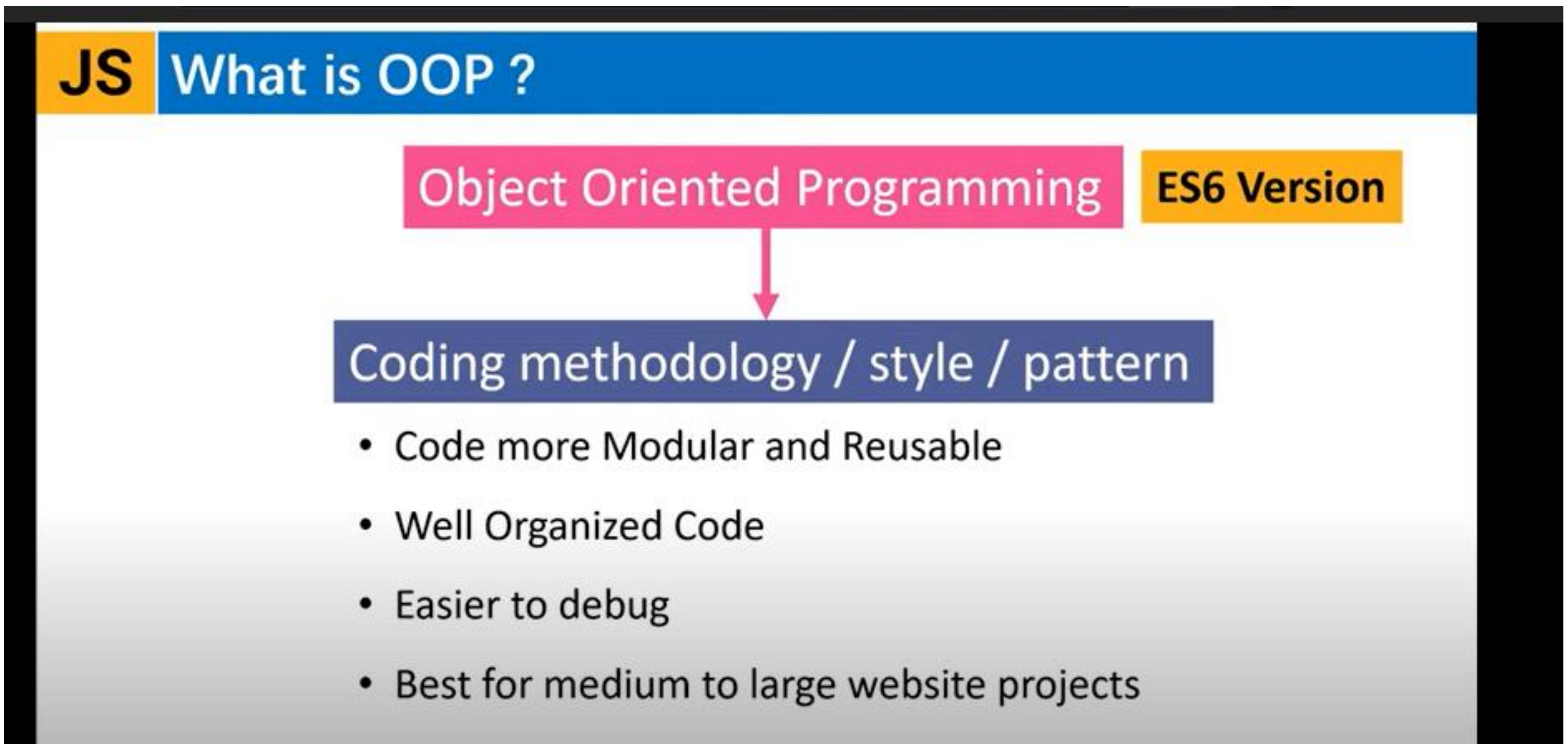


