



# Objets et chaînes de <prototypes> en JavaScript

Philippe Genoud

*Philippe.Genoud@univ-grenoble-alpes.fr*



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).

# <prototype> d'un objet

```
1 function Chat(nom,age,race,poids) {
2   this.nom = nom ;
3   this.age = age ;
4   this.race = race ;
5   this.poids = poids ;
6 }
7
8 Chat.prototype.miauler = function () {
9   console.log("Miaou ! Miaou !");
10 };
11
12 Chat.prototype.maignrir = function(deltaPoids) {
13   this.poids -= deltaPoids;
14 };
15
16 let felicite = new Chat("Félicité",10,"chat de gouttière", 3.5);
17
18 console.log(felicite);
19
```

définition du constructeur d'objets **Chat**

méthodes associées aux objets **Chat**

création d'un objet **Chat** (instance)

Sur la console du navigateur

L'objet référencé par **felicite** défini avec 4 propriétés: **nom, age, race, poids**

```
object
  Chat {nom: 'Félicité', age: 10, race: 'chat de gouttière', poids: 3.5}
```

[Voir le code](#)

Dans le débogueur du navigateur

Chrome DevTools: Niveau d'accès / Contrôle

- Suspendu sur point d'arrêt
- Script
  - felicite: Chat
    - age: 10
    - nom: "Félicité"
    - poids: 3.5
    - race: "chat de gouttière"
    - [[Prototype]]: Object
- Pile d'appel
  - (anonyme) protosObjetsCon...ructeur1.js:18
- Pile d'exécution
  - (global) protosObjetsConstructeur1.js:18

Firefox DevTools: Mis en pause par un point d'arrêt

- Expressions espionnes
- Portées
  - Bloc
    - <this>: Window
    - felicite: {...}
      - age: 10
      - nom: "Félicité"
      - poids: 3.5
      - race: "chat de gouttière"
      - <prototype>: {...}

En interne l'objet référencé par **felicite** possède une propriété supplémentaire que l'on peut observer à l'aide du débogueur : **[[Prototype]]** (Chrome ou Edge) ou **<prototype>** (Firefox)



L'attribut interne **<prototype>** ≠ attribut **prototype** d'une fonction constructeur

# Chaîne de <prototypes>

- Chaque objet possède un lien interne (masqué) nommé son **<prototype>** (Firefox) ou **[[Prototype]]** (Chrome, Edge, Node) qui le relie soit à un autre objet soit à **null**.

```
1 function Chat(nom,age,race,poids) {
2   this.nom = nom ;
3   this.age = age ;
4   this.race = race ;
5   this.poids = poids ;
6 }
7
8 Chat.prototype.miauler = function () {
9   console.log("Miaou ! Miaou !");
10 };
11
12 Chat.prototype.maignrir = function(deltaPoids) {
13   this.poids -= deltaPoids;
14 };
15
16 let felicite = new Chat("Félicité",10,"chat de gouttière");
17
18 console.log(felicite);
19
20
```

▼ Portées

- ▼ Bloc
  - ▶ <this>: Window
  - ▶ felicite: {\_-}
  - ▶ Window: Global

L'objet **felicite**

- ▼ felicite: {\_-}
- age: 10
- nom: "Félicité"
- poids: 3.5
- race: "chat de gouttière"
- ▶ <prototype>: {\_-}

<prototype> de l'objet **felicite**

- ▶ constructor:Chat()
- ▶ maigrir:Chat.prototype.maignrir()
- ▶ miauler:Chat.prototype.miauler()
- ▶ <prototype>: {\_-}

<prototype> du <prototype> de l'objet **felicite**

- ▶ \_\_defineGetter\_\_()
- ▶ \_\_defineSetter\_\_()
- ▶ \_\_lookupGetter\_\_()
- ▶ \_\_lookupSetter\_\_()
- ▶ \_\_proto\_\_: >>
- ▶ constructor:Object()
- ▶ hasOwnProperty()
- ▶ isPrototypeOf()
- ▶ propertyIsEnumerable()
- ▶ toLocaleString()
- ▶ toSource()
- ▶ toString()
- ▶ valueOf()

chaîne de <prototypes> de l'objet référencé par **felicite**

dernier objet de la chaîne son <prototype> est **null**

 [Voir le code](#)

# Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {}
```

<prototype> du <prototype> de l'objet **felicite**


```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité",10,  
17  
18 console.log(felicite);  
19  
20
```

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {}  
  ▶ Window: Global
```

lors de l'accès à une propriété (attribut) ou lors de l'exécution d'une méthode d'un objet, l'interpréteur JavaScript, s'il ne trouve pas la propriété ou la méthode dans l'objet lui-même, effectue une recherche de celle-ci en parcourant la chaîne de <prototypes>.

