user })
```

2) Using spread operator:

```
const user = { ...this.state.user, age: 42 }
this.setState({ user })
```

- Calling setState() with a function:

```
this.setState(prevState => ({
  user: {
    ...prevState.user,
    age: 42
  }
}))
```

41 Why function is preferred over object for setState()?

React may batch multiple setState() calls into a single update for performance. Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state.
This counter example will fail to update as expected:

```
// Wrong
this.setState({
  counter: this.state.counter + this.props.increment,
})
```

The preferred approach is to call setState() with function rather than object. That function will receive the previous state as the first argument, and the props at the time the update is applied as the second argument.

```
// Correct
this.setState((prevState, props) => ({
  counter: prevState.counter + props.increment
}))
```

42 What are Styled Components?

styled-components is a JavaScript library for styling React applications. It removes the mapping between styles and components, and lets you write actual CSS augmented with JavaScript.

Give an example of Styled Components? Lets create Title and Wrapper components with specific styles for each.

```
import React from 'react'
import styled from 'styled-components'

// Create a <Title> component that renders an <h1> which is centered, red and sized at 1.5em
const Title = styled.h1`
```

```
  font-size: 1.5em;
  text-align: center;
  color: palevioletred;
  `
```

```
// Create a <Wrapper> component that renders a <section> with some padding and a
papayawhip background
const Wrapper = styled.section`
  padding: 4em;
  background: papayawhip;
`
```

These two variables, Title and Wrapper, are now components that you can render just like any other react component.

```
<Wrapper>
  <Title>{'Lets start first styled component!'}</Title>
</Wrapper>
```

43 What is the purpose of registerServiceWorker in React?

React creates a service worker for you without any configuration by default. The service worker is a web API that helps you cache your assets and other files so that when the user is offline or on slow network, he/she can still see results on the screen, as such, it helps you build a better user experience, that's what you should know about service worker's for now. It's all about adding offline capabilities to your site.

```
   import React from 'react';
   import ReactDOM from 'react-dom';
   import App from './App';
   import registerServiceWorker from './registerServiceWorker';

   ReactDOM.render(<App />, document.getElementById('root'));
   registerServiceWorker();
```

44 Server side rendering vs client side rendering

On the bright side, server-side rendering is great for SEO. Your content is present before you get it, so search engines are able to index it and crawl it just fine. Something that is not so with client-side rendering. At least not simply.

Server-side pros:

- Search engines can crawl the site for better SEO.
- The initial page load is faster.
- Great for static sites.

Server-side cons:

- Frequent server requests.
- An overall slow page rendering.
- Full page reloads.
- Non-rich site interactions.

Client-side pros:

- Rich site interactions
- Fast website rendering after the initial load.
- Great for web applications.
- Robust selection of JavaScript libraries.

Client-side cons:

- Low SEO if not implemented correctly.
- Initial load might require more time.
- In most cases, requires an external library.

45 Do I need to keep all my state into Redux? Should I ever use react internal state?

It is up to developer decision. i.e, It is developer job to determine what kinds of state make up your application, and where each piece of state should live. Some users prefer to keep every single piece of data in Redux, to maintain a fully serializable and controlled version

of their application at all times. Others prefer to keep non-critical or UI state, such as “is this dropdown currently open”, inside a component's internal state.
Below are the thumb rules to determine what kind of data should be put into Redux

- Do other parts of the application care about this data?
- Do you need to be able to create further derived data based on this original data?
- Is the same data being used to drive multiple components?
- Is there value to you in being able to restore this state to a given point in time (ie, time travel debugging)?
- Do you want to cache the data (ie, use what's in state if it's already there instead of re-requesting it)?

46 What is render hijacking in react?

The concept of render hijacking is the ability to control what a component will output from another component. It actually means that you decorate your component by wrapping it into a Higher-Order component. By wrapping you can inject additional props or make other changes, which can cause changing logic of rendering. It does not actually enables hijacking, but by using HOC you make your component behave in different way.

47 What logic will you write in componentWillMount?

componentWillMount() is invoked just before mounting occurs. The componentWillMount is place to handle configuration, update state, and in general prepare for the first render.

48 Code Splitting?

Bundling is great, but as your app grows, your bundle will grow too. Especially if you are including large third-party libraries. (You need to keep an eye on the code you are including in your bundle so that you don’t accidentally make it so large that your app takes a long time to load.)

To avoid winding up with a large bundle, it’s good (to get ahead of the problem and) start “splitting” your bundle. Code-Splitting is a feature supported by bundlers like Webpack and Browserify (via factor-bundle) which can create multiple bundles that can be dynamically loaded at runtime.

Code-splitting your app can help you “lazy-load” just the things that are currently needed by the user, which can dramatically improve the performance of your app. While you haven’t reduced the overall amount of code in your app, you’ve avoided loading code that the user may never need, and reduced the amount of code needed during the initial load.

49 Context

Context provides a way to pass data through the component tree without having to pass props down manually at every level.
In a typical React application, data is passed top-down (parent to child) via props, but this can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.
example:

```
// Context lets us pass a value deep into the component tree
// without explicitly threading it through every component.
// Create a context for the current theme (with "light" as the default).
const ThemeContext = React.createContext('light');

class App extends React.Component {
  render() {
    // Use a Provider to pass the current theme to the tree below.
    // Any component can read it, no matter how deep it is.
    // In this example, we're passing "dark" as the current value.
    return (
      <ThemeContext.Provider value="dark">
        <Toolbar />
      </ThemeContext.Provider>
    );
  }
}

// A component in the middle doesn't have to
// pass the theme down explicitly anymore.
function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

