Programmation Web – Avancé

JavaScript & Node.js

L'orienté objet en JS

Attribution – Partage dans les Mêmes Conditions 4.0 International (CC BY-SA 4.0)

Presentation template by SlidesCarnival

L'orienté objet en JS

J'adore les classes... Puis-je en créer ?

Les objets en JS

- Prototype-based language (VS class-based, comme Java)
- Les chaînes, tableaux, APIs du browser, objets personnalisés...
- Pas de distinction entre une classe et une instance

Les objets en JS

- Tout objet peut être le prototype d'un autre objet, permettant à l'autre objet de partager les propriétés du 1er objet
- Détails [49.]

Création d'un objet

- Via un « object literal » = liste de paires nom de propriété / valeur

```
let raphael = {
  firstname: "Raphael",
  lastname: "Baroni",
  sayHello: () => "Hi everyone !",
};
```

Création d'un objet

- Via {} ou new

```
let sandra = {};
sandra.firstname = "Sandra";
sandra.lastname = "Parisi";
```

Accéder aux propriétés d'un objet

- Soit via un point

```
console.log(raphael.firstname, " :" , raphael.sayHello());
// Raphael  : Hi everyone !
```

- soit via des ["object_key"]

```
console.log(sandra["firstname"], ",", sandra["lastname"]);
// Sandra , Parisi
```

Création de “classes” en JS

1. Via **class** et **constructor()**
2. Via une fonction constructeur
3. Via une fonction normale

Création d'une "classe" de manière moderne

- Via **class** et **constructor()**
- Nouveau depuis ES6
- **constructor()** pas supporté par IE11 !
- 1 seul constructeur possible sinon erreur

Création d'une "classe" de manière moderne

```
class Car {
  constructor(brand, model) {
    this.brand = brand;
    this.model = model;
    this.id = Math.random();
  }
  getDescription() {
    return "Car's description : " + this.brand + ", " + this.model +
      " , ID:" + this.id
    );
  }
}
```

Création d'une "classe" via une fonction constructeur

```
function Auto(brand, model) {
  this.brand = brand;
  this.model = model;
  this.id = Math.random();
}

Auto.prototype.getDescription = function () {
  return (
    "Car's description : " + this.brand + ", " + this.model + " , ID:" +
     this.id
  );
};
```

Création d'une "classe" via une fonction normale

```
function AutoNotRecommended(brand, model) {
  let obj = {};
  obj.brand = brand;
  obj.model = model;
  obj.id = Math.random();
  obj.getDescription = function () {
    return (
      "Car's description : " + this.brand + ", " + this.model + " , ID:"
      + this.id
    );
  };
  return obj;
}
```

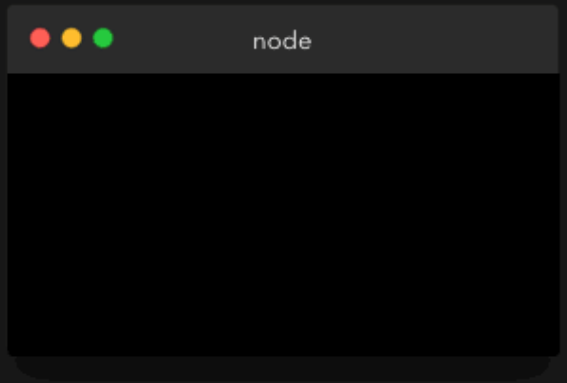
Création d'une instance et accès à ses propriétés

```
let dacia = new Car("Dacia", "Sandero");
let audi = new Auto("Audi", "A4");
let lada = AutoNotRecommended("Lada", "XRAY");
// let lada = new AutoNotRecommended("Lada", "XRAY"); // also works
```

