position: static; by default.

Static positioned elements are not affected by the top, bottom, left, and right properties.

## position: relative;

An element with position: relative; is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

## position: fixed;

s positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

## position: absolute;

## is positioned relative to the nearest positioned ancestor

## position: sticky;

 is positioned based on the user's scroll position.

## Overlapping Elements

The z-index property specifies the stack order of an element

An element with greater stack order is always in front of an element with a lower stack order.

**Note:** If two positioned elements overlap without a z-index specified, the element positioned last in the HTML code will be shown on top.

negative stack order: placed behind

|  |  |  |
| --- | --- | --- |
| auto | No clipping will be applied. This is default | [Play it »](https://www.w3schools.com/cssref/playit.asp?filename=playcss_clip&preval=auto) |
| *shape* | Clips an element. The only valid value is: rect (*top, right, bottom, left*) | [Play it »](https://www.w3schools.com/cssref/playit.asp?filename=playcss_clip&preval=rect(0px,25px,25px,0px)) |
| initial | Sets this property to its default value. [Read about *initial*](https://www.w3schools.com/cssref/css_initial.asp) | [Play it »](https://www.w3schools.com/cssref/playit.asp?filename=playcss_clip&preval=initial) |
| inherit | Inherits this property from its parent element. [Read about *inherit*](https://www.w3schools.com/cssref/css_inherit.asp) |  |

CSS Overflow

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

* visible - Default. The overflow is not clipped. The content 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, and a scrollbar is added to see the rest of the content
* auto - Similar to scroll, but it adds scrollbars only when necessary

## The float Property

The float property is used for positioning and formatting content e.g. let an image float left to the text in a container.

* left - The element floats to the left of its container
* right - The element floats to the right of its container
* none - The element does not float (will be displayed just where it occurs in the text). This is default
* inherit - The element inherits the float value of its parent

# CSS Layout - display: inline-block

Compared to display: inline, the major difference is that display: inline-block allows to set a width and height on the element.

Also, with display: inline-block, the top and bottom margins/paddings are respected, but with display: inline they are not.

Compared to display: block, the major difference is that display: inline-block does not add a line-break after the element, so the element can sit next to other elements.

## Image Sprites

An image sprite 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.

Using image sprites will reduce the number of server requests and save bandwidth.

#home {  
  width: 46px;  
  height: 44px;  
  background: url(img\_navsprites.gif) 0 0;  
}

#prev {  
  left: 63px;  
  width: 43px;  
  background: url('img\_navsprites.gif') -47px 0;  
}  
  
#next {  
  left: 129px;  
  width: 43px;  
  background: url('img\_navsprites.gif') -91px 0;  
}

**Text-Transform:**

text-transform: none|capitalize|uppercase|lowercase|initial|inherit;

**rounded corners**: border-radius

**Grid view:**

A responsive grid-view often has 12 columns, and has a total width of 100%, and will shrink and expand as you resize the browser window.

\* {  
  box-sizing: border-box;  
}

Ex:

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<style>

\* {

box-sizing: border-box;

}

.row::after {

content: "";

clear: both;

display: table;

}

[class\*="col-"] {

float: left;

padding: 15px;

}

.col-1 {width: 8.33%;}

.col-2 {width: 16.66%;}

.col-3 {width: 25%;}

.col-4 {width: 33.33%;}

.col-5 {width: 41.66%;}

.col-6 {width: 50%;}

.col-7 {width: 58.33%;}

.col-8 {width: 66.66%;}

.col-9 {width: 75%;}

.col-10 {width: 83.33%;}

.col-11 {width: 91.66%;}

.col-12 {width: 100%;}

html {

font-family: "Lucida Sans", sans-serif;

}

.header {

background-color: #9933cc;

color: #ffffff;

padding: 15px;

}

.menu ul {

list-style-type: none;

margin: 0;

padding: 0;

}

.menu li {

padding: 8px;

margin-bottom: 7px;

background-color: #33b5e5;

color: #ffffff;

box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);

}

.menu li:hover {

background-color: #0099cc;

