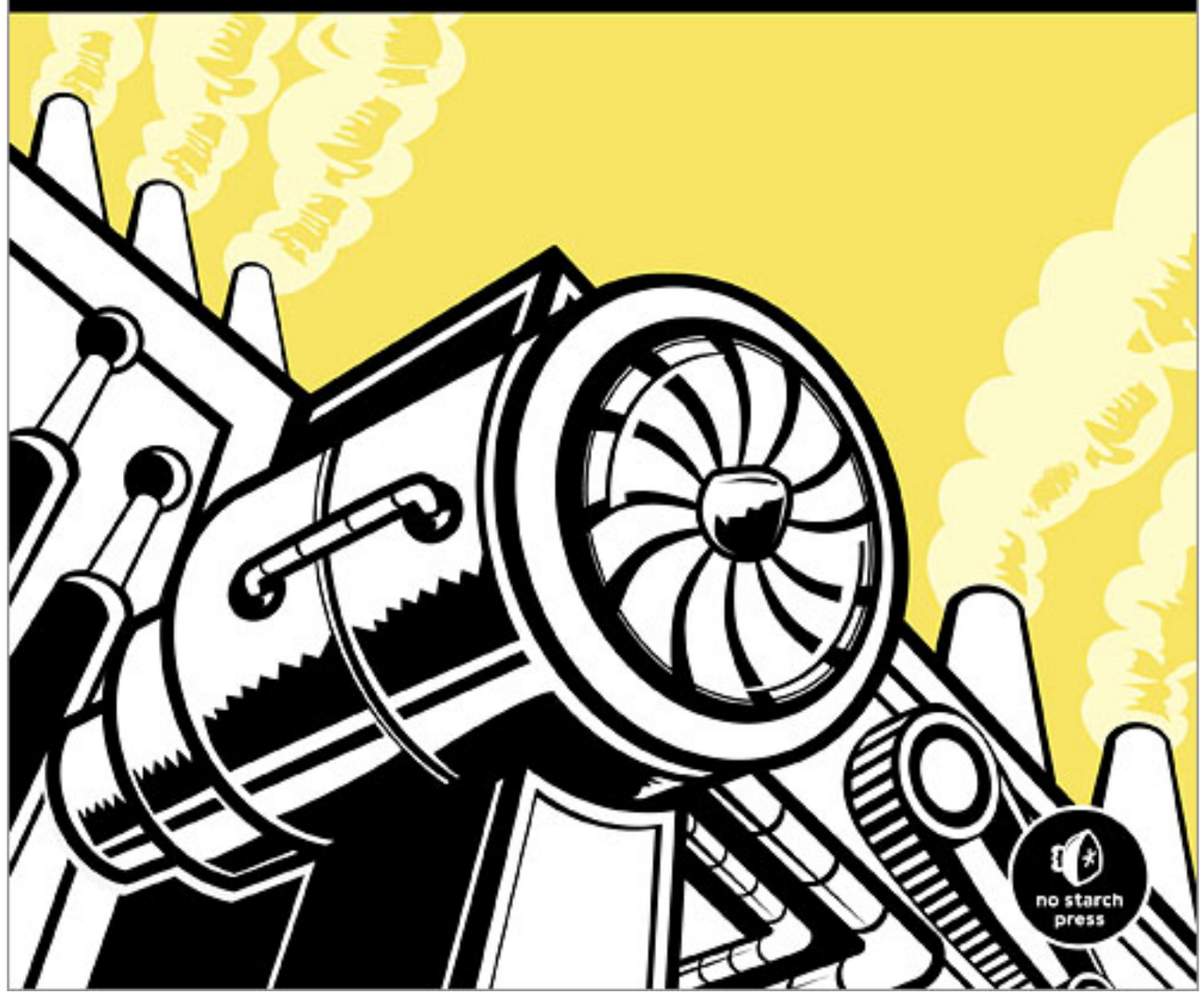
serious #**js**

to **infinity** and beyond!

plaatje: http://www.disneypictures.tk/disney-pictures/buzz-lightyear-pictures.html

THE PRINCIPLES OF OBJECT-ORIENTED JAVASCRIPT

NICHOLAS C. ZAKAS



constructor

a constructor is a function
to create an new object

```
var person1 = new Person();
console.log(person1 instanceof Person);              // true
console.log(person1.constructor === Person);         // true
```

#JS - CONSTRUCTOR

code

```
function Person(name) {
   Object.defineProperty(this, "name", {
        get: function() {
            return name;
        },
        set: function(newName) {
            name = newName;
        },
        enumerable: true,
        configurable: true
    });

    this.sayName = function() {
        console.log(this.name);
    };

}
```

```
var person1 = Person(“Nicholas");          // note: missing "new"
console.log(person1 instanceof Person);    // false
console.log(typeof person1);               // "undefined"
console.log(name);                         // "Nicholas"
```

assignment

Maak een **transport** object met daarin de properties: ‘owner’, ‘weigth’.

