Q

How are Objects defined in JavaScript?

Javascript objects

JavaScript is confusing for developers coming from Java or C++, as it's all dynamic, all runtime, and it has no classes at all.

It's all just instances (objects).

Even the "classes" we simulate (introduced in ES5) are just a function object.

from developer.mozilla.org

Javascript objects as data structures

Javascript objects are
collections of **named values** (**properties**)

```
var person = {firstName:"Dorothea",
  lastName:"Wierer", birthyear:1990};
```

Javascript objects can contain also **methods**

```
var person = {firstName:"Dorothea",
  lastName:"Wierer", birthyear:1990,
  fullName:function() {return this.firstName
  + " " + this.lastName;}};
```

Accessing properties

```
person.firstName
```

is equivalent to

```
person["firstName"];
```

and to

```
var x="firstName";
person[x];
```

Dynamic management of objects

You can dynamically add new properties to an existing object by simply giving it a value.

```
var person = {firstName:"Dorothea",
  lastName:"Wierer", birthyear:1990};
person.birthplace="Brunico;
```

You can also delete existing properties.

```
delete person.firstName;
```

Other ways to create objects

Hence, Javascript objects can also be created empty and populated later.

```
var person = {};
  person.firstName="Dorothea";
  person. lastName:"Wierer";
  person. birthyear:1990;
   person.fullName=function() {
      return this.firstName + " " +
  this.lastName;};
```

"Object" is a misleading name!

If you come from a language like Java, where objects are instances of classes, you will notice that the "Object" notion of Javascript is quite different!

For instance:

- you can not change the structure of a Java object, but you can change the structure of a JavaScript object!
- you cannot have objects without class in Java, but you do in JavaScript!

Other ways to create object

```
var object1 = new Object();
Object.defineProperty(
   object1, 'name', {
   value: "AA",
   writable: false
});
object1.name = 77;
document.write(object1.name);
```

OUTPUT:
AA

(but no error)

if writable:true:

OUTPUT:
77

Objects constructors

Object constructors: templates for creating objects of a certain type (somehow similar to the concept of "class").

```
function Rectangle(w, h) {
    this.width=w;
    this.height=h;
    this.area=function(){return this.width*this.height}
    }
a=new Rectangle(3,4);        a.area() => 12      a.width => 3
```

Instance variables

method

The constructor function is JavaScript's version of a class.

Odd consequences…

```
<script>
p = (n,s) => document.write(n+":"+s+"<br>");
function Rectangle(w, h) {
      this.width=w;
      this.height=h;
      this.area=function(){
            return this.width*this.height}
}
a=new Rectangle(2,3);
b=new Rectangle(2,3);
p(a.area(),b.area());
a.area=function(){return this.width*5}
p(a.area(),b.area());
</script>
```

OUTPUT:
6:6
10:6

What are prototypes?

Objects

The approach we have shown is not the most efficient in terms of memory allocation, as for every Rectangle we instantiate the area method!

It would be better to use the "prototype" feature.

```
Rectangle.prototype.area=function(){
        return this.w*this.h
}
```

Prototype

The Prototype is a **separate naming space**, shared by all instances of the same constructor/class

```
function Person(first, last ) {
      this.firstname = first;
      this.lastname = last;
}
Person.prototype.fullname= function () {
          return this.firstname+" "+this.lastname
      };
Person.prototype.nickname="The Best";
let person1 = new Person('Dorothea', 'Wierer');
let person2 = new Person('Valentino','Rossi');
console.log(person1.nickname);
console.log(person2.nickname);
Person.prototype.nickname="The Champion";
console.log(person1.nickname);
console.log(person2.nickname);
```

OUTPUT:
The Best
The Best
The Champion
The Champion

Prototype

The Object.create() method creates a new object, using an existing object as the prototype of the newly created object.

OUTPUT:
The Best
The Best
The Champion
The Champion

How can I inspect objects?