 [Voir le code](#)

# Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {_-}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {_-}
```

Prototype de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {_-}
```

Prototype du prototype de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
function Chat(nom,age,race,poids) {  
  this.nom = nom ;  
  this.age = age ;  
  this.race = race ;  
  this.poids = poids ;  
}  
  
Chat.prototype.miauler = function () {  
  console.log("Miaou ! Miaou !");  
};  
  
Chat.prototype.maigrir = function(deltaPoids) {  
  this.poids -= deltaPoids;  
};  
  
let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
console.log(felicite);
```

console.log(**felicite**.nom);

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {_-}  
  ▶ Window: Global
```

[Voir le code](#)

# Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {-}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {-}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {-}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
function Chat(nom,age,race,poids) {  
  this.nom = nom ;  
  this.age = age ;  
  this.race = race ;  
  this.poids = poids ;  
}  
  
Chat.prototype.miauler = function () {  
  console.log("Miaou ! Miaou !");  
};  
  
Chat.prototype.maigrir = function(deltaPoids) {  
  this.poids -= deltaPoids;  
};  
  
let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
console.log(felicite);
```

console.log(**felicite.nom**);

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {-}  
  ▶ Window: Global
```

 [Voir le code](#)

# Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {-}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {-}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {-}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
17  
18 console.log(felicite);  
19  
20
```

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {-}  
  ▶ Window: Global
```

```
console.log(felicite.nom);  
felicite.miauler();
```

[Voir le code](#)

# Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
felicitate: {  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  <prototype>: {  
    constructor: Chat()  
    maigrir: Chat.prototype.maigrir()  
    miauler: Chat.prototype.miauler()  
    <prototype>: {  
      __defineGetter__()  
      __defineSetter__()  
      __lookupGetter__()  
      __lookupSetter__()  
      __proto__: >>  
      constructor: Object()  
      hasOwnProperty()  
      isPrototypeOf()  
      propertyIsEnumerable()  
      toLocaleString()  
      toSource()  
      toString()  
      valueOf()  
    }  
  }  
}
```

<prototype> de l'objet **felicite**

```
constructor: Chat()  
maigrir: Chat.prototype.maigrir()  
miauler: Chat.prototype.miauler()  
<prototype>: {  
  __defineGetter__()  
  __defineSetter__()  
  __lookupGetter__()  
  __lookupSetter__()  
  __proto__: >>  
  constructor: Object()  
  hasOwnProperty()  
  isPrototypeOf()  
  propertyIsEnumerable()  
  toLocaleString()  
  toSource()  
  toString()  
  valueOf()  
}
```

<prototype> du <prototype> de l'objet **felicite**

```
__defineGetter__()  
__defineSetter__()  
__lookupGetter__()  
__lookupSetter__()  
__proto__: >>  
constructor: Object()  
hasOwnProperty()  
isPrototypeOf()  
propertyIsEnumerable()  
toLocaleString()  
toSource()  
toString()  
valueOf()
```

```
function Chat(nom,age,race,poids) {  
  this.nom = nom ;  
  this.age = age ;  
  this.race = race ;  
  this.poids = poids ;  
}  
  
Chat.prototype.miauler = function () {  
  console.log("Miaou ! Miaou !");  
};  
  
Chat.prototype.maigrir = function(deltaPoids) {  
  this.poids -= deltaPoids;  
};  
  
let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
console.log(felicite);
```

Portées

```
<this>: Window  
felicite: {  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  <prototype>: {  
    constructor: Chat()  
    maigrir: Chat.prototype.maigrir()  
    miauler: Chat.prototype.miauler()  
    <prototype>: {  
      __defineGetter__()  
      __defineSetter__()  
      __lookupGetter__()  
      __lookupSetter__()  
      __proto__: >>  
      constructor: Object()  
      hasOwnProperty()  
      isPrototypeOf()  
      propertyIsEnumerable()  
      toLocaleString()  
      toSource()  
      toString()  
      valueOf()  
    }  
  }  
}  
Window: Global
```

```
console.log(felicite.nom);  
felicite.miauler();
```

[Voir le code](#)



# Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {-}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  <prototype>: {-}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {-}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
▶ __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
17  
18 console.log(felicite);  
19  
20
```

