React

React is a JavaScript library for building user interfaces.

React is used to build single-page applications.

React allows us to create reusable UI components.

Key features and concepts of React.js include:

1. Components: React applications are built using reusable components. A component is a self-contained, modular piece of the user interface that can be composed and reused throughout the application.
2. Virtual DOM: React uses a virtual DOM to improve performance. Instead of updating the entire DOM when changes occur, React first updates a virtual representation of the DOM in memory and then efficiently updates only the parts that changed.
3. JSX: React uses JSX (JavaScript XML) syntax, which allows developers to write HTML elements and components in a syntax that looks similar to XML or HTML. JSX is then transpiled to JavaScript.
4. Unidirectional Data Flow: React follows a unidirectional data flow, where data flows in a single direction through a component hierarchy. This helps to manage state and props in a predictable way.
5. React Hooks: Hooks are functions that enable functional components to have state and lifecycle features. They allow developers to use state and other React features in functional components instead of class components.
6. React Router: React Router is a library that enables navigation and routing in React applications, allowing developers to create single-page applications with multiple views.

Learning by Examples

Our "Show React" tool makes it easy to demonstrate React. It shows both the code and the result.

Example:

import React from 'react';

import ReactDOM from 'react-dom/client';

function Hello(props) {

return <h1>Hello World!</h1>;

}

const container = document.getElementById("root");

const root = ReactDOM.createRoot(container);

root.render(<Hello />);

Learning by Exercises

React Exercises:

Exercise:-

Enter the correct ReactDOM method to render the React element to the DOM.

ReactDOM.

(myElement, document.getElementById('root'));

My Learning:-

Track your progress with the free "My Learning" program here at W3Schools.

Log in to your account, and start earning points!

This is an optional feature. You can study W3Schools without using My Learning.

Create React App:-

To learn and test React, you should set up a React Environment on your computer.

This tutorial uses the create-react-app.

The create-react-app tool is an officially supported way to create React applications.

Node.js is required to use create-react-app.

Open your terminal in the directory you would like to create your application.

Run this command to create a React application named my-react-app:

Run the React Application

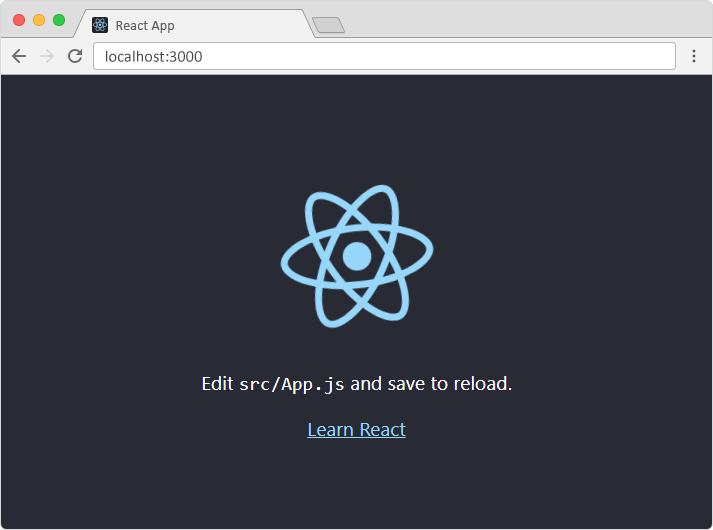
Run this command to move to the my-react-app directory:

cd my-react-app

Run this command to execute the React application my-react-app:

npm start

A new browser window will pop up with your newly created React App!

React ES6

What is ES6?

ES6 stands for ECMAScript 6.

ECMAScript was created to standardize JavaScript, and ES6 is the 6th version of ECMAScript, it was published in 2015, and is also known as ECMAScript 2015.

Why Should I Learn ES6?

React uses ES6, and you should be familiar with some of the new features like:

Classes

Arrow Functions

Variables (let, const, var)

Array Methods like .map()

Destructuring

Modules

Ternary Operator

Spread Operator

Classes

ES6 introduced classes.

A class is a type of function, but instead of using the keyword function to initiate it, we use the keyword class, and the properties are assigned inside a constructor() method.

Example:-

A simple class constructor:

class Car {

constructor(name) {

this.brand = name;

}

}

Now you can create objects using the Car class:

Example:-

Create an object called "mycar" based on the Car class:

class Car {

constructor(name) {

this.brand = name;

}

}

const mycar = new Car("Ford");

Method in Classes

You can add your own methods in a class:

Example:-

Create a method named "present":

class Car {

constructor(name) {

this.brand = name;

}

present() {

return 'I have a ' + this.brand;

}

}

const mycar = new Car("Ford");

mycar.present();

As you can see in the example above, you call the method by referring to the object's method name followed by parentheses (parameters would go inside the parentheses).