}

</style>

</head>

<body>

<div class="header">

<h1>Chania</h1>

</div>

<div class="row">

<div class="col-3 menu">

<ul>

<li>The Flight</li>

<li>The City</li>

<li>The Island</li>

<li>The Food</li>

</ul>

</div>

<div class="col-9">

<h1>The City</h1>

<p>Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.</p>

<p>Resize the browser window to see how the content respond to the resizing.</p>

</div>

</div>

</body>

</html>

**Media Queries:**

**CSS Lists**:

ul.a {  
  list-style-type: circle;  
}  
  
ul.b {  
  list-style-type: square;  
}  
  
ol.c {  
  list-style-type: upper-roman;  
}  
  
ol.d {  
  list-style-type: lower-alpha;  
}

CSS Selectors

* Simple selectors (select elements based on name, id, class)
* [Combinator selectors](https://www.w3schools.com/css/css_combinators.asp) (select elements based on a specific relationship between them)
* [Pseudo-class selectors](https://www.w3schools.com/css/css_pseudo_classes.asp) (select elements based on a certain state)
* [Pseudo-elements selectors](https://www.w3schools.com/css/css_pseudo_elements.asp) (select and style a part of an element)
* [Attribute selectors](https://www.w3schools.com/css/css_attribute_selectors.asp) (select elements based on an attribute or attribute value)

Simple selector:

* CSS element Selector(p)
* CSS id Selector(#)
* CSS class Selector(.)
* CSS Universal Selector (\*)
* CSS Grouping Selector(div,p)

Combinator selector:

* descendant selector (space)
* child selector (>)
* adjacent sibling selector (+)
* general sibling selector (~)

**Pseudo-class selector:**

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

* Style an element when a user mouses over it
* Style visited and unvisited links differently
* Style an element when it gets focus

a:hover a:link a:visited a:active

:first-child  matches a specified element that is the first child of another element

|  |  |  |
| --- | --- | --- |
| [:first-of-type](https://www.w3schools.com/cssref/sel_first-of-type.asp) | p:first-of-type | Selects every <p> element that is the first <p> element of its parent |

**Pseudo elements:**

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

For example, it can be used to:

* Style the first letter, or line, of an element
* Insert content before, or after, the content of an element

|  |  |  |
| --- | --- | --- |
| **elector** | **Example** | **Example description** |
| [::after](https://www.w3schools.com/cssref/sel_after.asp) | p::after | Insert something after the content of each <p> element |
| [::before](https://www.w3schools.com/cssref/sel_before.asp) | p::before | Insert something before the content of each <p> element |
| [::first-letter](https://www.w3schools.com/cssref/sel_firstletter.asp) | p::first-letter | Selects the first letter of each <p> element |
| [::first-line](https://www.w3schools.com/cssref/sel_firstline.asp) | p::first-line | Selects the first line of each <p> element |
| [::selection](https://www.w3schools.com/cssref/sel_selection.asp) | p::selection | Selects the portion of an element that is selected by a user |

**Attribute selector:**

The [attribute] selector is used to select elements with a specified attribute.

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Example description** |
| [[*attribute*]](https://www.w3schools.com/cssref/sel_attribute.asp) | [target] | Selects all elements with a target attribute |
| [[*attribute*=*value*]](https://www.w3schools.com/cssref/sel_attribute_value.asp) | [target=\_blank] | Selects all elements with target="\_blank" |
| [[*attribute*~=*value*]](https://www.w3schools.com/cssref/sel_attribute_value_contains.asp) | [title~=flower] | Selects all elements with a title attribute containing the word "flower" |
| [[*attribute*|=*value*]](https://www.w3schools.com/cssref/sel_attribute_value_lang.asp) | [lang|=en] | Selects all elements with a lang attribute value starting with "en" |
| [[*attribute*^=*value*]](https://www.w3schools.com/cssref/sel_attr_begin.asp) | a[href^="https"] | Selects every <a> element whose href attribute value begins with "https" |
| [[*attribute*$=*value*]](https://www.w3schools.com/cssref/sel_attr_end.asp) | a[href$=".pdf"] | Selects every <a> element whose href attribute value ends with ".pdf" |
| [[*attribute*\*=*value*]](https://www.w3schools.com/cssref/sel_attr_contain.asp) | a[href\*="w3schools"] | Selects every <a> element whose href attribute value contains the substring "w3schools" |

**DropDown:**

<form>

<select id="country" name="country">

<option value="au">Australia</option>

<option value="ca">Canada</option>

<option value="usa">USA</option>

</select>

</form>

**Box Model:**

MBPC(Margin, Border, Padding, Content)

**Text decoration:**

h1 {  
  text-decoration: overline;  
}  
  
h2 {  
  text-decoration: line-through;  
}  
  
h3 {  
  text-decoration: underline;  
}

Agile:

## [Acceptance Test Driven Development (ATDD)](https://www.agilealliance.org/glossary/atdd/)

Acceptance Test Driven Development (ATDD) involves team members with different perspectives (customer, development, testing) collaborating to write acceptance tests in advance of implementing the corresponding functionality.

## [Acceptance Testing](https://www.agilealliance.org/glossary/acceptance/)

An acceptance test is a formal description of the behavior of a software product, generally expressed as an example or a usage scenario.