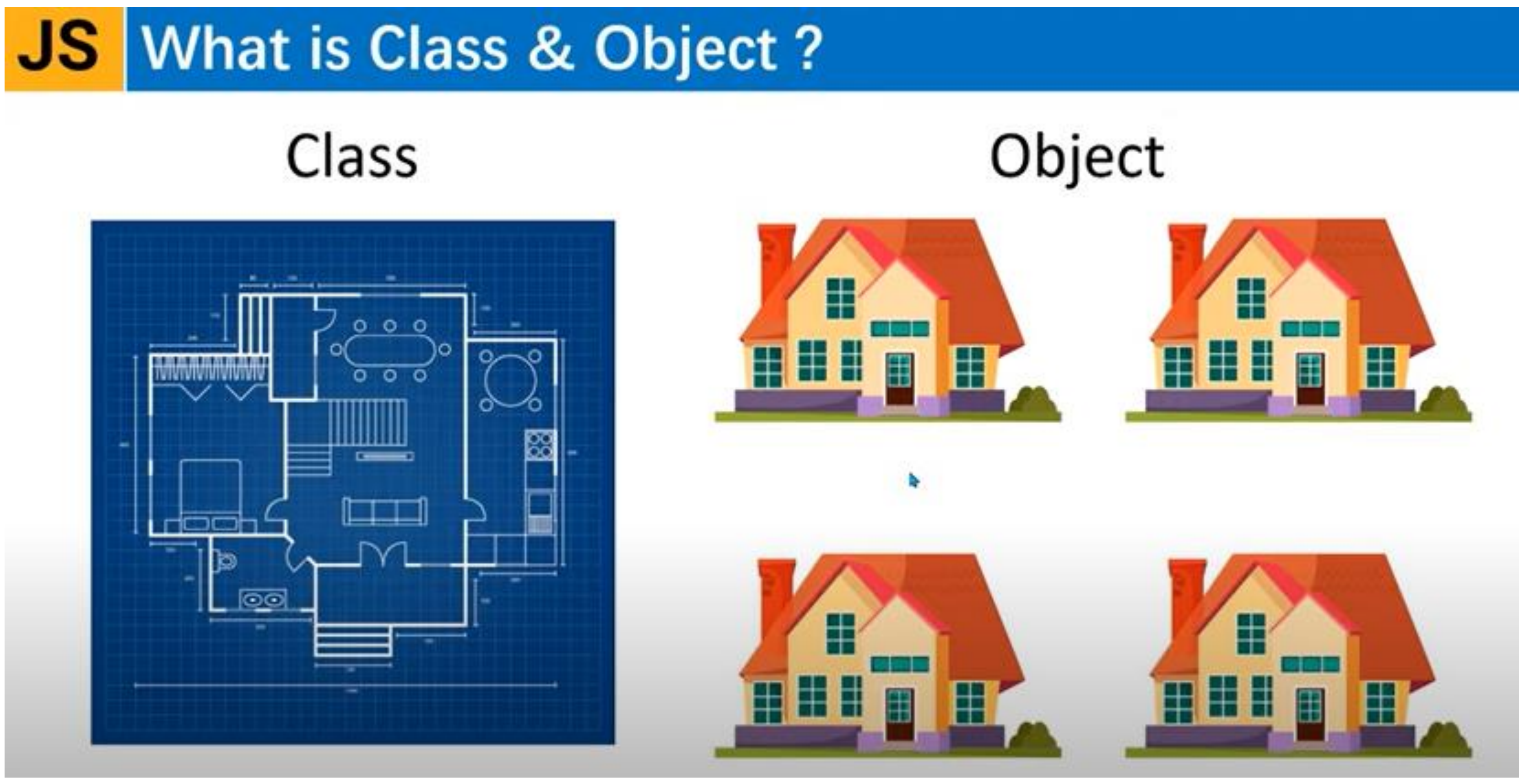
JS What is OOP ?

