Properties and Descriptor

Outline

- Types of property
- Property Descriptor
- Property Attributes (data property)
- Define / Modify Property
- Getters and Setters (accessor property)

Types of Property

JavaScript objects have two types of properties.

1. Data Properties
2. Accessor Properties

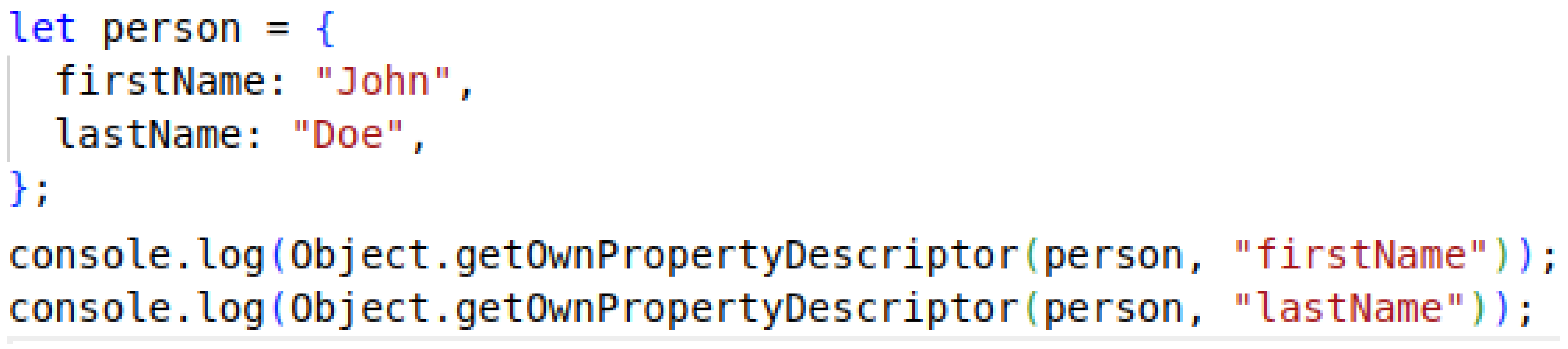
Property Descriptor

- Each property of an object has property descriptor which describes the nature of a property.
- Property descriptor for a particular object's property can be retrieved using Object.getOwnPropertyDescriptor() method.
- **Syntax:**

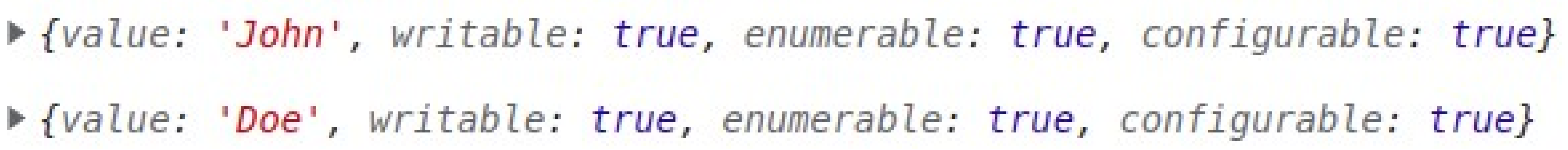
```
Object.getOwnPropertyDescriptor(object, 'property name')
```

- The getOwnPropertyDescriptor method returns a property descriptor for a property that directly defined in the specified object but not inherited from object's prototytpe.

