Source: https://www.robinwieruch.de/javascript-fundamentals-react-requirements
Most recent archive:
https://web.archive.org/web/20210217094557if_/https://www.robinwieruch.de/javascript-fundamentals-react-requirements

How to run a basic react file without making a backend

Source: https://stackoverflow.com/questions/46506332/run-simple-react-js-in-browser-locally

→Create Hello World node react app, then change the react file `App.js`

→Change into the newly created directory, then run `npm start`

How to debug a react file

1. Install the React Developmer Tools Browser Addon https://addons.mozilla.org/en-US/firefox/addon/react-devtools/
2. Run your hello world app
3. In your browser open the debugging stuff using `F12`
4. There should be two new tabs: `Components` and `Profiler`

How React files can look

Minimal setup:

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1>
            Hello React
          </h1>
          <a href="https://reactjs.org">
            Learn React
          </a>
        </header>
      </div>
    );
  }
}

export default App;
```

Using state and functions:

```
class Counter extends Component {
  state = {
    counter: 0,
  };

  onIncrement = () => {
    this.setState(state => ({ counter: state.counter + 1 }));
  }

  onDecrement = () => {
    this.setState(state => ({ counter: state.counter - 1 }));
  }

  render() {
    return (
      <div>
        <p>{this.state.counter}</p>

        <button
          onClick={this.onIncrement}
          type="button"
        >
          Increment
        </button>
        <button
          onClick={this.onDecrement}
          type="button"
        >
          Decrement
        </button>
      </div>
    );
  }
}
```

Classes

```
class Developer {
  constructor(firstname, lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }

  getName() {
    return this.firstname + ' ' + this.lastname;
  }
}

class ReactDeveloper extends Developer {
  getJob() {
    return 'React Developer';
  }
}

var me = new ReactDeveloper('Robin', 'Wieruch');

console.log(me.getName());
console.log(me.getJob());
```

→ The constructor method needs to be called `constructor`

→ To inherit from a class use `extends`

Arrow functions

```
// JavaScript ES5 function
function getGreeting() {
  return 'Welcome to JavaScript';
}

// JavaScript ES6 arrow function with body
const getGreeting = () => {
  return 'Welcome to JavaScript';
}

// JavaScript ES6 arrow function without body and implicit return
const getGreeting = () =>
  'Welcome to JavaScript';
```

Functions as Components in React

Instead of defining React components using classes, use functions!

```
// JavaScript ES5 function
function Greeting(props) {
  return <h1>{props.greeting}</h1>;
}

// JavaScript ES6 arrow function
const Greeting = (props) => {
  return <h1>{props.greeting}</h1>;
}

// JavaScript ES6 arrow function
// without body and implicit return
const Greeting = (props) =>
  <h1>{props.greeting}</h1>;
```

Template literals

```
getGreeting = (what) => {
  return `
    Welcome
    to
    ${what}
  `;
}

render() {
  return (
    <div>
      <p>{this.getGreeting("Hamster")}</p>
```

Map, Reduce, Filter

- `Map`: Iterate over an array. Per item do …
- `Filter`: Returns a subset of an array matching given conditions
- `Reduce`: Aggregates over an array. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce

```
var users = [
  { name: 'Robin', isDeveloper: true },
  { name: 'Markus', isDeveloper: false },
];

return (
  <ul>
    {users
      .filter(user => user.isDeveloper)
      .map(user => <li>{user.name}</li>)
    }
  </ul>
```

Variable types

- `var`: don’t use this anymore
- `let`: use when a variable needs to be reassigned (e.g. in a for loop)

- `const`: use when variable value is permanent

Ternary operators

Used for conditional rendering

2 ways of using them:

- `condition ? do if true : do if false`
- `condition && do if true`

```
const showUsers = false;

return (
  <div>
    {showUsers ? (
      <ul>
        {users.map(user => (
          <li>{user.name}</li>
        ))}
      </ul>
    ) : null}
  </div>
```

Import & Export

→When creating the hello world react app, `src/App.js` contains React code that is then imported in `src/index.js`

Export: Can be used to export variables, methods and classes

→`export default` is used to highlight the main functionality of the module we export from

Import: Can be used to import npm packages as well as variables, methods and classes from other files

→Can use relative paths when importing from other files

→Can use an alias when importing, i.e. `import x as y`

React packages and IDE addons

https://www.robinwieruch.de/react-libraries

Asynchronous JavaScript

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous

→ Use promises for asynchronous tasks

Higher-order functions

Idea: Functions can return other functions

Possible to use a shorthand for this, e.g.

```
const doFilter = query => user =>
  user.name.includes(query);
```

Useful shorthands

- Object property is called the same as another variable

```
const name = 'Robin';

const user = {
  name,
};
```

- Dynamic naming

```
const key = 'name';
const user = {
  [key]: 'Robin',
};
```

Destructuring

Idea: You can select a subset from an array of key value pairs, by specifying relevant keys

```
const state = { counter: 1, list: ['a', 'b'] }

// no object destructuring
const list = state.list;
const counter = state.counter;

// object destructuring
const { list, counter } = state;
```

This can be used to not carry `props` around all the time

```
// no destructuring
function Greeting(props) {
  return <h1>{props.greeting}</h1>;
}

// destructuring
function Greeting({ greeting }) {
  return <h1>{greeting}</h1>;
}
```

Alternatively, if the array doesn't have keys, you can imply the index

```
const list = ['a', 'b'];

// no array destructuring
const itemOne = list[0];
const itemTwo = list[1];

// array destructuring
const [itemOne, itemTwo] = list;
```

This is used by React hooks, e.g. for the state

```
const Counter = () => {
  const [count, setCount] = React.useState(0);

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

Spread Operator

The spread operator is 3 dots **...**

Its meaning is context dependent

When destructuring it can be used to bundle unspecified components of an array into the `...rest` variable

```
const state = { counter: 1, list: ['a', 'b'] };

// rest destructuring
const { list, ...rest } = state;

console.log(rest);
// output: { counter: 1 }
console.log(list);
// output: ['a', 'b']
```

When passing an object around this can save you from always explicitely writing each property that is being passed

```
const App = () => {
  const users = [
    { name: 'Robin', nationality: 'German' },
    { name: 'Markus', nationality: 'American' },
  ];

  return (
    <ul>
      {users.map(user => <li>
        <User {...user} />
      </li>)}
    </ul>
  );
};

const User = ({ name, nationality }) =>
  <span>{name} from {nationality}</span>;
```