Object Oriented Programming

ES6 Version

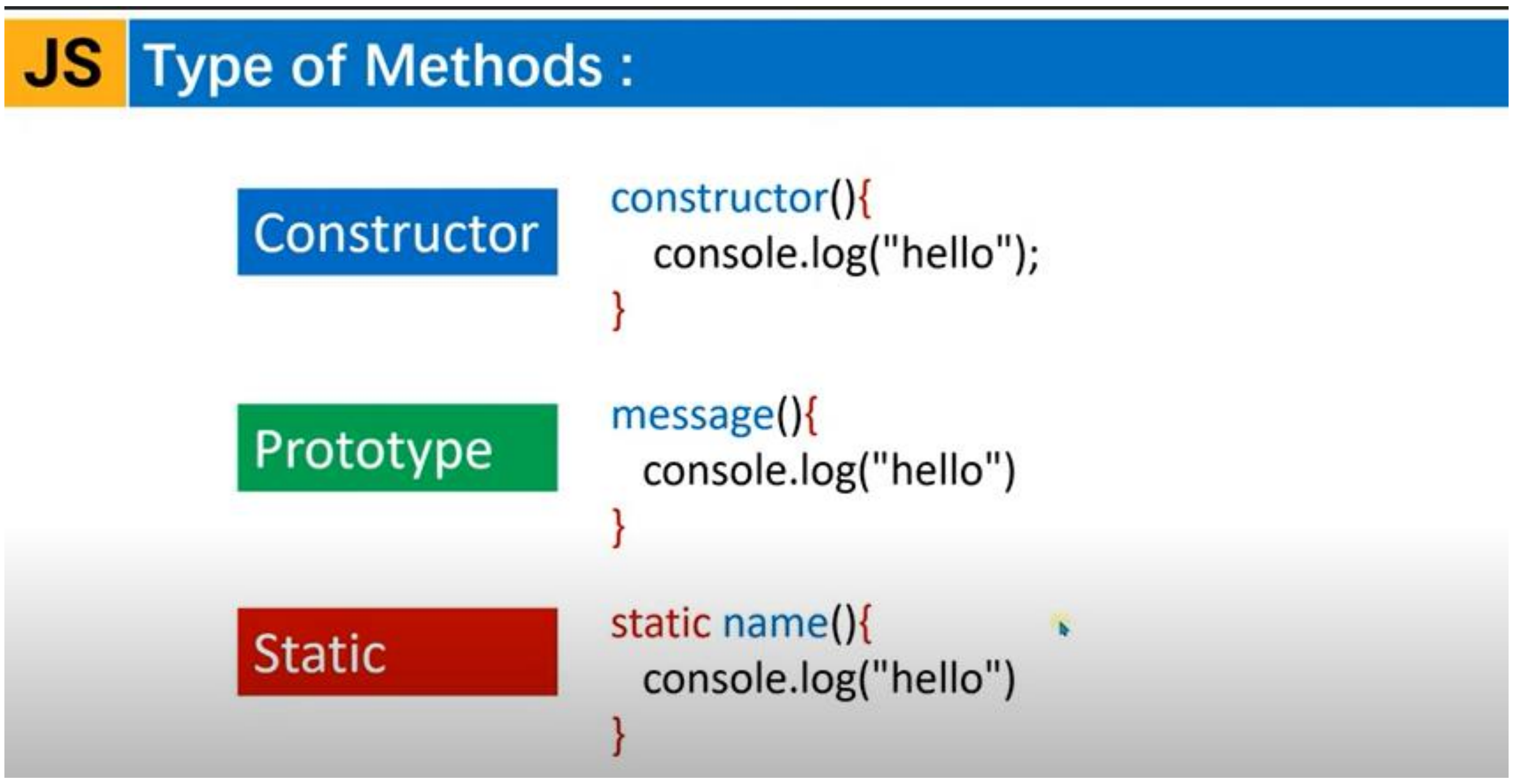
Coding methodology / style / pattern

- Code more Modular and Reusable
- Well Organized Code
- Easier to debug
- Best for medium to large website projects



JS What is Class & Object ?

Fig: Blue print class jake use kore object toiri hosse.



JS Type of Methods :

Constructor

```
constructor(){
  console.log("hello");
}
```

Prototype

```
message(){
  console.log("hello")
}
```

Static

```
static name(){
  console.log("hello")
}
```

- Constructor function aka e call hoi take call korte hoi na.

oop is not a programming language or tool. Its a style of programming or a programming paradigm.

onek programming language ase je gulo object-oriented-programming support kore. Jemon c#, java, ruby, python, javascript.

In oop programming we group related variables and functions that operate on them into object and this is what we call encapsulation.

kono object er jodi method thake ta hole bolte pari object er behavior ase ekhon manush er moto.

