**Introduction to ES5 & ES6**

* **ES5**

ES5 is also known as ECMAScript 2009 as it is released in 2009. It is a function contractors focus on how the objects are instantiated. For ES5 you have to write function keyword and return, to be used to define the function, like normal general JavaScript language.

* **ES6**

ES6 is also known as ECMAScript 2015 as it is released in 2015. Its class allows the developers to instantiate an object using the new operator, using an arrow function, in case it doesn’t need to use function keyword to define the function, also return keyword can be avoided to fetch the computer value.

* **ES5 VS ES6**

|  |  |  |
| --- | --- | --- |
| SR.NO. | ES5 | ES6 |
| 1. | ECMA script is a trademarked scripting language specification defined by Ecma international. The fifth edition of the  same is known as ES5 | ECMA script is a trademarked scripting language specification defined by Ecma international. The sixth edition of the same is known as ES6 |
| 2. | It was introduced in 2009. | It was introduced in 2015. |
| 3. | It supports primitive data types that are string, number, boolean, null, and undefined. | In ES6, there are some additions to JavaScript data types. It introduced a new primitive data type ‘symbol’ for supporting unique values. |
| 4. | There are only one way to define the variables by using the var keyword. | There are two new ways to define variables that are let and const. |
| 5. | It has a lower performance as compared to ES6. | It has a higher performance than ES5. |
| 6. | Object manipulation is time-consuming in ES5. | Object manipulation is less time-consuming in ES6. |
| 7. | In ES5, both function and return keywords are used to define a function. | An arrow function is a new feature introduced in ES6 by which we don’t require the function keyword to define the function. |
| 8. | It provides a larger range of community supports than that of ES6 | It provides a less range of community supports than that of ES5 |

* **ES5 VS ES6 Syntax Difference**

|  |  |
| --- | --- |
| Using React’s ES5 syntax | Using React’s ES6 syntax |
| var React = require("react"); // ES5  var ReactDOM = require("react-dom"); // ES5  var createReactClass = require("create-react-class"); // ES5  // ES5 Syntax  var CountComp = createReactClass({  getInitialState: function () {  return {  counter: 0,  };  },  Increase: function () {  this.setState({  counter: this.state.counter + 1,  });  },  Decrease: function () {  this.setState({  counter: this.state.counter - 1,  });  },  render: function () {  return (  <div>  <button onClick={this.Increase}>  Count increase  </button>  <h1> {this.state.counter} </h1>  <button onClick={this.Decrease}>  Count decrease  </button>  </div>  );  },  });  // ES5 Syntax  var Component = createReactClass({  render: function () {  return (  <div>  <CountComp />  </div>  );  },  });  ReactDOM.render(<Component />,  document.getElementById("root")); | import React from "react"; // ES6  import ReactDOM from "react-dom"; // ES6  let CountComp = (Compprop) =>  class extends React.Component {  constructor(props) {  super(props);  this.state = {  counter: 0,  };  this.Increase = this.Increase.bind(this);  this.Decrease = this.Decrease.bind(this);  } // ES6 Syntax  // ES6 Syntax  Increase() {  this.setState({  counter: this.state.counter + 1,  });  }  Decrease() {  this.setState({  counter: this.state.counter - 1,  });  }  render() {  return (  <div>  <Compprop  {...this.state}  increase={this.Increase}  decrease={this.Decrease}  />  </div>  );  }  };  // ES6 Syntax  const Button = (props) => {  return (  <div>  <button onClick={props.increase}>  Count increase  </button>  <h1> {props.counter} </h1>  <button onClick={props.decrease}>  Count decrease  </button>  </div>  );  };  // ES6 Syntax  let CompClick = CountComp(Button);  const Component = () => {  return (  <div>  <CompClick />  </div>  );  +  ReactDOM.render(<Component />,  document.getElementById("root")); |

* **ES6 Advantages**



* Arrows

Arrows are a function shorthand using the => syntax. There is a similarity between the syntactical features of C#, Java 8, CoffeeScript and ES6. Both statement block bodies and expression bodies are supported by them, which return the value of expression. Unlike functions, arrows share the same lexical this as their surrounding code.

* Classes

In ES6, having a single conducive declarative pattern makes class patterns painless thing to use and boosts interoperability. They are a simple sugar over the prototype-based OO pattern. Inheritance, instance and static methods are supported by Classes which makes ES6 more amiable version of Javascript.

* Enhanced Object Literals

Purpose of object literals is to support setting the prototype at construction, shorthand for foo: foo assignments, defining methods, making super calls, and computing property names with expressions. When combined they bring object literals and class declarations closer together, and benefits object-based design from some of the same conveniences.

* Modules

In ES6 modules provide a course to load and manage dependencies through the new import export keywords. Even after having good solutions like 3rd party libraries in ES5, modularity remains an important concept for large applications. The following goals of this version make it an indispensable language feature:

Anticipate and prevent need of globals

Easy reallocation from global code to modular code

Smooth interoperability with existing JS module systems like AMD, CommonJS, and Node.js

Quick collation

Standardized protocol for sharing libraries

Compatibility with browser and non-browser environments

Easy asynchronous external loading

* Block Scoping

Scoping becomes an ambiguous affair for developers who has background of C/C#/Java. Hoisting can add to that confusion. ES5 lacked a very essential feature of block scoping, which deranked it in comparison to ES6. Though need became the mother of invention and block scoping came into the scenario. ES6 emerged out with this feature, block scoping can be achieved through using let keyword.

* Promises

The ES6 is a promising language, where there is an asynchronous operation, there are promises to handle its results and errors. Though callbacks are also designed for the same, but promises provide improved readability via method chaining and succinct error handling. Many JavaScript libraries using Promises, RSVP.js, Q.js, and the$q  in [Angularjs service](https://www.cuelogic.com/product-development" \t "_blank) are a few of many examples.

The above features of ES6 are potent enough to make us pursue it.But let us evaluate the real scope of this technology before plunging into it.