Inspecting objects

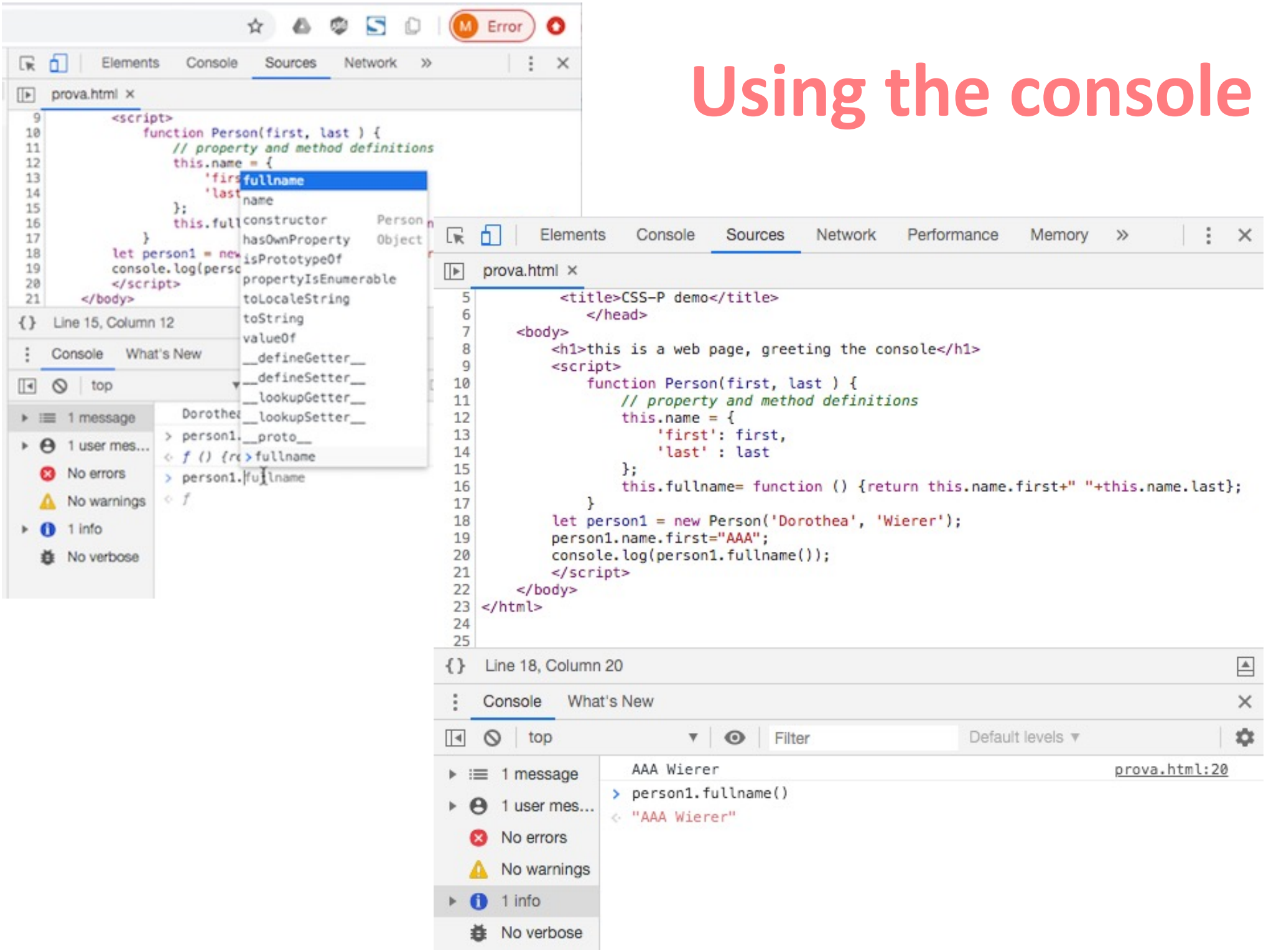
```
<script>f
x = new Rectangle(3,4);
for (e in x) { // loop over all properties
 console.log(e+" "+x[e]);
}
```

name of the property value of the property

If you only want to consider properties attached to the object itself,
and not its prototypes,
use getOwnPropertyNames()
or perform a hasOwnProperty() check.

propertyIsEnumerable() can also be used.