*Four core concept in javascript

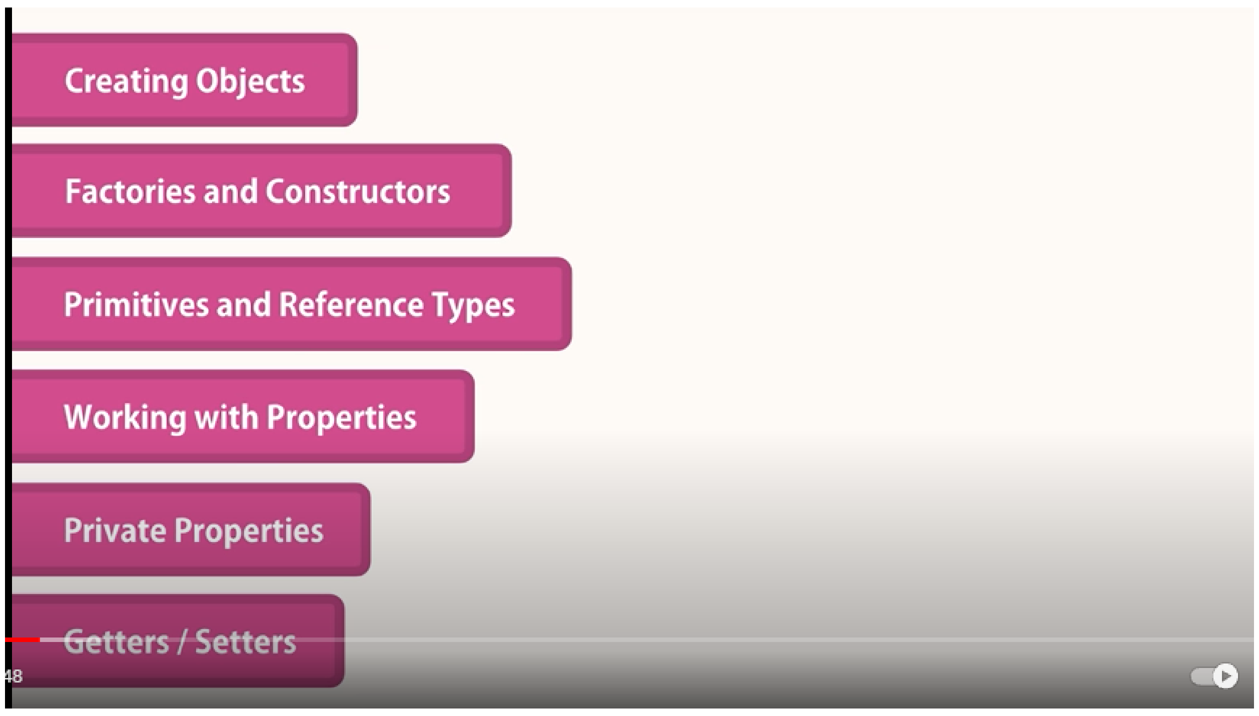
Encapsulation

Abstraction

Inheritance

Polymorphism = many+form

Fundamental concepts of oop:


Creating Objects

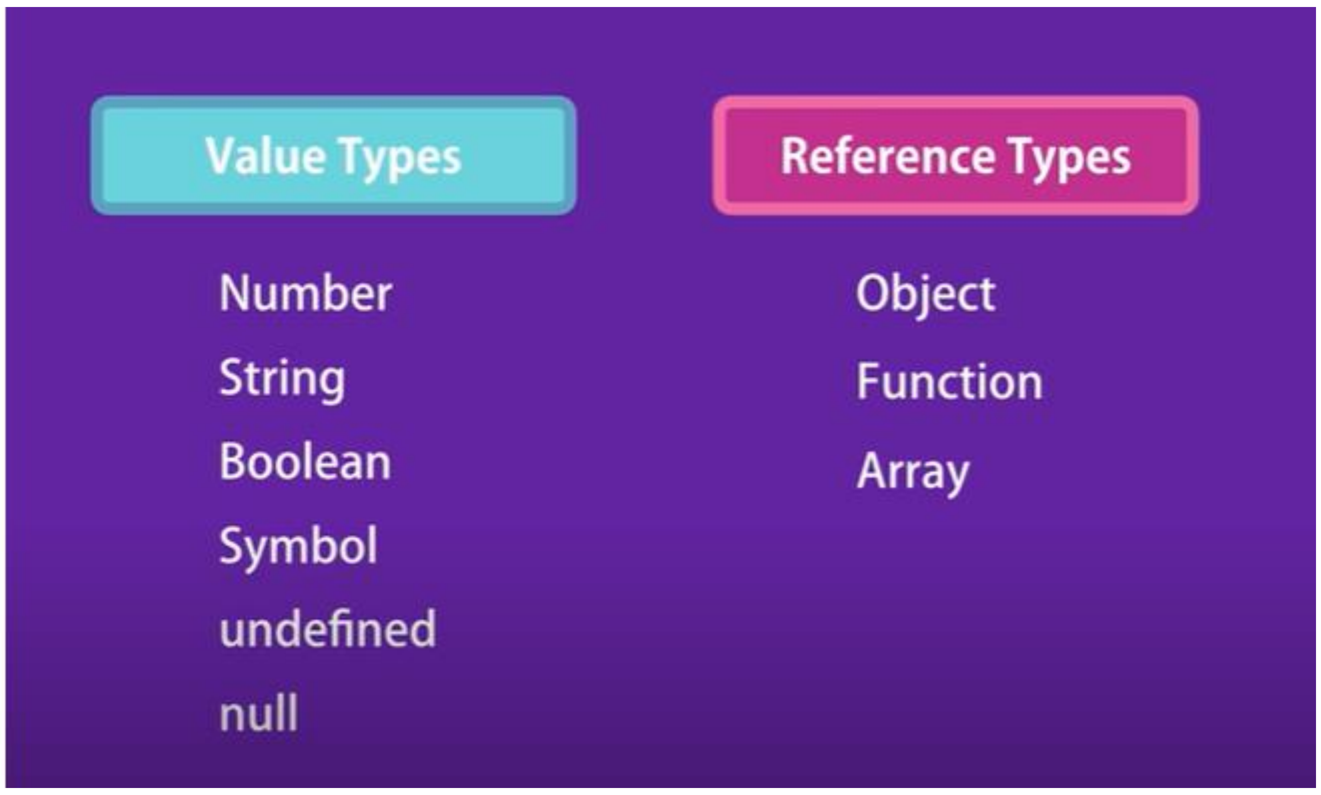
Factories and Constructors

Primitives and Reference Types

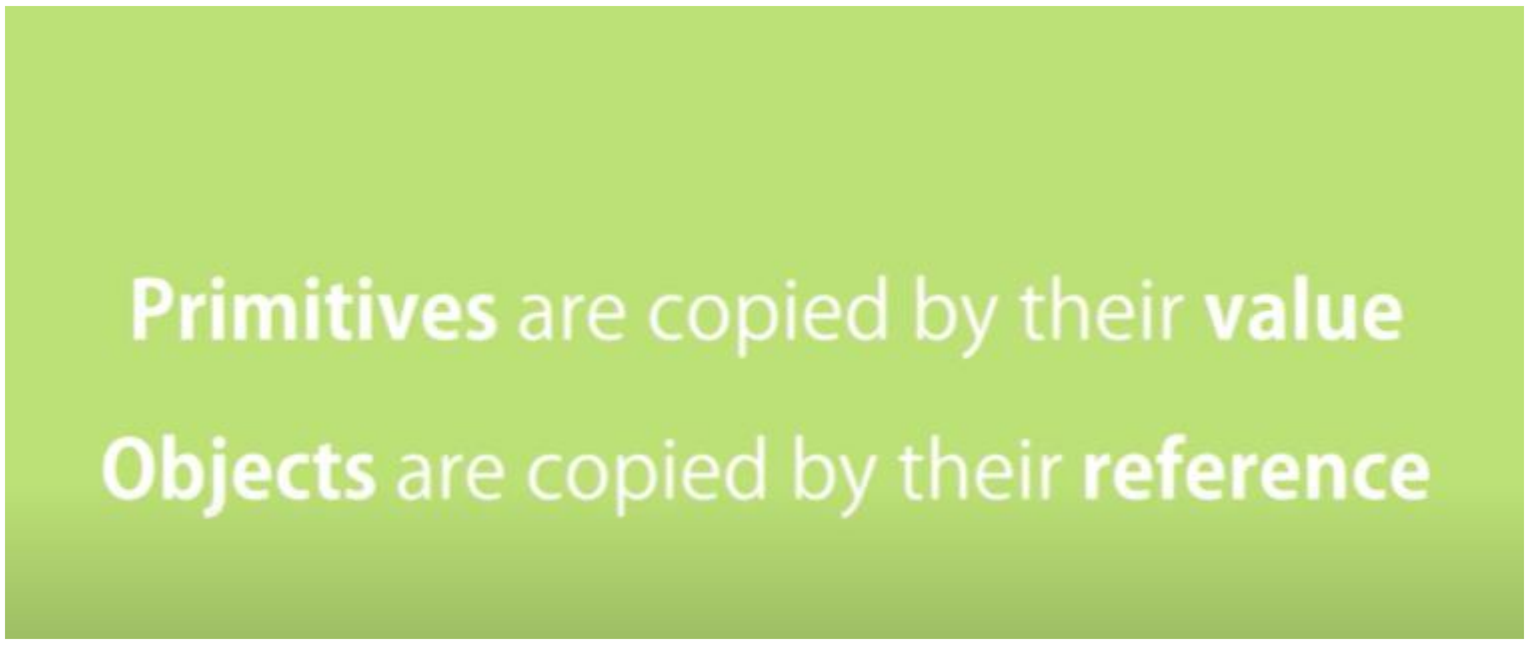
Working with Properties

Private Properties

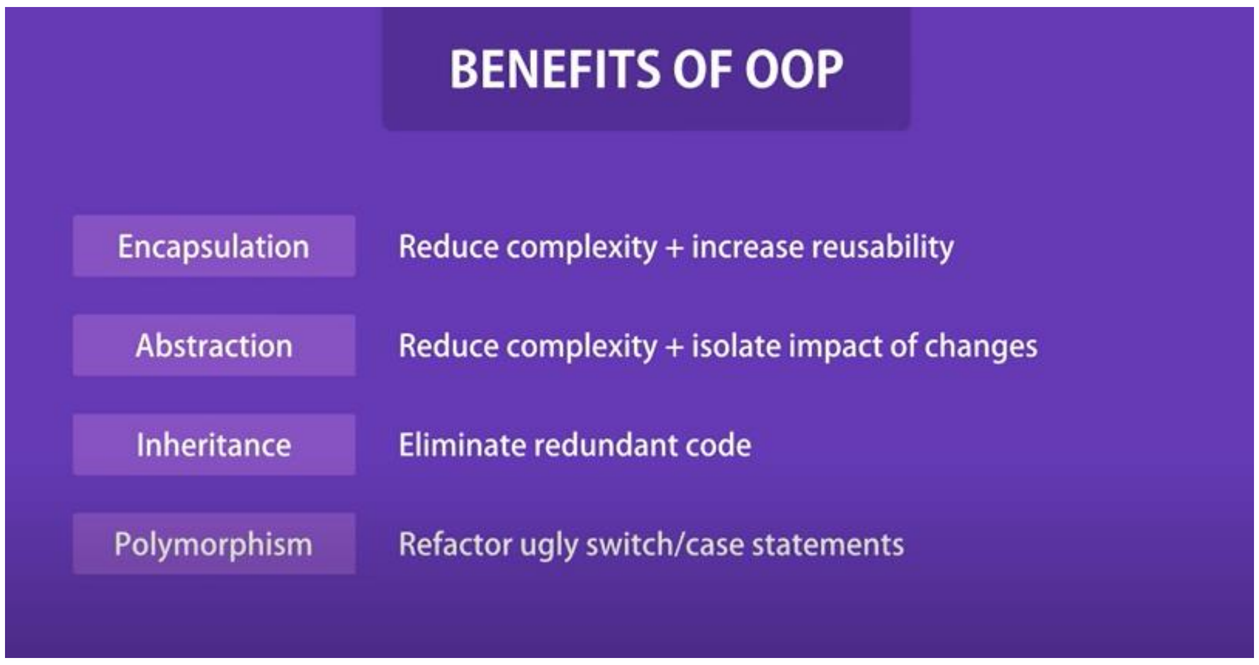
Getters / Setters


Value Types	Reference Types
Number	Object
String	Function
Boolean	Array
Symbol	
undefined	
null	

Premitive VS Reference type


Primitives are copied by their **value**

Objects are copied by their **reference**


BENEFITS OF OOP

Encapsulation	Reduce complexity + increase reusability
Abstraction	Reduce complexity + isolate impact of changes
Inheritance	Eliminate redundant code
Polymorphism	Refactor ugly switch/case statements

Encapsulation: **JavaScript Encapsulation**

The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data. It allows us to control the data and validate it. To achieve an encapsulation in JavaScript: -

- Use var keyword to make data members private.
- Use setter methods to set the data and getter methods to get that data.

The encapsulation allows us to handle an object using the following properties:

Read/Write - Here, we use setter methods to write the data and getter methods read that data.

Read Only - In this case, we use getter methods only.

Write Only - In this case, we use setter methods only.