Maak een **car** object met daarin de properties: ‘owner’, ‘weigth’ en ‘licenseplate’.

Maak een **Ship** object met daarin de properties: ‘owner’, ‘weigth’ en ‘cargo’.

prototype

a prototype is a recipe for
an object

```
var book = {
   title: "The Principles of Object-Oriented JavaScript"
};

console.log("title" in book);                                    //true
console.log(book.hasOwnProperty(“title"));                       //true
console.log("hasOwnProperty" in book);                           //true
console.log(book.hasOwnProperty(“hasOwnProperty”));              //false
console.log(Object.prototype.hasOwnProperty("hasOwnProperty"));//true
```

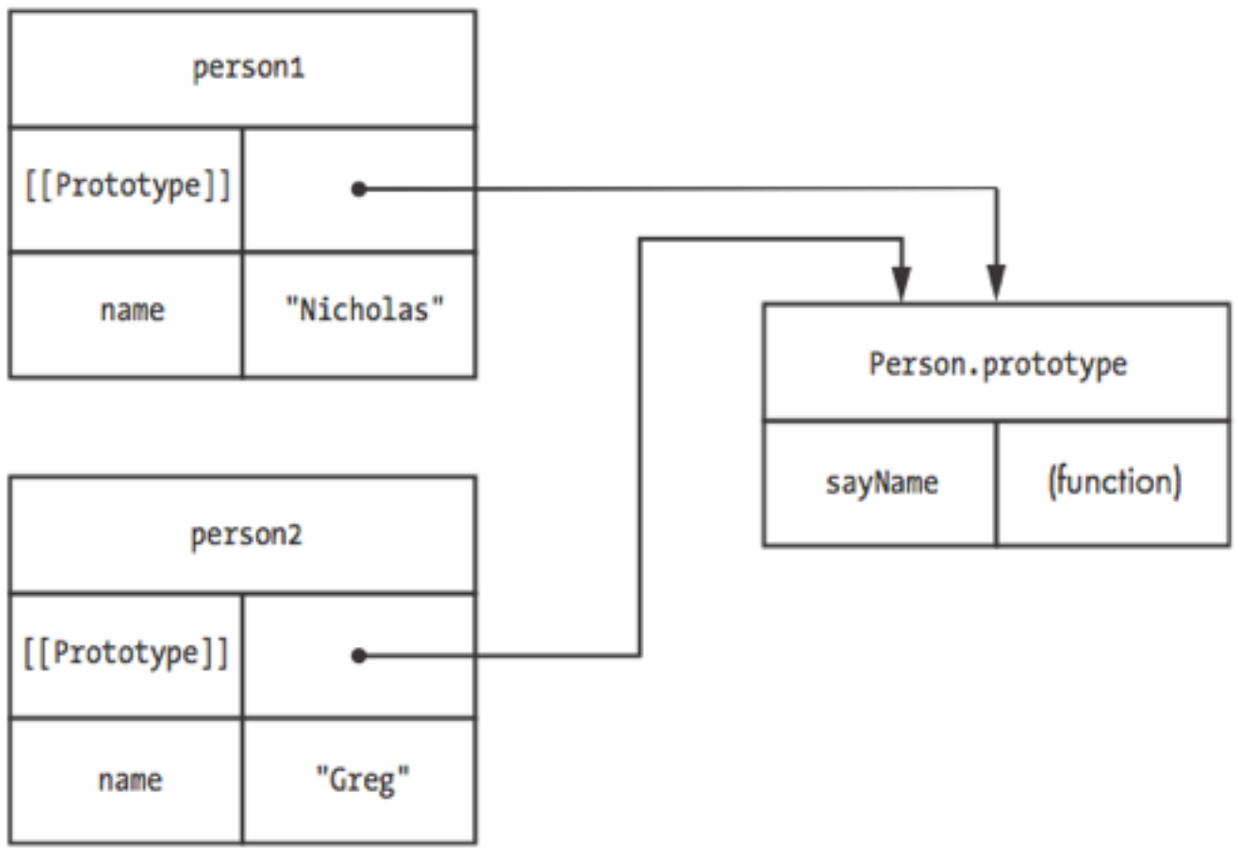


Figure 4-1: The [[Prototype]] properties for person1 and person2 point to the same prototype.