Class Inheritance

To create a class inheritance, use the extends keyword.

A class created with a class inheritance inherits all the methods from another class:

Example:-

Create a class named "Model" which will inherit the methods from the "Car" class:

class Car {

constructor(name) {

this.brand = name;

}

present() {

return 'I have a ' + this.brand;

}

}

class Model extends Car {

constructor(name, mod) {

super(name);

this.model = mod;

}

show() {

return this.present() + ', it is a ' + this.model

}

}

const mycar = new Model("Ford", "Mustang");

mycar.show();

The super() method refers to the parent class.

By calling the super() method in the constructor method, we call the parent's constructor method and get access to the parent's properties and methods.

To learn more about classes, check out our JavaScript Classes section.

React ES6 Arrow Functions

Arrow Functions

Arrow functions allow us to write shorter function syntax:

Example

Before:

hello = function() {

return "Hello World!";

}

Example

With Arrow Function:

hello = () => {

return "Hello World!";

}

It gets shorter! If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword:

Example

Arrow Functions Return Value by Default:

hello = () => "Hello World!";

Note: This works only if the function has only one statement.

If you have parameters, you pass them inside the parentheses:

Example

Arrow Function With Parameters:

hello = (val) => "Hello " + val;

In fact, if you have only one parameter, you can skip the parentheses as well:

Example

Arrow Function Without Parentheses:

hello = val => "Hello " + val;

What About this?

The handling of this is also different in arrow functions compared to regular functions.

In short, with arrow functions there is no binding of this.

In regular functions the this keyword represented the object that called the function, which could be the window, the document, a button or whatever.

With arrow functions, the this keyword always represents the object that defined the arrow function.

Let us take a look at two examples to understand the difference.

Both examples call a method twice, first when the page loads, and once again when the user clicks a button.

The first example uses a regular function, and the second example uses an arrow function.

The result shows that the first example returns two different objects (window and button), and the second example returns the Header object twice.

Example

With a regular function, this represents the object that called the function:

class Header {

constructor() {

this.color = "Red";

}

//Regular function:

changeColor = function() {

document.getElementById("demo").innerHTML += this;

}

}

const myheader = new Header();

//The window object calls the function:

window.addEventListener("load", myheader.changeColor);

//A button object calls the function:

document.getElementById("btn").addEventListener("click", myheader.changeColor);

Example:-

With an arrow function, this represents the Header object no matter who called the function:

class Header {

constructor() {

this.color = "Red";

}

//Arrow function:

changeColor = () => {

document.getElementById("demo").innerHTML += this;

}

}

const myheader = new Header();

//The window object calls the function:

window.addEventListener("load", myheader.changeColor);

//A button object calls the function:

document.getElementById("btn").addEventListener("click", myheader.changeColor);

React ES6 Variables

Variables

Before ES6 there was only one way of defining your variables: with the var keyword. If you did not define them, they would be assigned to the global object. Unless you were in strict mode, then you would get an error if your variables were undefined.

Now, with ES6, there are three ways of defining your variables: var, let, and const.

Example:-

var

var x = 5.6;

If you use var outside of a function, it belongs to the global scope.

If you use var inside of a function, it belongs to that function.

If you use var inside of a block, i.e. a for loop, the variable is still available outside of that block.

var has a function scope, not a block scope.

Example:-

let

let x = 5.6;

let is the block scoped version of var, and is limited to the block (or expression) where it is defined.

If you use let inside of a block, i.e. a for loop, the variable is only available inside of that loop.

The keyword const is a bit misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

Reassign a constant value

Reassign a constant array

Reassign a constant object

But you CAN:

Change the elements of constant array

Change the properties of constant object

Test Yourself With Exercises

Exercise:

Create a variable that cannot be changed.

X = 5.6;

React ES6 Array Methods

Array Methods:-

There are many JavaScript array methods.

One of the most useful in React is the .map() array method.

The .map() method allows you to run a function on each item in the array, returning a new array as the result.

In React, map() can be used to generate lists.

Example:-

Generate a list of items from an array:

const myArray = ['apple', 'banana', 'orange'];

const myList = myArray.map((item) => <p>{item}</p>)

Test Yourself With Exercises

Exercise:-

Complete the array method that will allow you to run a function on each item in the array and return a new array.

const myList = myArray.