Using the console



Q

Can I have object inheritance in JavaScript?

Using an object as prototype of another object

```
function Person() {
    this.isHuman=false;
    this.printIntroduction=function() {
        document.write("My name is "+this.name+
            ". Am I human? "+this.isHuman);
    }
}
pippo=new Person();
pippo.printIntroduction();

const me = Object.create(pippo);
// "name" is a property set on "me",
// but not on "person"
me.name = 'Matthew';
// inherited properties can be overwritten
me.isHuman = true;
me.printIntroduction();
```

```
My name is
undefined.
Am I human? false

My name is Matthew.
Am I human? true"
```

Prototype

Moreover, the Prototype is used for providing a sort of inheritance.

Prototype-based inheritance - 1

```
function Person(first, last ) {
            this.firstname = first;
            this.lastname = last;
}
Person.prototype.nickname="The Best";
Person.prototype.fullname= function () {
    return this.firstname+" "+this.lastname};
```

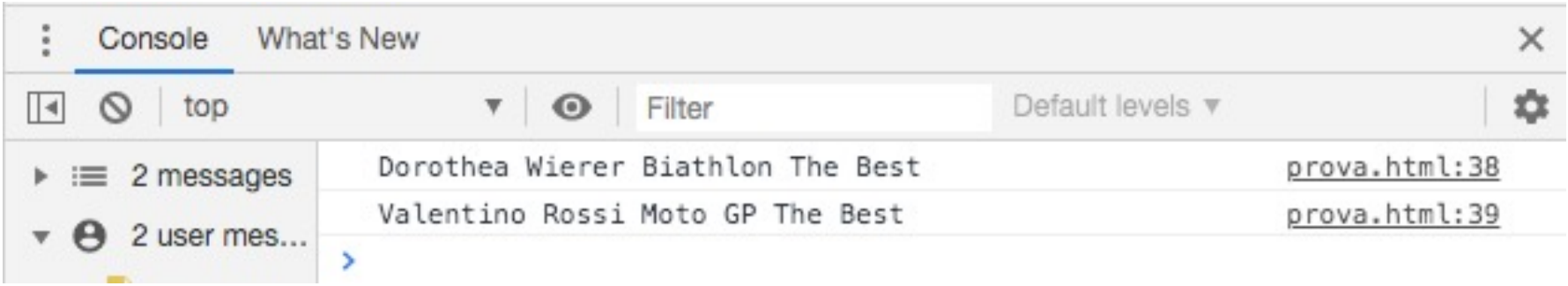
```
function Athlete(first, last, sport ) {
         Person.call(this,first,last);
         this.sport=sport;
}
Athlete.prototype = Object.create(Person.prototype);
Object.defineProperty(Athlete.prototype, 'constructor', {
          value: Athlete,
          enumerable: false,
          writable: true });
```

so that it does not appear in 'for in' loop

Prototype-based inheritance - 2

```
let person1 = new Athlete('Dorothea', 'Wierer','Biathlon');
let person2 = new Athlete('Valentino','Rossi','Moto GP');

console.log(person1.fullname()+" "+person1.sport+" "+person1.nickname);
console.log(person2.fullname()+" "+person2.sport+" "+person2.nickname);
```