Property Descriptor : Example

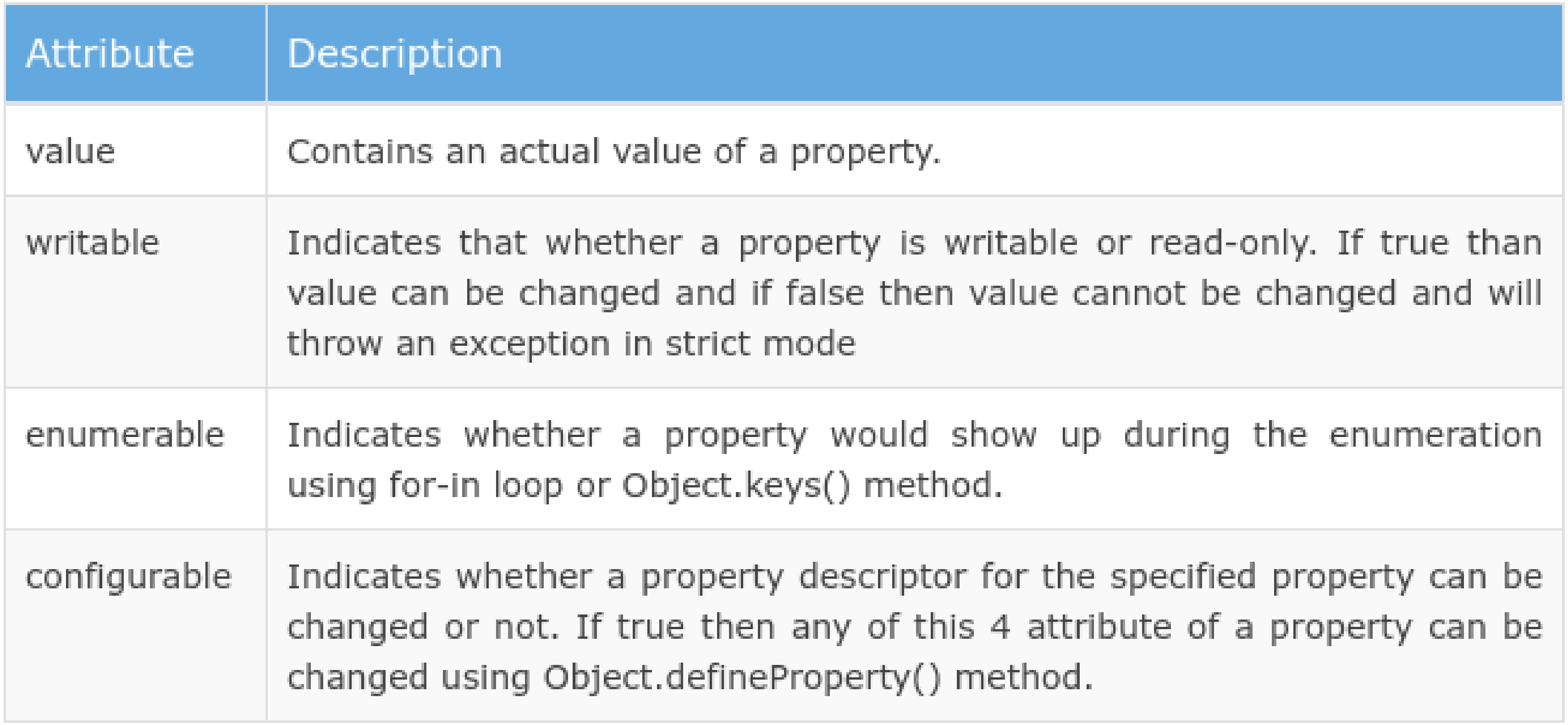


```
let person = {
  firstName: "John",
  lastName: "Doe",
};
console.log(Object.getOwnPropertyDescriptor(person, "firstName"));
console.log(Object.getOwnPropertyDescriptor(person, "lastName"));
```



```
▶ {value: 'John', writable: true, enumerable: true, configurable: true}
▶ {value: 'Doe', writable: true, enumerable: true, configurable: true}
```