Prototype

assignment

Voeg aan het **car** en **ship** object een functie ‘horn’ toe die bij een **car** ‘beep en beep’ en bij **ship** ‘Whaaaaaamp’ zegt.

```
var object = {};

console.log(object.toString()); // "[object Object]"

object.toString = function() {
       return "[object Custom]";
};

console.log(object.toString()); // "[object Custom]"

// delete own property
delete object.toString;

console.log(object.toString()); // "[object Object]"

// no effect - delete only works on own properties

delete object.toString;
console.log(object.toString());    // "[object Object]"
```

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

Person.prototype.sayName = function() {
    console.log(this.name);
};


Person.prototype.favorites = [];

var person1 = new Person("Nicholas");

var person2 = new Person("Greg");

person1.favorites.push("pizza");
person2.favorites.push("quinoa");
console.log(person1.favorites);     // "pizza,quinoa"
console.log(person2.favorites);     // "pizza,quinoa"
```

```
function Person(name) {
       this.name = name;
}
Person.prototype = {
   sayName: function() {
      console.log(this.name);
   },
   toString: function() {
      return "[Person " + this.name + "]";
   }
}
var person1 = new Person("Nicholas");

console.log(person1 instanceof Person);          // true
console.log(person1.constructor === Person);     // false
console.log(person1.constructor === Object);     // true
```

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

Person.prototype = {
   constructor: Person,
      sayName: function() {
         console.log(this.name);
   },
      toString: function() {
         return "[Person " + this.name + "]";
      }
};
var person1 = new Person("Nicholas");
var person2 = new Person("Greg");

console.log(person1 instanceof Person); // true
console.log(person1.constructor === Person); // true
console.log(person1.constructor === Object); // false
console.log(person2 instanceof Person); // true
console.log(person2.constructor === Person); // true
console.log(person2.constructor === Object); // false
```