Prototype-based inheritance - 3

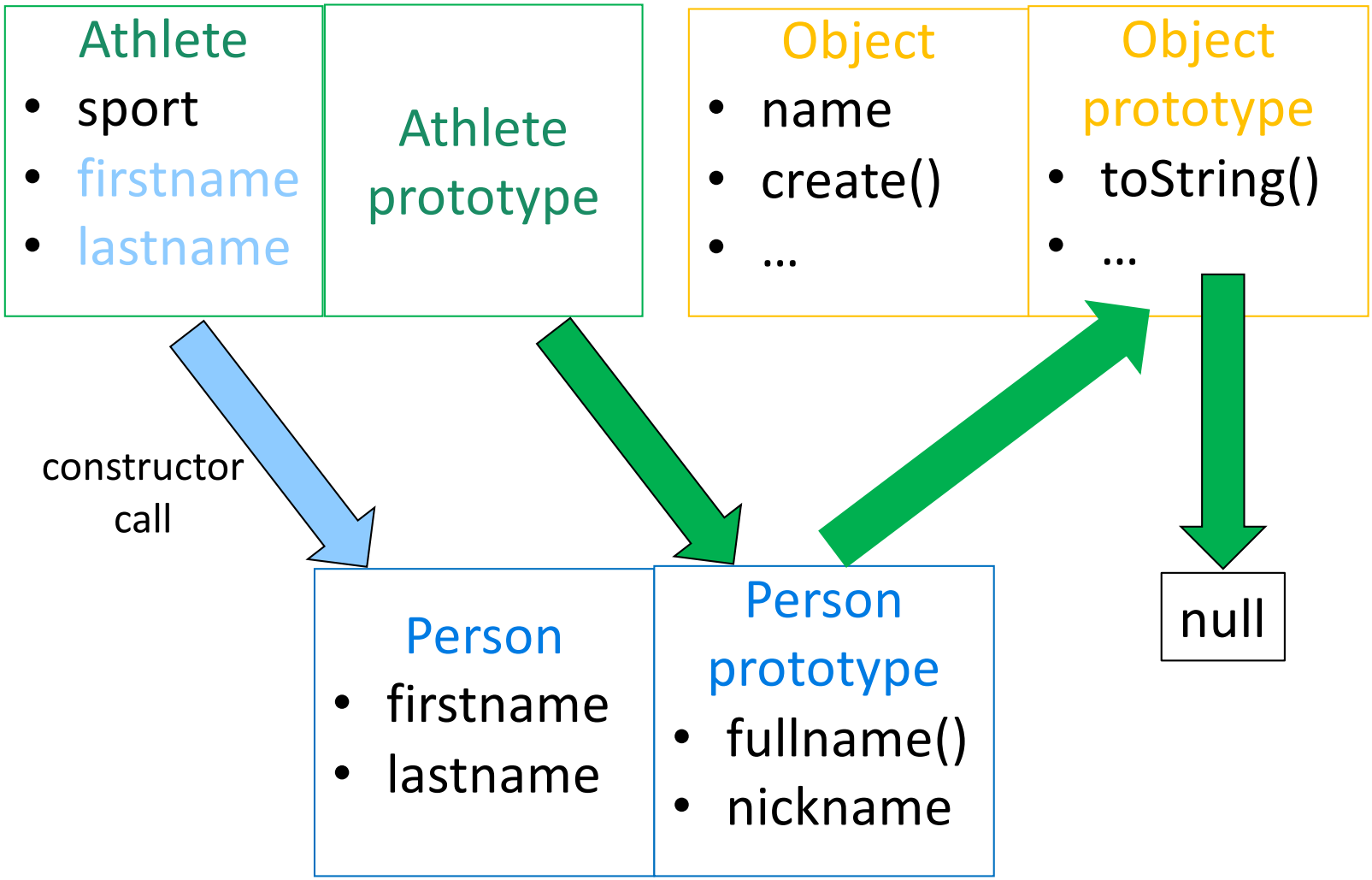
```
function Person(first, last ) {
            this.firstname = first;
            this.lastname = last;
}
Person.prototype = Object.create(Object.prototype);
Object.defineProperty(Person.prototype, 'constructor', {
          value: Person,
          enumerable: false,
          writable: true });
Person.prototype.nickname="The Best";
Person.prototype.fullname= function () {
    return this.firstname+" "+this.lastname};
```

