Getting started with React

Dr. Vivek Bhardwaj

Assistant Professor

Introduction to React and JSX

Introduction to React

- React is an open-source front-end framework
- React introduces JSX, or JavaScript XML
    - JSX can be used to create React components
    - React, sometimes referred to as a frontend JavaScript framework, is a JavaScript library created by Facebook.
    - React is a tool for building UI components.

What is ES6?

JavaScript ES6 brings new features and new syntax which has made your code more readable and more modern. It enables you to do more by writing less code. ES6 has also introduced to us a lot of great features like template strings, modules, arrow functions, class destruction… and more

Javascript ES6 is the version of Javascript released in 2015. Informally, many developers call any version of Javascript released after 2015 “ES6” because of the many new features the 2015 update added to the language.

ES6

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

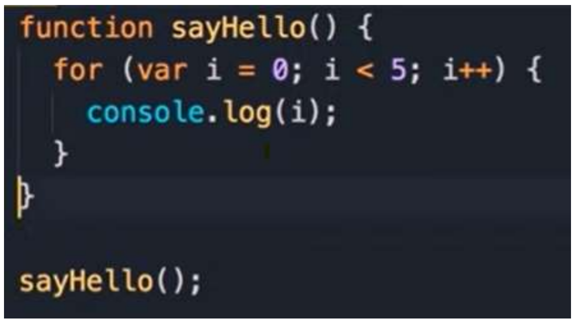
- Classes
- Arrow Functions
- Variables (let, const, var)
- Array Methods like .map()
- Destructuring
- Modules
- Ternary Operator
- Spread Operator

Variables

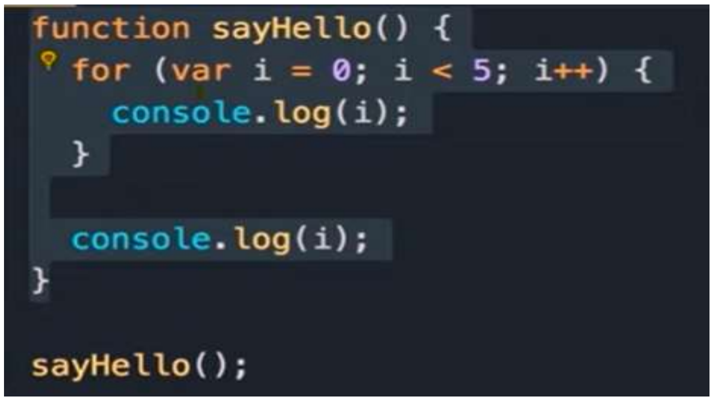
Before ES6 there was only one way of defining your variables: with the var keyword

```
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.



```
function sayHello() {
  for (var i = 0; i < 5; i++) {
    console.log(i);
  }
}

sayHello();
```



```
function sayHello() {
  for (var i = 0; i < 5; i++) {
    console.log(i);
  }

  console.log(i);
}

sayHello();
```

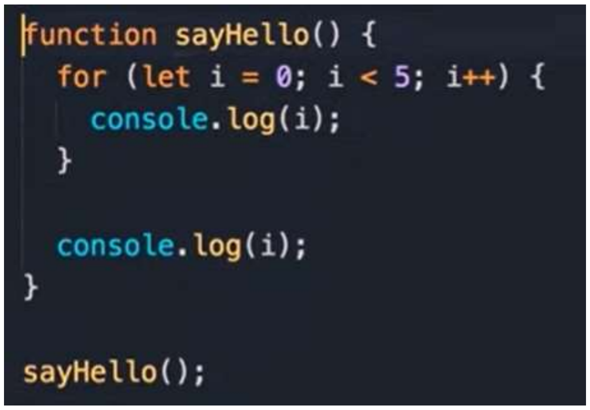
Variables

ES6, there are three ways of defining your variables: `var`, `let`, and `const`

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.
`let` has a *block* scope.



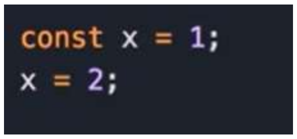
```
function sayHello() {
  for (let i = 0; i < 5; i++) {
    console.log(i);
  }

  console.log(i);
}

sayHello();
```

const

```
const x = 5.6;
```

`const` is a variable that once it has been created, its value can never change.

`const` has a *block* scope.