class ThemedButton extends React.Component {
  // Assign a contextType to read the current theme context.
  // React will find the closest theme Provider above and use its value.
  // In this example, the current theme is "dark".
  static contextType = ThemeContext;
  render() {
    return <Button theme={this.context} />;
  }
}
```

50 What is diff algorithm?

ReactJS using diff algorithm compares both the Virtual DOM to find the minimum number of steps to update the Real DOM. When diffing two trees, React first compares the two root elements. If the root elements have different types, React will tear down the old tree and build the new tree from scratch. When comparing two React DOM elements of the same type, React looks at the attributes of both, keeps the same DOM node, and only updates the changed attributes.

51 Controlled Component vs Uncontrolled Component

Controlled Component: it's value is controlled in React state

Uncontrolled Component: it's value is not controlled in React state, instead, the value is saved in the DOM. If you want to know the value of the input, you have to pull the value from DOM.

52 Render Props

The term “render prop” refers to a technique for sharing code between React components using a prop whose value is a function. Reusable React Components.
A component with a render prop takes a function that returns a React element and calls it instead of implementing its own render logic.

53 Server side rendering vs Client side rendering

Server-side rendering is the most common method for displaying information onto the screen. It works by converting HTML files in the server into usable information for the browser. Whenever you visit a website, your browser makes a request to the server that contains the contents of the website.
When developers talk about client-side rendering, they’re talking about rendering content in the browser using JavaScript. So instead of getting all of the content from the HTML document itself, you are getting a basic HTML document with a JavaScript file that will render the rest of the site using the browser.

54 Lifecycle method

constructor: gets called first, also call super and pass in props. initialize your state, set default values. bind this. But, use constructor is optional, class field, just initialize state.

getDerivedStateFromProps: during mounting, returns a state object based on the initial props.

render: Most important, return jsx component, the content that actually display on screen.
componentDidMount: after render for the first time, it is called, start AJAX call to load data.

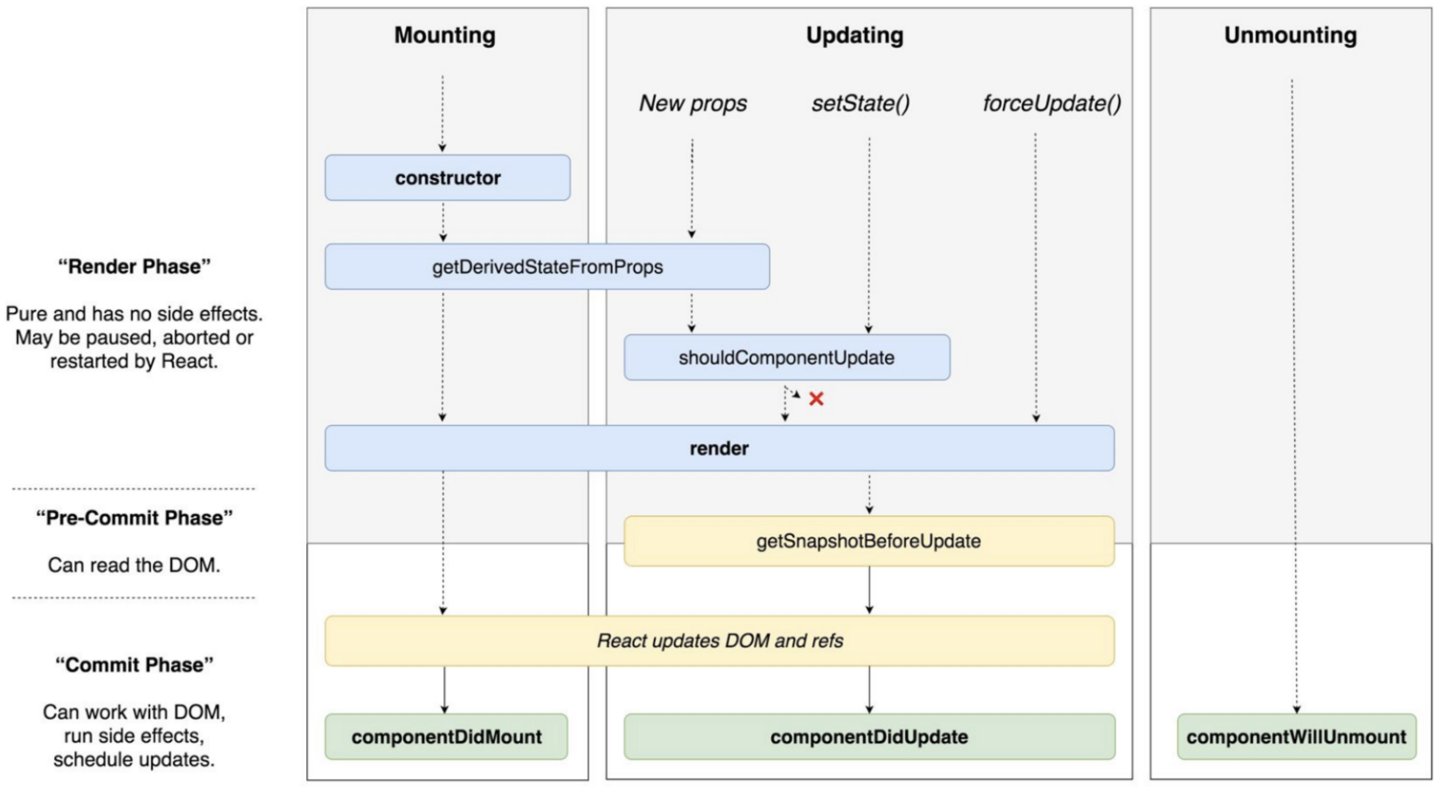
getDerivedStateFromProps: Updating state based on props when the props themselves aren’t enough.

shouldComponentUpdate: optimize the performance, (nextProps, nextState) {return bollean}, react will decide if the component will re-render.

render: the same

getSnapshotBeforeUpdate:
componentDidUpdate: committed upadates and changes to the DOM.

55 Lifecycle picture



56 What is dynamic import?

The dynamic import() syntax is a ECMAScript proposal not currently part of the language standard. It is expected to be accepted in the near future. You can achieve code-splitting into your app using dynamic import(). Let's take an example of addition,

i. Normal Import

```
import { add } from './math';
console.log(add(10, 20));
```

i. Dynamic Import

```
import("./math").then(math => {
  console.log(math.add(10, 20));
});
```

57 What are hooks?

Hooks are a new feature proposal that lets you use state and other React features without writing a class. Let's see an example of useState hook example,

```
import { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

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

Redux

1 What is Redux?

Redux is one of the hottest libraries for front-end development in today’s marketplace. It is a predictable state container for JavaScript applications and is used for the entire applications state management. Applications developed with Redux are easy to test and can run in different environments showing consistent behavior.

2 How are Actions defined in Redux?

Actions in React must have a type property that indicates the type of ACTION being performed. They must be defined as a String constant and you can add more properties to it as well. In Redux, actions are created using the functions called Action Creators. Below is an example of Action and Action Creator:

```
function addTodo(text) {
      return {
              type: ADD_TODO,
               text
   }
}
```

3 Explain the role of Reducers?

Reducers are pure functions which specify how the application's state changes in response to an ACTION. Reducers work by taking in the previous state and action, and then it returns a new state. It determines what sort of update needs to be done based on the type of the action, and then returns new values. If no work needs to be done, it returns the previous state as it is,

4 What is the significance of Store in Redux?

A store is a JavaScript object which can hold the application's state and provide a few helper methods to access the state, dispatch actions and register listeners. The entire state/ object tree of an application is saved in a single store. As a result of this, Redux is very simple and predictable. We can pass middleware to the store to handle the processing of data as well as to keep a log of various actions that change the state of stores. All the actions return a new state via reducers.

5 How is Redux different from Flux?

Flux:
1.The store contains state and change logic. 2.There are multiple stores 3.All the stores are disconnected and flat. 4.Has singleton dispatcher 5.React components subscribe to the store 6.State is mutable

Redux: 1. Store and change logic are separate. 2. There is only one store 3. Single store with hierarchical reducers 4. No concept of dispatcher 5. Container components utilize connect 6. State is immutable

6 What are the advantages of Redux?

- Predictability of outcome- since there is always one source of truth, i.e. the store, there is no confusion about how to sync the current state with actions and other parts of the application.
- Maintainability- The code becomes easier to maintain with a predictable outcome and strict structure.
- Server-side rendering - You just need to pass the store created on the server, to the client side. This is very useful for initial redner and provides a better user experience as it optimizes the application performance.
- Developer tools - From actions to state changes, developers can track everything going on in the application in real time.
- Community and ecosystem - Redux has a huge community behind it which makes it even more captivating to use. A large community of talented individuals contributes to the betterment of the library and develop various development with it.
- Ease of testing - Redux's code is mostly functions which are small. pure and isolated. This makes the code testable and independent.
- Organization - Redux is precise about how code should be organized, this makes the code more consistent d easier when a team works with it.

7 What are the main features of Reselect library?

- Selectors can compute derived data, allowing Redux to store the minimal possible state.
- Selectors are efficient. A selector is not recomputed unless one of its arguments changes.
- Selectors are composable. They can be used as input to other selectors.

What are Redux selectors and why to use them?