## [Antipattern](https://www.agilealliance.org/glossary/antipattern/)

Antipatterns are common solutions to common problems where the solution is ineffective and may result in undesired consequences.

## [Burndown Chart](https://www.agilealliance.org/glossary/burndown-chart/)

Burndown charts and burnup charts track the amount of output (in terms of hours, story points, or backlog items) a team has completed across an iteration or a project.

## [CRC Cards](https://www.agilealliance.org/glossary/crc-cards/)

Class Responsibility Collaborator (CRC) Cards are an object oriented design technique teams can use to discuss what a class should know and do and what other classes it interacts with

## [Daily Meeting](https://www.agilealliance.org/glossary/daily-meeting/)(standup)

The daily meeting is one of the most commonly practiced Agile techniques and presents opportunity for a team to get together on a regular basis to coordinate their activities.

## [Epic](https://www.agilealliance.org/glossary/epic/)

An epic is a large user story.

## [Heartbeat Retrospective](https://www.agilealliance.org/glossary/heartbeatretro/)

The team meets regularly to reflect on the most significant events that occurred since the previous such meeting, and identify opportunities  for improvement.

## [Iteration](https://www.agilealliance.org/glossary/iteration/)

An iteration is a timebox during which development takes place. The  duration may vary from project to project and is usually fixed.

## [Kanban Board](https://www.agilealliance.org/glossary/kanban-board/)

A Kanban Board is a visual workflow tool consisting of multiple columns. Each column represents a different stage in the workflow process.

## [Points (estimates in)](https://www.agilealliance.org/glossary/points-estimates-in/)

Agile teams generally prefer to express estimates in units other than the time-honored “man-hours.” Possibly the most widespread unit is “story points.”

## [Product Backlog](https://www.agilealliance.org/glossary/backlog/)

A product backlog is a list of the new features, changes to existing features, bug fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome.

## [Product Owner](https://www.agilealliance.org/glossary/product-owner/)

The product owner is a role created by the Scrum Framework responsible for making sure the team delivers the desired outcome.

## [Scrum Master](https://www.agilealliance.org/glossary/scrum-master/)

The scrum master is responsible for ensuring the team lives agile values and principles and follows the practices that the team agreed they would use.

## [Sign Up for Tasks](https://www.agilealliance.org/glossary/sign-up-for-tasks/)

Members of an Agile development team normally choose which tasks to work on, rather than being assigned work by a manager

## [Sprint Backlog](https://www.agilealliance.org/glossary/sprint-backlog/)

A sprint backlog is the subset of product backlog that a team targets to deliver during a sprint in order to accomplish the sprint goal and make progress toward a desired outcome

## [Sprint Planning](https://www.agilealliance.org/glossary/sprint-planning/)

Sprint planning is an event that occurs at the beginning of a sprint where the team determines the product backlog items they will work on during that sprint.

## [Story Mapping](https://www.agilealliance.org/glossary/storymap/)

Story mapping consists of ordering user stories along two independent dimensions

## [Story Splitting](https://www.agilealliance.org/glossary/split/)

Splitting consists of breaking up one user story into smaller ones, while preserving the property that each user story separately has measurable business value.

## [Test Driven Development (TDD)](https://www.agilealliance.org/glossary/tdd/)

“Test-driven development” is a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring).

## [Unit Testing](https://www.agilealliance.org/glossary/unit-test/)

A unit test is a short program fragment written and maintained by the developers on the product team, which exercises some narrow part of the product’s source code and checks the results.

## [Usability Testing](https://www.agilealliance.org/glossary/usability/)

Usability testing is an empirical, exploratory technique to answer questions such as “how would an end user respond to our software under realistic conditions?”

## [User Stories](https://www.agilealliance.org/glossary/user-stories/)

In consultation with the customer or product owner, the team divides up the work to be done into functional increments called “user stories.”

## [Velocity](https://www.agilealliance.org/glossary/velocity/)

At the end of each iteration, the team adds up effort estimates associated with user stories that were completed during that iteration. This total is called velocity. [(see more)](https://www.agilealliance.org/glossary/velocity/)

## [Version Control](https://www.agilealliance.org/glossary/version-control/)

Version control is not strictly an Agile “practice” insofar as it is now widespread in the industry as a whole. But it is mentioned here for several reasons.

## How does React work?

React creates a virtual DOM. When state changes in a component it firstly runs a “diffing” algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM with the results of diff.

## What are the advantages of using React?

* It is easy to know how a component is rendered, you just need to look at the render function.
* JSX makes it easy to read the code of your components. It is also really easy to see the layout, or how components are plugged/combined with each other.
* You can render React on the server-side. This enables improves SEO and performance.
* It is easy to test.
* You can use React with any framework (Backbone.js, Angular.js) as it is only a view layer.