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {-}  
  ▶ Window: Global
```

```
console.log(felicite.nom);  
felicite.miauler();
```

[Voir le code](#)

# Chaîne de prototypes

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
17  
18 console.log(felicite);  
19  
20
```

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {}  
  ▶ Window: Global
```

```
console.log(felicite.nom);  
felicite.miauler();  
console.log(felicite.toString());
```

[Voir le code](#)

# Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet `felicite`

```
felicitate: {  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  <prototype>: {  
    constructor: Chat()  
    maigrir: Chat.prototype.maigrir()  
    miauler: Chat.prototype.miauler()  
    <prototype>: {  
      __defineGetter__()  
      __defineSetter__()  
      __lookupGetter__()  
      __lookupSetter__()  
      __proto__: >>  
      constructor: Object()  
      hasOwnProperty()  
      isPrototypeOf()  
      propertyIsEnumerable()  
      toLocaleString()  
      toSource()  
      toString()  
      valueOf()  
    }  
  }  
}
```

<prototype> de l'objet `felicite`

```
constructor: Chat()  
maigrir: Chat.prototype.maigrir()  
miauler: Chat.prototype.miauler()  
<prototype>: {  
  <prototype>: {  
    __defineGetter__()  
    __defineSetter__()  
    __lookupGetter__()  
    __lookupSetter__()  
    __proto__: >>  
    constructor: Object()  
    hasOwnProperty()  
    isPrototypeOf()  
    propertyIsEnumerable()  
    toLocaleString()  
    toSource()  
    toString()  
    valueOf()  
  }  
}
```

<prototype> du <prototype> de l'objet `felicite`

```
__defineGetter__()  
__defineSetter__()  
__lookupGetter__()  
__lookupSetter__()  
__proto__: >>  
constructor: Object()  
hasOwnProperty()  
isPrototypeOf()  
propertyIsEnumerable()  
toLocaleString()  
toSource()  
toString()  
valueOf()
```

```
function Chat(nom,age,race,poids) {  
  this.nom = nom ;  
  this.age = age ;  
  this.race = race ;  
  this.poids = poids ;  
}  
  
Chat.prototype.miauler = function () {  
  console.log("Miaou ! Miaou !");  
};  
  
Chat.prototype.maigrir = function(deltaPoids) {  
  this.poids -= deltaPoids;  
};  
  
let felicitate = new Chat("Félicité",10,"chat de gouttière",3.5);  
console.log(felicitate);
```

Portées

```
<this>: Window  
felicitate: {  
  <prototype>: {  
    constructor: Chat()  
    maigrir: Chat.prototype.maigrir()  
    miauler: Chat.prototype.miauler()  
    <prototype>: {  
      __defineGetter__()  
      __defineSetter__()  
      __lookupGetter__()  
      __lookupSetter__()  
      __proto__: >>  
      constructor: Object()  
      hasOwnProperty()  
      isPrototypeOf()  
      propertyIsEnumerable()  
      toLocaleString()  
      toSource()  
      toString()  
      valueOf()  
    }  
  }  
}  
Window: Global
```

```
console.log(felicitate.nom);  
felicitate.miauler();  
console.log(felicitate.toString());
```

[Voir le code](#)

# Chaîne de <prototypes> : fallback

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
17  
18 console.log(felicite);  
19  
20
```

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {}  
  ▶ Window: Global
```

```
console.log(felicite.nom);  
felicite.miauler();  
console.log(felicite.toString());
```

[Voir le code](#)

# Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
▶ __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
17  
18 console.log(felicite);  
19  
20
```

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {}  
  ▶ Window: Global
```

```
console.log(felicite.nom);  
felicite.miauler();  
console.log(felicite.toString());
```

[Voir le code](#)

# Chaîne de <prototypes> : fallback

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
17  
18 console.log(felicite);  
19  
20
```

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {}  
  ▶ Window: Global
```

```
console.log(felicite.nom);  
felicite.miauler();  
console.log(felicite.toString());  
felicite.grossir(1.0);
```

[Voir le code](#)

# Chaîne de <prototypes> : fallback

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet **felicite**

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
▶ __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité",10,"chat de gouttière");  
17  
18 console.log(felicite);  
19  
20
```

