

JAVASCRIPT PATTERN

ECMA SCRIPT 5

THE IMMEDIATE INVOKED FUNCTION EXPRESSION

THE IMMEDIATE INVOKED FUNCTION EXPRESSION

```
(function () {  
    'use strict';  
    // - - - - -  
    var onLoad = null;  
  
    onLoad = function () {  
        console.log('window loaded!');  
    };  
  
    window.addEventListener('load', onLoad)  
    // - - - - -  
})();
```

ECMA SCRIPT 5

THE MODULE BLOCK PATTERN

THE MODULE BLOCK PATTERN

```
(function () {  
    'use strict';  
    // - - - - -  
    var fn = {},  
        a = 42;  
  
    fn.log = function (message) {  
        console.log(message);  
    };  
  
    fn.alert = function () {};  
  
    window.myNamespace = window.myNamespace || {};  
    window.myNamespace.myModuleName = fn;  
    // - - - - -  
})();
```

INHERITANCE

```
(function () {  
    // - - - - -  
    var fn = {};  
  
    fn.log = function (message) {  
        console.log(message);  
    };  
  
    console.log('Tut was in Module 2!');  
  
    window.app = window.app || {}; // Defaultoperator  
    window.app.myModule2 = fn;  
  
    window.app.myModule1.log('Aufruf an 1 aus 2');  
  
    // - - - - -  
})();
```

ECMA SCRIPT 5

CONSTRUCTOR PATTERN

OBJEKT ALS KONSTRUKTOR

```
function Person(firstName, lastName) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
  
  this.fullName = function() {  
    return this.firstName + ' ' + this.lastName;  
  };  
}  
  
// Use it like this:  
var john = new Person('John', 'Doe');  
john.firstName;    // "John"  
john.fullName();   // "John Doe"  
  
john.firstName = 'John';  
john.fullName();   // John Doe  
  
// adding methods  
john.prototype.doSomething = function () {}
```


KONSTRUKTOR OHNE THIS

```
function Person(firstName, lastName) {  
    var _firstName = firstName,  
        _lastName = lastName;  
  
    var my = {  
        firstName: _firstName,  
        lastName: _lastName  
    };  
  
    my.fullName = function() {  
        return _firstName + ' ' + _lastName;  
    };  
}
```

KONSTRUKTOR OHNE THIS

```
// Getter/setters

my.firstName = function(value) {
  if (!arguments.length) return _firstName;
  _firstName = value;

  return my;
};

my.lastName = function(value) {
  if (!arguments.length) return _lastName;
  _lastName = value;

  return my;
};

return my;
}
```

KONSTRUKTOR OHNE THIS

```
// Use it like this:  
  
var chuck = Person('Chuck', 'Norris');  
  
chuck.firstName('Jackie');  
chuck.lastName('Chan');  
  
chuck.fullName(); // Jackie Chan
```

„<http://www.samselickoff.com/blog/some-javascript-constructor-patterns/>“

ECMA SCRIPT 5

INHERITANCE PATTERN

PROTOTYPES IN KONSTRUKTOREN

```
var Shipment = function (state, type) {  
  var  
    _state = undefined,  
    _type  = undefined,  
  endvar;  
  
  this.state = state || undefined;  
  this.type  = type  || undefined;  
  
  this.setState = function () {};  
};
```

PROTOTYPES IN KONSTRUKTOREN

```
// Nachträglich hinzugefügte Methoden
Shipment.prototype.setState = function (value) {
  this.state = value;
};

Shipment.prototype.setType = function (value) {
  this.type = value;
};

Shipment.prototype.save =
  function () {
    console.log( 'saving shipment '
      + this.state + ', '
      + this.type + '.' );
  };
};
```

```
var shipment = [];  
  
shipment[0] = new Shipment(3, 1);  
shipment[1] = new Shipment();  
  
shipment[0].save();  
  
shipment[1].setState(4);  
shipment[1].setType(2);  
  
shipment[1].save();
```


