Basics of JavaScript

Variables and types

Objects and arrays

Functions and Classes

Loops and conditionals

Why React?

It’s a JavaScript library for building user interfaces

React is small and it's not a complete solution, we will need to use other libraries with react to form solutions.

It focuses on one thing building user interfaces.

How exactly is NOT being a framework a good thing?

Frameworks serves a great need for young projects and startups. Many smart designs decisions are already made, which gives a clear path to focus on writing good application logic.

However frameworks come with some disadvantages as well

Limited flexibility

- Do things a certain way

- Hard to deviate

Large and full of features

- Hard to customize

- Use the whole thing

How react gained popularity?

The "virtual" browser (vs DOM API) that is friendlier than real browser.

Performance with the virtual browser.

"Just JavaScript" - a very small react API to learn after that the JavaScript skills are what make you a better react developer.

React Native (for the win)

Learning React pays off big time for IOS and Android mobile applications. React Native allows to use same react skills to build native application.

Battle-tested

The react and Facebook team tests all the improvements, right on the facebook.com, which increases the trust in the library among the community

Declarative language (model UI and state)

React established a new language between developers and browsers that allow developers to declaratively describe stateful user interfaces

ECMA Script

These days, we preprocess our JavaScript with Babel in order to access experimental features and language extensions like JSX.

1. Variables and Block Scopes

JavaScript used to be a different language few years ago,

{{{{

// Block Scope

var a = 40;

}}}}

a

if (true) {

// Block Scope

}

for (var i = 1; i <= 10; i++) {

// Block Scope

}

i

function sum(a, b) {

// Function Scope

var result = a + b;

}

sum(4 + 3);

result;

Variables of a function doesn't leaks out of the function scope even mentioned with var

1. Arrow Functions

// "this" here is the caller of X

}

the value of this is determined by the calling environment

const Y = () => {

// "this" here is NOT the caller of Y

//It's the same "this" found in Y's scope

}

an arrow function on the other hand not care on who has called it

const testerObj = {

func1: function(){

console.log('func1', this);

},

func2:() => {

console.log('func2', this);

},

};

testerObj.func1();

testerObj.func2();

const squarel = (a) => {

return a\*a;

}

const square2 = (a) => a\*a;

const square2 = a => a\*a;

[1,2,3,4].map(a => console.log(a\*a));

1. Object Literals

We can create a JavaScript object in a few different ways, but the most common way is to use the object literal.

// [1,2,3,4].map(a => console.log(a\*a));

const mystery = 'answer';

const obj = {

p1 : 10, //default properties

p2: 20,

f1() {},

f2: () => {},

[mystery]: 42, // mystery is defined with dynamic property syntax

}

If we want a property to hold a function use this short syntax while defining object literals

If you want to use an arrow function use this regular syntax in object literal

modern objects also support dynamic properties using this syntax, it looks like an array literal but don’t confuse with that. Let’s evaluate what’s within the square brackets

Assuming that we define a “mystery” before the object liters let’s check the out in the playground

console.log(obj.answer);

Objects are very popular in JavaScript, they are used to manage and communicate data and using these features will make the code shorter and easier to read.

We can create a javascript objetc in a few different ways,but the most common way is to use the object literal

This is a lot easier than creating something like a new object

1. Destructing and Rest/Spread

const [first, ...restOfItems] = [10, 20, 30, 40];

console.log(first);

console.log(restOfItems);

1. Template Strings

Sample

We can define string literals is JavaScript in single quotes or double quotes

The modern JavaScript has third way of defining strings, and that’s by using back tick character, they are called as template string as they can be used for dynamic value. They support something called interpolation. We can inject any dynamic values within JavaScript between these curly braces

const html = `<div>${Math.random()}</div>`;

We can have multiple lines breakdown with the template strings, which is not often seen in other string formats.

1. Classes or Class Components

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

You can also use an ES6 class to define a component:

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

The above two components are equivalent from React’s point of view.

1. Promises and Async/Await

To work with asynchronous object we need to deal with promises objects, a promise is an object that might deliver the data at the later point in program.

const fetchData = async () => {

const resp = await fetch('https://api.github.com');

const data = await resp.json();

console.log(data);

};

fetchData();

Example of a async function that returns a promise is the web fetch api which is natively available in the some browser. Here we are fetching the top level github api, when the fetch returns a promise we then call .ten function to return the return of fetch and supply the callback function in here, this callback functon will recieve the data from the API

The fetch API has the raw response, if we want to parse the data in JSON format we need to call the .json() method on the response object.

1. View Function

ECMAScript is the language specification used to implement the JavaScript language.

However, there are many new features in the latest versions of JavaScript that we'd like to use. Thanks to Babel, we can use them today! Babel transforms newer features into ES5 for cross-platform compatibility.