Arrow Functions

- Arrow functions are a new feature in ES6.
- They are a shorter syntax for writing functions.
- They are a little bit more concise.
- Lets take an example to make a function expression to add two numbers.
- Prior to ES6, we would write this:

```
let addTwo = function (a, b) {
  return a + b;
};
```

- After ES6, we can write this:

```
let addTwo = (a, b) => a + b;
```

- The main difference is that we don't need to use the return keyword and we need not require to write the function keyword to define an arrow function.
- `=>` is known as fat arrow or lambda.
- Comprehensively speaking, an arrow function is a function expression that does not have a name.
- Arrow functions are always anonymous.

Syntax

- Without parameters we need to write `() => {expression}`

```
let greet = () => {
  console.log("Hello");
};
```

- With parameters we need to write `(param1, param2) => {expression}`

```
let addTwo = (a, b) => a + b;
```

- With just 1 parameter we can choose to omit () and write `param => {expression}`

```
let incByTwo = a => a + 2;
```

Table of values of `this` keyword in different environments and different modes

- Node Environment

`this` Context	Non Strict Mode	Strict Mode
Global Context	Empty Object {}	Empty Object {}
Function Context	Global Object {}	undefined
Object Context	Object itself	Object itself
Object Function Function Context	Global Object	undefined

- Browser Environment

`this` Context	Non Strict Mode	Strict Mode
Global Context	Window Object	Window Object
Function Context	Window Object	undefined
Object Context	Object itself	Object itself
Object Function Function Context	Window Object	undefined

- The thing with react is it by default runs on strict mode.
- Arrow functions are not bound to the `this` keyword, we can use `bind` method to bind the `this` keyword.
- Normally, we can use variables simply by using the name of the variable but in arrow functions, we cannot access the variables.
- this in context of arrow functions is empty object.

```
let test = () => {
  console.log(this); // Empty Object {}
};
test();
```

- With arrow function inside an object when we access object props using this, we cant as this is empty object.

```
let person = {
  name: "John",
  age: 30,
  sayName: function () {
    console.log(this.name);
  },
  sayNameArrow: () => {
```

```
    console.log(this.name);
  },
};
person.sayName(); // John
person.sayNameArrow(); // undefined
```

- Advantages of arrow functions:
  - They are shorter.
  - They are anonymous.
  - this is handled differently in different environments, here in arrow function is empty object (sorted, phew!)

Class and Constructor in JavaScript

- Sorry to brek it to you but, Classes in JavaScript are syntactic sugar for creating objects.
- Classes in other programming languages are called as `blueprints of objects` or `prototypes`.
- Classes in js are used as templates for creating objects.

For example

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  sayName() {
    console.log(this.name);
  }
}
```

- The above class is called as a constructor function.
- The constructor function is called when we create an object using the class.
- The constructor function is used to initialize the properties of the object.
- The constructor function is basically used to create the object.

Constructor function

```
function car(name, model, year) {
  this.name = name;
  this.model = model;
  this.year = year;
  this.test = function driving() {
    console.log(`I am driving ${this.name}, ${this.model} from
${this.year}`);
  };
  }
}
```

```
let car1 = new car("Honda", "Civic", "2018");
let car2 = new car("Toyota", "Corolla", "2019");
console.log(car1); // { name: 'Honda', model: 'Civic', year: '2018' }
console.log(car2); // { name: 'Toyota', model: 'Corolla', year: '2019' }
car1.test(); // I am driving Honda, Civic from 2018
car2.test(); // I am driving Toyota, Corolla from 2019
```

Class in JavaScript

- Understanding constructor function will help us understand class in JavaScript.
- Lets create a class Person.

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  sayName() {
    console.log(this.name);
  }
}
```

- constructor function mandatorily needs to be named using constructor keyword.
- constructor function mandatorily needs to be called using new keyword in order to create an object.

```
let p1 = new Person("John", 30);
let p2 = new Person("Jane", 25);
console.log(p1); // Person { name: 'John', age: 30 }
console.log(p2); // Person { name: 'Jane', age: 25 }
p1.sayName(); // John
p2.sayName(); // Jane
```

Example 2

```
class Teacher {
  constructor(name, age, subject) {
    this.name = name;
    this.age = age;
    this.subject = subject;
  }
  sayName() {
    console.log(this.name);
  }
}
```

Inheritance in JavaScript

- Inheritance is a way to create new classes from existing classes.
- Inheritance is a way to reuse the code of existing classes.
- In here comes the usage of super keyword.
- The super keyword is used to call the parent class constructor, methods and properties.

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  sayName() {
    console.log(this.name);
  }
}

class Teacher extends Person {
  constructor(name, age, subject) {
    super(name, age);
    this.subject = subject;
  }
  saySubject() {
    console.log(this.subject);
  }
}
let t1 = new Teacher("John", 30, "Math");
t1.sayName(); // John
t1.saySubject(); // Math

class Student extends Person {
  constructor(name, age, grade) {
    super(name, age);
    this.grade = grade;
  }
  sayGrade() {
    console.log(this.grade);
  }
}
let s1 = new Student("Jane", 25, "A");
s1.sayName(); // Jane
s1.sayGrade(); // A
```

- The super keyword is used to call the parent class constructor, methods and properties.
- Syntax: `super(parameters)` where parameters are the parameters of the parent class constructor.
- These super call are done in child class constructor which is called when we create an object using the child class.
- These properties gets inherited from the parent class.