Selectors are functions that take Redux state as an argument and return some data to pass to the component.

For example, to get user details from the state:

```
const getUserData = state => state.user.data
```

8 How to set initial state in Redux?

You need to pass initial state as second argument to createStore:

```
const rootReducer = combineReducers({
  todos: todos,
  visibilityFilter: visibilityFilter
})

const initialState = {
  todos: [{ id: 123, name: 'example', completed: false }]
}

const store = createStore(
  rootReducer,
  initialState
)
```

9 What is redux-saga?

redux-saga is a library that aims to make side effects (asynchronous things like data fetching and impure things like accessing the browser cache) in React/Redux applications easier and better.

It is available in NPM:

```
$ npm install --save redux-saga
```

10 What is Redux Thunk?

Redux Thunk middleware allows you to write action creators that return a function instead of an action. The thunk can be used to delay the dispatch of an action, or to dispatch only if a certain condition is met. The inner function receives the store methods dispatch() and getState() as parameters.

11 What are the differences between redux-saga and redux-thunk?

Both Redux Thunk and Redux Saga take care of dealing with side effects. In most of the scenarios, Thunk uses Promises to deal with them, whereas Saga uses Generators. Thunk is simple to use and Promises are familiar to many developers, Sagas/Generators are more powerful but you will need to learn them. But both middleware can coexist, so you can start with Thunks and introduce Sagas when/if you need them.

12 What are the different ways to write mapDispatchToProps()?

There are a few ways of binding action creators to dispatch() in mapDispatchToProps(). Below are the possible options:

```
const mapDispatchToProps = (dispatch) => ({
 action: () => dispatch(action())
})

const mapDispatchToProps = (dispatch) => ({
 action: bindActionCreators(action, dispatch)
})

const mapDispatchToProps = { action }
```

The third option is just a shorthand for the first one.

13 What is the difference between component and container in React Redux?

Component is a class or function component that describes the presentational part of your application.

Container is an informal term for a component that is connected to a Redux store. Containers subscribe to Redux state updates and dispatch actions, and they usually don't render DOM elements; they delegate rendering to presentational child components.

14 What is the difference between React context and React Redux?

You can use Context in your application directly and is going to be great for passing down data to deeply nested components which what it was designed for. Whereas Redux is much more powerful and provides a large number of features that the Context API doesn't provide. Also, React Redux uses context internally but it doesn't expose this fact in the public API.

15 How to make AJAX request in Redux?

You can use redux-thunk middleware which allows you to define async actions.
Let's take an example of fetching specific account as an AJAX call using fetch API:

```
export function fetchAccount(id) {
  return dispatch => {
    dispatch(setLoadingAccountState()) // Show a loading spinner
    fetch(`/account/${id}`, (response) => {
      dispatch(doneFetchingAccount()) // Hide loading spinner
      if (response.status === 200) {
        dispatch(setAccount(response.json)) // Use a normal function to set the received state
      } else {
        dispatch(someError)
      }
    })
  }
}

function setAccount(data) {
 return { type: 'SET_Account', data: data }
}
```

16 What are the core principles of Redux?

Redux follows three fundamental principles:

- Single source of truth: The state of your whole application is stored in an object tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.
- State is read-only: The only way to change the state is to emit an action, an object describing what happened. This ensures that neither the views nor the network callbacks will ever write directly to the state.
- Changes are made with pure functions: To specify how the state tree is transformed by actions, you write reducers. Reducers are just pure functions that take the previous state and an action as parameters, and return the next state.

17 Can I dispatch an action in reducer?

Dispatching an action within a reducer is an anti-pattern. Your reducer should be without side effects, simply digesting the action payload and returning a new state object. Adding listeners and dispatching actions within the reducer can lead to chained actions and other side effects.

React-Redux

How to integrate redux with react component?

- Install react-redux library
- Design presentational components and container components

Generate container components with the React Redux library's connect() function
Use the react redux component called Provider to wrap the root component, to make the store available to all container components without passing it explicitly.

- To use connect(), you need to define a special function called mapStateToProps that describes how to transform the current Redux store state into the props you want to pass to a presentational component you are wrapping. you can define a function called mapDispatchToProps that receives the dispatch() method and returns callback props that you want to inject into the presentational component.

React-Router

1 What is React Router?

React Router is a powerful routing library on top of React, which helps in adding new screens and flows to the application. This keeps the URL in sync with data that's being displayed on the web page. It maintained a standardized structure and behavior and is used for developing single page web applications. React Router has a simple API.

2 Why is switch keyword used in react Router v4?

Although a div is used to encapsulate multiple routers inside the Router. The 'switch' is used when you want to display only a single route to be rendered amongst the several defined routes. The 'switch' tag when in use matches the typed URL with the defined routes in sequential order. When the first match is found, it renders the specified route. Thereby bypassing the remaining routes.

3 Why do you need a Router in React?

A Router is used to define multiple routers and when a user types a specific URL. if the URL matches the path of any 'route' defined inside the router, then the user is redirected to that particular route. So basically, we need to add a Router library to our app that allows creating multiple routes with each leading us to a unique view.

```
<switch>
   <route exact path=’/’ component={Home}/>
   <route path=’/posts/:id’ component={Newpost}/>
```

```
    <route path='/posts'   component={Post}/>
</switch>
```

4 What are the advantages of React Router?

1.Just like how React is based on components, in React Router v4, the API is 'All About Components'. A Router can be visualized as a single root component (BrowserRouter) in which we enclose the specific child routes(route)
2.No need to manually set History value: In React Router v4, all we need to do is wrap our routes within the BrowserRouter component.
3.The packages are split: Three packages one each for Web, Native and Core. This supports the compact size of our application.It is easy to switch over based on a similar coding style.

5. BrowserRouter vs HashRouter

BrowserRouter
It uses history API, i.e. it's unavailable for legacy browsers (IE 9 and lower and contemporaries). Client-side React application is able to maintain clean routes like example.com/react/route but needs to be backed by web server. Usually this means that web server should be configured for single-page application, i.e. same index.html is served for /react/route path or any other route on server side. On client side, window.location.pathname is parsed by React router. React router renders a component that it was configured to render for /react/route.
Additionally, the setup may involve server-side rendering, index.html may contain rendered components or data that are specific to current route.
HashRouter
It uses URL hash, it puts no limitations on supported browsers or web server. Server-side routing is independent from client-side routing.
Backward-compatible single-page application can use it as example.com/#/react/route.
The setup cannot be backed up by server-side rendering because it's / path that is served on server side, #/react/route URL hash cannot be read from server side. On client side, window.location.hash is parsed by React router. React router renders a component that it was configured to render for /react/route, similarly to BrowserRouter.
Most importantly, HashRouter use cases aren't limited to SPA. A website may have legacy or search engine-friendly server-side routing, while React application may be a widget that maintains its state in URL like example.com/server/side/route#/react/route. Some page that contains React application is served on server side for /server/side/route, then on

client side React router renders a component that it was configured to render for /react/ route, similarly to previous scenario.

Restful api

Web service APIs that adhere to the REST architectural constraints are called RESTful APIs. HTTP-based RESTful APIs are defined with the following aspects:

- a base URL, such as http://api.example.com/resources;
- a media type that defines state transition data elements (e.g., Atom, microformats, application/vnd.collection+json). The current representation tells the client how to compose requests for transitions to all the next available application states. This could be as simple as a URL or as complex as a Java applet;
- standard HTTP methods (e.g., OPTIONS, GET, PUT, POST, and DELETE).

