



# **JavaScript 2**

## **Lektion => 9**

Utbildare: Mahmud Al Hakim

**NACKADEMIN**

# Lektionstillfällets mål

## Mål med lektionen

- JavaScripts objekt
- Prototyper och prototypkedja
- Klasser
- **Bra att läsa**  
<https://javascript.info/classes>
- **Arbetsmetod**
  - Teori och praktik varvas under lektionen

# Kort summering av föregående lektion

- Vi har gått igenom
  - Exekveringskontext
  - Scope
  - Closure

# What is JavaScript?

- JavaScript® (often shortened to JS) is a lightweight, interpreted, **object-oriented language** with first-class functions, and is best known as the scripting language for Web pages, but it's used in many non-browser environments as well.
- It is a **prototype-based**, multi-paradigm scripting language that is dynamic, and supports object-oriented, imperative, and functional programming styles.
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/About\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript)

# Functional class pattern

```
function User(name) {  
    this.sayHi = function() {  
        return 'Hello ' + name;  
    }  
}
```

# Privata egenskaper

- `name` är en privat egenskap.
- Du har tillgång till privata egenskaper inne i en konstruktor.

```
let user = new User("Mahmud");
console.log(user.sayHi()); // Hello Mahmud
console.log(user.name);    // undefined
```

# Publika egenskaper

```
let user = new User("Mahmud");
user.name = "Kalle";
console.log(user.name);    // Kalle
console.log(user.sayHi()); // Mahmud
```

Varför?

Kolla i Chrome DevTools

# Privata egenskaper lagras i en closure

```
↳ ▼User {sayHi: f, name: "Kalle"} ⓘ  
  name: "Kalle"  
  ▼sayHi: f ()  
    arguments: null  
    caller: null  
    length: 0  
    name: ""  
    ►prototype: {constructor: f}  
    ►__proto__: f ()  
    [[FunctionLocation]]: demo.html:14  
  ▼[[Scopes]]: Scopes[3]  
    ►0: Closure (User) {name: "Mahmud"}  
    ►1: Script {user: User}  
    ►2: Global {postMessage: f, blur: f  
  ►__proto__: Object
```



# Privata metoder

```
function User(name, birthday) {  
    function calcAge() {  
        let birthDate = new Date(birthday);  
        return new Date().getFullYear() - birthDate.getFullYear();  
    }  
  
    this.sayHi = function() {  
        return (`Hello ${name}, age: ${calcAge()}`);  
    }  
}
```

# Privata metoder - fortsättning

- Metoden calcAge() är en privat metod.
- Du får använda metoden inuti konstruktorn.

```
let user = new User("Mahmud", "1973 02 02");  
console.log(user.sayHi());
```

```
user.calcAge();
```



► **Uncaught TypeError:**  
**user.calcAge is not a function**

# Privata metoder lagras i en closure

```
▼ User {sayHi: f} ⓘ  
  ▼ sayHi: f ()  
    arguments: null  
    caller: null  
    length: 0  
    name: ""  
  ► prototype: {constructor: f}  
  ► __proto__: f ()  
    [[FunctionLocation]]: demo.html:18  
  ▼ [[Scopes]]: Scopes[3]  
    ▼ 0: Closure (User)  
      birthday: "1973 02 02"  
    ► calcAge: f calcAge()  
      name: "Mahmud"
```



# Factory class pattern

```
function User(name, birthday) {  
    function calcAge() {  
        let birthDate = new Date(birthday);  
        return new Date().getFullYear() - birthDate.getFullYear();  
    }  
    return {  
        sayHi() {  
            return `${name}, age:${calcAge()}`;  
        }  
    }  
}
```

# Factory class pattern – fortstättning

- Men hjälp av en factory returnerar vi publika egenskaper och metoder.
- Vi behöver inte använda nyckelordet **new**  
(Jämför med föregående exempel).
- Allt annat är precis samma som "Functional class pattern".

```
let user = User("Mahmud", "1973 02 02");
console.log(user.sayHi());
```

# Prototype-based classes



```
function User(name) {  
    this._name = name;  
}
```

```
User.prototype.sayHi = function() {  
    return (`${this._name}`);  
};
```

# Konstruktorn skapar objekt med data

```
let user = new User("Mahmud");
```

```
> user
< ▶User {_name: "Mahmud"} ⓘ
  _name: "Mahmud"
  ▶ __proto__: Object
```



