**What is OOP?**

**Now** that we know and understand the fundamentals of JavaScript, it is time to move on to the big guns! We are about to learn the magic of Object-Oriented Programming.

Object Oriented Programming is a programming style that is centered around using objects, rather than functions.

In OOP, you write your code in blocks called objects that make your life a lot easier. Once you make them connect and work together, you can make some amazing programs using objects.

With OOP, we can:

* Write cleaner code
* Make the code simpler
* Build reusable blocks of code
* Prepare for technical interviews
* Use the latest tools in JavaScript
* Be an amazing programmer!

In this course, we will start with learning basic information about OOP and the four pillars that make up its foundation. Before we go in-depth, we will first learn what objects are, how to make them, how to use them and how to make sure they are as good as possible. After that, we will learn more complex topics, including connections between objects, inheritance between objects, and lots of other fun stuff.

I hope you enjoy this challenging but very fun course! Take it at your own pace. Don't stress too much if you don't understand something. It took us all lots of time!

**The 4 pillars of OOP**

The four pillars are the foundation of OOP. They are:

* Abstraction
* Encapsulation
* Inheritance
* Polymorphism

Let's take a closer look at each of them.

**Abstraction**

To abstract something away means to hide away the implementation details inside something – sometimes a function.

Think of a laptop. It has all these crazy boards and technologies inside it, yet we can operate it with just the keys of a keyboard. That is what Abstraction is. Hiding complex implementations and only showing what is really useful. It basically means hiding some elements of an object that shouldn't be global, but showing the rest.

**Encapsulation**

Encapsulation means that each object in your code should control its own code. The keys, the methods on your object, properties, and so on. If you were to reset a Boolean or delete a key from the object, they're all changes. Encapsulation means combining a bunch of functions and variables into an object and then making that object control them and only them. Code from outside should not be able to mess it up.

**Inheritance**

Inheritance lets one object acquire the properties and methods of another object. So, we can have an object called Animal which has some properties. Then, we can have a Dog object and a Cat object which both inherit the same properties as Animal. But they still have their own properties and methods. This can be very useful.

**Polymorphism**

Polymorphism is connected to Inheritance. It basically means many forms. It says that objects that inherit from the same parent should be able to do different things, even with the same methods. So, if we have a Baby that inherits the method eat() from a Human object and an Adult object that inherits that same method from the Human object, they should do different things. The baby should drink milk and the adult should eat a burger.  It means overriding the methods you just inherited.

**Introduction**

We already talked a bit about objects. Let's define exactly what an object is.

As we said, in real life, anything can be an "object" with its own properties and methods. A car has the properties: model, make, color, miles, and more. It has the methods: start, break, and more.

Let's express these values as name:value pairs, separated by a comma. It would look like this:

model: Audi A6,  
color: Gray,  
miles: 10,780,  
start: () => startEngine(),  
break: () => hitBrakes()

This is a lot closer to an object in javascript.

Objects are variable containers that can contain many variables. An object is defined with curly braces {}. It resembles a comma-separated list of  name:value pairs. Here is an example:

{  
    name: "John Doe",  
    age: 41,  
    occupation: "Web Developer",  
    car: "Audi A6",  
    salary: 120000  
}

**Object Literals**

Let's start with the most basic form of objects.

An object literal is a comma-separated list of name-value pairs inside of curly braces. We can attach that object to a variable. That is an object literal. Inside of the object literal, you can have variables called properties and functions called methods. Here is an example:

const myObject = {  
    type: 'Addition',  
    addTwoNumbers: function(num1, num2) {  
        return num1 + num2;  
    }  
}

So, how do we access these values inside of the object?

If you remember, we have name:value pairs. We can use the name of each value to get the corresponding value.

The first method for that is using dot syntax (.). We can use the name of the variable + . + name of the value inside of the variable. This is what it looks like:

console.log(myObject.type);

Output: Addition.

We can also execute the methods using this syntax:

console.log(myObject.addTwoNumbers(5, 10));

Output: 15.

The second method is using brackets, just like an array. Except now, instead of using indexes, we use the name of the values. Like this:

console.log(myObject["type"]);

Output: Addition.

**The first method is the one that you will use the most. You should only use the second one when the name has special characters, or when you want to use a variable to access or set a value. For example:**

const key = "type";  
console.log(myObject[key]);  
// Addition

1. **Challenge**(Easy)

Write an object literal with the name myObject. Inside you should have the following pairs:

key name with value ObjectsAreCool!, key lesson with value learned, and a method called calculate that takes in two numbers, multiplies them and returns the answer.

**Object Factory**

Now, let's say we want to create multiple objects. Should we just write multiple object literals?

Actually, we can write a function to create objects for us! Think of how you could write this function before you read the rest of this lecture.

So, to create that function, we need to have a function that takes in the parameters we want, and then creates and returns an object literal.

Let's make a factory for people. Each person object should have a name, age and occupation. This is what that function would look like:

function createPerson(personName, personAge, personOccupation) {  
    return {  
        name: personName,  
        age: personAge,  
        occupation: personOccupation  
    }  
}

Now, we can use this function to create multiple objects:

const Jill = createPerson("Jill Hemings", 21, "Hair Stylist");  
const Jack = createPerson("Jack Smith", 52, "Engineer");  
const Jonah = createPerson("Jonah Hill", 25, "Singer");  
  
console.log(Jill.name);  
console.log(Jack.age);  
console.log(Jonah.occupation);

Output: Jill Hemings, 52, Singer.

**Bonus Tip:** We can put object literals inside object literals:

const person = {  
    name: {  
        first: "John",  
        last: "Doe"  
    },  
    age: 42  
}  
  
console.log(person.name.first);  
console.log(person.age);

Output: John, 42.

1. **Challenge**(Easy)

Write a function that creates objects. It should take in three parameters. The first one should have the key name, the second one age and the third one should be numberOfToes. Return the object.