IMPLICIT

```
function Athlete(first, last, sport ) {
         Person.call(this,first,last);
         this.sport=sport;
}
Athlete.prototype = Object.create(Person.prototype);
Object.defineProperty(Athlete.prototype, 'constructor', {
          value: Athlete,
          enumerable: false,
          writable: true });
```

Prototype-based inheritance - 4

In the constructor we define the instance variables of our instances.
In the prototype we define the methods, and the variables shared by all our instances.
By invoking the "superclass" constructor, we inherit its instance variables.
By associating or prototype to the "superclass", we inherit its prototype.



Warning!

```
function Person(first, last ) {
      this.firstname = first;
      this.lastname = last;
}
Person.prototype.nickname="The Best";
let person1 = new Person('Dorothea', 'Wierer');
let person2 = new Person('Valentino','Rossi');
console.log(person1.nickname);
console.log(person2.nickname);
Person.prototype.nickname="The Champion";
console.log(person1.nickname);
console.log(person2.nickname);
person1.nickname="The Magic";
console.log(person1.nickname);
console.log(person2.nickname);
delete person1.nickname;
console.log(person1.nickname);
console.log(person2.nickname);
```

OUTPUT:

The Best
The Best
The Champion
The Champion
The Magic
The Champion
The Champion
The Champion

Property inheritance: demo

```
function A(p) {
    this.a1 = p;
    this.a2 = "due";
}
A.prototype.a3="tre";
A.prototype.a4="quattro";
```

```
function B(p) {
    A.call(this,p);
    this.b1="cinque";
}
B.prototype = Object.create(A.prototype);
Object.defineProperty(B.prototype,
    'constructor',
    { value: B,
      enumerable: false,
      writable: true });
B.prototype.a4="sei";
```

```
let v1 = new A("uno_A");
let v2 = new B("uno_B");
for (e in v1) {
    console.log("=> A "+e+"
    "+v1[e]);
}
for (e in v2) {
    console.log("B "+e+"
    "+v2[e]);
}
```

```
=> A a1 uno_A
=> A a2 due
=> A a3 tre
=> A a4 quattro
B a1 uno_B
B a2 due
B b1 cinque
B a4 sei
B a3 tre
```

from A.constructor param

def. in A.constructor

in A.prototype

from B.constructor param

def. in A.constructor

def. in B.constructor

def. in A.prototype, overwrt in B.prototype

def. in A.prototype

Prototype-based inheritance - 5

For an in-depth discussion, see

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain#Using_prototypes_in_JavaScript

Q

Are there predefined objects in JavaScript?

Predefined objects

JavaScript has a number of predefined objects, e.g.:

```
var x1 = new Object();
var x2 = new String();     // do not use it!
var x3 = new Number();     // do not use it!
var x4 = new Boolean();    // do not use it!
var x5 = new Array();      // do not use it! Use []
var x6 = new RegExp();     // do not use it! Use /( )/
var x7 = new Function();   // do not use it!
var x8 = new Date();
```

The native prototypes should **never** be extended unless it is for the sake of compatibility with newer JavaScript features.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

Predefined objects

- Math is an object you never instantiate – it's a container for math functions.
  - https://www.w3schools.com/js/js_math.asp
  - https://www.w3schools.com/js/js_random.asp
- Date
  - https://www.w3schools.com/js/js_dates.asp
- RegExpr
  - https://www.w3schools.com/js/js_regexp.asp

https://www.w3schools.com/js/js_math.asp

Q

So, there are no classes in JavaScript?