## What is the difference between a Presentational component and a Container component?

Presentational components are concerned with how things look. They generally receive data and callbacks exclusively via props. These components rarely have their own state, but when they do it generally concerns UI state, as opposed to data state.

Container components are more concerned with how things work. These components provide the data and behavior to presentational or other container components. They call Flux actions and provide these as callbacks to the presentational components. They are also often stateful as they serve as data sources.

## What are the differences between a class component and functional component?

* Class components allows you to use additional features such as local state and lifecycle hooks. Also, to enable your component to have direct access to your store and thus holds state.
* When your component just receives props and renders them to the page, this is a ‘stateless component’, for which a pure function can be used. These are also called dumb components or presentational components.

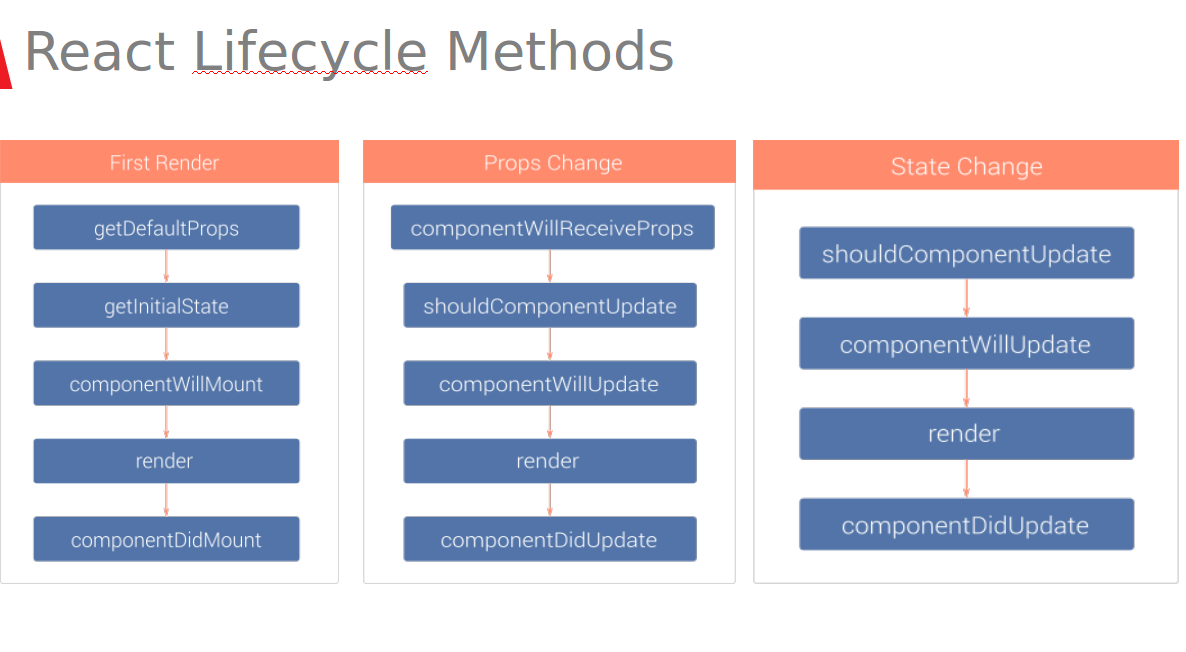
## What is the difference between state and props?

The state is a data structure that starts with a default value when a Component mounts. It may be mutated across time, mostly as a result of user events.

Props (short for properties) are a Component’s configuration. They are received from above and immutable as far as the Component receiving them is concerned. A Component cannot change its props, but it is responsible for putting together the props of its child Components. Props do not have to just be data — callback functions may be passed in as props.

## Name the different lifecycle methods.

* componentWillMount- this is most commonly used for App configuration in your root component.
* componentDidMount - here you want to do all the setup you couldn’t do without a DOM, and start getting all the data you need. Also if you want to set up eventListeners etc. this lifecycle hook is a good place to do that.
* componentWillReceiveProps - this lifecyclye acts on particular prop changes to trigger state transitions.
* shouldComponentUpdate - if you’re worried about wasted renders shouldComponentUpdate is a great place to improve performance as it allows you to prevent a rerender if component receives new prop. shouldComponentUpdate should always return a boolean and based on what this is will determine if the component is rerendered or not.
* componentWillUpdate - rarely used. It can be used instead of componentWillReceiveProps on a component that also has shouldComponentUpdate (but no access to previous props).
* componentDidUpdate - also commonly used to update the DOM in response to prop or state changes.
* componentWillUnmount - here you can cancel any outgoing network requests, or remove all event listeners associated with the component.