INHERITANCE

```
var ShipmentRainbow =  
function (state, type, typeOfDoc, reference){  
    this.state      = state || undefined;  
    this.type       = type  || undefined;  
    this.typeOfDoc  = typeOfDoc || 'HTML';  
    this.reference  = reference || 'Hallo';  
};  
  
ShipmentRainbow.prototype    = new Shipment();  
ShipmentRainbow.constructor = ShipmentRainbow;
```

INHERITANCE

```
var shipmentRainbow = new  
ShipmentRainbow(3,1,'typeOfDocText','referenceText');  
  
shipmentRainbow.save();  
  
console.dir(shipmentRainbow);  
  
console.log( shipmentRainbow instanceof Shipment );  
console.log( shipmentRainbow instanceof ShipmentRainbow );
```

ECMA SCRIPT 5

CHAINING PATTERN

CHAINING PATTERN

```
(function() {  
  'use strict';  
  // - - - - -  
  
  // define the class  
  let _fn = function() {  
    this.version = '0.1';  
    this.os = 'MacOS X';  
    this.browser = 'Chrome';  
  };  
};
```

CHAINING PATTERN

```
_fn.prototype.setVersion = function(version) {  
    this.version = version;  
    return this;  
};  
  
_fn.prototype.setOs = function(os) {  
    this.os = os;  
    return this;  
};  
  
_fn.prototype.setBrowser = function(browser) {  
    this.browser = browser;  
    return this;  
};  
  
_fn.prototype.save = function() {  
    console.log(  
        'saving ' + this.version + ', on ' +  
        this.os + ' with ' + this.browser + '.'  
    );  
    return this;  
};
```

CHAINING PATTERN

```
Let fn = new _fn();  
  
fn.setVersion('1');  
fn.setOs('MacOS XI');  
fn.setBrowser('Chrome Xtra');  
fn.save();  
  
fn  
  .setVersion('2')  
  .setOs('MacOS XII')  
  .setBrowser('Chrome Xtra Large')  
  .save();
```

ECMA SCRIPT 5

SINGLETON

SINGLETON

```
var mySingleton = (function () {

    // Instance stores a reference to the Singleton
    var instance;

    function init() {

        // Singleton
        // Private methods and variables
        function privateMethod(){
            console.log( "I am private" );
        }
        var privateVariable = "Im also private";
        var privateRandomNumber = Math.random();

        return {
            // Public methods and variables
            publicMethod: function () {
                console.log( "The public can see me!" );
            },
            publicProperty: "I am also public",

            getRandomNumber: function() {
                return privateRandomNumber;
            }
        };
    };

};
```


SINGLETON

```
return {  
  
    // Get the Singleton instance if one exists  
    // or create one if it doesn't  
    getInstance: function () {  
  
        if ( !instance ) {  
            instance = init();  
        }  
  
        return instance;  
    }  
  
};  
  
})();
```

„ <https://addyosmani.com/resources/essentialjsdesignpatterns/book/#constructorpatternjavascript>“

ECMA SCRIPT 5

FACTORY PATTERN

FACTORY PATTERN

- Factories erzeugen Objekte aus Klassen.
- Erst bei der Erzeugung wird die Klasse ausgewählt.

FACTORY PATTERN

```
var creator = {}; jsp.dom = {};  
  
creator.dom.Text = function () {  
  this.insert = function (where) {  
    var txt = document.createTextNode(this.url);  
    where.appendChild(txt);  
  };  
};  
creator.dom.Link = function () {  
  this.insert = function (where) {  
    var link = document.createElement('a');  
    link.href = this.url;  
    link.appendChild(document.createTextNode(this.url));  
    where.appendChild(link);  
  };  
};  
creator.dom.Image = function () {  
  this.insert = function (where) {  
    var im = document.createElement('img');  
    im.src = this.url;  
    where.appendChild(im);  
  };  
};
```

FACTORY PATTERN

```
// the factory method:  
creator.dom.factory = function (type) {  
    return new creator.dom[type];  
}
```

```
var o = creator.dom.factory('Link');  
o.url = 'http://google.com';  
o.insert(document.body);
```

„[https://github.com/shichuan/javascript-patterns/
blob/master/design-patterns/factory.html](https://github.com/shichuan/javascript-patterns/blob/master/design-patterns/factory.html)“

