MALMÖ UNIVERSITY

INLEDANDE WEBBPROGRAMMERING MED JAVASCRIPT

INTRODUCTION TO WEB PROGRAMING USING JAVASCRIPT

ME152A

L2: DATA STRUCTURES AND OBJECTS

OUTLINE

- What did we learn so far?
- Data structures and objects
- Creating objects
- JavaScript prototypes

WHAT DID WE LEARN SO FAR?

- Conditional execution?
- Loops?
- Functions?

WHY DATA STRUCTURES AND OBJECTS?

- Numbers, Booleans, and strings are the bricks that data structures are built from.

- *Objects* allow us to group values—including other objects—together and thus build more complex structures.

DATA SETS

- Represent a collection of numbers: 2, 3, 5, 7, and 11.

- How should we represent these numbers?
- Strings?

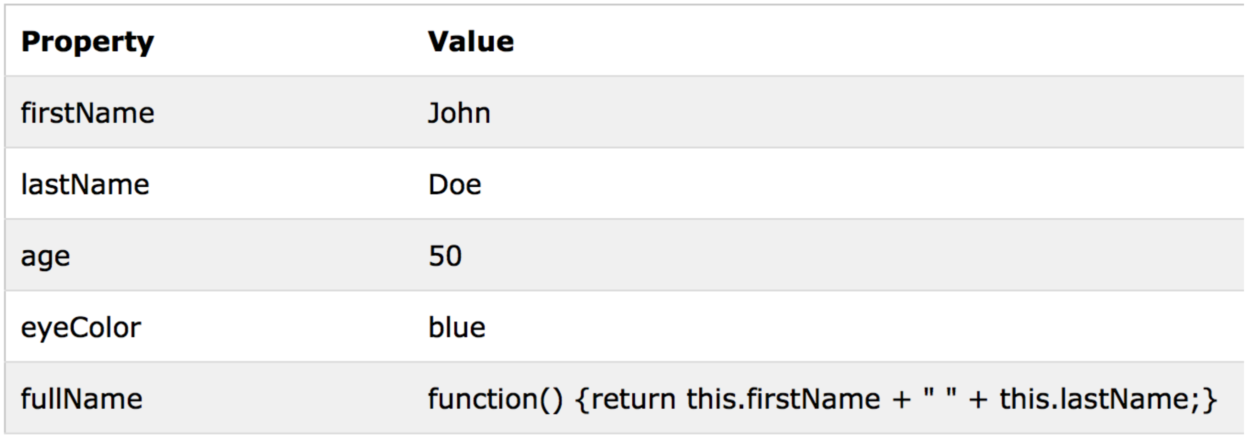
```
var listOfNumbers = [2, 3, 5, 7, 11];
console.log(listOfNumbers[1]);
// → 3
console.log(listOfNumbers[1 - 1]);
// → 2
```

PROPERTIES

- Almost all JavaScript values have properties.
- The exceptions are `null` and `undefined`.
- The two most common ways to access properties in JavaScript are with a dot and with square brackets.
- The two most common ways to access properties in JavaScript are with a dot and with square brackets.
  - `value.x` and `value[x]`
    - `value.x` fetches the property of value named "`x`"
    - `value[x]` tries to evaluate the expression x and uses the result as the property name.
- The elements in an array are stored in properties.

METHODS

- Properties that contain functions are generally called *methods* of the value they belong to. In our previous examples: “`toUpperCase` is a method of a string”.



Property	**Value**
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

http://www.w3schools.com/js/js_object_methods.asp

OBJECTS

- Values of the type *object* are arbitrary collections of properties, and we can add or remove these properties as we please.
- One way to create an object is by using a curly brace notation.

```
var day1 = {
  squirrel: false,
  events: ["work", "touched tree", "pizza", "running",
           "television"]
};
```

- Properties whose names are not valid variable names or valid numbers have to be quoted.

```
var descriptions = {
  work: "Went to work",
  "touched tree": "Touched a tree"
};
```

REAL LIFE OBJECTS, PROPERTIES, AND METHODS

- In real life, a car is an **object**.