## Where in a React component should you make an AJAX request?

componentDidMount is where an AJAX request should be made in a React component. This method will be executed when the component “mounts” (is added to the DOM) for the first time. This method is only executed once during the component’s life. Importantly, you can’t guarantee the AJAX request will have resolved before the component mounts. If it doesn't, that would mean that you’d be trying to setState on an unmounted component, which would not work. Making your AJAX request in componentDidMount will guarantee that there’s a component to update.

## What are controlled components?

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. When a user submits a form the values from the aforementioned elements are sent with the form. With React it works differently. The component containing the form will keep track of the value of the input in it's state and will re-render the component each time the callback function e.g. onChange is fired as the state will be updated. An input form element whose value is controlled by React in this way is called a "controlled component".

## What are refs used for in React?

Refs are used to get reference to a DOM node or an instance of a component in React. Good examples of when to use refs are for managing focus/text selection, triggering imperative animations, or integrating with third-party DOM libraries. You should avoid using string refs and inline ref callbacks. Callback refs are advised by React.

## What is a higher order component?

A higher-order component is a function that takes a component and returns a new component. HOC’s allow you to reuse code, logic and bootstrap abstraction. The most common is probably Redux’s connect function. Beyond simply sharing utility libraries and simple composition, HOCs are the best way to share behavior between React Components. If you find yourself writing a lot of code in different places that does the same thing, you may be able to refactor that code into a reusable HOC.

Exercises

* Write an HOC that reverses it’s input
* Write an HOC that supplies data from an API to it’s Passed Component
* Write an HOC that implements shouldComponentUpdate to avoid reconciliation.
* Write an HOC that uses React.Children.toArray to sort the children passed to it’s Passed Component.

## What advantages are there in using arrow functions?

* Scope safety: Until arrow functions, every new function defined its own this value (a new object in the case of a constructor, undefined in strict mode function calls, the base object if the function is called as an “object method”, etc.). An arrow function does not create its own this, the this value of the enclosing execution context is used.
* Compactness: Arrow functions are easier to read and write.
* Clarity: When almost everything is an arrow function, any regular function immediately sticks out for defining the scope. A developer can always look up the next-higher function statement to see what the thisObject is.

## Why is it advised to pass a callback function to setState as opposed to an object?

Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state.

## What is the alternative of binding this in the constructor?

You can use property initializers to correctly bind callbacks. This is enabled by default in create react app. you can use an arrow function in the callback. The problem here is that a new callback is created each time the component renders.

## How would you prevent a component from rendering?

Returning null from a component’s render method does not affect the firing of the component’s lifecycle methods.

## When rendering a list what is a key and what is it’s purpose?

Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity. The best way to pick a key is to use a string that uniquely identifies a list item among its siblings. Most often you would use IDs from your data as keys. When you don’t have stable IDs for rendered items, you may use the item index as a key as a last resort. It is not recommend to use indexes for keys if the items can reorder, as that would be slow.

## What is the purpose of super(props)?

A child class constructor cannot make use of this until super() has been called. Also, ES2015 class constructors have to call super() if they are subclasses. The reason for passing props to super() is to enable you to access this.propsin the constructor.

## What is JSX?

JSX is a syntax extension to JavaScript and comes with the full power of JavaScript. JSX produces React “elements”. You can embed any JavaScript expression in JSX by wrapping it in curly braces. After compilation, JSX expressions become regular JavaScript objects. This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions:

## What is equivalent of the following using React.createElement?

Question:

const element = (  
 <h1 className="greeting">  
 Hello, world!  
 </h1>  
);

Answer:

const element = React.createElement(  
 'h1',  
 {className: 'greeting'},  
 'Hello, world!'  
);

## What is Children?

In JSX expressions that contain both an opening tag and a closing tag, the content between those tags is passed to components automatically as a special prop: props.children.

There are a number of methods available in the React API to work with this prop. These include React.Children.map, React.Children.forEach, React.Children.count, React.Children.only, React.Children.toArray.

## What is state in react?

State is similar to props, but it is private and fully controlled by the component. State is essentially an object that holds data and determines how the component renders and behaves.

## Why would you eject from create-react-app?

Until you eject you are unable to configure webpack or babel presets.

## What is redux?

The basic idea of redux is that the entire application state is kept in a single store. The store is simply a javascript object. The only way to change the state is by firing actions from your application and then writing reducers for these actions that modify the state. The entire state transition is kept inside reducers and should not have any side-effects.