((item) => <p>{item}</p>)

React ES6 Destructuring

Destructuring

To illustrate destructuring, we'll make a sandwich. Do you take everything out of the refrigerator to make your sandwich? No, you only take out the items you would like to use on your sandwich.

Destructuring is exactly the same. We may have an array or object that we are working with, but we only need some of the items contained in these.

Destructuring makes it easy to extract only what is needed.

Destructing Arrays:-

Here is the old way of assigning array items to a variable:

Example:-

Before:

const vehicles = ['mustang', 'f-150', 'expedition'];

// old way

const car = vehicles[0];

const truck = vehicles[1];

const suv = vehicles[2];

Here is the new way of assigning array items to a variable:

Example:-

With destructuring:

const vehicles = ['mustang', 'f-150', 'expedition'];

const [car, truck, suv] = vehicles;

When destructuring arrays, the order that variables are declared is important.

If we only want the car and suv we can simply leave out the truck but keep the comma:

const vehicles = ['mustang', 'f-150', 'expedition'];

const [car,, suv] = vehicles;

Destructuring comes in handy when a function returns an array:

Example:-

function calculate(a, b) {

const add = a + b;

const subtract = a - b;

const multiply = a \* b;

const divide = a / b;

return [add, subtract, multiply, divide];

}

const [add, subtract, multiply, divide] = calculate(4, 7);

Destructuring Objects

Here is the old way of using an object inside a function:

Example:-

Before:

const vehicleOne = {

brand: 'Ford',

model: 'Mustang',

type: 'car',

year: 2021,

color: 'red'

}

myVehicle(vehicleOne);

// old way

function myVehicle(vehicle) {

const message = 'My ' + vehicle.type + ' is a ' + vehicle.color + ' ' + vehicle.brand + ' ' + vehicle.model + '.';

}

Here is the new way of using an object inside a function:

Example:-

With destructuring:

const vehicleOne = {

brand: 'Ford',

model: 'Mustang',

type: 'car',

year: 2021,

color: 'red'

}

myVehicle(vehicleOne);

function myVehicle({type, color, brand, model}) {

const message = 'My ' + type + ' is a ' + color + ' ' + brand + ' ' + model + '.';

}

Notice that the object properties do not have to be declared in a specific order.

We can even destructure deeply nested objects by referencing the nested object then using a colon and curly braces to again destructure the items needed from the nested object:

Example:-

const vehicleOne = {

brand: 'Ford',

model: 'Mustang',

type: 'car',

year: 2021,

color: 'red',

registration: {

city: 'Houston',

state: 'Texas',

country: 'USA'

}

}

myVehicle(vehicleOne)

function myVehicle({ model, registration: { state } }) {

const message = 'My ' + model + ' is registered in ' + state + '.';

}

Test Yourself With Exercises

Exercise:-

Use destructuring to extract only the third item from the array, into a variable named suv.

const vehicles = ['mustang', 'f-150', 'expedition'];

const [

] = vehicles;

React ES6 Spread Operator

Spread Operator

The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

Example:-

const numbersOne = [1, 2, 3];

const numbersTwo = [4, 5, 6];

const numbersCombined = [...numbersOne, ...numbersTwo];

The spread operator is often used in combination with destructuring.

Example:-

Assign the first and second items from numbers to variables and put the rest in an array:

const numbers = [1, 2, 3, 4, 5, 6];

const [one, two, ...rest] = numbers;

We can use the spread operator with objects too:

Example:-

Combine these two objects:

const myVehicle = {

brand: 'Ford',

model: 'Mustang',

color: 'red'

}

const updateMyVehicle = {

type: 'car',

year: 2021,

color: 'yellow'

}

const myUpdatedVehicle = {...myVehicle, ...updateMyVehicle}

Notice the properties that did not match were combined, but the property that did match, color, was overwritten by the last object that was passed, updateMyVehicle. The resulting color is now yellow.

Test Yourself With Exercises

Exercise:-

Use the spread operator to combine the following arrays.

const arrayOne = ['a', 'b', 'c'];

const arrayTwo = [1, 2, 3];

const arraysCombined = [

];

React ES6 Modules

Modules

JavaScript modules allow you to break up your code into separate files.

This makes it easier to maintain the code-base.

ES Modules rely on the import and export statements.

Export

You can export a function or variable from any file.

Let us create a file named person.js, and fill it with the things we want to export.

There are two types of exports: Named and Default.

Named Exports

You can create named exports two ways. In-line individually, or all at once at the bottom.