```
const x = 1;
x = 2;
```

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

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.

Syntax: Class Declaration

```
class Class_name{
}
```

Example - Class Declaration

```
class Student{
    constructor(name, age){
    this.name = name;
    this.age = age;
    }

}
```

Instantiating an Object from class

Syntax

```
var obj_name = new class_name([arguements])
```

Example

```
var stu = new Student('Peter', 22)
```

Accessing functions

The object can access the attributes and functions of a class. We use the **'.' dot notation (**or **period)** for accessing the data members of the class.

Syntax

```
obj.function_name();
```

```
class Student {
  constructor(name, age) {
    this.n = name;
    this.a = age;
  }
  stu() {
    console.log("The Name of the student is: ", this.n)
    console.log("The Age of the student is: ",this. a)
  }
}

var stuObj = new Student('Peter',20);
stuObj.stu();
```

Classes

Example

A simple class constructor:

```
class Car { constructor(name) {
this.brand = name;
 }
}
```

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

```
class Car { constructor(name) {
this.brand = name;
      }
} const mycar = new Car("Ford");
```

The constructor function is called automatically when the object is initialized

You can add your own methods in a class:

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();
```

you call the method by referring to the object's method name followed by parentheses

The Static keyword

The **static** keyword is used for making the static functions in the class. Static functions are referenced only by using the class name.

Example

```
'use strict'
class Example {
  static show() {
    console.log("Static Function")
  }
}

Example.show() //invoke the static method
```

React ES6 Arrow Functions

Arrow functions allow us to write shorter function syntax:

Syntax for defining the arrow function

```
const functionName = (arg1, arg2, ?..) => {
    //body of the function
}
```

There are three parts to an Arrow Function or Lambda Function:

- **Parameters:** Any function may optionally have the parameters.
- **Fat arrow notation/lambda notation:** It is the notation for the **arrow (=>).**
- **Statements:** It represents the instruction set of the function.

```
/ function expression

var myfun1 = function show() {
 console.log("It is a Function Expression");
}

//Arrow function

var myfun2 = () => {
    console.log("It is an Arrow Function");
  };

myfun1();
myfun2();
```

Syntactic Variations

There are some syntactic variations for the arrow functions that are as follows:

•**Optional parentheses for the single parameter**

```
var num = x => {
    console.log(x);
}
num(140);
```

•**Optional braces for single statement and the empty braces if there is not any parameter required.**

```
var show = () => console.log("Hello World");
show();
```

Arrow Function with Parameters

If you require to pass more than one parameter by using an arrow function, then you have to pass them within the parentheses.

For example

```
var show = (a,b,c) => {
    console.log(a + " " + b + " " + c );
}
show(100,200,300);
```

Arrow function with default parameters

```
var show = (a, b=200) =>
 { console.log(a + " " + b);
}
show(100);
```

```
var show = (a, b=200) => {
    console.log(a + " " + b);
}
show(100,500);
```

Arrow Function without Parentheses

```
var show = x => {
    console.log(x);
}
show("Hello World");
```

React ES6 Arrow Functions

Arrow functions allow us to write shorter function syntax:

Before:

```
hello = function() {
return "Hello World!"; }
```

With Arrow Function:

```
hello = () => {
 return "Hello World!";
}
```

Arrow Functions Return Value by Default:

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

Arrow Function With Parameters:

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

Arrow Function Without Parentheses:

```
hello = val => "Hello " + val;
```

Arrays

Syntax

There are two ways of declaring an array.

```
var array_name = new Array();  // By using the new keyword

var array_name = [value1, value2,....valueN];  //By using Array literals
                  or,
var array_name;   //Declaration
array_name=[value1, value2,…..valueN]; //Initialization
```

Accessing Array elements

Arrays are indexed from 0. The array name followed by the subscript is used for referring an array element.

