

# Object Oriented Programming in JavaScript and Error Handling





# Kind of Programming paradigms

- **imperative:** in which the programmer instructs the machine how to change its state,
  - **procedural** which groups instructions into procedures,
  - **object-oriented** which groups instructions together with the part of the state they operate on,
- **declarative** in which the programmer merely declares properties of the desired result, but not how to compute it
  - **functional** in which the desired result is declared as the value of a series of function applications,
  - **logic** in which the desired result is declared as the answer to a question about a system of facts and rules,
  - **mathematical** in which the desired result is declared as the solution of an optimization problem



# **OOP in Javascript**

- Class Base
- Prototype Base



# Class Base

*In object-oriented programming, a class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods).*

Syntax:

```
class MyClass {  
    // class methods  
    constructor(param1, param2) { ... }  
    method1() { ... }  
    method2() { ... }  
    method3() { ... }  
    ...  
}  
  
Class childClass extends myClass{  
    constructor() {  
        super(param1, param2)  
    }  
    childMethod() {  
    }  
}
```



# Prototype

A prototype is a kind of object which functions always have. This is saved inside the hidden prototype property of a function

```
/* Prototype class base */
function Animal(className) {
    this.className = className
}

/* add method to Animal Prototype */
Animal.prototype.getClass = function () {
    return `Animal class is : ${this.className}`;
}

/* create class Dog with extends Animal */
function Dog(name) {
    /* execute parent constructor */
    Animal.call(this, 'Animal'); // pass this
    this.name = name // add new property
}

/* 1. extends property prototype */
Dog.prototype = Object.create(Animal.prototype)
/* 2. override constructor with Dog */
Dog.prototype.constructor = Dog;

/* 3. add new method */
Dog.prototype.getName = function () {
    return this.name;
}

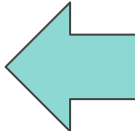
var d = new Dog("Tommy")
console.log(d)
console.log(d.getClass);
```



# Error Handling in Javascript

- Error Categories :
  - Syntact eror
  - Runtime Errors / Exception : the program runs but behaves not as expected or gives wrong results
  - Logical Errors : when you make a mistake or flaw in the logic that controls your program's script,when you make a mistake or flaw in the logic that controls your program's script

```
var myFunc = function(cb) {  
  doSomething(function (err, a) {  
    if (err) return cb(err)  
    doSomethingElse(function (err, b)  
  {  
    if (err) return cb(err)  
    return cb(null, [a, b])  
  })  
})  
}
```



What's bad  
about this  
kind of  
pattern ?



# The try...catch...finally Statement

- It's possible to catch logical and runtime errors but not syntax errors

```
try {  
    // Code to run  
    break;  
} catch (e) {  
    // Code to run if an exception occurs  
    break;  
}  
  
finally {  
    // Code that is always  
    // executed regardless of  
    // an exception occurring  
}
```

```
function myFunction() {  
    var passenger = 50;  
    try {  
        alert("The coach can sit  
" + passenger + " people.");  
    } catch (e) {  
        alert("Error: " +  
e.description);  
    }  
}
```



# The Throw Statement

- The `throw` statement is used to generate user-defined exceptions.

```
function myFunction()  
{  
    var winPoints = 3;  
    var drawPoints = 1;  
  
    try{  
        if ( drawPoints == 1 ){  
            throw( "Real Madrid will not qualify." );  
        }  
        else  
        {  
            var qualification = winPoints + drawPoints;  
        }  
    }  
  
    catch ( e ) {  
        alert("Error: " + e );  
    }  
}
```





# Handling Errors in Asynchronous Code

- The `throw` statement is used to generate user-defined exceptions.

```
function myFunction()  
{  
    var winPoints = 3;  
    var drawPoints = 1;  
  
    try{  
        if ( drawPoints == 1 ){  
            throw( "Real Madrid will not qualify." );  
        }  
        else  
        {  
            var qualification = winPoints + drawPoints;  
        }  
    }  
  
    catch ( e ) {  
        alert("Error: " + e );  
    }  
}
```