Classes

JavaScript classes, introduced in ECMAScript 2015, are primarily syntactic sugar over JavaScript's existing prototype-based inheritance. The class syntax *does not* introduce a new object-oriented inheritance model to JavaScript.

Unlike function declarations, and class declarations are not hoisted.

Classes

```
class Person {
      constructor(first, last ) {
            this.firstname = first;
            this.lastname = last;
       }
       fullname() {
            return this.firstname+" "+this.lastname;
       }
}
```

```
class Athlete extends Person {
      constructor(first, last, sport ) {
            super(first,last);
            this.sport=sport;
      }
}
```

```
let person1 = new Athlete('Dorothea', 'Wierer','Biathlon');
 let person2 = new Athlete('Valentino','Rossi','Moto GP');

 console.log(person1.fullname()+" "+person1.sport);
 console.log(person2.fullname()+" "+person2.sport);
```

Classes

Instance variables can ONLY be defined within methods. "Regular" instance variables have been proposed for future versions, see: https://tc39.es/

Class variables (like the java static ones) can be defined, but they cannot be called on instances.

```
Person.nickname="The Champion";
let person1 = new Athlete('Dorothea', 'Wierer','Biathlon');
let person2 = new Athlete('Valentino','Rossi','Moto GP');
console.log(person1.fullname()+" "+person1.sport+" "+Person.nickname);
console.log(person2.fullname()+" "+person2.sport+" "+person2.nickname);
```

```
Dorothea Wierer Biathlon The Champion
Valentino Rossi Moto GP undefined
```

```
Person.nickname="The Champion";
Person.prototype.nickname="The Best";
let person1 = new Athlete('Dorothea', 'Wierer','Biathlon');
let person2 = new Athlete('Valentino','Rossi','Moto GP');
console.log(person1.fullname()+" "+person1.sport+" "+Person.nickname);
console.log(person2.fullname()+" "+person2.sport+" "+person2.nickname);
```

```
Dorothea Wierer Biathlon The Champion
Valentino Rossi Moto GP The Best
```

Classes

Class methods (like the java static ones) can be defined, but they cannot be called on instances.

```
class Person {
  constructor(first, last, birthYear) {
     this.firstname = first;
     this.lastname = last;
     this.birthYear = birthYear;
  }
  static getAge(p) { // we cannot use this inside here!
     let dt=new Date();
     let thisyear=dt.getFullYear();
     return thisyear-p.birthYear;
  }
  fullname() {
     return this.firstname+" "+this.lastname;
  }
}
let person1 = new Person('Dorothea', 'Wierer',1990);
console.log(person1.fullname()+" "+Person.getAge(person1));
```

OUTPUT:

```
Dorothea Wierer 30
```

Q

What are, exactly, arrays in Javascript?

Arrays

```
a=[];
a = new Array(); //discouraged
a[0]=3; a[1]="hello"; a[10]=new Rectangle(2,2);
a.length() => 11;

a["name"]="Jaric"; ⇔  a.name="Jaric";
```

Arrays are

SPARSE, INHOMOGENEOUS , ASSOCIATIVE

See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Arrays

Array give you functions to (e.g.):

- add/remove an element at the end
- add/remove an element at the front
- add/remove an element by index
- remove a number of elements starting from an index
- find the index of an element
- make a copy of an array

See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Arrays

See also a set of examples of common tasks performed on array (such as e.g. find the maximum, sort):

- https://www.w3schools.com/js/js_array_sort.asp
- https://www.w3schools.com/js/js_array_iteration.asp

Q

How does the + operator work on objects?

+ Operator with objects

We already know that the first rule with the + operator is to convert objects to primitive values.
The rule to execute the conversion is:

Check if the object:

- is not a Date AND
- has a valueOf() method AND
- its valueOf() method returns a primitive value.

If yes, use the valueOf() method.

Else, use the toString() method.

Note: the array [1,"a",2] would be converted to "**1,a,2**".
An empty object {} is converted to **"[object Object]"**