Object	Properties	Methods

	car.name = Fiat
	car.model = 500
	car.weight = 850kg
	car.color = white
		car.start()
		car.drive()
		car.brake()
		car.stop()

http://www.w3schools.com/js/js_objects.asp

OBJECTS

- In JavaScript, almost "everything" is an object.
  - Booleans can be objects (or primitive data treated as objects)
  - Numbers can be objects (or primitive data treated as objects)
  - Strings can be objects (or primitive data treated as objects)
  - Dates are always objects
  - Maths are always objects
  - Regular expressions are always objects
  - Arrays are always objects
  - Functions are always objects
  - Objects are objects
- In JavaScript, all values, except primitive values, are objects.
- Primitive values are: `strings ("John Doe"), numbers (3.14), true, false, null, and undefined.`

http://www.w3schools.com/js/js_object_definition.asp

CREATING OBJECT

- With JavaScript, you can define and create your own objects.
- There are different ways to create new objects:
  - Define and create a single object, using an object literal.
  - Define and create a single object, with the keyword new.
  - Define an object constructor, and then create objects of the constructed type.

http://www.w3schools.com/js/js_object_definition.asp

USING AN OBJECT LITERAL

- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

```
var person = {
        firstName: "John”  ,
        lastName: "Doe”  ,
        age: 50,
        eyeColor: "blue”
};
console.log (person.firstName + " is " + person.age
+ " years old.” );
```

http://www.w3schools.com/js/js_object_definition.asp

USING THE KEYWORD NEW

- The following example also creates a new JavaScript object with four properties:

```
var person = new Object();
person.firstName = "John”;
person.lastName = "Doe”;
person.age = 50;
person.eyeColor = "blue”;
```

http://www.w3schools.com/js/js_object_definition.asp

USING AN OBJECT CONSTRUCTOR

- Sometimes we like to have an "object type" that can be used to create many objects of one type.
- The standard way to create an "object type" is to use an object constructor function:

```
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
var myFather = new person ( "John", "Doe", 50,
"blue” );
var myMother = new person("Sally", "Rally", 48,
"green");
```

http://www.w3schools.com/js/js_object_definition.asp

OBJECTS

```
// An object can be created with// Attribute: value pairs
        var dog = {                                   value
            name: "Boo",
            type: "Pomeranian",
attribute   age: 11
        };
        // Or, these can be added later
        var dog = {};
        dog.name = "Boo”;
        dog.type = "Pomeranian”;
        dog.age = 11;
```

OBJECTS

```
//An object, like an array, can contain all
kinds of values
      var obj = {
           text: "ABC",
           number: 1997,
           bool: true,
           list: [1, 2, 3, "e”]
        };
```

“THIS” KEYWORD

- **this** is the object that "owns" the JavaScript code.
- The value of **this**, when used in a function, is the object that "owns" the function.
- The value of **this**, when used in an object, is the object itself.
- The **this** keyword in an object constructor does not have a value. It is only a substitute for the new object.
- The value of **this** will become the new object when the constructor is used to create an object.

- Note: that **this** is not a variable. It is a keyword. You cannot change the value of **this**.

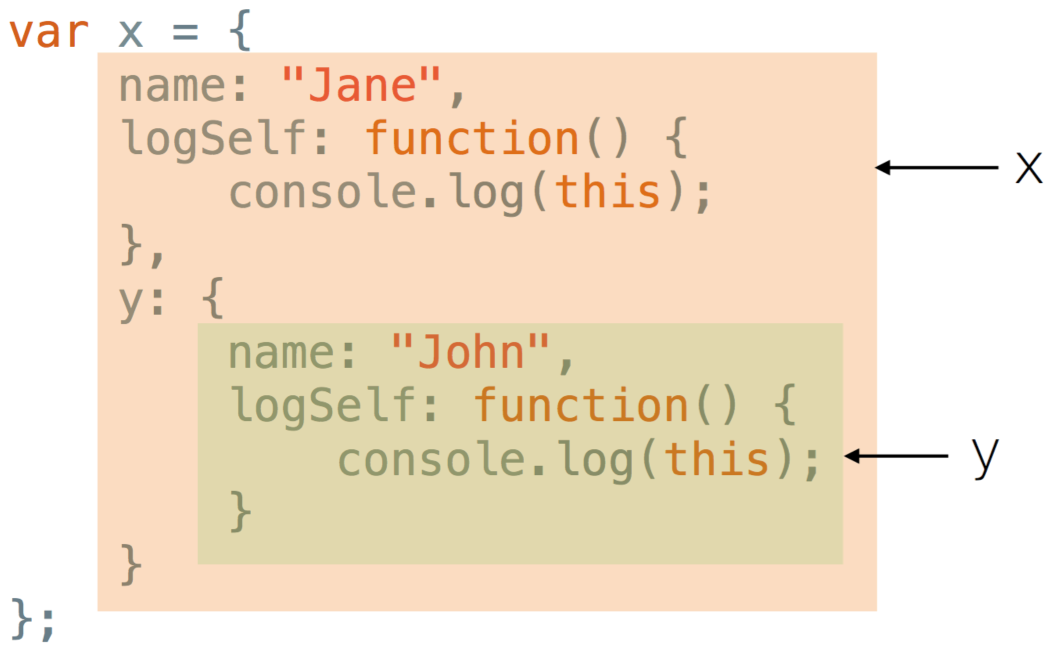
http://www.w3schools.com/js/js_object_definition.asp