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {}  
  ▶ Window: Global
```

```
console.log(felicite.nom);  
felicite.miauler();  
console.log(felicite.toString());  
felicite.grossir(1.0);
```

[Voir le code](#)

# Chaîne de <prototypes> : *fallback*

- Les chaînes de <prototypes> mécanisme utilisé par JavaScript pour implémenter le partage de propriétés entre objet issus d'un même constructeur et une forme d'héritage (mécanisme de *fallback*)

L'objet `felicite`

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  ▶ <prototype>: {}
```

<prototype> de l'objet `felicite`

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {}
```

<prototype> du <prototype> de l'objet `felicite`

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
▶ __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
function Chat(nom,age,race,poids) {  
  this.nom = nom ;  
  this.age = age ;  
  this.race = race ;  
  this.poids = poids ;  
}  
  
Chat.prototype.miauler = function () {  
  console.log("Miaou ! Miaou !");  
};  
  
Chat.prototype.maigrir = function(deltaPoids) {  
  this.poids -= deltaPoids;  
};  
  
let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
console.log(felicite);  
felicite.grossir(1.0);
```

Portées

```
▼ Bloc  
  ▶ <this>: Window  
  ▶ felicite: {}  
  ▶ Window: Global
```

```
console.log(felicite.nom);  
felicite.miauler();  
console.log(felicite.toString());  
felicite.grossir(1.0);
```

! ▶ `TypeError: felicite.grossir is not a function`

[Voir le code](#)



# <prototype> et constructeur

- Les objets créés à partir d'un même constructeur partagent le même <prototype>

L'objet **felicite**

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  <prototype>: {}
```

<prototype> de l'objet **felicite**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {}
```

<prototype> du <prototype> de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité");  
17 let felix = new Chat("Felix");  
18  
19 console.log(felicite);  
20
```

Portées

```
▼ Bloc  
  <this>: Window  
  felicite: {}  
  felix: {}  
  Window: Global
```

[Voir le code](#)

# <prototype> et constructeur

- Les objets créés à partir d'un même constructeur partagent le même <prototype>

L'objet **felicite**

```
▼ felicity: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  > <prototype>: {}
```

<prototype> de l'objet **felicite**  
et de l'objet **felix**

```
▶ constructor: Chat()  
▶ maigrir: Chat.prototype.maigrir()  
▶ miauler: Chat.prototype.miauler()  
▶ <prototype>: {}
```

<prototype> du <prototype>  
de l'objet **felicite**

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

```
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicite", 10, "chat de gouttière", 3.5);  
17 let felix = new Chat("Felix", 6, "siamois", 3);  
18  
19 console.log(felicite);  
20
```

Portées

```
▼ Bloc  
  <this>: Window  
  felicity: {}  
  felix: {}  
Window: Global
```

[Voir le code](#)

# <prototype> et constructeur

- Les objets créés à partir d'un même constructeur partagent le même <prototype>

The image shows a JavaScript IDE with the following code:

```
1 function Chat(nom,age,race,poids) {
2   this.nom = nom ;
3   this.age = age ;
4   this.race = race ;
5   this.poids = poids ;
6 }
7
8 Chat.prototype.miauler = function () {
9   console.log("Miaou ! Miaou !");
10 };
11
12 Chat.prototype.maignrir = function(deltaPoids) {
13   this.poids -= deltaPoids;
14 };
15
16 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
17 let felix = new Chat("Felix", 6, "siamois", 3);
18
19 console.log(felicite);
```

Callouts and annotations:

- L'objet felicite**: Shows the object structure: `felicite: {nom: "Félicité", age: 10, poids: 3.5, race: "chat de gouttière", <prototype>: {}}`.
- L'objet felix**: Shows the object structure: `felix: {nom: "Felix", age: 6, poids: 3, race: "siamois", <prototype>: {}}`.
- Portées**: Shows the global scope containing `Window: Global`, `felicite: {...}`, and `felix: {...}`.
- constructor:Chat()**: Shows the constructor function with properties: `arguments: null, caller: null, length: 4, name: "Chat", prototype: {...}, <prototype>():`.
- <prototype> de l'objet felicite et de l'objet felix**: Points to the `Chat.prototype` object, which contains methods `maigrir` and `miauler`.
- <prototype> du <prototype> de l'objet felicite**: Points to the `Object.prototype` object, which contains methods like `__defineGetter__`, `__defineSetter__`, `__lookupGetter__`, `__lookupSetter__`, `__proto__: >>`, `constructor: Object()`, `hasOwnProperty()`, `isPrototypeOf()`, `propertyIsEnumerable()`, `toLocaleString()`, `toSource()`, `toString()`, and `valueOf()`.

La fonction constructeur **Chat** est aussi un objet

Le <prototype> des objets créés avec le constructeur **Chat** est le **prototype** de l'objet constructeur **Chat**

[Voir le code](#)

# <prototype> et constructeur

- Les objets créés à partir d'un même constructeur partagent le même <prototype>