Data Property Attributes



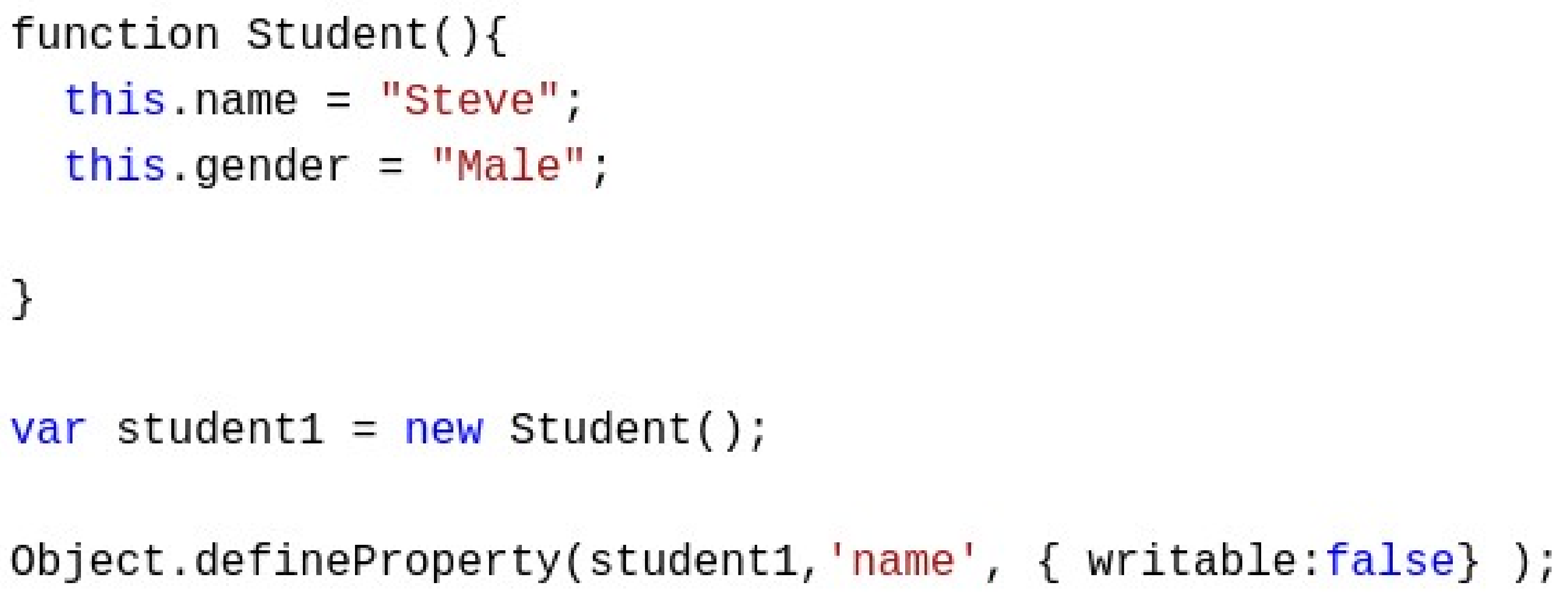
Attribute	Description
value	Contains an actual value of a property.
writable	Indicates that whether a property is writable or read-only. If true than value can be changed and if false then value cannot be changed and will throw an exception in strict mode
enumerable	Indicates whether a property would show up during the enumeration using for-in loop or Object.keys() method.
configurable	Indicates whether a property descriptor for the specified property can be changed or not. If true then any of this 4 attribute of a property can be changed using Object.defineProperty() method.

- The three attributes (writable, enumerable, and configurable) are all optional and all default to true.
- These will be false by default when we use Object.defineProperty() method

Define Property

- This method allows you to define a new property on an object
- It can also be used to change the descriptor of an existing property
- Syntax:

Object.defineProperty(object, 'property name', descriptor)



```
function Student(){
  this.name = "Steve";
  this.gender = "Male";

}

var student1 = new Student();

Object.defineProperty(student1,'name', { writable:false} );
```

Object.defineProperty() (Ex.1.0)

```
user = { };
Object.defineProperty(user, 'name', {
  value: 'John',
  writable: false,
  enumerable: true,
  configurable: true
})
console.log(user.name); // 'John'
user.name = 'Jack’;    // Exception if in ‘strict’ mode
console.log(user.name); // 'John'
```

Object.defineProperty() (Ex.1.1)

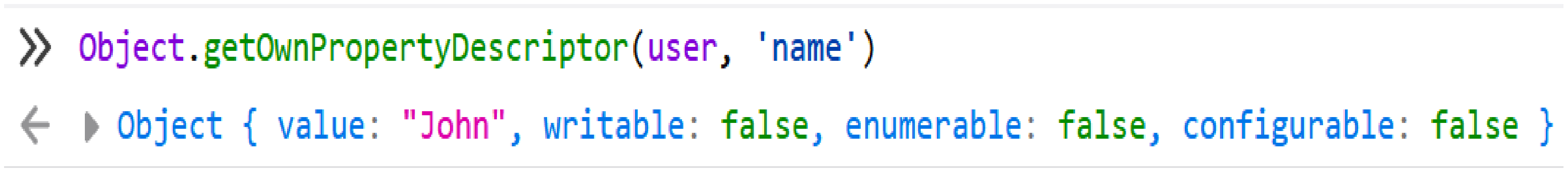
```
Object.defineProperty(user, 'name', {
  writable: true,
  configurable: false
});

user.name = 'Jack'
console.log(user.name);  // "Jack"

delete user.name         // Won’t work
```