Syntax

```
1.array_name[subscript];
```

Example

```
1.var num;
2.num = [2,4,6,8];
3.console.log(num[0]);
4.console.log(num[1]);
```

Array Constructor

You create an array by using the array constructor. Array constructor can be passed as:

•A list of values separated by comma or,

•A numeric value which indicates the size of array

Example - Single Numeric Value

```
var num = new Array(5); // This single numeric value indicates the size of array.
var i;
for(i=0;i<num.length;i++){
num[i]=i*5;
console.log(num[i]);
}
```

Example - Comma Separated Values

```
var num = new Array(1,2,3,4,5);
var i;
for(i=0;i<num.length;i++){
console.log(num[i]);
}
```

ES6 Multidimensional Arrays

ES6 also supports the multidimensional array concept. A multidimensional array can be defined as an array reference to another array for its value.
a two-dimensional array is the simplest form of a multidimensional array.

Declaration

The following syntax illustrates you how to declare two-dimensional array in JavaScript.

```
1.var array_name = [[value1,value2,value3],[val1,val2,val3]];
```

Accessing of Two-dimensional element of array

```
1.var array_name[initial_array_index][referenced_array_index]
```

```
var multi = [[2,3,4],[4,9,16]]
console.log(multi[0][0])
console.log(multi[0][1])
console.log(multi[0][2])
console.log(multi[1][0])
console.log(multi[1][1])
console.log(multi[1][2])
```

Passing Array to function

Passing array as an argument to a function, you have to specify the array name (a reference to an array) without brackets. Let us try to understand it with the following example.

Example

```
var rainbow = new Array("Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red“);
function show(rainbow) {
  for(var i = 0;i<rainbow.length;i++) {
    console.log(rainbow[i])
  }
}
show(rainbow)
```

```
var rainbow = new Array("Violet", "Indigo", "Blue",
"Green", "Yellow", "Orange", "Red");
var show =(rainbow)=>{
   for(var i = 0;i<rainbow.length;i++)
{
      console.log(rainbow[i])
   }
}
show(rainbow)
```

Return Array from function

```
function show() {
   return new Array("Blue", "Red", "Green", "Yellow")
}
var colors = show()
for(var i in colors) {
   console.log(colors[i])
}
```

ES6 Array methods

The array methods introduced in ES6 are tabulated below.

S.no.	Methods	Description
1.	Array.from()	It converts array-like values and iterable values into arrays.
2.	Array.of()	It creates an instance from a variable number of arguments instead of the number of arguments or type of arguments.
3.	Array.prototype.copyWithin()	It copies the part of an array to a different location within the same array.
4.	Array.prototype.find()	It finds a value from an array, based on the specific criteria that are passed to this method.
5.	Array.prototype.findIndex()	The Array.prototype.findIndex() returns the index of the first element of the given array that satisfies the given condition.
6.	Array.prototype.entries()	It returns an array iterator object, which can be used to loop through keys and values of arrays.
7.	Array.prototype.keys()	It returns an array iterator object along with the keys of the array.
8.	Array.prototype.values()	it provides the value of each key.
9.	Array.prototype.fill()	It fills the specified array elements with a static value

Array.from()

It converts array-like values and iterable values into arrays.

Example

```
let name = Array.from('javaTpoint')

console.log(name)
```

Array.of()

In ES5, when a single numeric value gets passed in the array constructor, then it will create an array of that size. **Array.of()** is a new way of creating an array which fixes this behavior of ES5. By using this method, if you are creating an array only with a single numeric value, then it will create an array only with that value instead of creating the array of that size.

```
let name = Array.of(42,10)

console.log(name)
console.log(name.length)
)
```

Array.copyWithin()

This method copies the part of an array to a different location within the same array. It returns the modified array without any modification in its length.

Syntax

```
array.copyWithin(target, start, end)
```

```
const num = [1,2,3,4,5,6,7,8,9,10];
const num1 = [1,2,3,4,5,6,7,8,9,10];
const num2 = [1,2,3,4,5,6,7,8,9,10];
console.log(num.copyWithin(1,3,5));
console.log(num1.copyWithin(1,3)); //omitting the parameter end
console.log(num2.copyWithin(1)); //omitting the parameters start and end
```

Array.prototype.find()

It finds a value from an array, based on the specific criteria that are passed to this method. It returns the first element value that satisfies the given condition.

```
const numbers4 = [4, 9, 16, 25, 29];
let first = numbers4.find(myFunction);

function myFunction(value,index, array) {
  return value > 7;
}
console.log(first);
```