```
protosObjetsConstructeur1_1.js X
1 function Chat(nom,age,race,poids) {
2   this.nom = nom ;
3   this.age = age ;
4   this.race = race ;
5   this.poids = poids ;
6 }
7
8 Chat.prototype.miauler = function () {
9   console.log("Miaou ! Miaou !");
10 };
11
12 Chat.prototype.maignrir = function(deltaPoids) {
13   this.poids -= deltaPoids;
14 };
15
16 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
17 let felix = new Chat("Felix", 6, "siamois", 3);
18
19 console.log(felicite);
20
```

 [Voir le code](#)

L'objet felicite

```
▼ felicite: {}
  age: 10
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {}
```

```
▼ felix: {}
  age: 6
  nom: "Felix"
  poids: 3
  race: "siamois"
  <prototype>: {}
```

Portées

```
▼ Bloc
  <this>: Window
  felicite: {}
  felix: {}
  Window: Global
```

```
▼ constructor:Chat()
  arguments: null
  caller: null
  length: 4
  name: "Chat"
  prototype: {}
  <prototype>()
```

<prototype> de l'objet felicite et de l'objet felix

```
▼ constructor:Chat()
  maigrir:Chat.prototype.maignrir()
  miauler:Chat.prototype.miauler()
  <prototype>: {}
```

Comme tout objet le constructeur **Chat**<sup>1</sup> possède une chaîne de <prototypes>

Toutes les chaînes de <prototypes> partagent le même objet terminal

```
▶ apply()
  arguments: >>
▶ bind()
▶ call()
  caller: >>
▶ constructor:Function()
  length: 0
  name: ""
▶ toSource()
▶ toString()
▶ Symbol(Symbol.hasInstance):[Symbol]
▶ <get arguments(): arguments()
▶ <set arguments(): arguments()
▶ <get caller(): caller()
▶ <set caller(): caller()
▶ <prototype>: {}
```

<prototype> du <prototype> de l'objet felicite

```
▼ <prototype>: {}
  __defineGetter__()
  __defineSetter__()
  __lookupGetter__()
  __lookupSetter__()
  __proto__: >>
  constructor:Object()
  hasOwnProperty()
  isPrototypeOf()
  propertyIsEnumerable()
  toLocaleString()
  toSource()
  toString()
  valueOf()
```

<sup>1</sup> Chat est une fonction et en JavaScript les fonctions sont des objets

# <prototype> vs. Constructeur.prototype

Ne pas confondre

- la propriété (attribut) **prototype** d'un objet constructeur
- le lien prototype (**<prototype>**) de cet objet constructeur



L'objet felicite

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  <prototype>: {}
```

```
▼ felix: {}  
  age: 6  
  nom: "Felix"  
  poids: 3  
  race: "siamois"  
  <prototype>: {}
```

```
▼ Portées  
  ▼ Bloc  
    ▶ <this>: Window  
    ▶ felicite: {}  
    ▶ felix: {}  
    ▶ Window: Global
```

```
protosObjetsConstructeur1_1.js X  
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maignrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité",10,"chat de gouttière",3.5);  
17 let felix = new Chat("Felix",6,"siamois",3);  
18  
19 console.log(felicite);  
20
```

[Voir le code](#)

<prototype> de l'objet felicite  
et de l'objet felix

```
▼ constructor:Chat()  
  arguments: null  
  caller: null  
  length: 4  
  name: "Chat"  
  prototype: {}  
  <prototype>()
```

```
▶ apply()  
  arguments: >>  
▶ bind()  
▶ call()  
  caller: >>  
▶ constructor:Function()  
  length: 0  
  name: ""  
▶ toSource()  
▶ toString()  
▶ Symbol(Symbol.hasInstance):[Symbol()]  
▶ <get arguments(): arguments()  
▶ <set arguments(): arguments()  
▶ <get caller(): caller()  
▶ <set caller(): caller()  
▶ <prototype>: {}
```

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
  __proto__: >>  
▶ constructor:Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