“THIS” KEYWORD EXAMPLE

```
var person = {
    firstname: "Jane” ,
    lastname: "Doe” ,
    fullname: function() {
        return this.firstname + " " +
this.lastname;
     }
};
console.log (person.firstname);
console.log (person.fullname());
```

http://www.w3schools.com/js/js_object_definition.asp

“THIS” KEYWORD EXAMPLE

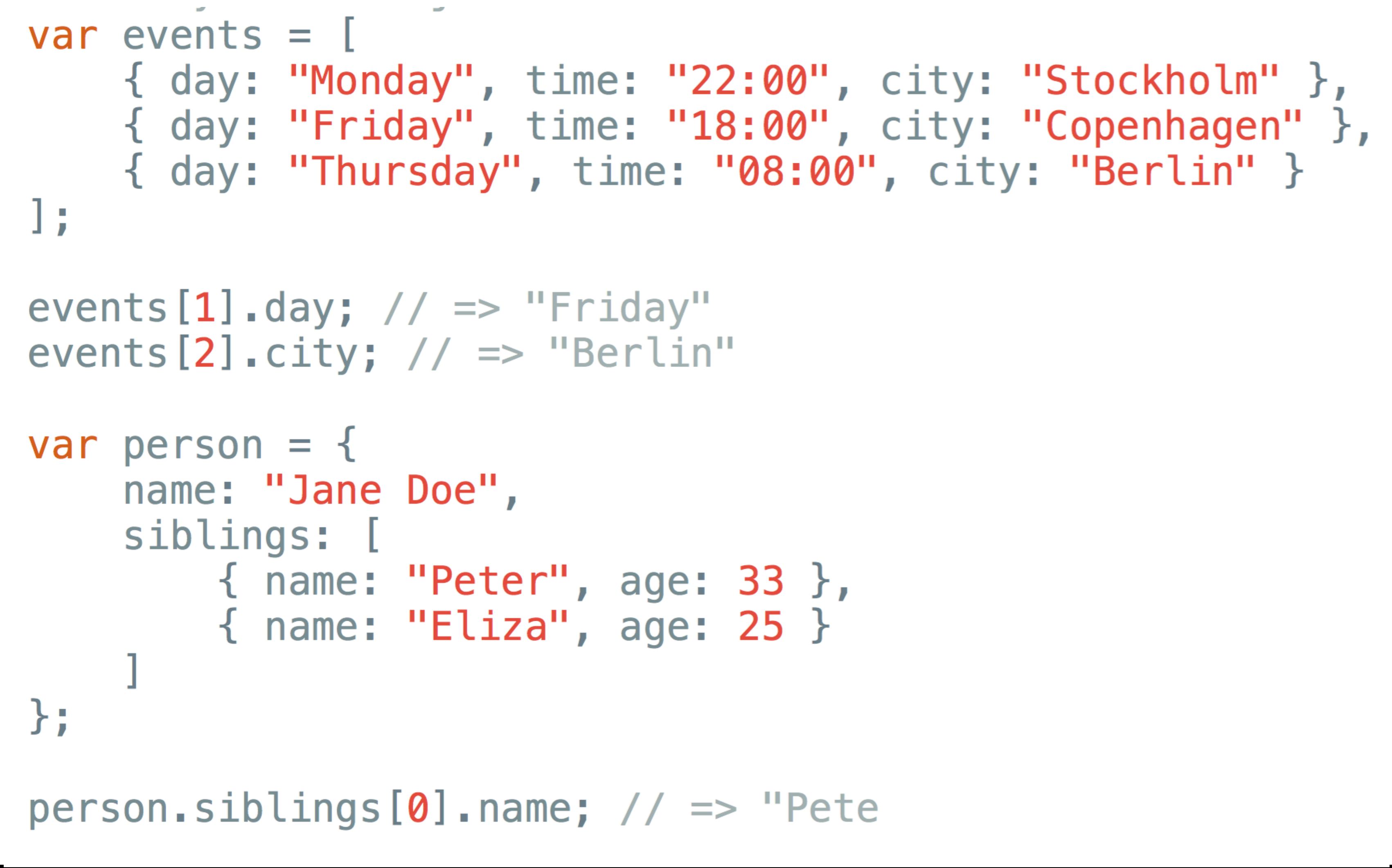


```
var x = {
    name: "Jane",
    logSelf: function() {
        console.log(this);
    },
    y: {
        name: "John",
        logSelf: function() {
            console.log(this);
        }
    }
};

x.logSelf();
// => { name: "Jane", logSelf: [Function], y: [Object] }

x.y.logSelf();
// => { name: "John", logSelf: [Function] }
```