Uniform Resource Locator(URL)

Collection, such as https://api.example.com/resources/

- GET List the URIs and perhaps other details of the collection's members.
- PUT Replace the entire collection with another collection.
- PATCH Not generally used
- POST Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation
- DELETE Delete the entire collection

Element, such as https://api.example.com/resources/item17

- GET Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.
- PUT Replace the addressed member of the collection, or if it does not exist, create it.
- PATCH Update the addressed member of the collection.
- POST Not generally used. Treat the addressed member as a collection in its own right and create a new entry within it.
- DELETE Delete the addressed member of the collection.

Unlike SOAP-based Web services, there is no "official" standard for RESTful Web APIs. This is because REST is an architectural style, while SOAP is a protocol. REST is not a standard in itself, but RESTful implementations make use of standards, such as HTTP, URI, JSON, and XML.

Webpack

What is webpack?

webpack is a module bundler for javascript applications. Webpack recursively builds every module in your application, then packs all those modules into a small number of bundles.

What is a bundle in webpack?

Bundle is the output file generated by webpack. It contains all of the modules which are used in application. Bundles generation process is regulated by webpack config file.

In which environment does webpack work?

node.js

What is an entry point?

The entry object is where webpack looks to start building the bundle, at this point the application starts executing.

What is a dependency graph and how does webpack build it?

Any time one file depends on another, webpack treats this as a dependency. Starting from entry points, webpack recursively builds a dependency graph that includes every module your application needs, using import and require statements, then packages all of those modules into bundle.

Coding Questions:

1.Explain how the value can be truthy, and yet not loosely equal to true.

```
var a = 13;
var b = true;
if (a) {
console.log (Boolean(a)); //true
}
console.log (a == b); // false
```

First one because a is a Not Zero number, so the Boolean of a is true.
Second one because as a number only "1 == true" is true. For double equal condition, when other number except 0 and 1 compare with Boolean value. In JS firstly it will transform Boolean to number. So in this question true will transform to number 1 and of course it is not equal to number 13. So the result is false.

2.The following utility function extends an object by adding properties and methods from an extension object. Unfortunately, extend has a bug. Can you spot it?

```
var extend = function (obj, extension) {
if (typeof obj === "object" &&typeof extension === "object") {
for (var i in extension) {
if (extension.hasOwnProperty(i) &&
!obj.hasOwnProperty(i))
{
obj[i] = extension[i];
}
}
return obj;
}
}
```

Bug:

1: "hasOwnProperty()" will only check the properties in current object. If we also want to check the properties in its parent's object, we have to code like this: "i in extension".

2: If extension is not an object, the console is undefined. So here we add an else condition: else{ return obj; }
So if the extension is not an object, it will return the original obj object.

3.if obj or extension is an array, it will add the index of the array in the object, it is not right, we should change typeof obj === 'object' to obj.constructor === Object

When should you use JSON Web Tokens?

Here are some scenarios where JSON Web Tokens are useful:
Authorization: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.
(Single sign-on is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to a connected system or accomplished using the Lightweight Directory Access Protocol and stored LDAP databases on servers.)
Information Exchange: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

Can we caching redux state when close tags?

When close the tag, send a request to backend and store the state in the database. Everytime we bootstraps the application, we call the backend service to get the state using componentDidMount().
We can use local storage to keep the state when close the tags. But we can not use the session storage because it expires once the browser closes. Both storages work the same by using key value pairs.

Talking Project problem the tech lead met in Project (mainly about render optimization)

Bundle size is too large (react-loadable)

We can use code splitting, which is a feature of webpack. This feature allows you to split your code into different bundles which can then be loaded on demand. For example, we can implement the route-based code splitting with react-loadable library, in order to load the code for the different routes on-demand.

Long List (react-window)

renders long lists of data such as hundreds or thousands of rows, we can use a technique known as “windowing”. This technique only renders a small subset of your rows at any given time, and can reduce the time it takes to re-render the components. React-window

Large image

we can use task runner to compress the image. Also we can use srcset and sizes attribute of img tag to implement responsive images, which means provide several source images along sizes to let the browser pick the right one

Too many useless re-render

Use shouldComponentUpdate with PureComponent Add keys Reselect?

What is the difference between one-way data binding and two-way data binding?

Why one-way data binding is better than two-way data binding?

Two way data binding means that UI fields are bound to model data dynamically such that when a UI field changes, the model data changes with it and vice-versa.
One way data flow means that the model is the single source of truth.
Changes in the UI trigger messages that signal user intent to the model (or “store” in React). Only the model has the access to change the app’s state. The effect is that data always flows in a single direction, which makes it easier to understand. One way data flows are deterministic, whereas two-way binding can cause side-effects which are harder to follow and understand.

How do you unit test your axios call?

How do you unit test your axios call?

Behavior Questions?

What is the biggest challenge? (How to solve it?)

Node

Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Code

```
console.log('hello'.repeatify(3));
```

Should print hellohellohello

Answer

A possible implementation is shown below:

```
String.prototype.repeatify = function (howManyTimes) {
        var repeatedString = '';

  for (var i =0; i < howManyTimes; i++) {
        repeatedString += this.toString();
  }

        return repeatedString;
}



console.log('hello'.repeatify(3));
```

Unit Test

100% coverage 100%

150-200 components 1000多个test

advantages to write Unit test: n a team different branch, 改动可能使其他branch test 不通过

PWA
AMP Accelerated Mobile Pages
electron
mobile: react-native framework ionic
Mobx

AMD and CommonJS

AMD stands for "Asynchronous Module Definition" used in browser, it will not block our browser. Asynchronous module definition (AMD) is a specification for the programming language JavaScript. It defines an application programming interface (API) that defines code module and their dependencies, and loads them asynchronously if desired
require.js use AMD
CommonJS used in server, node.js

HTML and CSS

1.Positions

fixed: relative to browser, which means it always stays in the same place even if the page is scrolled.

relative: relative to it's original(normal) position

absolute: relative to its non-static parent, if can't find, it will relative to the browser. It will find the most closed parents element that is non-static. Absolute positioned elements are removed from the normal flow, which means it can be overlapped by other elements.

static: default value. Elements render in order, as they appear in the document flow. Not affected by the top, bottom, left, right and z-index properties. Other content will not be adjusted to fit into any gap left by the element.

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. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

2 Float

The elements after the floating element will flow around it. The elements before the floating element will not be affected. If an image is floated to the right, a following text flows around it, to the left. The float CSS property places an element on the left or right side of its container, allowing text and inline elements to wrap around it. The element is removed from the normal flow of the page, though still remaining a part of the flow

3 Avoid visual differences in different browsers

Reset the body margin and padding at the very first of the css style sheet: (asterisk) `*{margin:0; padding:0}`

4 Combinators in CSS3:

Combinator: explains the relationship between the selectors.

Descendant selector: matches all elements that are descendants of a specified element. div p{ color:black;} (All p in the div)

Child selector: selects all elements that are the immediate children of a specified element. div > p{...}

Adjacent sibling selector: selects all elements that are the adjacent siblings of a element. Adjacent means the immediately following. div + p{...}

General sibling selector: selects all elements that are siblings of a specified element. div ~ p{...}

5 Pseudo selectors

Used to define a special state of an element. eg. a:hover{...} means do the set when hover over the link.

active focus hover link

A pseudo-class is used to define a special state of an element.