Héritage par prototype en JS

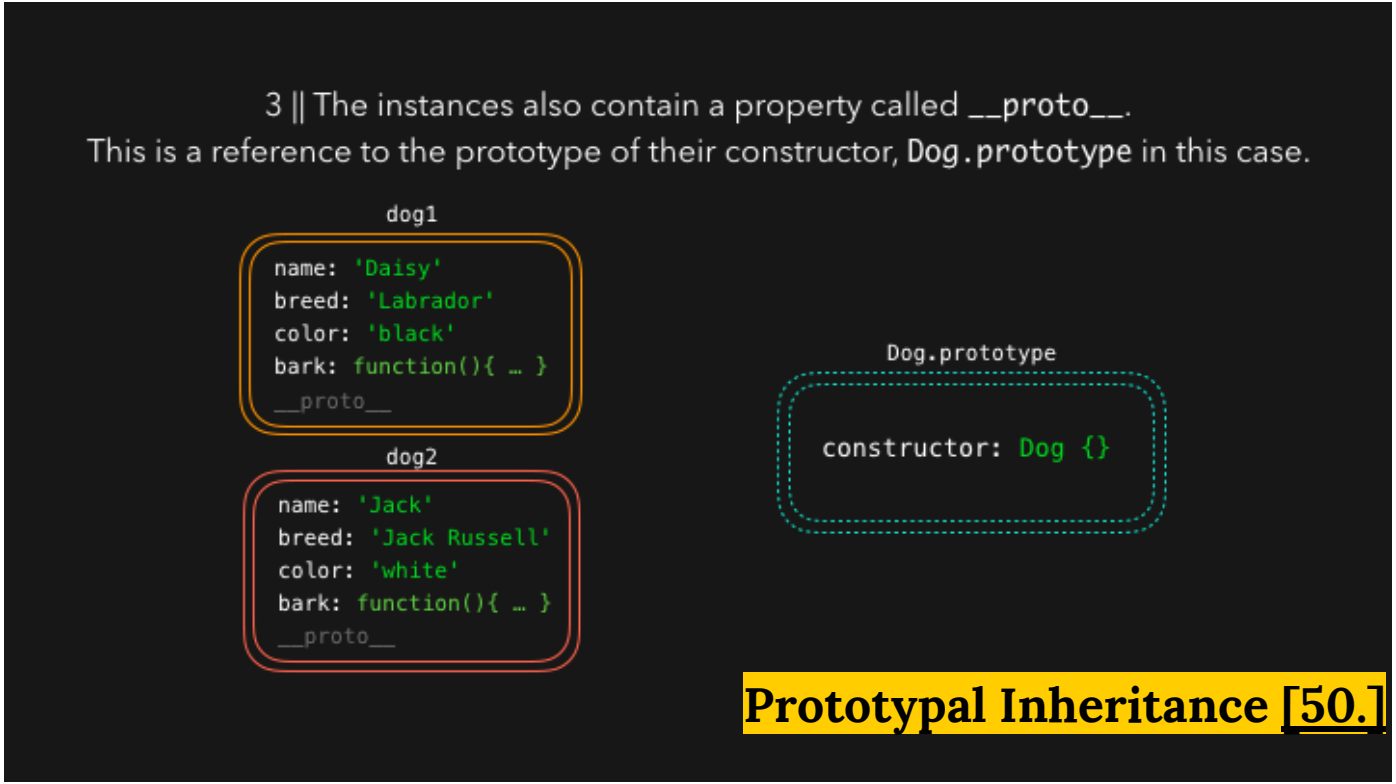
1 || When we create a constructor function, a **prototype** object gets created as well. The constructor's prototype has a reference to the original constructor function.

```
function Dog(name, breed, color) {
  this.name = name
  this.breed = breed
  this.color = color
  this.bark = function() {
    return 'Woof!'
  }
}
```



Prototypal Inheritance [50.]