ECMA SCRIPT 6

KLASSEN

CLASS

```
class SimpleDate {  
    constructor(year, month, day) {  
        // Check that (year, month, day) is a valid date  
        // ...  
  
        this._year = year;  
        this._month = month;  
        this._day = day;  
    }  
  
    addDays(nDays) {  
        // Increase "this" date by n days  
        // ...  
    }  
  
    getDay() {  
        return this._day;  
    }  
}
```

INSTANTIIERUNG

```
let today = new SimpleDate(2000, 2, 28);  
today.addDays(1);
```

PRIVATE PROPERTIES

```
class SimpleDate {  
  constructor(year, month, day) {  
    let _year = year;  
    let _month = month;  
    let _day = day;  
  }  
  
  this.getDay = function(){  
    return _day;  
  }  
}
```

STATIC PROPERTIES AND METHODS

```
class SimpleDate {
  static setDefaultDate(year, month, day) {
    SimpleDate._defaultDate = new SimpleDate(year, month, day);
  }

  constructor(year, month, day) {
    if (arguments.length === 0) {
      this._year = SimpleDate._defaultDate._year;
      this._month = SimpleDate._defaultDate._month;
      this._day = SimpleDate._defaultDate._day;

      return;
    }

    // Check that (year, month, day) is a valid date
    // ...

    this._year = year;
    this._month = month;
    this._day = day;
  }

  addDays(nDays) {
    // Increase "this" date by n days
    // ...
  }

  getDay() {
    return this._day;
  }
}
```

```
SimpleDate.setDefaultDate(1970, 1, 1);
let defaultDate = new SimpleDate();
```

INHERITANCE

```
class Employee {
  constructor(firstName, familyName) {
    this._firstName = firstName;
    this._familyName = familyName;
  }

  getFullName() {
    return `${this._firstName} ${this._familyName}`;
  }
}

class Manager {
  constructor(firstName, familyName) {
    this._firstName = firstName;
    this._familyName = familyName;
    this._managedEmployees = [];
  }

  getFullName() {
    return `${this._firstName} ${this._familyName}`;
  }

  addEmployee(employee) {
    this._managedEmployees.push(employee);
  }
}
```

INHERITANCE

```
class Employee {  
    constructor(firstName, familyName) {  
        this._firstName = firstName;  
        this._familyName = familyName;  
    }  
  
    getFullName() {  
        return `${this._firstName} ${this._familyName}`;  
    }  
}
```

INHERITANCE

```
class Manager extends Employee {  
    constructor(firstName, familyName) {  
        super(firstName, familyName);  
        this._managedEmployees = [];  
    }  
  
    addEmployee(employee) {  
        this._managedEmployees.push(employee);  
    }  
}
```

„ <https://www.sitepoint.com/object-oriented-javascript-deep-dive-es6-classes/>“

ECMA SCRIPT 6

PROMISES

PROMISES

- Promises werden zur Steuerung von asynchronen Abläufen verwendet, zum Beispiel `setTimeout()` oder `XMLHttpRequest()`.
- `New Promise` erzeugt eine `resolve()` und eine `reject()` Callback-Funktion zur Steuerung des asynchronen Ergebnisses.

PROMISES

```
var p = new Promise(function(resolve, reject) {  
    // Do an async task async task and then...  
  
    if(/* good condition */) {  
        resolve('Success!');  
    }  
    else {  
        reject('Failure!');  
    }  
});  
  
p.then(function() {  
    /* do something with the result */  
})  
.catch(function() {  
    /* error */  
})
```

PROMISES

- Wenn der async Task nicht innerhalb der Promise beendet werden soll,
- Sometimes you don't need to complete an async tasks within the promise -- if it's possible that an async action will be taken, however, returning a promise will be best so that you can always count on a promise coming out of a given function. In that case you can simply call `Promise.resolve()` or `Promise.reject()` without using the new keyword. For example:

BASIC PROMISES

```
var userCache = {};  
  
function getUserData(username) {  
    if (userCache[username]) {  
        // Return a promise without the "new" keyword  
        return Promise.resolve(userCache[username]);  
    }  
  
    // Use the fetch API to get the information  
    // fetch returns a promise  
    return fetch('users/' + username + '.json')  
        .then(function(result) {  
            userCache[username] = result;  
            return result;  
        })  
        .catch(function() {  
            throw new Error('Could not find user: ' + username);  
        });  
}
```

THEN() VERARBEITET DAS POSITIVE ERGEBNIS

```
new Promise(function(resolve, reject) {  
    // A mock async action using setTimeout  
    setTimeout(function() { resolve(10); }, 3000);  
})  
.then(function(result) {  
    console.log(result);  
});  
  
// 10
```

THEN() IST KASKADIERBAR

```
new Promise(function(resolve, reject) {  
    // A mock async action using setTimeout  
    setTimeout(function() { resolve(10); }, 3000);  
})  
  .then(function(num) {  
    console.log('first then: ', num);  
    return num * 2;  
  })  
  .then(function(num) {  
    console.log('second then: ', num);  
    return num * 2;  
  })  
  .then(function(num) {  
    console.log('last then: ', num);  
  });  
  
// From the console:  
// first then:    10  
// second then:   20  
// last then:     40
```

CATCH() FÄNGT NICHT ERFÜLLTE PROMISES AB

```
new Promise(function(resolve, reject) {  
  // A mock async action using setTimeout  
  setTimeout(function() {'Error!';}, 3000);})  
  .then(function(e) { console.log('done', e); })  
  .catch(function(e) { console.log('catch: ', e); });  
  
// From the console:  
// 'catch: Error!'  
  
// better sending an error  
reject(Error('Data could not be found'));
```


PROMISE.ALL

- Bei mehreren asynchronen Aktionen kann eine Promise auf alle Tasks eingerichtet werden.
- Nur wenn alle Tasks abgearbeitet sind, gilt eine Promise als eingelöst.
- **Promise.all** stellt ein Promise-Array zusammen und startet einen Callback, wenn alle Promises eingelöst sind.

PROMISE.ALL()

```
Promise.all([promise1, promise2]).then(function(results) {  
    // Both promises resolved  
})  
.catch(function(error) {  
    // One or more promises was rejected  
});
```

PROMISE.ALL()

```
var request1 = fetch('/users.json');  
var request2 = fetch('/articles.json');  
  
Promise.all([request1, request2])  
  .then(function(results) {  
    // Both promises done!  
  });
```

REJECT BEI PROMISE.ALL()

```
var req1 = new Promise(function(resolve, reject) {  
    // A mock async action using setTimeout  
    setTimeout(function() { resolve('First!'); }, 4000);  
});  
var req2 = new Promise(function(resolve, reject) {  
    // A mock async action using setTimeout  
    setTimeout(function() { reject('Second!'); }, 3000);  
});  
Promise.all([req1, req2]).then(function(results) {  
    console.log('Then: ', results);  
}).catch(function(err) {  
    console.log('Catch: ', err);  
});  
  
// From the console:  
// Catch: Second!
```

PROMISE.RACE

- **`Promise.race()`** startet einen Callback sobald irgendeine Promise aus einer Reihe erfüllt oder abgelehnt wird.

PROMISE.RACE

```
var req1 = new Promise(function(resolve, reject) {  
  // A mock async action using setTimeout  
  setTimeout(function() { resolve('First!'); }, 8000);  
});  
var req2 = new Promise(function(resolve, reject) {  
  // A mock async action using setTimeout  
  setTimeout(function() { resolve('Second!'); }, 3000);  
});  
Promise.race([req1, req2]).then(function(one) {  
  console.log('Then: ', one);  
}).catch(function(one, two) {  
  console.log('Catch: ', one);  
});  
  
// From the console:  
// Then: Second!
```

API'S MIT PROMISES

API'S, DIE MIT PROMISES ARBEITEN

```
fetch(  
  'https://davidwalsh.name/some/url',  
  { method: 'get' }  
)  
  .then(function(response) { /* Success! */ })  
  .catch(function(err) { /* Error! */ });  
  
navigator.getBattery().then(function(result) {});
```


„<https://davidwalsh.name/promises>“

„<https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-promise-27fc71e77261>“