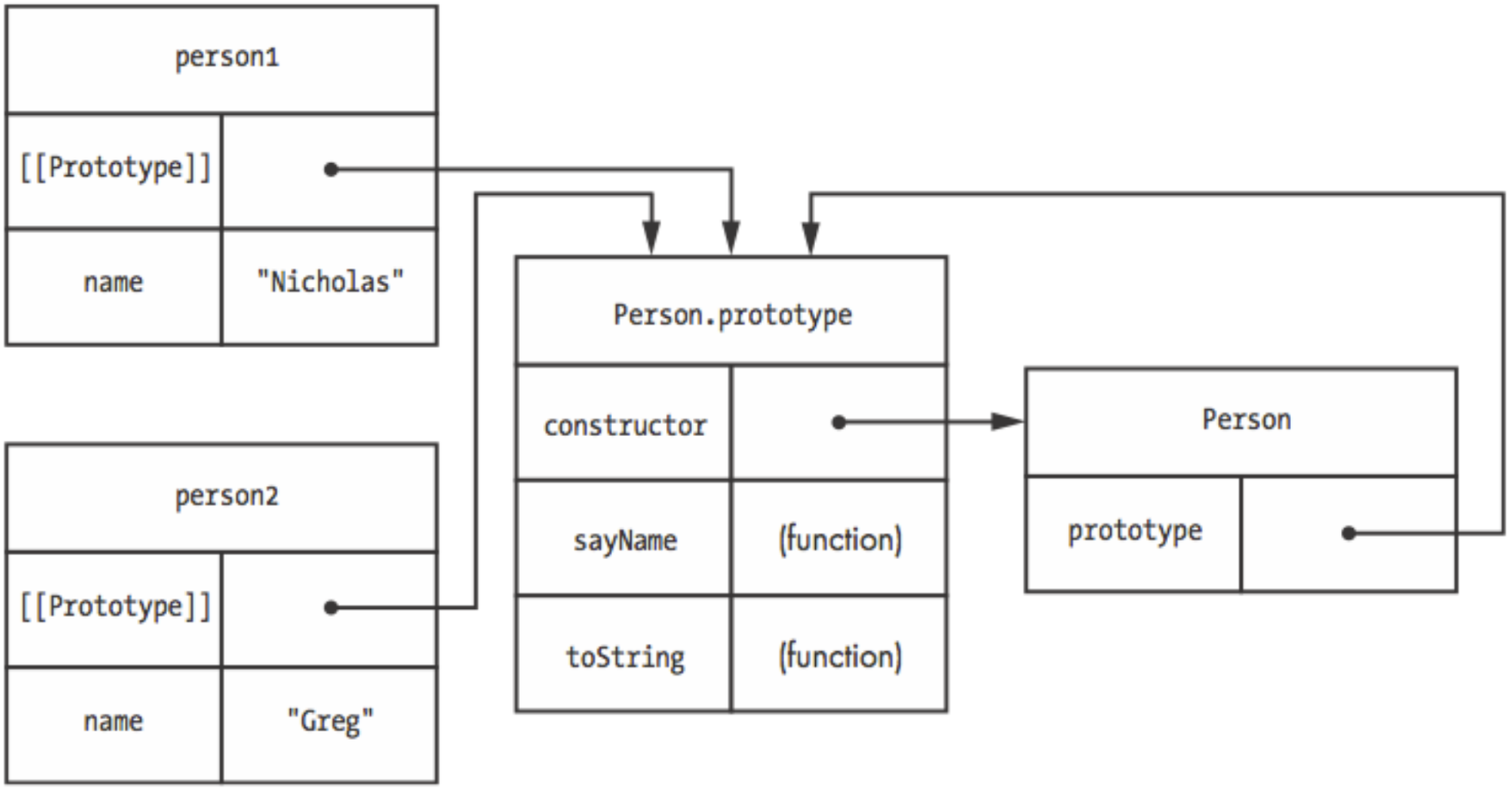


Figure 4-3: An instance and its constructor are linked via the prototype.

Prototype

inheritance

Inheritance in JS is accomplished through the prototype. An object inherits via its prototype. The prototype inherits via its own prototype. And so on

Methods Inherited from Object.prototype

hasOwnProperty() Determines whether an own property with the given name exists

propertyIsEnumerable() Determines whether an own property is enumerable

isPrototypeOf() Determines whether the object is the prototype of another

valueOf() Returns the value representation of the object **toString()** Returns a string representation of the object

```
var book = {
        title: "The Principles of Object-Oriented JavaScript",
        toString: function() {
            return "[Book " + this.title + "]"
        }
};
var message = "Book = " + book;

// "Book = [Book The Principles of Object-Oriented JavaScript]"
console.log(message);
```

```
var book = {
    title: "The Principles of Object-Oriented JavaScript"
};

// is the same as
var book = Object.create(Object.prototype, {
              title: {
                 configurable: true,
                 enumerable: true,
                 value: "The Principles of Object-Oriented JavaScript",
                 writable: true
              }
});
```

```
var person1 = {
    name: "Nicholas",
    sayName: function() {
        console.log(this.name);
    }
};

var person2 = Object.create(person1, {
    name: {
        configurable: true,
        enumerable: true,
        value: "Greg",
        writable: true
} });

person1.sayName(); // outputs "Nicholas"
person2.sayName(); // outputs "Greg"
```