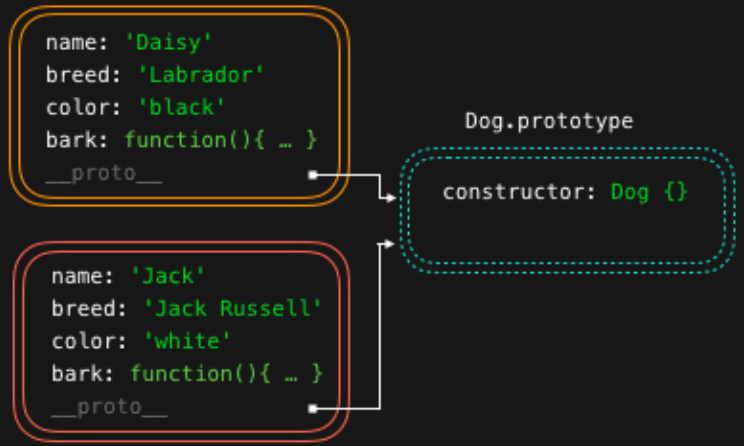
Héritage par prototype en JS



Héritage par prototype en JS

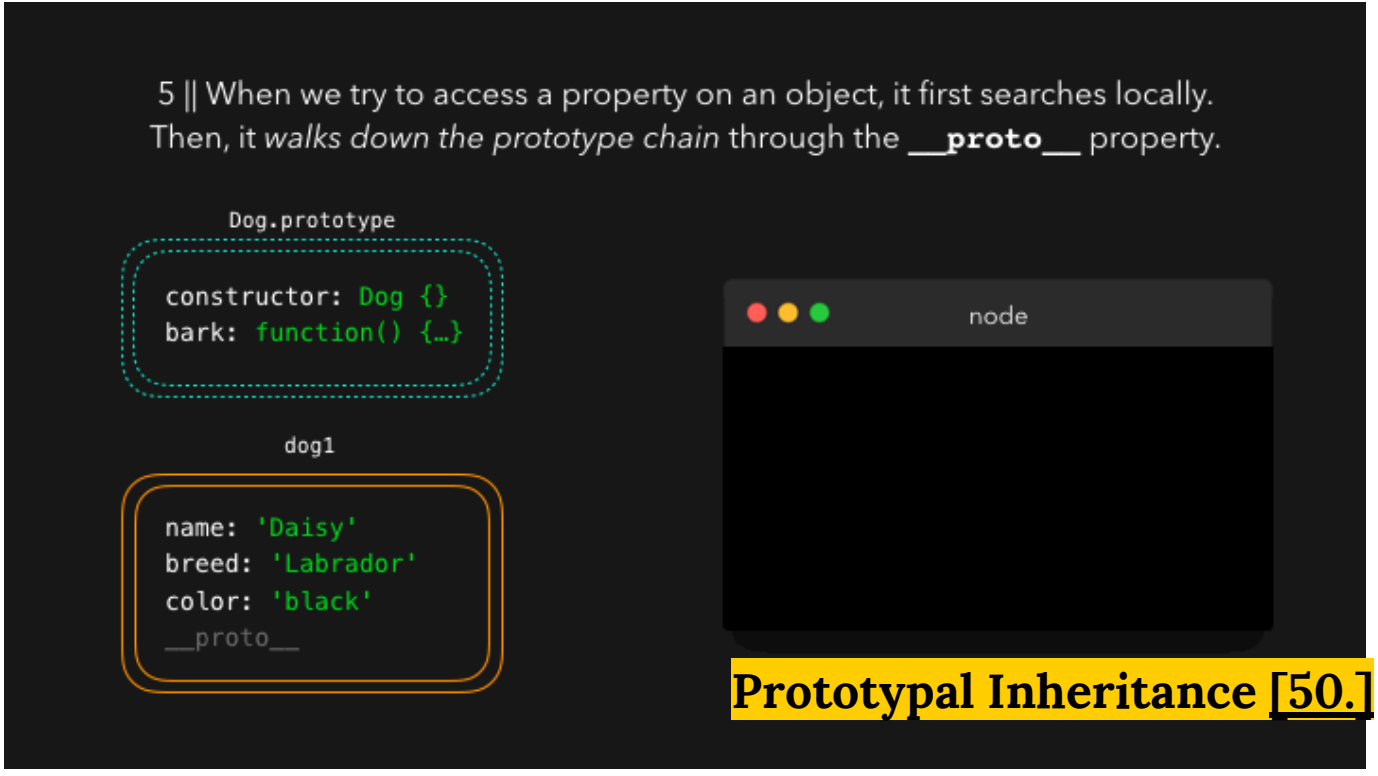
4 || We can save memory by adding properties to the prototype that all instances can share, instead of creating new copies of that property each time.

```
function Dog(name, breed, color) {
  this.name = name
  this.breed = breed
  this.color = color
  this.bark = function() {
    return 'Woof!'
  }
}
```



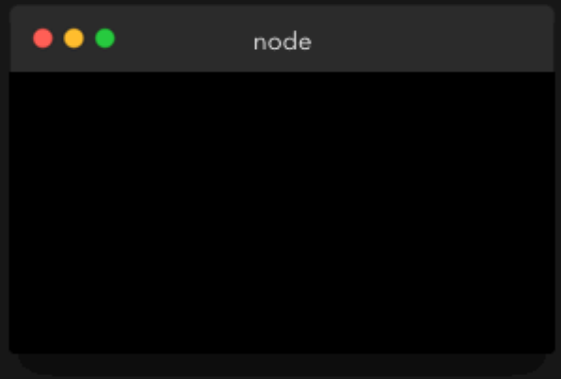
Prototypal Inheritance [50.]

Héritage par prototype en JS



Héritage par prototype en JS

6 || The prototype chain can have several steps. For example, `Dog.prototype` itself is an object, thus inherits properties from the built-in `Object.prototype`



Prototypal Inheritance [50.]