## What is a store in redux?

The store is a javascript object that holds application state. Along with this it also has the following responsibilities:

* Allows access to state via getState();
* Allows state to be updated via dispatch(action);
* Registers listeners via subscribe(listener);
* Handles unregistering of listeners via the function returned by subscribe(listener).

## What is an action?

Actions are plain javascript objects. They must have a type indicating the type of action being performed. In essence, actions are payloads of information that send data from your application to your store.

## What is a reducer?

A reducer is simply a pure function that takes the previous state and an action, and returns the next state.

## What is Redux Thunk used for?

Redux thunk is middleware that allows you to write action creators that return a function instead of an action. The thunk can then be used to delay the dispatch of an action if a certain condition is met. This allows you to handle the asyncronous dispatching of actions.

## What is a pure function?

A pure function is a function that doesn’t depend on and doesn’t modify the states of variables out of its scope. Essentially, this means that a pure function will always return the same result given same parameters.

## What do you like about react?

…..

## What don’t you like about react?

…..React Interview Questions

## How does React work?

React creates a virtual DOM. When state changes in a component it firstly runs a “diffing” algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM with the results of diff.

## What are the advantages of using React?

* It is easy to know how a component is rendered, you just need to look at the render function.
* JSX makes it easy to read the code of your components. It is also really easy to see the layout, or how components are plugged/combined with each other.
* You can render React on the server-side. This enables improves SEO and performance.
* It is easy to test.
* You can use React with any framework (Backbone.js, Angular.js) as it is only a view layer.

## What is the difference between a Presentational component and a Container component?

Presentational components are concerned with how things look. They generally receive data and callbacks exclusively via props. These components rarely have their own state, but when they do it generally concerns UI state, as opposed to data state.

Container components are more concerned with how things work. These components provide the data and behavior to presentational or other container components. They call Flux actions and provide these as callbacks to the presentational components. They are also often stateful as they serve as data sources.

## What are the differences between a class component and functional component?

* Class components allows you to use additional features such as local state and lifecycle hooks. Also, to enable your component to have direct access to your store and thus holds state.
* When your component just receives props and renders them to the page, this is a ‘stateless component’, for which a pure function can be used. These are also called dumb components or presentational components.

## What is the difference between state and props?

The state is a data structure that starts with a default value when a Component mounts. It may be mutated across time, mostly as a result of user events.

Props (short for properties) are a Component’s configuration. They are received from above and immutable as far as the Component receiving them is concerned. A Component cannot change its props, but it is responsible for putting together the props of its child Components. Props do not have to just be data — callback functions may be passed in as props.

## Name the different lifecycle methods.

* componentWillMount- this is most commonly used for App configuration in your root component.
* componentDidMount - here you want to do all the setup you couldn’t do without a DOM, and start getting all the data you need. Also if you want to set up eventListeners etc. this lifecycle hook is a good place to do that.
* componentWillReceiveProps - this lifecyclye acts on particular prop changes to trigger state transitions.
* shouldComponentUpdate - if you’re worried about wasted renders shouldComponentUpdate is a great place to improve performance as it allows you to prevent a rerender if component receives new prop. shouldComponentUpdate should always return a boolean and based on what this is will determine if the component is rerendered or not.
* componentWillUpdate - rarely used. It can be used instead of componentWillReceiveProps on a component that also has shouldComponentUpdate (but no access to previous props).
* componentDidUpdate - also commonly used to update the DOM in response to prop or state changes.
* componentWillUnmount - here you can cancel any outgoing network requests, or remove all event listeners associated with the component.

## Where in a React component should you make an AJAX request?

componentDidMount is where an AJAX request should be made in a React component. This method will be executed when the component “mounts” (is added to the DOM) for the first time. This method is only executed once during the component’s life. Importantly, you can’t guarantee the AJAX request will have resolved before the component mounts. If it doesn't, that would mean that you’d be trying to setState on an unmounted component, which would not work. Making your AJAX request in componentDidMount will guarantee that there’s a component to update.

## What are controlled components?

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. When a user submits a form the values from the aforementioned elements are sent with the form. With React it works differently. The component containing the form will keep track of the value of the input in it's state and will re-render the component each time the callback function e.g. onChange is fired as the state will be updated. An input form element whose value is controlled by React in this way is called a "controlled component".

## What are refs used for in React?

Refs are used to get reference to a DOM node or an instance of a component in React. Good examples of when to use refs are for managing focus/text selection, triggering imperative animations, or integrating with third-party DOM libraries. You should avoid using string refs and inline ref callbacks. Callback refs are advised by React.