FUNCTION TO DYNAMICALLY CREATE AN OBJECT - WHEN WE NEED IT

```
function createPerson(first, last, age) {
    return {
        firstname: first,
        lastname: last,
        age: age,
        fullname: function() {
            return this.firstname + " " + this.lastname;
        }
    };
}

var jane = createPerson("Jane", "Doe", 23);

jane.fullname(); // => "Jane Doe"
```

ARRAYS AND OBJECTS IN COMBINATION



```
var events = [
    { day: "Monday", time: "22:00", city: "Stockholm" },
    { day: "Friday", time: "18:00", city: "Copenhagen" },
    { day: "Thursday", time: "08:00", city: "Berlin" }
];

events[1].day; // => "Friday"
events[2].city; // => "Berlin"

var person = {
    name: "Jane Doe",
    siblings: [
        { name: "Peter", age: 33 },
        { name: "Eliza", age: 25 }
    ]
};

person.siblings[0].name; // => "Pete
```

JAVASCRIPT OBJECTS ARE MUTABLE

- Objects are mutable: They are addressed by reference, not by value.
- If y is an object, the following statement will not create a copy of y :

```
var x = y;  // This will not create a copy of y.
```

- The object x is not a **copy** of y. It **is** y. Both x and y points to the same object.
- Any changes to y will also change x, because x and y are the same object.

```
var person = {firstName:"John” , lastName: "Doe” ,
age:50, eyeColor: "blue” }
var x = person;
x.age = 10;   // This will change both x.age and
person.age
```

http://www.w3schools.com/js/js_object_definition.asp

JAVASCRIPT PROTOTYPES

- Every JavaScript object has a prototype. The prototype is also an object.
- All JavaScript objects inherit their properties and methods from their prototype.
- All JavaScript objects inherit the properties and methods from their prototype.
- `Objects` created using an object literal, or with new `Object()`, inherit from a prototype called `Object.prototype`.
- `Objects` created with `new Date()` inherit the `Date.prototype`.
- The `Object.prototype` is on the top of the prototype chain.
- All JavaScript objects (Date, Array, RegExp, Function, ...) inherit from the `Object.prototype`.

http://www.w3schools.com/js/js_object_prototypes.asp

JAVASCRIPT OBJECT PROTOTYPES

- The standard way to create an object prototype is to use an object constructor function:

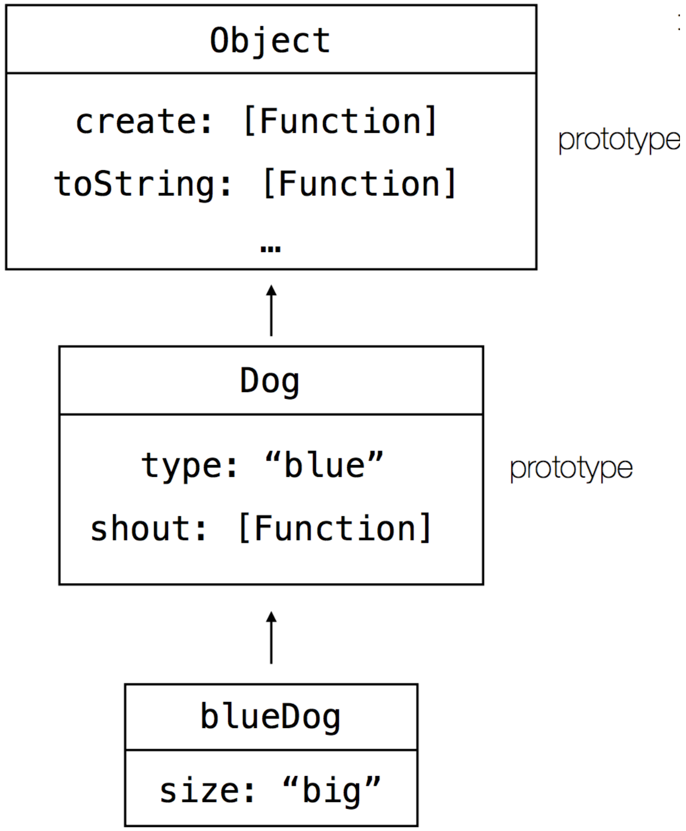
```
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
```

- With a constructor function, you can use the **new** keyword to create new objects from the same prototype:

```
var myFather = new person ( "John", "Doe", 50,
"blue” );
var myMother = new person("Sally", "Rally", 48,
"green");
```

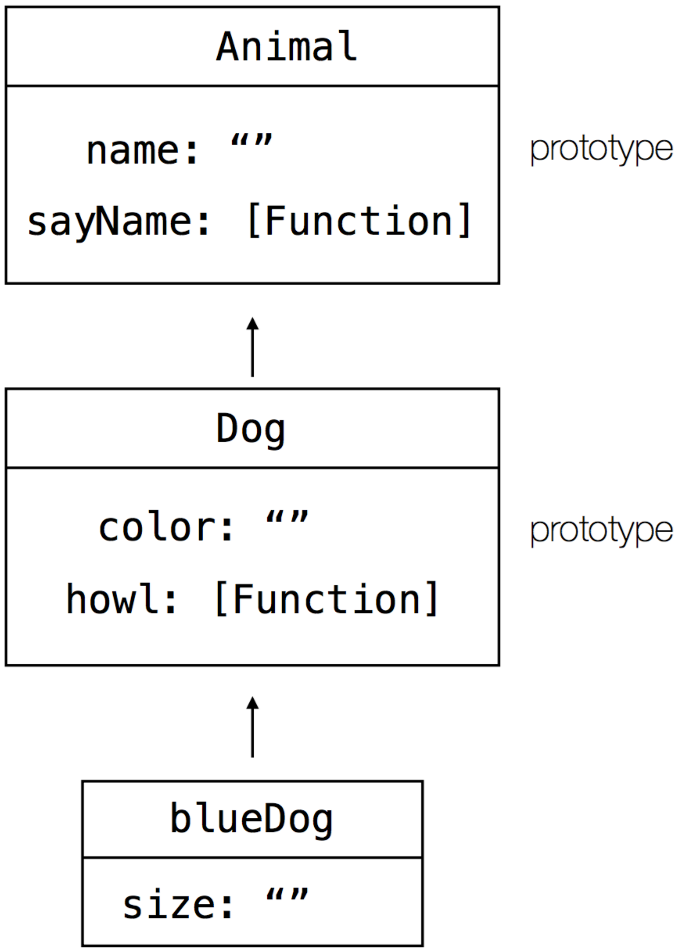
http://www.w3schools.com/js/js_object_prototypes.asp

JAVASCRIPT OBJECT PROTOTYPES



```
// An attribute that is inherited by all objects
Object.prototype.random = "Hello!";

function Dog(type) {
    this.type = type;
}

var blueDog = new Dog("blue");
blueDog.size = "big";

// Now we attribute "random"
for (var prop in blueDog) {
    console.log( prop );
}

for (var prop in blueDog) {
    // Check that the attributes come directly
    from our own object (i.e.: BlueDog)
    if (blueDog.hasOwnProperty(prop)) {
        console.log( prop );
    }
}
```

JAVASCRIPT OBJECT PROTOTYPES



```
function Animal(name) {
    this.name = name;
}

Animal.prototype.sayName = function() {
    console.log("My name is " + this.name);
};

function Dog(name, color) {
    Animal.call(this, name);
    this.color = color;
}

Dog.prototype = new Animal();

Dog.prototype.howl = function() {
    console.log("The " + this.color + " dog howls!");
};

var blueDog = new Dog("Snappy", "blue");
blueDog.size = "big";
```