- :active a: active Selects the active link
- :checked   input:checked Selects every checked input element
- :disabled input: disabled Selects every disabled input element
- :empty p: empty Selects every p element that has no children
- :enabled input: enabled Selects every enabled input element
- :focus input: focus Selects the input element that has focus
- :hover a: hover Selects links on mouse over
- :in-range input: in-range Selects input elements with a value within a specified range

6 Pseudo-element

A CSS pseudo-element is used to style specified parts of an element.

Used to style specified parts of an element. eg. p::first-letter{...}

```
p::after, insert after the content of each <p> element
p::before, insert before the content of each<p> element
p::first-letter, selects the first letter of each <p> element
p::first-line, selects the first line of each <p> elemtn
p::selection, selects the portion of an element that is selected by a user
```

7 What is the difference between rem and em?

rem means the foot-size is relative to the root or the html, it means you can define a font-size in root, then you can use rem, 1rem is the size that you define. em means the size is relative to its parent.

1 rem = 16 px

8 What is the value for box-sizing CSS property?

(Box-Model): The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The CSS box model describes the rectangular boxes. Each box has a content area and optional surrounding padding, border, and margin areas.

the default boxing-sizing is content-box, we often use border-box instead.

In the border-box model, width and height including the size of the content and also the padding and border.

9 Specificity(Priority)

Specificity means a browser decides which property values are the most relevant to an element and gets to be applied. CSS styles follow an order of specificity. It specifies when styles override another or take precedence.

CSS styles follow an order of specificity. It specifies when styles override another or take precedence.

- Inline css is higher than style tag
- style tag is higher than css introduced by the tag
- Id selector is higher than class and then higher than tag name
- Tow color properties in the same curly brakets, the later one will override the previous one.
- You can use "!important" statement to make the highest priority specificity.
- Inherited css has the lowest priority.(Inherited property uses the property value from its parent property)

Inline style > id > class == Attribute == pseudo class > simple element

10 CSS reset

A CSS Reset is a short, compressed set of CSS rules that resets the styling of all HTML elements to a consist baseline. Using a CSS RESET, css authors can force every browser to have all its styles reset to null, thus avoiding cross-browser differences as much as possible.

11 CSS Performances

repaint: it occurs when changes are made to an element skin that changes visibility, but do not affect its layout.(eg. outline, visibility, background color)

reflow: it involves changes that affect the layout of a portion of the page. It is expensive and will slow the performance because most of the time it means to re-render the whole page. Resize window, change font, add or remove a stylesheet, content changes, activation of CSS pseudo classes and many more will cause the reflow.

12 Overflow

It specifies what happens if content overlfows an element's box:

```
visiable, hidden, scroll, auto, initial, inherit
```

The CSS overflow property controls what happens to content that is too big to fit into an area

The overflow property specifies whether to clip content or to add scrollbars when the content of an element is too big to fit in a specified area.

The overflow property has the following values:

- visible - Default. The overflow is not clipped. It renders outside the element's box
- hidden - The overflow is clipped, and the rest of the content will be invisible
- scroll - The overflow is clipped, but a scrollbar is added to see the rest of the content. Note that this will add a scrollbar both horizontally and vertically
- auto - If overflow is clipped, a scrollbar should be added to see the rest of the content

The overflow property only works for block elements with a specified height.

In OS X Lion (on Mac), scrollbars are hidden by default and only shown when being used (even though "overflow:scroll" is set).

13 What is the difference between SVG and Canvas?

https://www.sitepoint.com/how-to-choose-between-canvas-and-svg/

SVG stands for Scalable Vector Graphics

SVG is used to define graphics for the Web

SVG is a W3C recommendation

SVG is a document format for scalable vector graphics.

Canvas is a javascript API for drawing vector graphics to a bitmap of a specific size.

To elaborate a bit, on format versus API:

With svg you can view, save and edit the file in many different tools. With canvas you just draw, and nothing is retained about what you just did apart from the resulting image on the screen. You can animate both, SVG handles the redrawing for you by just looking at the elements and attributes specified, while with canvas you have to redraw each frame yourself using the API. You can scale both, but SVG does it automatically, while with canvas again, you have to re-issue the drawing commands for the given size.

14 visibility hidden vs display:none

display none:don't take space. remove from DOM

visibility hidden:take space because it's still in DOM

Property	Occupies Place	Click Event
Opacity:0	yes	yes
visibility:hidden	yes	no
display:none	no	yes

15 What's new in HTML5 ?

semantic elements like (header nav footer section article)

inline-element: time meter progress

new exciting element: video audio canvas new form type: date number email

16 Would you like to talk about responsive web design?

Responsive websites use media queries, flexible grids, and responsive images to create a user experience that flexes and changes based on a multitude of factors.

Media query is a CSS technique introduced in CSS3.

It uses the @media rule to include a block of CSS properties only if a certain condition is true.

Media queries can help with that. We can add a breakpoint where certain parts of the design will behave differently on each side of the breakpoint.

17 flexbox

The Flexbox Layout officially called CSS Flexible Box Layout Module is new layout module in CSS3 made to improve the items align, directions and order in the container even when they are with dynamic or even unknown size. The prime characteristic of the flex container is the ability to modify the width or height of its children to fill the available space in the best possible way on different screen sizes.

Flexbox is a one-dimensional layout method for laying out items in rows or columns. Items flex to fill additional space and shrink to fit into smaller spaces.

flex: It specifies the length of the item, relative to the rest of the flexible items inside the same container. The flex property is a shorthand for the flex-grow, flex-shrink, and flex-basis properties. If the element is not a flexible item, the flex property has no effect.(display:flex)

flex-direction flex-wrap:

You can combine the two properties flex-direction and flex-wrap into the flex-flow shorthand. The first value specified is flex-direction and the second value is flex-wrap.

justify-content: The justify-content property is used to align the items on the main axis, the direction in which flex-direction has set the flow.

- flex-start
- flex-end
- center
- space-around
- space-between
- space-evenly

align-items:

18 Margin and Padding

Margin is the outer space of an element, while padding is the inner space of an element. In other words, margin is the space outside of an element's border, while padding is the space inside of its border.

Margin:

```
margin: top right bottom left (margin: 25px 50px 75px 100px;)
margin: top right_and_left bottom( margin: 25px 50px 75px;)
margin: top_bottom right_left (margin: 25px 50px;)
margin: top_bottom_right_left (margin: 25px;)

margin: 0 auto (0 for top and bottom margin, and left and right margin should be equal)
```

19 HTML Attributes?

· All HTML elements can have attributes
· Attributes provide additional information about an element
· Attributes are always specified in the start tag
· Attributes usually come in name/value pairs like: name="value"

The alt Attribute

The alt attribute specifies an alternative text to be used, when an image cannot be displayed. The value of the attribute can be read by screen readers.

The style Attribute

The style attribute is used to specify the styling of an element, like color, font, size etc.

The title Attribute

Here, a title attribute is added to the

element. The value of the title attribute will be displayed as a tooltip when you mouse over the element.

id

Specifies a unique id for an element

src

Specifies the URL (web address) for an image

20 Use CSS to hide HTML element:

(1). visibility: hidden

(2). opacity: 0

(3). display:none

(4). position:absolute, then adjust the top, left, bottom, right properties

21 HTML Accessibility

Make your HTML code as semantic as possible, so that the code is easy to understand for visitors and screen readers.

Semantic HTML

Semantic HTML gives context to screen readers, which read the contents of a web page out loud.

Examples of semantic elements: form, table, and article - Clearly defines its content.