JavaScript Encapsulation Example

Let's see a simple example of encapsulation that contains two data members with its setter and getter methods.

```
<script>
class Student
  {
    constructor()
    {
       var name;
       var marks;
    }
        getName()
        {
          return this.name;
        }
      setName(name)
      {
        this.name=name;
      }

      getMarks()
      {
        return this.marks;
      }
    setMarks(marks)
    {
      this.marks=marks;
    }

  }
  var stud=new Student();
   stud.setName("John");
   stud.setMarks(80);
   document.writeln(stud.getName()+" "+stud.getMarks());
</script>
```

Output:

```
John 80
```

Abstruction: Handle complexity & Hide unnecessary details from user. Object er bitor je internal implementation details hide kore rakhe. It reduces the duplication of code.

Example

Let's see an example to achieve abstraction.

```
<script>
//Creating a constructor function
 function Vehicle()
{
    this.vehicleName="vehicleName";
    throw new Error("You cannot create an instance of Abstract Class");
}
Vehicle.prototype.display=function()
{
    return "Vehicle is: "+this.vehicleName;
}
//Creating a constructor function
function Bike(vehicleName)
{
    this.vehicleName=vehicleName;
}
//Creating object without using the function constructor
Bike.prototype=Object.create(Vehicle.prototype);
var bike=new Bike("Honda");
document.writeln(bike.display());


 </script>
```

Output:

```
Vehicle is: Honda
```

Polymorphism: It's means normally... Something occurs in several different forms. It provides an ability to call the same method on different JavaScript objects.

Let's see an example where a child class object invokes the parent class method.

Example:

```
<script>
class A
 {
   display()
  {
    document.writeln("A is invoked");
  }
 }
class B extends A
 {
 }
var b=new B();
b.display();
</script>
```

Output:

```
A is invoked
```

Inheritance in JavaScript

Inheritance is an important concept in object oriented programming. In the classical inheritance, methods from base class get copied into derived class.

Let's see how we can achieve inheritance like functionality in JavaScript using prototype object.

Let's start with the Person class which includes FirstName & LastName property as shown below.

```
function Person(firstName, lastName) {
    this.FirstName = firstName || "unknown";
    this.LastName = lastName || "unknown";
};

Person.prototype.getFullName = function () {
    return this.FirstName + " " + this.LastName;
}
```

In the above example, we have defined Person class (function) with FirstName & LastName properties and also added getFullName method to its prototype object.

Code:

```
//object literal syntex
// object literal syntex is simple way to define a an objevt.
const circle = {
    radiou: 1,
    location: {
        x: 1,
        y: 2
    },
    draw: function () {
```

```javascript
        console.log('draw');
    }
};
circle.draw();

//ai uporer kode tuko abr use korle duplicate korte hobe. And er pore jodi error hoi fix korte kothin multiple jaiga thaka fix korte hobe So ........object literal syntex is not good awy to create an object. Er solution holo factory or constructor function use kora


//factory function (uporer code tuko e)
function createCircle(radius) {
    return {
        radious,
        draw: function () {
            console.log('draw');
        }
   }
}
const circle = createCircle(1);
circle.draw;


//construction function
function Circle(radius) {
    this.radius = radios;
    this.drow = function () {
        console.log('draw');
    }
}
const another = new Circle(1);


let x = {};
//behind the sceen JS translate it let x = new Object();

//In javascript functions are object




//value/premitive VS Reference Types

//premitive example
let number = 10;

function increase(number) {
    number++;
}
increase(number);
console.log(number);
```

```
//Output: 10


//reference example(reference types are object)
let obj = { value: 10 };

function increase(obj) {
    obj.value++;
}
increase(obj);
console.log(obj);
//Output: {value: 11}





function Circle(radius) {
  this.radius = radios;
  this.drow = function () {
    console.log("draw");
  };
}
const another = new Circle(1);

circle.location = { x: 1 };
circle["location"] = { x: 1 };
//both same

//Thard bracket use er karon koro...
const propertyName = 'Centeer Location'; // aita ami circle.propertyName ai vabe
pabo na. tai...
circle[propertyName] = { x: 1 };



//Type of method:
class student{
    constructor(name) {
      //constructor method
        this.fName = name;
      console.log("construction Function");
    }
    hello() {
      //eta ekta prototype method
      console.log(`Hello ${this.fName}`);
    }
}
let a = new student('My baba');
a.hello();
//Output: construction Function
//Output: Hello My baba
```