Array Methods

`map()` creates a new array from calling a function for every array element.
`map()` calls a function once for each element in an array.
`map()` does not execute the function for empty elements.
`map()` does not change the original array.

```
const numbers = [65, 44, 12, 4];
const newArr = numbers.map(myFunction)

function myFunction(num) {
  return num * 10;
}
console.log(newArr);
```

```
const numbers1 = [4, 9, 16, 25];
const newArr1 = numbers1.map(Math.sqrt)
console.log(newArr1);
```

The **map()** method **creates a new array** populated with the results of calling a provided function on every element in the calling array.

```
const array1 = [1, 4, 9, 16];// Pass a function to map

const map1 = array1.map(x => x * 2);

console.log(map1);

// Expected output: Array [2, 8, 18, 32]
```

React ES6 Destructuring

Destructuring means to break down a complex structure into simpler parts. With the syntax of destructuring, you can extract smaller fragments from objects and arrays. It can be used for assignments and declaration of a variable.

ES6 Array destructuring

Object destructuring

Destructuring is an efficient way to extract multiple values from data that is stored in arrays or objects. When destructuring an array, we use their positions (or index) in an assignment.

Before:

```
const vehicles = ['mustang', 'f-150', 'expedition']; // old way
const car = vehicles[0];
const truck = vehicles[1];
const suv = vehicles[2];
```

With destructuring:

```
const vehicles = ['mustang', 'f-150', 'expedition’];
const [car, truck, suv] = vehicles;
```

Array destructuring

Example

```
var arr = ["Hello", "World"]

// destructuring assignment
var [first, second] = arr;

console.log(first); // Hello
console.log(second); // World
```

Example

```
var colors = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"];

// destructuring assignment
var[color1, color2, color3] = colors;

console.log(color1); // Violet
console.log(color2); // Indigo
console.log(color3); // Blue
```

Example

If you want to choose random elements from the given array then in array destructuring you can perform it as follows:

```
var colors = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"];

// destructuring assignment
var[color1, ,color3, ,color5] = colors; //Leave space for unpick elements
console.log(color1); // Violet
console.log(color3); // Blue
console.log(color5); // Yellow
```

Array destructuring and Rest operator

By using the rest operator (…) in array destructuring, you can put all the remaining elements of an array in a new array.

```
var colors = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"];

// destructuring assignment
var [a,b,...args] = colors;
console.log(a);
console.log(b);
console.log(args);
```

Array destructuring and Default values

If you are taking a value from the array and that value is undefined, then you can assign a default value to a variable.

```
var x, y;

Var [x=50, y=70] = [100];
console.log(x); // 100
console.log(y); // 70
```

Swapping Variables

The values of the two variables can be swapped in one destructuring expression. The array destructuring makes it easy to swap the values of variables without using any temporary variable.

```
var x = 100, y = 200;
Var [x, y] = [y, x];
console.log(x); // 200
console.log(y); // 100
```

Parsing returned array from functions

A function can return an array of values. It is always possible to return an array from a function, but array destructuring makes it more concise to parse the returned array from functions.

Example

```
function array() {
    return [100, 200, 300];
}

var [x, y, z] = array();

console.log(x); // 100
console.log(y); // 200
console.log(z); // 300
```

```
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);
console.log(add);
console.log(subtract);
```

React ES6 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.

Syntax

```
1.var variablename1 = [...value];
```

Constructing array literal

When we construct an array using the literal form, the spread operator allows us to insert another array within an initialized array.

Example

```
let colors = ['Red', 'Yellow'];
let newColors = [...colors, 'Violet', 'Orange', 'Green'];
console.log(newColors);
```

Concatenating arrays

```
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];
console.log(numbersCombined);
```

Copying an array

```
let colors = ['Red', 'Yellow'];
let newColors = [...colors];
console.log(newColors);
```

Example

Without using spread operator

```
let colors = ['Red', 'Yellow'];
let newColors = colors;
newColors.push('Green');
console.log(newColors);
console.log(colors);
```

Using spread operator

```
let colors = ['Red', 'Yellow'];
let newColors = [...colors];
newColors.push('Green');
console.log(newColors);
console.log(colors);
```

If we copy the array elements without using the spread operator, then inserting a new element to the copied array will affect the original array.

But if we are copying the array by using the spread operator, then inserting an element in the copied array will not affect the original array.

Destructuring Objects

```
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 + '.';
}
```

```
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 + '.';
}
```