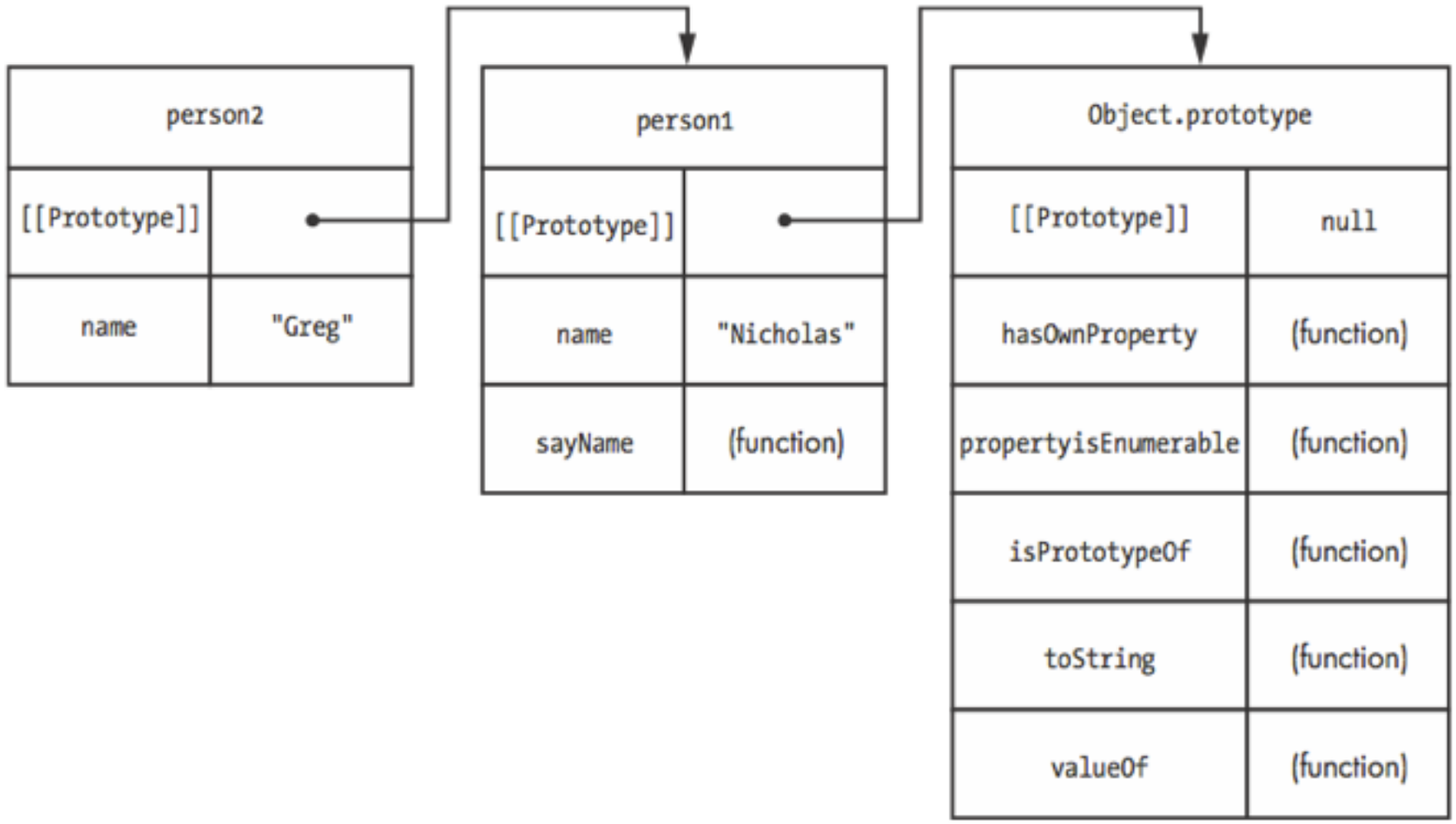


Figure 5-1: The prototype chain for `person2` includes `person1` and `Object.prototype`.

Inheritance

```
function Rectangle(length, width) {
   this.length = length;
   this.width = width;
}

Rectangle.prototype.getArea = function() {
   return this.length * this.width;
};

Rectangle.prototype.toString = function() {
   return "[Rectangle " + this.length + "x" + this.width + "]";
};
```

#JS - INHERITANCE

code

```
function Square(size) {
   this.length = size;
   this.width = size;
}

Square.prototype = new Rectangle();
Square.prototype.constructor = Square;

Square.prototype.toString = function() {
   return "[Square " + this.length + "x" + this.width + "]";
};
```

```
var rect = new Rectangle(5, 10);
var square = new Square(6);

console.log(rect.getArea());     // 50
console.log(square.getArea());   // 36
console.log(rect.toString());    // "[Rectangle 5x10]"
console.log(square.toString());  // "[Square 6x6]"

console.log(rect instanceof Rectangle);   // true
console.log(rect instanceof Object);      // true
console.log(square instanceof Square);    // true
console.log(square instanceof Rectangle); // true
console.log(square instanceof Object);    // true
```