Example:-

In-line individually:

person.js

export const name = "Jesse"

export const age = 40

All at once at the bottom:

person.js

const name = "Jesse"

const age = 40

export { name, age }

Get Certified!

Complete the React modules, do the exercises, take the exam and become w3schools certified!!

Default Exports

Let us create another file, named message.js, and use it for demonstrating default export.

You can only have one default export in a file.

Example:-

message.js

const message = () => {

const name = "Jesse";

const age = 40;

return name + ' is ' + age + 'years old.';

};

export default message;

Import

You can import modules into a file in two ways, based on if they are named exports or default exports.

Named exports must be destructured using curly braces. Default exports do not.

Example:-

Import named exports from the file person.js:

import { name, age } from "./person.js";

Example:-

Import a default export from the file message.js:

import message from "./message.js";

React JSX

What is JSX?

JSX stands for JavaScript XML.

JSX allows us to write HTML in React.

JSX makes it easier to write and add HTML in React.

Coding JSX

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.

JSX converts HTML tags into react elements.

You are not required to use JSX, but JSX makes it easier to write React applications.

Here are two examples. The first uses JSX and the second does not:

Example 1:-

JSX:

const myElement = <h1>I Love JSX!</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(myElement);

Example 2:-

Without JSX:

const myElement = React.createElement('h1', {}, 'I do not use JSX!');

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(myElement);

As you can see in the first example, JSX allows us to write HTML directly within the JavaScript code.

JSX is an extension of the JavaScript language based on ES6, and is translated into regular JavaScript at runtime.

Get Certified!

Complete the React modules, do the exercises, take the exam and become w3schools certified!!

Expressions in JSX:-

With JSX you can write expressions inside curly braces { }.

The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result:

Example:-

Execute the expression 5 + 5:

const myElement = <h1>React is {5 + 5} times better with JSX</h1>;

Inserting a Large Block of HTML

To write HTML on multiple lines, put the HTML inside parentheses:

Example:-

Create a list with three list items:

const myElement = (

<ul>

<li>Apples</li>

<li>Bananas</li>

<li>Cherries</li>

</ul>

);

One Top Level Element

The HTML code must be wrapped in ONE top level element.

So if you like to write two paragraphs, you must put them inside a parent element, like a div element.

Example:-

Wrap two paragraphs inside one DIV element:

const myElement = (

<div>

<p>I am a paragraph.</p>

<p>I am a paragraph too.</p>

</div>

);

JSX will throw an error if the HTML is not correct, or if the HTML misses a parent element.

Alternatively, you can use a "fragment" to wrap multiple lines. This will prevent unnecessarily adding extra nodes to the DOM.

A fragment looks like an empty HTML tag: <></>.

Example:-

Wrap two paragraphs inside a fragment:

const myElement = (

<>

<p>I am a paragraph.</p>

<p>I am a paragraph too.</p>

</>

);

Elements Must be Closed

JSX follows XML rules, and therefore HTML elements must be properly closed.

Example:-

Close empty elements with />

const myElement = <input type="text" />;

JSX will throw an error if the HTML is not properly closed.

Attribute class = className

The class attribute is a much used attribute in HTML, but since JSX is rendered as JavaScript, and the class keyword is a reserved word in JavaScript, you are not allowed to use it in JSX.

Use attribute className instead.

JSX solved this by using className instead. When JSX is rendered, it translates className attributes into class attributes.

Example:-

Use attribute className instead of class in JSX:

const myElement = <h1 className="myclass">Hello World</h1>;

Conditions - if statements

React supports if statements, but not inside JSX.

To be able to use conditional statements in JSX, you should put the if statements outside of the JSX, or you could use a ternary expression instead:

Option 1:

Write if statements outside of the JSX code:

Example:-

Write "Hello" if x is less than 10, otherwise "Goodbye":

const x = 5;

let text = "Goodbye";

if (x < 10) {

text = "Hello";

}

const myElement = <h1>{text}</h1>;

Option 2:

Use ternary expressions instead:

Example:-

Write "Hello" if x is less than 10, otherwise "Goodbye":

const x = 5;

const myElement = <h1>{(x) < 10 ? "Hello" : "Goodbye"}</h1>;

Note that in order to embed a JavaScript expression inside JSX, the JavaScript must be wrapped with curly braces, {}.

Test Yourself With Exercises

Exercise:

Render a <p> element without using JSX.

const paragraph = React.createElement(

, {}, 'This is a paragraph without using JSX!');

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(paragraph);