# Metoder lagras i prototypen

Metoder lagras inte i varje objekt.

```
console.log(user.sayHi());
```

```
▼ User {_name: "Mahmud"} ⓘ  
  _name: "Mahmud"  
  ▼ __proto__:  
    ► sayHi: f ()  
    ► constructor: f User(name)  
    ► __proto__: Object
```

“Prototypal pattern is more memory-efficient” \*

\* <https://javascript.info/class-patterns>

# Prototype-based classes – fortsättning

```
function User(name, birthday) {  
    this._name = name;  
    this._birthday = birthday;  
}
```

```
User.prototype._calcAge = function() {  
    let birthDate = new Date(this._birthday);  
    return new Date().getFullYear() - birthDate.getFullYear();  
};
```

```
User.prototype.sayHi = function() {  
    return (`${this._name}, age: ${this._calcAge()}`);  
};
```

Data lagras i objektet

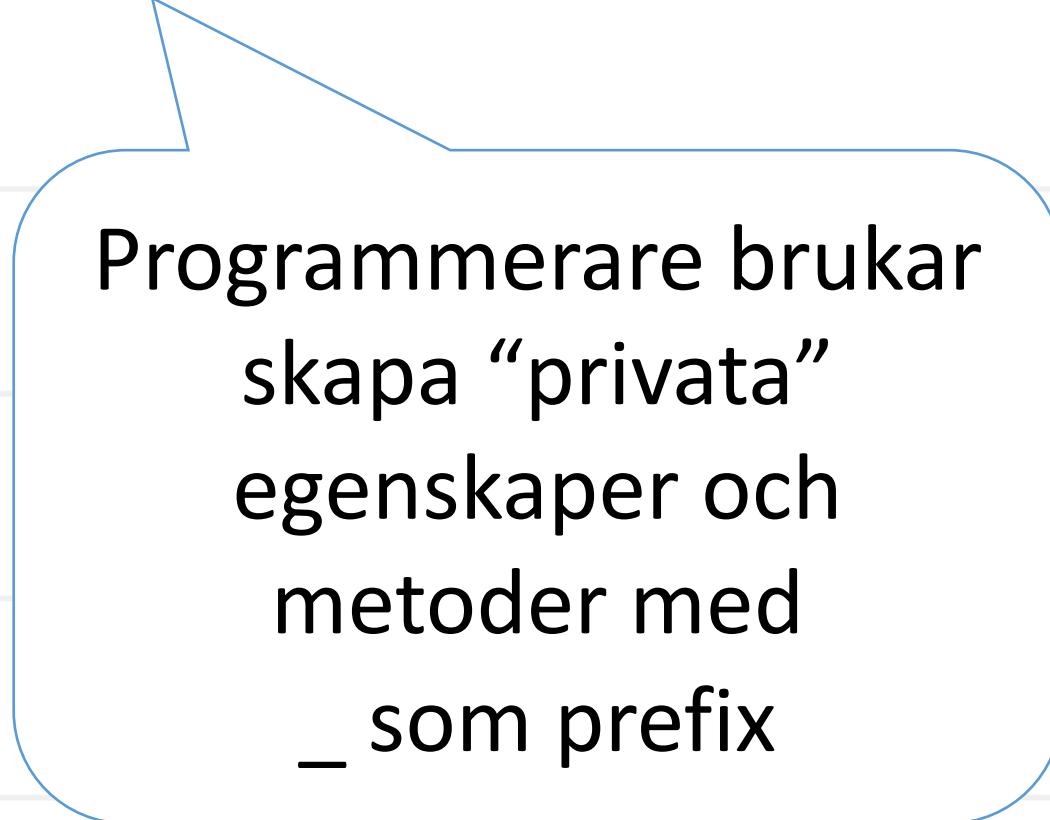
Metoder lagras i prototypen

```
let user = new User("Mahmud", "1973 02 02");
console.log(user.sayHi());
```

```
▼ User {_name: "Mahmud", _birthday: "1973 02 02"}
  _birthday: "1973 02 02"
  _name: "Mahmud"
▼ __proto__:
  ► sayHi: f ()
  ► _calcAge: f ()
  ► constructor: f User(name, birthday)
  ► __proto__: Object
```