Alternative Text

The alt attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

The value of the alt attribute should describe the image:

If a browser cannot find an image, it will display the value of the alt attribute

Link Titles

The title attribute specifies extra information about an element. The information is most often shown as a tooltip text when the mouse moves over the element.

Declare the Language

Declaring a language is important for screen readers and search engines, and is declared with the lang attribute. Use the following to show a web page in English

22 Bootstrap Grid System

Bootstrap's grid system allows up to 12 columns across the page.

Bootstrap grid system is robust flexbox grid to build responsive layouts of all shapes and sizes.

Bootstrap’s grid system uses a set of containers, rows, and columns to layout and align content. It’s developed with flexbox and is fully responsive. Below is a Bootstrap grid system example and an in-depth look at how the grid comes together.

Grid Classes

The Bootstrap grid system has four classes:

- xs (for phones - screens less than 768px wide)
- sm (for tablets - screens equal to or greater than 768px wide)
- md (for small laptops - screens equal to or greater than 992px wide)
- lg (for laptops and desktops - screens equal to or greater than 1200px wide)

23 ID and Classes

ID's are unique

· Each element can have only one ID
· Each page can have only one element with that ID

Classes are NOT unique

· You can use the same class on multiple elements.
· You can use multiple classes on the same element.

24 3 way to add css

There are three ways of inserting a style sheet:

· External style sheet

Each page must include a reference to the external style sheet file inside the link element. The link element goes inside the head section:

```
Example
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

· Internal style sheet

An internal style sheet may be used if one single page has a unique style.

Internal styles are defined within the style element, inside the head section of an HTML page

· Inline style

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

What style will be used when there is more than one style specified for an HTML element?

All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1.Inline style (inside an HTML element)

2.External and internal style sheets (in the head section)

3.Browser default

So, an inline style has the highest priority, and will override external and internal styles and browser defaults.

25 Difference between span&div?

div is a block element, span is inline element.

This means that to use them semantically, divs should be used to wrap sections of a document, while spans should be used to wrap small portions of text, images, etc.

block elements

· p
· h1, h2, h3, h4, h5, h6
· ol, ul
· pre
· address
· blockquote
· dl
· div
· fieldset
· form
· hr
· noscript
· table

incline elements

· b, big, i, small, tt
· abbr, acronym, cite, code, dfn, em, kbd, strong, samp, var
· a, bdo, br, img, map, object, q, script, span, sub, sup
· button, input, label, select, textarea

26 Table benefits? Bad way to use table.

most screen readers read web pages in the order that they are displayed in the HTML, and tables can be very hard for screen readers to parse.

Many table designs don't print well because they are simply too wide for the printer.

not good in layout

27 Sass

CSS on its own can be fun, but stylesheets are getting larger, more complex, and harder to maintain. This is where a preprocessor can help. Sass lets you use features that don't exist in CSS yet like variables, nesting, mixins, inheritance and other nifty goodies that make writing CSS fun again.

Once Sass is installed, you can compile your Sass to CSS using the sass command. You'll need to tell Sass which file to build from, and where to output CSS to. For example, running sass input.scss output.css from your terminal would take a single Sass file, input.scss, and compile that file to output.css.

Variables

Think of variables as a way to store information that you want to reuse throughout your stylesheet. You can store things like colors, font stacks, or any CSS value you think you'll want to reuse. Sass uses the $ symbol to make something a variable. Here's an example:

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

When the Sass is processed, it takes the variables we define for the $font-stack and $primary-color and outputs normal CSS with our variable values placed in the CSS. This can be extremely powerful when working with brand colors and keeping them consistent throughout the site.

Nesting

When writing HTML you've probably noticed that it has a clear nested and visual hierarchy. CSS, on the other hand, doesn't.

Import

CSS has an import option that lets you split your CSS into smaller, more maintainable portions. The only drawback is that each time you use @import in CSS it creates another HTTP request. Sass builds on top of the current CSS @import but instead of requiring an HTTP request, Sass will take the file that you want to import and combine it with the file you're importing into so you can serve a single CSS file to the web browser.

Mixins

Some things in CSS are a bit tedious to write, especially with CSS3 and the many vendor prefixes that exist. A mixin lets you make groups of CSS declarations that you want to reuse throughout your site. You can even pass in values to make your mixin more flexible. A good use of a mixin is for vendor prefixes. Here's an example for transform.

(property 有dollar符号)

```
@mixin transform(property) {
  -webkit-transform: property;
  -ms-transform: property;
  transform: property;
}

.box { @include transform(rotate(30deg)); }
```

Extend/Inheritance

This is one of the most useful features of Sass. Using @extend lets you share a set of CSS properties from one selector to another.It helps keep your Sass very DRY. In our example we're going to create a simple series of messaging for errors, warnings and successes using another feature which goes hand in hand with extend, placeholder classes. A placeholder class is a special type of class that only prints when it is extended, and can help keep your compiled CSS neat and clean.

```
/* This CSS will print because %message-shared is extended. */
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

// This CSS won't print because %equal-heights is never extended.
%equal-heights {
  display: flex;
  flex-wrap: wrap;
}

.message {
  @extend %message-shared;
}

.success {
  @extend %message-shared;
  border-color: green;
}

.error {
  @extend %message-shared;
  border-color: red;
}

.warning {
  @extend %message-shared;
  border-color: yellow;
}
```

Operators

Doing math in your CSS is very helpful. Sass has a handful of standard math operators like +, -, *, /, and %. In our example we're going to do some simple math to calculate widths for an aside & article.

```
.container {
  width: 100%;
}

article[role="main"] {
  float: left;
  width: 600px / 960px * 100%;
}
```

```
aside[role="complementary"] {
  float: right;
  width: 300px / 960px * 100%;
}
```

28.Optimize css:

(1). No more than one external CSS stylesheet

(2). Inline small CSS into HTML using style tags for above the fold content

(3). No @IMPORT CALLS FOR CSS

29.Have you heard about progressive web app?

Progressive Web App (PWA) is a term used to denote a new software development methodology. Unlike traditional applications, progressive web apps are a hybrid of regular web pages and a mobile application.This new application model attempts to combine features offered by most modern browsers with the benefits of mobile experience.

https://en.wikipedia.org/wiki/Progressive_web_applications

https://developers.google.com/web/progressive-web-apps/

30 CSS3 web fonts:the @font-face rule

Web fonts allow developers to use fonts that are not installed on the user's computer. When you have found/bought the font you want to use, just include the font file on your web server, and it will be automatically downloaded to the user when needed.