Object.defineProperties()

```
user = {}
Object.defineProperties(user, {
  "name" :   { value: "John" },
  "gender" : { value: "male" }
})
```



```
>> Object.getOwnPropertyDescriptor(user, 'name')
← ▸ Object { value: "John", writable: false, enumerable: false, configurable: false }
```

```
console.log(name);  // "John"
user.name = "Jack"; // Won't work
console.log(name);  // "John"
```

Accessor Property

- Similar to data properties, accessor properties also have **Configurable** and **Enumerable** attributes.
- But the accessor properties have the **Get** and **Set** attributes instead of Value and Writable.
- When you read data from an accessor property, the **Get** function is called automatically to return a value. The default return value of the Get function is undefined.
- If you assign a value to an accessor property, the **Set** function is called automatically.

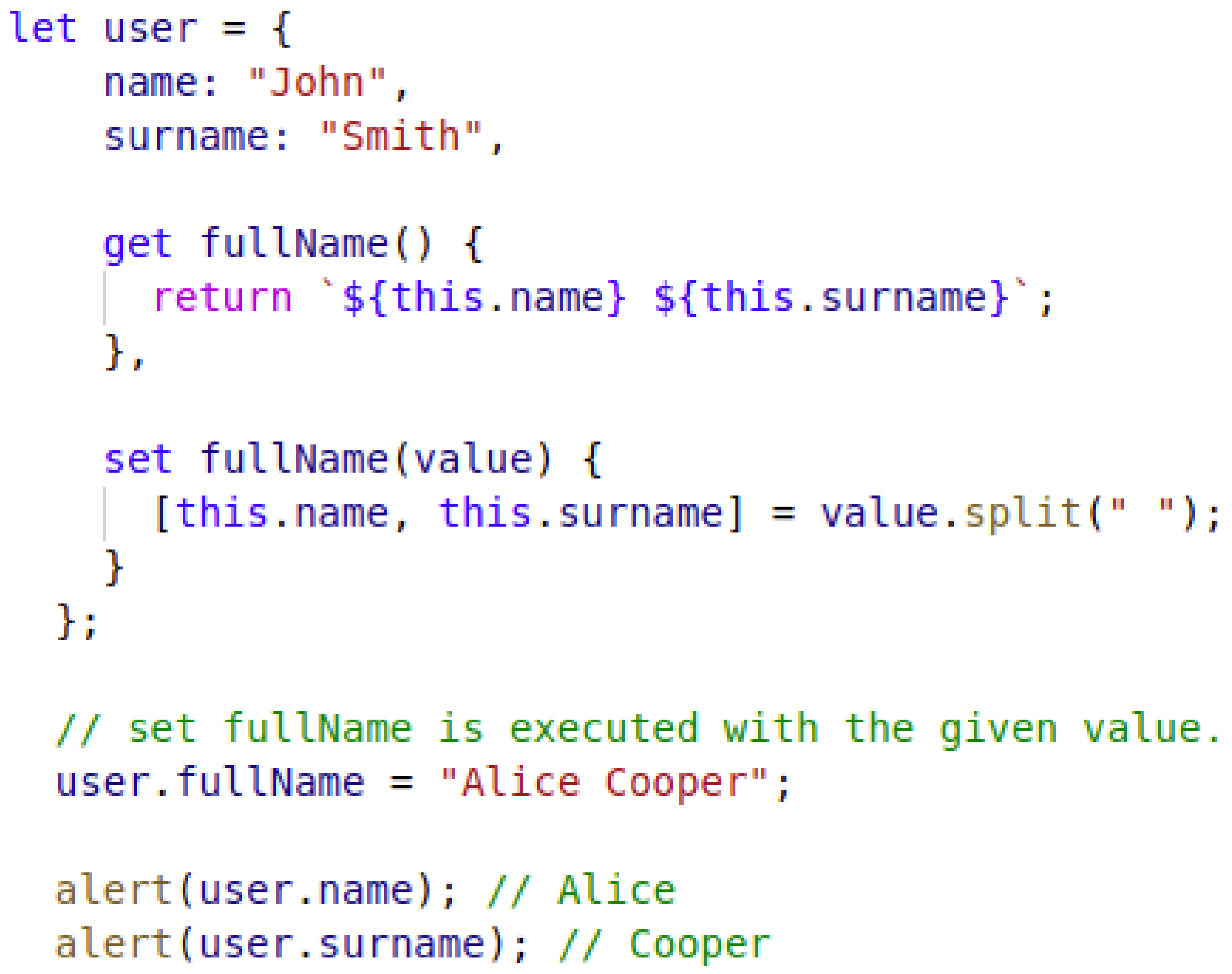
Defining Getters and Setters

- A getter is a method that gets the value of a specific property
- A setter is a method that sets the value of a specific property
- You can define getters and setters on any predefined core object or user-defined object that supports the addition of new properties.
- Getters and setters can be either
  - defined using object initializers, or
  - added later to any object at any time using a getter or setter adding method

Defining Getters and Setters

```
var user = {
  firstName: 'John',
  get fName() {
    return this.firstName;
  },
  set fName(fName) {
    this.firstName = fName.toUpperCase();
  }
}
user.fName = 'jack';
user.fName;  // JACK
```

Defining Getters and Setters