ADDING PROPERTIES AND METHODS TO OBJECTS

Adding a new property to an existing object is easy:

```
myFather.nationality = "English";
```

Adding a new method to an existing object is also easy:

```
myFather.name = function () {
    return this.firstName + " " +
this.lastName;
};
```

Keep in mind:
You cannot add a new property to a prototype the same way as you add a new property to an existing object, because the prototype is not an existing object.

```
person.nationality = "English";
```

http://www.w3schools.com/js/js_object_prototypes.asp

ADDING PROPERTIES TO A PROTOTYPE

To add a new property to a constructor, you must add it to the constructor function:

```
function person(first, last, age,
eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
    this.nationality = "English"
}
```

http://www.w3schools.com/js/js_object_prototypes.asp

ADDING METHODS TO A PROTOTYPE

Your constructor function can also define methods:

```
function person(first, last, age,
eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
    this.name = function() {return
this.firstName + " " +
this.lastName;};
}
```

http://www.w3schools.com/js/js_object_prototypes.asp

USING THE “PROTOTYPE” PROPERTY (ADD NEW PROPERTIES)

The JavaScript prototype property allows you to add new properties to an existing prototype:

```
function person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
person.prototype.nationality = "English";
```

http://www.w3schools.com/js/js_object_prototypes.asp

USING THE “PROTOTYPE” PROPERTY (ADD NEW METHODS)

The JavaScript prototype property also allows you to add new methods to an existing prototype:

```
function person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
person.prototype.name = function() {
    return this.firstName + " " + this.lastName;
};
```

http://www.w3schools.com/js/js_object_prototypes.asp

OBJECT-ORIENTED PROGRAMMING

- Object-oriented programming (OOP) is a programming paradigm that uses abstraction to create models based on the real world.
- OOP envisions software as a collection of cooperating objects rather than a collection of functions or simply a list of commands
- OOP promotes greater flexibility and maintainability in programming, and is widely popular in large-scale software engineering.
- Object-oriented code promotes more direct analysis, coding, and understanding of complex situations and procedures

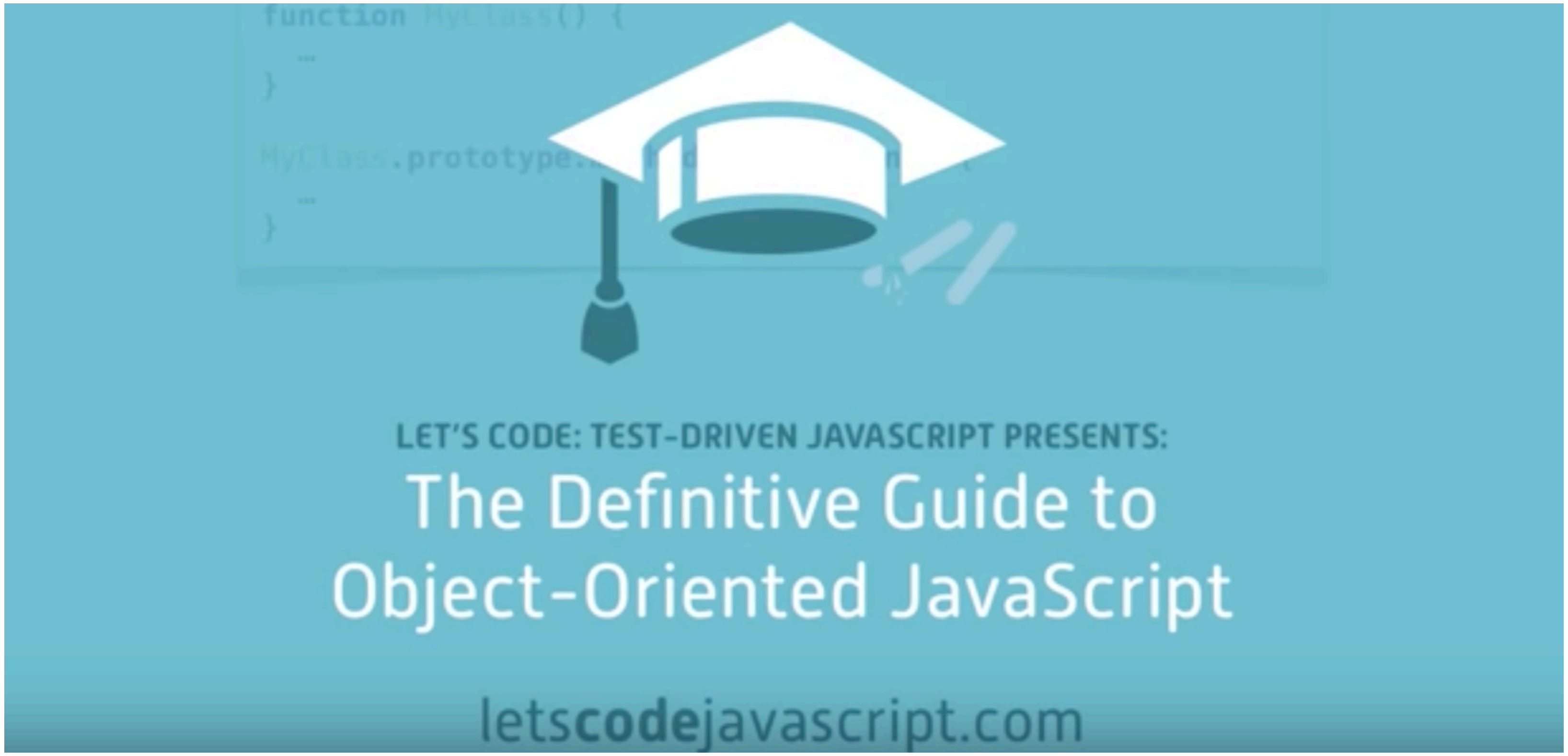
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript

TERMINOLOGY

- **Namespace**: A container which lets developers bundle all functionality under a unique, application-specific name.
- **Class:** Defines the object's characteristics. A class is a template definition of an object's properties and methods.
- **Object:** An instance of a class.PropertyAn object characteristic, such as color.
- **Method:** An object capability, such as walk. It is a subroutine or function associated with a class.
- **Constructor:** A method called at the moment an object is instantiated. It usually has the same name as the class containing it.
- **Inheritance:** A class can inherit characteristics from another class.
- **Encapsulation:** A method of bundling the data and methods that use the data.
- **Abstraction:** The conjunction of an object's complex inheritance, methods, and properties must adequately reflect a reality model.
- **Polymorphism:** Poly means "many" and morphism means "forms". Different classes might define the same method or property.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript

VIDEO TUTORIAL: THE DEFINITIVE GUIDE TO OBJECT-ORIENTED JAVASCRIPT


function MyClass() {
...
}
MyClass.prototype.
...
}
LET'S CODE: TEST-DRIVEN JAVASCRIPT PRESENTS:
The Definitive Guide to Object-Oriented JavaScript
letscodejavascript.com

https://www.youtube.com/watch?v=PMfcsYzj-9M

REFLECTION

- Objects
- JavaScript prototype
    - Add new methods
    - Add new properties
    - …
- Number of examples
- OOP in general

NEXT WEEKS SEMINAR (FRIDAY13:15, NI:B0308)

- I look forward to discuss the following two topics on next week:
  - Web vs native mobile app development:
    - a. What are the benefits of HTML5 in mobile platforms?
    - b. What are the drawbacks of HTML5 in mobile platforms?
  - Choosing the right mobile platform/architecture and user experience design:
    - a. What mobile platforms/architectures would you choose, and why
    - b. How would you design the mobile user experience in order to make you app more usable?
- Articles to read:
  - Native vs Web vs Hybrid: How to Select the Right Platform for Your Enterprise's Mobile Apps
  - Mobile Development Overview:
    - CHAPTER 1: Choosing the Right Architecture and
    - CHAPTER 2: Designing Your User Experience.
- Link to the materials: https://www.dropbox.com/sh/rgtwf5bqafhe3u0/AADyAURs0YZS-OLrfSGdYa7La?dl=0
- Videos:
  - Native, Web or Hybrid Mobile Apps?: https://www.youtube.com/watch?v=Ns-JS4amlTc
  - Native, HTML5, and Hybrid Mobile App Development: Real-Life Experiences - Eran Zinman: https://www.youtube.com/watch?v=We0byPckthQ

THANK YOU

QUESTIONS?

Literature:

Haverbeke, M. (2014). *Eloquent JavaScript: A Modern Introduction to Programming*. No Starch Press.