# Objektet har tillgång till allt som finns i prototypen

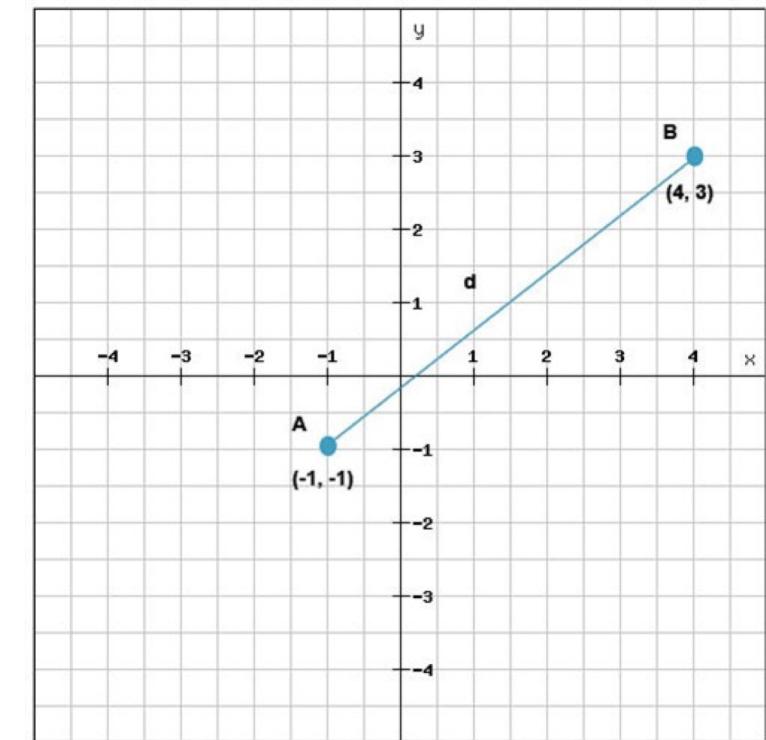
```
> user
<- ►User {_name: "Mahmud", _birthday: "1973 02 02"}
> user._name
<- "Mahmud"
> user._birthday
<- "1973 02 02"
> user._calcAge()
<- 46
> user.sayHi()
<- "Mahmud, age: 46"
> |
```



Programmerare brukar skapa “privata” egenskaper och metoder med `_` som prefix

# Övning

1. Skapa en konstruktor som beskriver en punkt i en tvådimensionell värld.
2. Skapa en metod i prototypen som beräknar avståndet mellan två punkter.
3. Skapa två punkter (två objekt).
4. Beräkna avståndet mellan punkterna.



- Bildkälla:  
<https://www.matteboken.se/lektioner/matte-2/geometri/avstandsformeln>

# Inheritance (arv)

```
▼ User {_name: "Mahmud", _birthday: "1973 02 02"} ⓘ  
  _birthday: "1973 02 02"  
  _name: "Mahmud"  
▼ __proto__:  
  ► sayHi: f ()  
  ► _calcAge: f ()  
  ► constructor: f User(name, birthday)  
▼ __proto__:  
  ► constructor: f Object()  
  ► hasOwnProperty: f hasOwnProperty()  
  ► isPrototypeOf: f isPrototypeOf()  
  ► propertyIsEnumerable: f propertyIsEnumerable()  
  ► toLocaleString: f toLocaleString()  
  ► toString: f toString()  
  ► valueOf: f valueOf()
```

Alla objekt i  
JavaScript ärver från  
**Object**

# Object.prototype.hasOwnProperty()

```
> user.hasOwnProperty('_name')
< true
> user.hasOwnProperty('_birthday')
< true
```

- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Object/hasOwnProperty](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwnProperty)

# Arv – En superklass som beskriver ett djur

```
function Animal(name) {  
  this.name = name;  
}  
  
Animal.prototype.eat = function() {  
  return (`${this.name} eats.`);  
};  
  
let animal = new Animal("My animal");  
console.log(animal.eat());
```

# Animal

*Animal*

*constructor*

*prototype*



*Animal.prototype*

*eat: function*

*new Animal()*

*name: "My animal"*



**[[Prototype]]**

# Arv – En klass som beskriver en kanin

```
function Rabbit(name) {  
    this.name = name;  
}  
  
Rabbit.prototype.jump = function() {  
    return (`${this.name} jumps!`);  
};  
  
let rabbit = new Rabbit("My rabbit");  
console.log(rabbit.jump());
```