```
let user = {
    name: "John",
    surname: "Smith",

    get fullName() {
      return `${this.name} ${this.surname}`;
    },

    set fullName(value) {
      [this.name, this.surname] = value.split(" ");
    }
  };

  // set fullName is executed with the given value.
  user.fullName = "Alice Cooper";

  alert(user.name); // Alice
  alert(user.surname); // Cooper
```

Object.defineProperty()

```
var obj = { counter: 0 };

Object.defineProperty(obj, "reset", {
   get: function () { this.counter = 0; }
});
Object.defineProperty(obj, "increment", {
  get: function () { this.counter++; }
});
Object.defineProperty(obj, "decrement", {
  get: function () { this.counter--; }
});
```

Object.defineProperty()

```
Object.defineProperty(obj, "add", {
  set: function (value) { this.counter += value;
 }
});
Object.defineProperty(obj, "subtract", {
  set: function (value) { this.counter -
= value; }
});

obj.reset;   // counter = 0
obj.add = 10;  // counter = 10
obj.subtract = 3;// counter = 7
obj.increment; // counter = 8
obj.decrement; // counter = 7
```

Properties with Getters and Setters

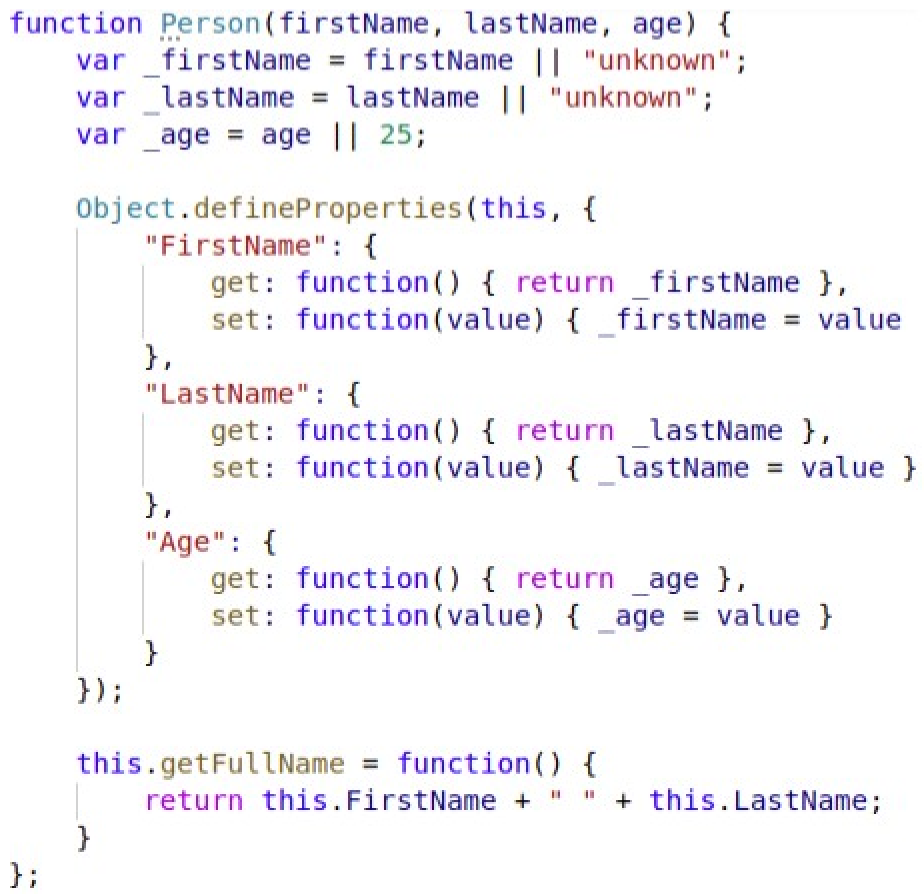
```
function Person() {
    var _firstName = "unknown";

    Object.defineProperties(this, {
        "FirstName": {
            get: function () {
                return _firstName;
            },
            set: function (value) {
                _firstName = value;
            }
        }
    });
};
```

```
var person1 = new Person();
person1.FirstName = "Steve";
alert(person1.FirstName );

var person2 = new Person();
person2.FirstName = "Bill";
alert(person2.FirstName );
```