## What is a higher order component?

A higher-order component is a function that takes a component and returns a new component. HOC’s allow you to reuse code, logic and bootstrap abstraction. The most common is probably Redux’s connect function. Beyond simply sharing utility libraries and simple composition, HOCs are the best way to share behavior between React Components. If you find yourself writing a lot of code in different places that does the same thing, you may be able to refactor that code into a reusable HOC.

Exercises

* Write an HOC that reverses it’s input
* Write an HOC that supplies data from an API to it’s Passed Component
* Write an HOC that implements shouldComponentUpdate to avoid reconciliation.
* Write an HOC that uses React.Children.toArray to sort the children passed to it’s Passed Component.

## What advantages are there in using arrow functions?

* Scope safety: Until arrow functions, every new function defined its own this value (a new object in the case of a constructor, undefined in strict mode function calls, the base object if the function is called as an “object method”, etc.). An arrow function does not create its own this, the this value of the enclosing execution context is used.
* Compactness: Arrow functions are easier to read and write.
* Clarity: When almost everything is an arrow function, any regular function immediately sticks out for defining the scope. A developer can always look up the next-higher function statement to see what the thisObject is.

## Why is it advised to pass a callback function to setState as opposed to an object?

Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state.

## What is the alternative of binding this in the constructor?

You can use property initializers to correctly bind callbacks. This is enabled by default in create react app. you can use an arrow function in the callback. The problem here is that a new callback is created each time the component renders.

## How would you prevent a component from rendering?

Returning null from a component’s render method does not affect the firing of the component’s lifecycle methods.

## When rendering a list what is a key and what is it’s purpose?

Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity. The best way to pick a key is to use a string that uniquely identifies a list item among its siblings. Most often you would use IDs from your data as keys. When you don’t have stable IDs for rendered items, you may use the item index as a key as a last resort. It is not recommend to use indexes for keys if the items can reorder, as that would be slow.

## What is the purpose of super(props)?

A child class constructor cannot make use of this until super() has been called. Also, ES2015 class constructors have to call super() if they are subclasses. The reason for passing props to super() is to enable you to access this.propsin the constructor.

## What is JSX?

JSX is a syntax extension to JavaScript and comes with the full power of JavaScript. JSX produces React “elements”. You can embed any JavaScript expression in JSX by wrapping it in curly braces. After compilation, JSX expressions become regular JavaScript objects. This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions:

## What is equivalent of the following using React.createElement?

Question:

const element = (  
 <h1 className="greeting">  
 Hello, world!  
 </h1>  
);

Answer:

const element = React.createElement(  
 'h1',  
 {className: 'greeting'},  
 'Hello, world!'  
);

## What is Children?

In JSX expressions that contain both an opening tag and a closing tag, the content between those tags is passed to components automatically as a special prop: props.children.

There are a number of methods available in the React API to work with this prop. These include React.Children.map, React.Children.forEach, React.Children.count, React.Children.only, React.Children.toArray.

## What is state in react?

State is similar to props, but it is private and fully controlled by the component. State is essentially an object that holds data and determines how the component renders and behaves.

## Why would you eject from create-react-app?

Until you eject you are unable to configure webpack or babel presets.

## What is redux?

The basic idea of redux is that the entire application state is kept in a single store. The store is simply a javascript object. The only way to change the state is by firing actions from your application and then writing reducers for these actions that modify the state. The entire state transition is kept inside reducers and should not have any side-effects.

## What is the difference between Redux & Flux

Flux is very similar to Redux. The main difference is that Flux has multiple stores that change the state of the application, and it broadcasts these changes as events. Components can subscribe to these events to sync with the current state. Redux doesn’t have a dispatcher, which in Flux is used to broadcast payloads to registered callbacks. Another difference in Flux is that many varieties are available, and that creates some confusion and inconsistency.

## What is a store in redux?

The store is a javascript object that holds application state. Along with this it also has the following responsibilities:

* Allows access to state via getState();
* Allows state to be updated via dispatch(action);
* Registers listeners via subscribe(listener);
* Handles unregistering of listeners via the function returned by subscribe(listener).

## What is an action?

Actions are plain javascript objects. They must have a type indicating the type of action being performed. In essence, actions are payloads of information that send data from your application to your store.

## What is a reducer?

A reducer is simply a pure function that takes the previous state and an action, and returns the next state.

## What is Redux Thunk used for?

Redux thunk is middleware that allows you to write action creators that return a function instead of an action. The thunk can then be used to delay the dispatch of an action if a certain condition is met. This allows you to handle the asyncronous dispatching of actions.

## What is a pure function?

A pure function is a function that doesn’t depend on and doesn’t modify the states of variables out of its scope. Essentially, this means that a pure function will always return the same result given same parameters.