# Rabbit

*Rabbit*

`constructor`

`prototype`



*Rabbit.prototype*

`jump: function`

*new Rabbit()*

`name: "My rabbit"`



`[[Prototype]]`

# Inheritance chain (Prototypkedja)

*Animal.prototype*

*methods of Animal*



**[[Prototype]]**

*Rabbit.prototype*

*methods of Rabbit*



**[[Prototype]]**

*rabbit*

*properties of rabbit*

# En kanin ärver djurens egenskaper och metoder

// OBS! Sätt detta före kaninens egna metoder!

```
Rabbit.prototype = Object.create(Animal.prototype);
```

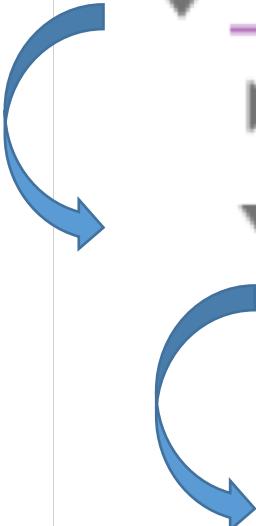
// Eller så här. OBS! Deprecated (Rekommenderas ej)

```
// Rabbit.prototype.__proto__ = Animal.prototype;
```

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Object/proto](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/proto)

# Prototypkedja

```
▼ Rabbit {name: "My rabbit"} ⓘ  
  name: "My rabbit"  
  ▼ __proto__: Animal  
    ► jump: f ()  
    ▼ __proto__:  
      ► eat: f ()  
      ► constructor: f Animal(name)  
      ► __proto__: Object
```



# Övning 1 – Hitta felet

```
1 function Animal(name) {  
2     this.name = name;  
3 }  
4  
5 Animal.prototype.walk = function() {  
6     alert(this.name + ' walks');  
7 };  
8  
9 function Rabbit(name) {  
10    this.name = name;  
11 }  
12  
13 Rabbit.prototype = Animal.prototype;  
14  
15 Rabbit.prototype.walk = function() {  
16     alert(this.name + " bounces!");  
17 };
```

Lösning finns här: <https://javascript.info/class-patterns>

## Övning 2 – Rewrite to prototypes

- The Clock class is written in functional style.
- Rewrite it using prototypes.
- P.S. The clock ticks in the console, open it to see.
- [Open a sandbox for the task.](#)
- <https://javascript.info/task/rewrite-to-prototypes>

# The “class” syntax

The “class” construct allows one to define prototype-based classes with a clean, nice-looking syntax.

<https://javascript.info/class>

# Prototype-based class

```
function User(name) {  
  this.name = name;  
}  
  
User.prototype.sayHi = function() {  
  alert(this.name);  
};
```

# class syntax

```
class User {  
  constructor(name) {  
    this.name = name;  
  }  
  
  sayHi() {  
    alert(this.name);  
  }  
}
```

# Babel – <https://babeljs.io/repl>

The screenshot shows the Babel REPL interface. On the left, there's a sidebar with settings like Evaluate, Line Wrap, Minify, Prettify, File Size, and Time Travel. Under PRESETS, 'es2015-loose' is selected. The main area has two panes: 'Try it out' on the left and 'Output' on the right. In the 'Try it out' pane, the following ES6 class is written:

```
1 class User {  
2     constructor(name) {  
3         this.name = name;  
4     }  
5  
6     sayHi() {  
7         alert(this.name);  
8     }  
9 }
```

In the 'Output' pane, the transformed ES5 code is shown:

```
1 "use strict";  
2  
3 var User =  
4 /*#__PURE__*/  
5 function () {  
6     function User(name) {  
7         this.name = name;  
8     }  
9  
10    var _proto = User.prototype;  
11  
12    _proto.sayHi = function sayHi() {  
13        alert(this.name);  
14    };  
15  
16    return User;  
17 }();
```

# Övning – Rewrite to class

- Rewrite the Clock class from prototypes to the modern “class” syntax.
- P.S. The clock ticks in the console, open it to see.
- [Open a sandbox for the task.](#)
- <https://javascript.info/task/rewrite-to-class>

# JavaScript OOP Crash Course (ES5 & ES6)

Brad Traversy <https://youtu.be/vDJpGenyHaA>



Object-oriented Programming in JavaScript:  
Mosh Hamedani <https://youtu.be/PFmuCDHHpwk>



# Summering av dagens lektion

- JavaScripts objekt
- Prototyper och prototypkedja
- Klasser
- **Bra att läsa**  
<https://javascript.info/classes>
- Reflektioner kring dagens lektion?

NACKADEMIN

Framåtblick inför nästa lektion

Lycka till med projektarbetet

NACKADEMIN