```
@font-face {font-family: myFirstFont; src: url(sansation_light.woff);}
div{font-family: myFirstFont;}
```

31 @import:

The @import CSS at-rule allows to import style rules from other style sheets`eg. @import url; @import url list-of-media-queries;`

The @import can make your css code clearer, however, it slows down your application speed because it causes files to load sequentially instead of in parallel. This wastes times and round trips and makes your web page load slower. You can use a stylesheet link to avoid the @import

32 Image Sprites

It is a collection of images put into a single image.

A web page with many images can take a long time to load and generates multiple server requests. With image sprites, it will reduce the number of server requests and save bandwidth. And you can use the width, height and other properties to choose a part of the combined image to form the needed image. With CSS, we can show just the part of the image we need.

33 Background image size:

Before CSS3, the size was determined by the actual size of the image. In CSS3, it is possible to specify the size of the background image, which allows us to reuse background images in different contexts.

34 IE, firefox render the box model

Firefox and other standards compliant browsers will add the size of the padding to the width or height of the box, while IE will place the padding inside of the box without adjusting the width or height.

Solution: Place one element inside of a container, where container does not have any padding but has a width defined, and the inside element does have padding defined

35.Text-align

Justify, each line is stretched so that every line has equal width, and the left and right margins are straight.

36.What is meant by cascade in CSS

The cascade is a fundamental feature of CSS. It is an algorithm defining how to combine properties values originating from different sources. It defines the order of precedence for how conflicting styles should be applied

37. make scroll smoother

Limit the minimum execution interval time of ONSCROLL event

The onscroll event trigger interval time is around 10~20ms when we scrolling the mouse wheel. However, in most cases we don't need the onscroll event to execute such frequently. We can use following code to set a minimum execution interval time for onscroll event.

1.Performance

web page response time can be improved by 25 to 50 percent by following these rules:

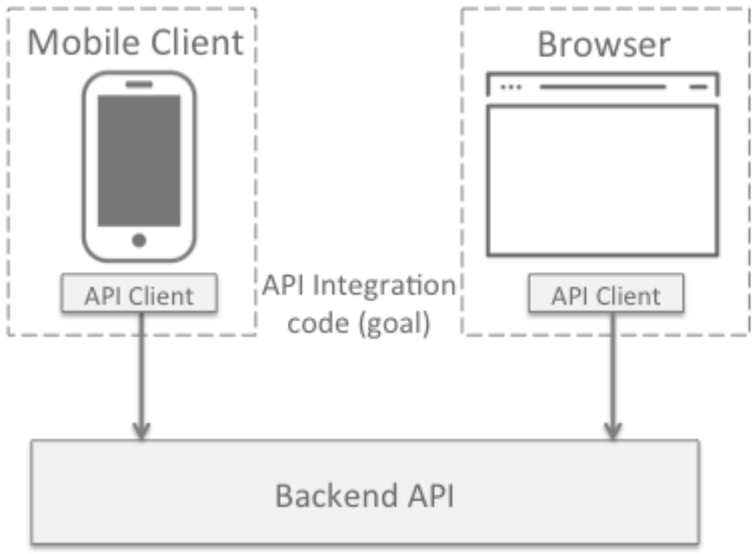
Minimize HTTP requests.
Use a content delivery network.
Add an Expires or a Cache-Control header.
Gzip components.
Put stylesheets at the top.
Put scripts at the bottom.
Avoid CSS expressions.
Make JavaScript and CSS external.
Reduce DNS lookups.
Minify JavaScript and CSS.

Avoid redirects.
Remove duplicate scripts.
Configure ETags.
Make Ajax cacheable.
Use GET for Ajax requests.
Reduce the number of DOM elements.
Eliminate 404s.
Reduce cookie size.
Use cookie-free domains for components.
Avoid filters.
Do not scale images in HTML.
Make favicon.ico small and cacheable.

2.Back end team vs. front end team issue

Conflict resolution through layered API design

The hope is that our integration between API clients and server will look like the following:



As you may have noticed, this isn’t always the case. Instead, a few issues may arise that are at the root of this conflict:

The frontend team want to reduce the burden of coding by requiring that payloads fit the UI design and limit API calls where appropriate

Backend developers want to optimize the endpoints for both reuse and rapid development

To resolve this conflict, we need to find a way to support both parties. A common approach to doing this is by using layered API design. Within the microservices space, this is commonly known as the Backend For Frontend (BFF) pattern.

The BFF pattern allows the backend API to remain as-is, with a new API constructed and consumed by frontend developers to meet the needs of the UI. There are two primary approaches to use the BFF pattern:

3.RESTful API time out solution

You can set REST API call-out time to maximum 120 second. If your .NET application is unable to process request in 120 seconds then you will have to optimize .net webservice. Default timeout is 10 seconds.
Sample:

```
    HttpRequest req = new HttpRequest();
    req.setEndpoint(endpoint_url);mvc
    req.setMethod('GET');
    req.setTimeout(120000);
```

4. Software Design Pattern

MVC, MVP, and MVVM are three popular design patterns in software development.
MVC design pattern divides an application into three major aspects: Model, View, and Controller.

Model: Model means data that is required to display in the view. Model represents a collection of classes that describes the business logic (business model and the data model). It also defines the business rules for data means as how the data can be changed and manipulated.

View: The View represents UI components like XML, HTML etc. View displays the data that is received from the controller as the outcome. In MVC pattern View monitors the model for any state change and displays updated model. Model and View interact with each other using the Observer pattern.

Controller: The Controller is responsible to process incoming requests. It processes the user’s data through the Model and passing back the results to View. It normally acts as a mediator between the View and the Model.

1、js 统计一个字符串出现频率最高的字母/数字

```
let str = 'asdfghjklaqwertyuiopiaia';
const strChar = str => {
  let string = [...str],
    maxValue = '',
    obj = {},
    max = 0;
  string.forEach(value => {
    obj[value] = obj[value] == undefined ? 1 : obj[value] + 1
    if (obj[value] > max) {
      max = obj[value]
      maxValue = value
    }
  })
return maxValue;
}
console.log(strChar(str))    // a
```

复制代码

2、数组去重

2.1、forEach

```
let arr = ['1', '2', '3', '1', 'a', 'b', 'b']
const unique = arr => {
   let obj = {}
   arr.forEach(value => {
      obj[value] = 0
   })
   return Object.keys(obj)
}
console.log(unique(arr))  // ['1','2','3','a','b']
```
复制代码

2.2、filter

```
let arr = ['1', '2', '3', '1', 'a', 'b', 'b']
const unique = arr => {
   return arr.filter((ele, index, array) => {
      return index === array.indexOf(ele)
   })
}
console.log(unique(arr))  // ['1','2','3','a','b']
```
复制代码

2.3、set

```
let arr = ['1', '2', '3', '1', 'a', 'b', 'b']
const unique = arr => {
   return [...new Set(arr)]
}
console.log(unique(arr))  // ['1','2','3','a','b']
```
复制代码

2.4、reduce

```
let arr = ['1', '2', '3', '1', 'a', 'b', 'b']
const unique = arr.reduce((map, item) => {
   map[item] = 0
   return map
}, {})
console.log(Object.keys(unique))  // ['1','2','3','a','b']
```
复制代码

3、翻转字符串

```
let str ="Hello Dog";
const reverseString = str =>{
   return [...str].reverse().join("");
}
console.log(reverseString(str))   // goD olleH
```
复制代码

4、数组中最大差值

4.1、forEach

```
let arr = [23, 4, 5, 2, 4, 5, 6, 6, 71, -3];
const difference = arr => {
   let min = arr[0],
      max = 0;
   arr.forEach(value => {
      if (value < min) min = value
      if (value > max) max = value
   })
   return max - min ;
}
console.log(difference(arr))  // 74
```
复制代码

4.2、max、min

```
let arr = [23, 4, 5, 2, 4, 5, 6, 6, 71, -3];
const difference = arr => {
   let max = Math.max(...arr),
      min = Math.min(...arr);
   return max - min ;
}
console.log(difference(arr)) // 74
```
复制代码

5、不借助临时变量，进行两个整数的交换

5.1、数组解构

```
let a = 2,
   b = 3;
   [b,a] = [a,b]
   console.log(a,b)   // 3 2
```
复制代码

5.2、算术运算（加减）

输入a = 2,b = 3,输出 a = 3,b = 2