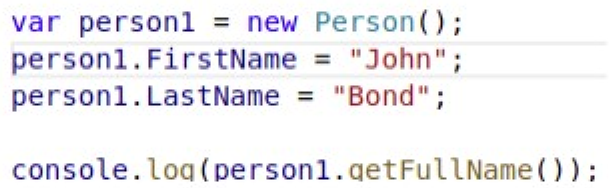
Multiple Properties



```
function Person(firstName, lastName, age) {
    var _firstName = firstName || "unknown";
    var _lastName = lastName || "unknown";
    var _age = age || 25;

    Object.defineProperties(this, {
        "FirstName": {
            get: function() { return _firstName },
            set: function(value) { _firstName = value
        },
        "LastName": {
            get: function() { return _lastName },
            set: function(value) { _lastName = value }
        },
        "Age": {
            get: function() { return _age },
            set: function(value) { _age = value }
        }
    });

    this.getFullName = function() {
        return this.FirstName + " " + this.LastName;
    }
};
```

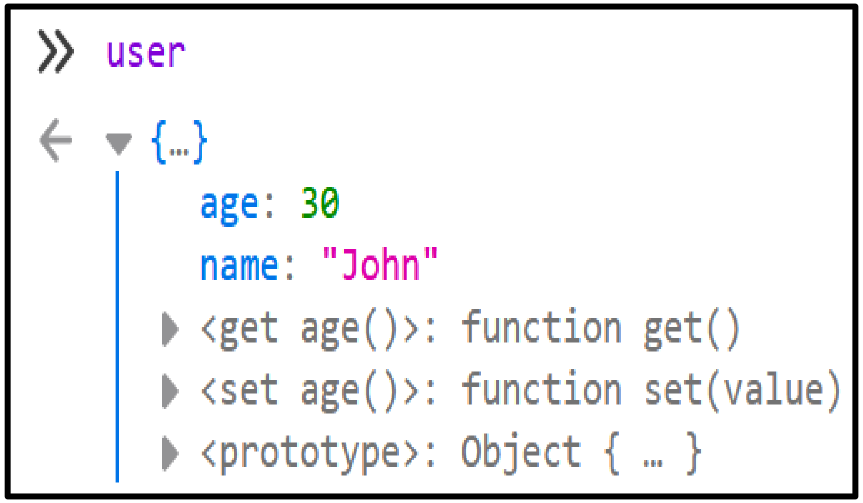


```
var person1 = new Person();
person1.FirstName = "John";
person1.LastName = "Bond";

console.log(person1.getFullName());
```

Object.defineProperty() (Ex.2)

```
user = { name: 'John' };
var age = 30;
Object.defineProperty(user, 'age', {
  get: () => age,
  set: value => { age = value }
});
```



```
» user
← ▾ {…}
      age: 30
      name: "John"
    ▶ <get age()>: function get()
    ▶ <set age()>: function set(value)
    ▶ <prototype>: Object { … }
```

References

- https://www.tutorialsteacher.com/javascript