## What do you like about react?

…..

## What don’t you like about react?

…..

**Question 44-** *Explain the Error boundaries in React 16.3?***Answer-** Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed.

A class component becomes an error boundary if it defines either (or both) of the lifecycle methods static getDerivedStateFromError() or componentDidCatch(). Updating state from these lifecycles lets you capture an unhandled JavaScript error in the below tree and display a fallback UI.

**static getDerivedStateFromError()**  
This lifecycle is invoked after an error has been thrown by a component. It receives the error that was thrown as a parameter and should return a value to update state.  
*As it is called during the render phase so side-effects like this.setState are not allowed.*

**componentDidCatch()**  
This lifecycle is invoked after an error has been thrown by a descendant component. It receives two parameters:

1. error - The error that was thrown.
2. info - An object with a componentStack key containing information about the component which threw the error.

*As it is called during the commit phase so side-effects like this.setState are allowed.*

Below is the example where we define a component called ErrorBoundary

class ErrorBoundary extends React.Component {  
 constructor(props) {  
 super(props);  
 this.state = { hasError: false };  
 }  
  
 static getDerivedStateFromError(error) {  
 // Update state so the next render will show the fallback UI.  
 return { hasError: true };  
 }  
  
 componentDidCatch(error, info) {  
 logComponentStackToMyService(info.componentStack);  
 }  
  
 render() {  
 if (this.state.hasError) {  
 return <h1>Something went wrong.</h1>;  
 }  
  
 return this.props.children;   
 }  
}

After this we need to wrap any component with it and try will work as the classic try..catch block.

<ErrorBoundary>  
 <ProductCard />  
</ErrorBoundary>

**Question 46-** *Explain “refs” in React?***Answer-**Refs in React provides a way to access the React elements/DOM nodes created in “render()” method.

When parent components need to interact with children components, we use **props.**However, *in some cases we might need to modify a child without re-rendering it with new props*. That’s when we use refs.

We should minimise the use of refs in React project as it’s directly interact with DOM. We should use refs in following situations.

* Interaction with third party DOM interacting libraries like jQuery
* Triggering animations
* Managing focus, text selection

There are various ways to use “refs” but we will see the latest way to use “ref” which was introduced in React 16.3 and it is using React.createRef()

In the below example we have a simple input box, where we can input a value.Then, when the submit button is clicked, we’ll read this value and log it to the console.

class CustomTextInput extends React.Component {  
 constructor(props) {  
 super(props);  
 **// create a ref to store the textInput DOM element**  
 this.textInput = React.createRef();  
 }  
 handleSubmit = e => {  
 e.preventDefault();  
  
 console.log(this.textInput.current.value);  
 };  
  
 render() {  
 **// tell React that we want to associate the <input> ref  
 // with the `textInput` that we created in the constructor**  
 return (  
 <div>  
 <form onSubmit={e => this.handleSubmit(e)}>  
 <input type="text" ref={this.textInput} />  
 <button>Submit</button>  
 </form>  
 </div>  
 );  
 }  
}

React hooks:

|  |  |
| --- | --- |
|  | import React, { useState, useEffect } from 'react'; |
|  | import axios from 'axios'; |
|  |  |
|  | const ResourceList = ({ item }) => { |
|  | const [resources, setResources] = useState([]); |
|  |  |
|  | const fetchResource = async (item) => { |
|  | const response = await axios.get(`https://jsonplaceholder.typicode.com/${item}`); |
|  | setResources(response.data); |
|  | } |
|  |  |
|  | useEffect(() => { |
|  | fetchResource(item); |
|  | }, [item]) |
|  |  |
|  | return ( |
|  | <ul> |
|  | {resources.map(record => <li key={record.id}>{record.title}</li>)} |
|  | </ul> |
|  | ) |
|  | } |
|  |  |
|  | export default ResourceList; |

# Scheduling: setTimeout and setInterval

There are two methods for it:

* setTimeout allows us to run a function once after the interval of time.
* setInterval allows us to run a function repeatedly, starting after the interval of time, then repeating continuously at that interval.

## [setTimeout](https://javascript.info/settimeout-setinterval" \l "settimeout)

let timerId = setTimeout(func|code, [delay], [arg1], [arg2], ...)

ex:

function sayHi() {

alert('Hello');

}

setTimeout(sayHi, 1000);

setTimeout(() => alert('Hello'), 1000);

## [setInterval](https://javascript.info/settimeout-setinterval" \l "setinterval)

let timerId = setInterval(func|code, [delay], [arg1], [arg2], ...)

// repeat with the interval of 2 seconds

let timerId = setInterval(() => alert('tick'), 2000);