```
let a = 2,
   b = 3;
const swop = (a, b) => {
   b = b - a;
   a = a + b;
   b = a - b;
   return [a,b];
}
console.log(swop(2,3)) // [3,2]
```
复制代码

5.3、逻辑运算（异或）

```
let a = 2,
    b = 3;
const swop = (a, b) => {
    a ^= b; //x先存x和y两者的信息
    b ^= a; //保持x不变，利用x异或反转y的原始值使其等于x的原始值
    a ^= b; //保持y不变，利用x异或反转y的原始值使其等于y的原始值
    return [a,b];
}
console.log(swop(2,3)) // [3,2]
```
复制代码

6、排序 (从小到大)

6.1、冒泡排序

```
let arr = [43, 32, 1, 5, 9, 22];
const sort = arr => {
    let res = []
    arr.forEach((v, i) => {
        for (let j = i + 1; j < arr.length; j++) {
            if (arr[i] > arr[j]) {
                [arr[i],arr[j]] = [arr[j],arr[i]]
            }
        }
    })
    return arr
}
console.log(sort(arr))  // [1, 5, 9, 22, 32, 43]
```
复制代码

For Agile Development: https://reqtest.com/agile-blog/agile-scrum-guide/

What is Agile?

Agile software development methodology is more adaptable to changes as there is no in-depth planning at the beginning of a project rather there are changing requirements throughout the course of the project. A constant feedback from the end users is encouraged. In agile, there is an incremental and iterative development approach. The work is prioritized on the basis of business or customer value. There are cross-functional teams that work on the iterations of the product over a period of time. Each iteration is focused on producing a working product.

Agile refers to a process that is aligned with the concepts listed in the Agile Manifesto.

The 12 principles of the Agile Manifesto:

1. Customer satisfaction is of highest priority which is achieved through the continuous delivery of valuable software.
2. Accommodate changing requirements even in later phases of development.
3. Deliver working software frequently in a shorter timescale.
4. Business team and developers must collaborate on a daily basis throughout the project.
5. Higher autonomy is given to the team members with greater support and trust.
6. Face-to-face interaction is critical for conveying information within a development team.
7. The progress of the project is measured by working software.
8. Promote sustainable development by maintaining a constant pace indefinitely.
9. Technical excellence and good design should be the main focus.
10. Simplicity is essential for progress.
11. Self-organizing teams are required for the best architectures and designs.
12. The teams should reflect on how to become more effective regularly and adopt the changes to increase effectiveness.

What are the approaches to implement agile?

Agile is a framework and there are various approaches to implement agile.

Scrum

It's one of the most popular ways to implement agile. In this iterative approach, there are sprints that last one to two weeks and allow the team to deliver software on regular basis. Scrum uses a software model that follows a set of roles, responsibilities, and meetings.

Kanban

In this approach, a visual framework is used to implement agile. This approach promotes small yet continuous changes to the current system. The principles of Kanban include visualizing workflow, limiting work in progress, managing and enhancing the workflow and continuous improvement.

Extreme Programming (XP)

Extreme Programming focuses on improving the quality and responsiveness to evolving customer requirements. Its principles include feedback, simplicity, and embracing change.

Feature Driven Development (FDD)

FDD involves 5 basic activities: develop the overall model, build a feature list, plan by feature, design by feature, and build by feature. FDD blends the industries best practices into a single approach.

Adaptive System Development (ASD)

ASD approach focuses on the idea that the project should always be in a state of continuous adaptation. There is a cycle of 3 repeating series: speculate, collaborate and learn in ASD.

Dynamic Systems Development Method (DSDM)

DSDM is an approach to address the common failures of IT projects, like missing deadlines, going over budget and no user involvement. This approach focuses on the business, being stringent about the timeliness, collaboration, never compromising on quality, build & develop iteratively, communicate continuously and demonstrate control.

Lean Software Development (LSD)

Lean Software Development approach (LSD) has 7 principles. These principles include an elimination of any waste, focusing on learning, take decisions as late as possible, delivering as soon as possible, empowering the team, building integrity and seeing the whole picture.

Crystal Clear

Crystal Clear approach is used with teams of six to eight developers. This approach focuses on the people involved in the project, not processes or artifacts. In this process, delivery of usable code to the user is frequent. The efficiency improvement and communication is achieved by being co-located.

What is Scrum?

Scrum is the most popular approach to implement agile. It helps to manage software development with an iterative approach. There are fixed-length iterations known as a sprint that allows shipping software frequently. A sprint lasts one to two weeks and at the

end of each sprint, the stakeholders and team members conduct a meeting to plan the next steps.
The roles, responsibilities, and meetings are fixed in a Scrum. In each sprint, there is sprint planning, daily stand-up, sprint demo and sprint retrospective. There are task boards and burndown charts to follow up on the progress of the sprint as well as to receive incremental feedback.

Roles in a scrum:

Product Owner

The vision for the software to be built is communicated by the Product Owner. Product Owner not only focuses on the work to be completed but also focuses on business and market requirements. The PO interacts with the team as well as other stakeholders to build and manage the backlog. The role of a PO is to motivate the team to align them with the goal and vision of the project.

Scrum Master

Scrum Master is responsible for organizing meetings, dealing with challenges and bottlenecks. The Scrum Master interacts with Product Owner to ensure that the product backlog is ready for the next sprint. He or she is also responsible to ensure that the team follows the Scrum process.

Scrum Team

The Scrum Team can be comprised of 5 to 7 members. In a Scrum team, there are no distinct roles as a programmer, designer or tester rather everyone has a set of tasks that they complete together. The Scrum Team plans the amount of work they can complete in each iteration.

Steps in the Scrum flow:

Product backlog

The product backlog comprises a list of all the desired features of the product. The Product Owner and Scrum Master prioritize the items on the basis of user stories and requirements. The development team refers to the product backlog to complete the task during each sprint.

Sprint planning

In the sprint planning meeting, the Product Owner provides a list of high priority items on the backlog. The team chooses the task they can complete during the sprint and transfer the tasks from product backlog to the sprint backlog.

Backlog refinement

The team and Product Owner meet at the end of each sprint to prepare the backlog for the next sprint. The team splits the user stories into a smaller chunk of tasks and removes any user stories that are irrelevant. The team also accesses the priority of stories to reprioritize tasks.

Daily Scrum

A 15-minute stand-up meeting known as Daily Scrum is conducted daily. The team member discusses the goals and issues related to the development. The Daily Scrum is held every day during the sprint to keep the team on track.

Sprint review meeting

A live demonstration is given at the end of each sprint to showcase the work the team has completed during the sprint in the sprint review meeting.

Sprint retrospective meeting

This meeting is held to reflect on the success of the Scrum process and is there any changes required to be made in the next sprint. The team discusses the highs and lows of the earlier sprint and all the improvements for the next sprint.

Summary

Agile is the software development methodology that focuses on customer satisfaction by delivery shippable software frequently. Scrum is one of the many approaches to implement Agile. Scrum is suitable for certain type of projects where there are rapidly changing requirements.

Agile development tools:(scrum or kanban methodology)
Jira https://www.atlassian.com/agile/tutorials/how-to-do-scrum-with-jira-software
MS Project https://www.mpug.com/articles/newly-released-agile-capabilities-in-ms-project/
MS Planner https://sharepointmaven.com/how-to-use-microsoft-planner-for-agile-and-scrum-projects/

https://github.com/aditiraj/hackerrankSolutions-JavaScript