academeez

Object Oriented Programming

Class, inheritance, etc.

Object Oriented Programming

- In OOP Programming we structure our code in classes which are objects containing common methods and properties for each class
- We can look at a class as a recipe for creating an object with common behaviour
- OOP is a programming paradigm that is commonly used for building modern Javascript apps
- In this lesson we will learn how OOP works in JavaScript

Before ES6

- Before ES6 class in Javascript was represented as a function
- The following example shows a simple class, with a property and method, and we create an instance of that class

```
function Person(name) {
    this.name = name;
}

Person.prototype.sayHello = function() {
    console.log(`Hello ${this.name}`);
}

const me = new Person('Yariv');
me.sayHello()
```

After ES6

academeez

- ES6 gave us a new syntax for defining a class
- The following example shows the same class defined in ES6

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

    sayHello() {
        console.log(`Hello ${this.name}`);
    }
}

const me = new Person('Yariv');
me.sayHello()
```

What we will learn - class

- The new syntax is simply a syntax sugarcoat of the old one
- The old one can still be used
- We will focus on this lesson on the new **class** syntax which today is the more common way to describe our OOP classes
- It’s highly recommended to view the prototype lesson before this lesson

Inheritance

- A class can inherit the methods and properties of a parent class using the **extends**
- If constructor is defined in child it has to call **super()** to trigger the constructor on the parent

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

class Student extends Person {
    constructor(name, grade = 100) {
        super(name);
        this.grade = grade;
    }
}

const me = new Student('yariv', 80);
console.log(me.grade);
```

super

academeez

- We can also use super to call methods on the parent

```
class Person {
    constructor(name) { ... }

    sayHello() {
        console.log(`Hello my name is ${this.name}`);
    }
}

class Student extends Person {
    constructor(name, grade = 100) { ... }

    sayHello() {
        super.sayHello();
        console.log(`and my grade is ${this.grade}`);
    }
}
```

Class properties

- Class properties are attached to this, can also be defined in the class body
- Experimental feature: private properties

```
class Person {
    country = 'Israel';
    #age = 35;
    constructor(name, age) {
        this.name = name;
        this.#age = age;
    }
}
```

Getter setter properties

- Getters and setters are functions that are accessed like properties

```
class Person {
    constructor(firstName, lastName) { ... }

    get fullName() {
        return `${this.firstName} ${this.lastName}`
    }

    set fullName(value) {
        this.firstName = value.split(' ')[0];
        this.lastName = value.split(' ')[1];
    }
}

const me = new Person('yariv', 'katz');
console.log(me.fullName);
me.fullName = 'Pigletshvilly Chaitovski';
console.log(me.fullName);
```

static

- Static methods are methods attached to the class object
- Used for utilities function, create methods, that are not related to instances

```
class Person {
    static nameOfClass() {
        return 'Person';
    }
}

console.log(Person.nameOfClass());
```

this

- **this** in Javascript is usually dynamic and can change
- It’s determined by the object calling the method

```
class Person {
    printThis() {
        console.log(this);
    }
}

const me = new Person();
me.printThis(); // this is me
const a = { printThis: me.printThis};
a.printThis(); // this is a
const printThis = me.printThis;
printThis(); // this is undefined
```

Binded this

academeez

- You can bind **this** to the instance using **bind** or **arrow class method**

```
class Person {
    printThis = () => {
        console.log(this);
    }
}

const me = new Person();
me.printThis(); // this is me
const a = { printThis: me.printThis};
a.printThis(); // this is me
const printThis = me.printThis;
printThis(); // this is me
```

Binded this

academeez

- The same can be achieved with **Bind**

```
class Person {
    constructor() {
        this.printThis = this.printThis.bind(this);
    }

    printThis() {
        console.log(this);
    }
}

const me = new Person();
me.printThis(); // this is me
const a = { printThis: me.printThis};
a.printThis(); // this is me
const printThis = me.printThis;
printThis(); // this is me
```

Binded this - caveats

- The binded method will be attached to the object so it will be created again for every object
- On child parent inheritance the child won't be able to call the method with super

```
class Person {
    printThis = () => {
        console.log(this);
    }
}

class Student extends Person {
    printThis = () => {
        super.printThis();
    }
}

const me = new Student();
me.printThis(); // ERROR
```

Summary

- Javascript is an OOP language
- With the class syntax it is now easier to declare classes

academeez

Thank You

Next Lesson: If