Héritage par prototype en JS

7 || ES6 introduced classes, which is syntactical sugar for constructor functions.

```
function Dog(name, breed, color) {
  this.name = name
  this.breed = breed
  this.color = color
}

Dog.prototype.bark = function() {
  return 'Woof!'
}
```

```
class Dog {
  constructor(name, breed, color) {
    this.name = name
    this.breed = breed
    this.color = color
  }

  bark() {
    return 'Woof!'
  }
}
```

Prototypal Inheritance [50.]

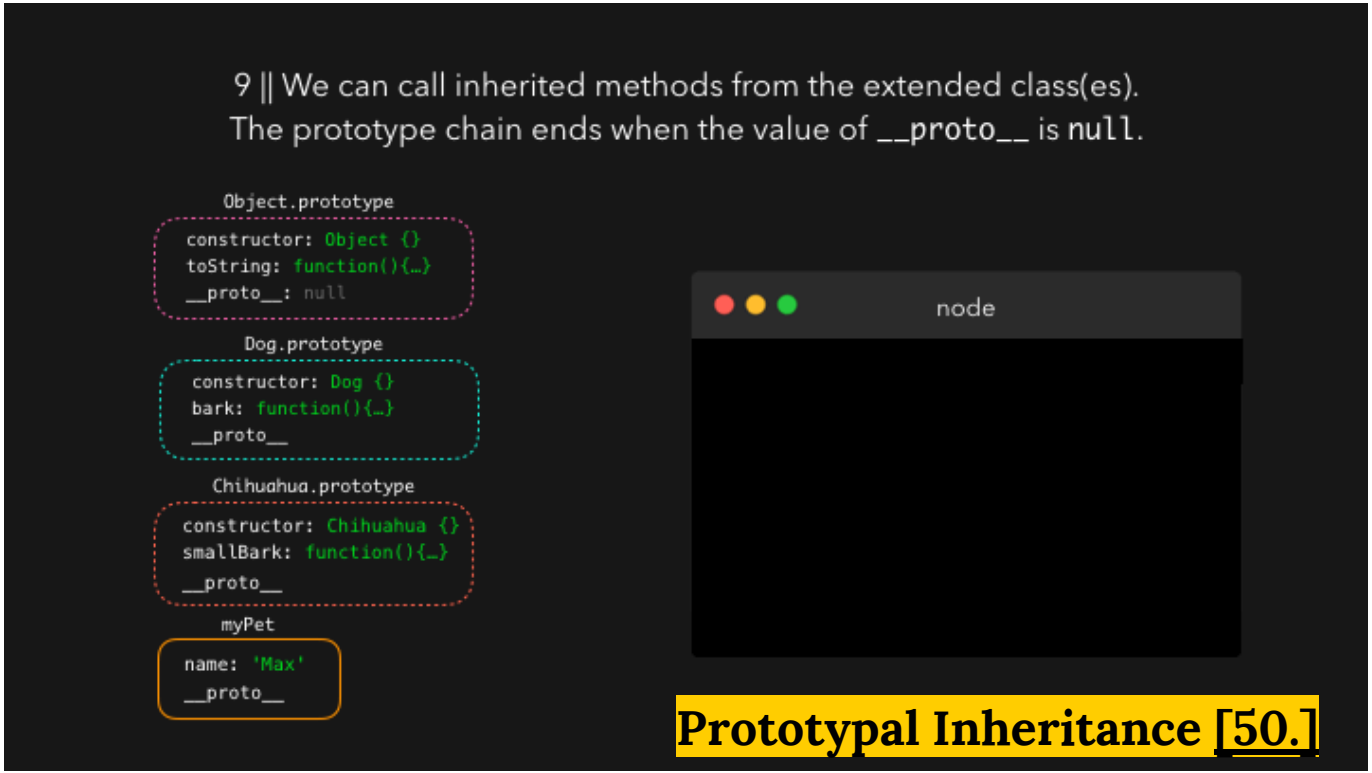
Héritage par prototype en JS

8 || Prototypal inheritance works the same way with classes as with ES5 constructors. With the **super** keyword, we can call the class that the sub-class extends.

```
class Dog {
  constructor(name) {
    this.name = name
  }

  bark() {
    return 'Woof!'
  }
}

class Chihuahua extends Dog {
  constructor(name) {
    super(name)
  }

  smallBark() {
    return 'Small woof!'
  }
}

const myPet = new Chihuahua("Max")
```

Prototypal Inheritance [50.]

Héritage par prototype en JS



DEMO : Programmation orienté objet en JS

- Création d'un objet représentant une personne spécifique, avec les différentes syntaxes.
- Création d'un modèle objet « Car » et d'une de ses instances.