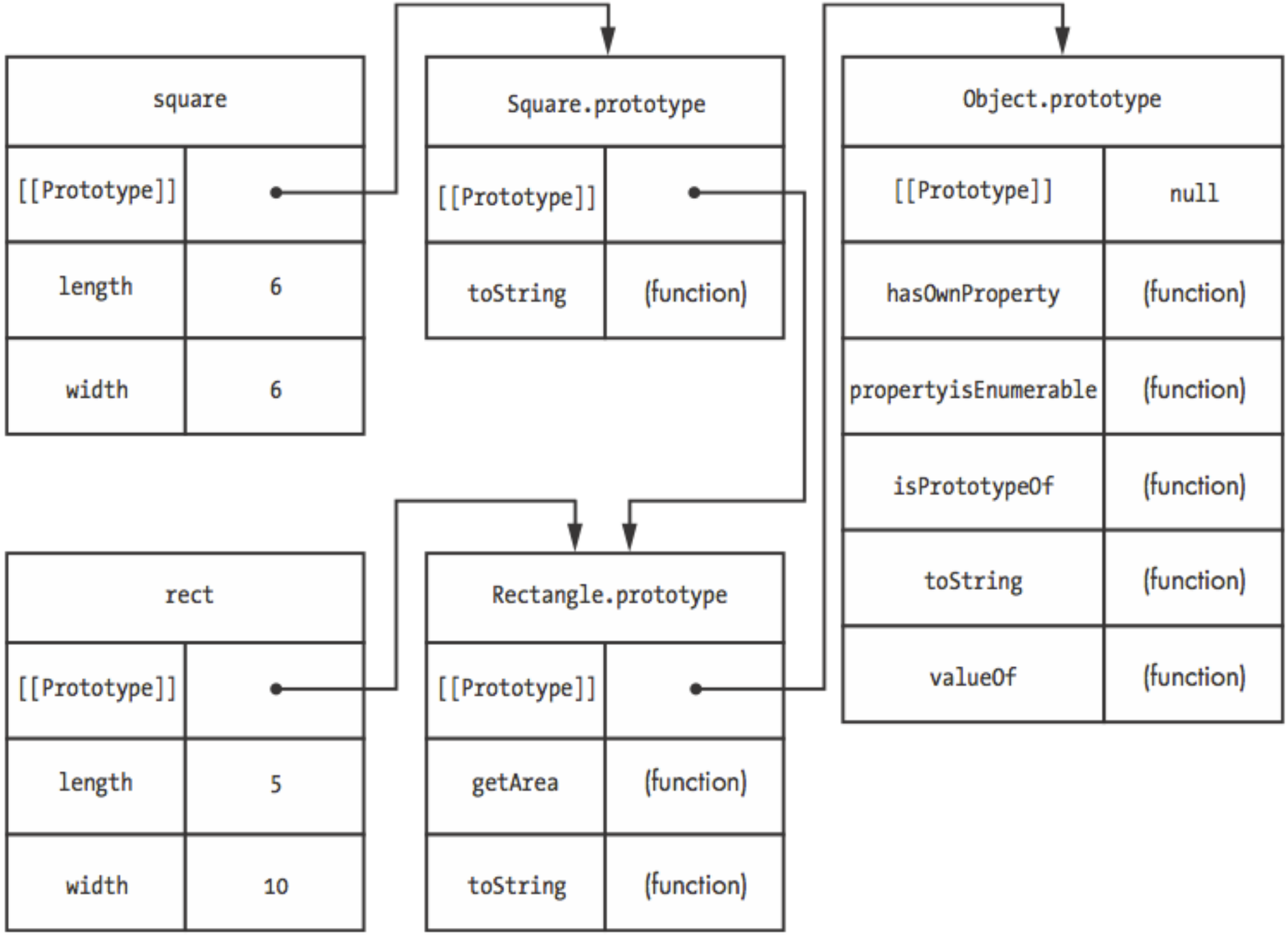


Figure 5-2: The prototype chains for square and rect show that both inherit from Rectangle.prototype and Object.prototype, but only square inherits from Square.prototype.

Inheritance

assignment

Zorg voor een overervingsrelatie tussen **car, ship** en **transport.**

Probeer de functie horn bij het object **transport** onder te brengen.

Zorg er voor dat in de constructor van **car** en **ship**, de constructor van **tranport** wordt aangeroepen zodat de overeenkomstige properties via **transport** worden opgeslagen.

```
function Rectangle(length, width) {
   this.length = length;
   this.width = width;
}


Rectangle.prototype.getArea = function() {
   return this.length * this.width;
};

Rectangle.prototype.toString = function() {
   return "[Rectangle " + this.length + "x" + this.height + "]";
};
```

#JS - INHERITANCE

code

```
// inherits from Rectangle
function Square(size) {
   Rectangle.call(this, size, size);
}

Square.prototype = Object.create(Rectangle.prototype, {
                              constructor: {
                                 configurable: true,
                                 enumerable: true,
                                 value: Square,
                                 writable: true
   }

});

// call the supertype method
Square.prototype.toString = function() {
   var text = Rectangle.prototype.toString.call(this);
   return text.replace("Rectangle", “Square”);

};
```

#JS - INHERITANCE

code

huiswerk

Maak je todo app zoveel mogelijk OO voor volgende week af. Gebruik de volgende objecten: **TodoList**, **TodoListItem.**

Zie voor meer uitdagingen het huiswerk op github.