# <prototype> vs. Constructeur.prototype

Ne pas confondre



1 la propriété (attribut) **prototype** d'un objet constructeur

- le lien prototype (<prototype>) de cet objet constructeur

L'objet felicite

```
▼ felicite: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  <prototype>: {}
```

```
▼ felix: {}  
  age: 6  
  nom: "Felix"  
  poids: 3  
  race: "siamois"  
  <prototype>: {}
```

<prototype> de l'objet felicite et de l'objet felix

```
▼ constructor: Chat()  
  maigrir: Chat.prototype.maigrir()  
  miauler: Chat.prototype.miauler()  
  <prototype>: {}
```

1 le prototype de l'objet constructeur Chat est le <prototype> des objets créés avec le constructeur Chat

```
protosObjetsConstructeur1_1.js X  
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maigrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);  
17 let felix = new Chat("Felix", 6, "siamois", 3);  
18  
19 console.log(felicite);  
20
```

```
▼ Portées  
▼ Bloc  
  <this>: Window  
  felicite: {}  
  felix: {}  
  Window: Global
```

```
▼ constructor: Chat()  
  arguments: null  
  caller: null  
  length: 4  
  name: "Chat"  
  prototype: {}  
  <prototype>()
```

```
▶ apply()  
▶ arguments: >>  
▶ bind()  
▶ call()  
▶ caller: >>  
▶ constructor: Function()  
▶ length: 0  
▶ name: ""  
▶ toSource()  
▶ toString()  
▶ Symbol(Symbol.hasInstance): [Symbol()]  
▶ <get arguments(): arguments()  
▶ <set arguments(): arguments()  
▶ <get caller(): caller()  
▶ <set caller(): caller()  
▶ <prototype>: {}
```

```
▶ __defineGetter__()  
▶ __defineSetter__()  
▶ __lookupGetter__()  
▶ __lookupSetter__()  
▶ __proto__: >>  
▶ constructor: Object()  
▶ hasOwnProperty()  
▶ isPrototypeOf()  
▶ propertyIsEnumerable()  
▶ toLocaleString()  
▶ toSource()  
▶ toString()  
▶ valueOf()
```

[Voir le code](#)

# <prototype> vs. Constructeur.prototype

Ne pas confondre



- 1 la propriété (attribut) **prototype** d'un objet constructeur
- 2 le lien prototype (**<prototype>**) de cet objet constructeur

L'objet **felicite**

```
▼ felicity: {}  
  age: 10  
  nom: "Félicité"  
  poids: 3.5  
  race: "chat de gouttière"  
  <prototype>: {}
```

```
▼ felix: {}  
  age: 6  
  nom: "Felix"  
  poids: 3  
  race: "siamois"  
  <prototype>: {}
```

```
▼ Portées  
▼ Bloc  
  <this>: Window  
  felicity: {}  
  felix: {}  
  Window: Global
```

```
protosObjetsConstructeur1_1.js X  
1 function Chat(nom,age,race,poids) {  
2   this.nom = nom ;  
3   this.age = age ;  
4   this.race = race ;  
5   this.poids = poids ;  
6 }  
7  
8 Chat.prototype.miauler = function () {  
9   console.log("Miaou ! Miaou !");  
10 };  
11  
12 Chat.prototype.maignrir = function(deltaPoids) {  
13   this.poids -= deltaPoids;  
14 };  
15  
16 let felicity = new Chat("Félicité",10,"chat de gouttière",3.5);  
17 let felix = new Chat("Felix",6,"siamois",3);  
18  
19 console.log(felicite);  
20
```

[Voir le code](#)

```
▼ constructor:Chat()  
  arguments: null  
  caller: null  
  length: 4  
  name: "Chat"  
  1 prototype: {}  
  2 <prototype>()
```

<prototype> de l'objet **felicite**  
et de l'objet **felix**

```
▼ constructor:Chat()  
  maigrir:Chat.prototype.maignrir()  
  miauler:Chat.prototype.miauler()  
  <prototype>: {}
```

- 1 le **prototype** de l'objet constructeur **Chat** est le **<prototype>** des objets créés avec le constructeur **Chat**
- 2 **<prototype>** du constructeur est la propriété **prototype** de la fonction qui a servi à créer l'objet constructeur **Chat()**

```
▼ apply()  
  arguments: >>  
  bind()  
  call()  
  caller: >>  
  constructor:Function()  
  length: 0  
  name: ""  
  toSource()  
  toString()  
  Symbol(Symbol.hasInstance):[Symbol()]  
  <get arguments(): arguments()  
  <set arguments(): arguments()  
  <get caller(): caller()  
  <set caller(): caller()  
  <prototype>: {}
```

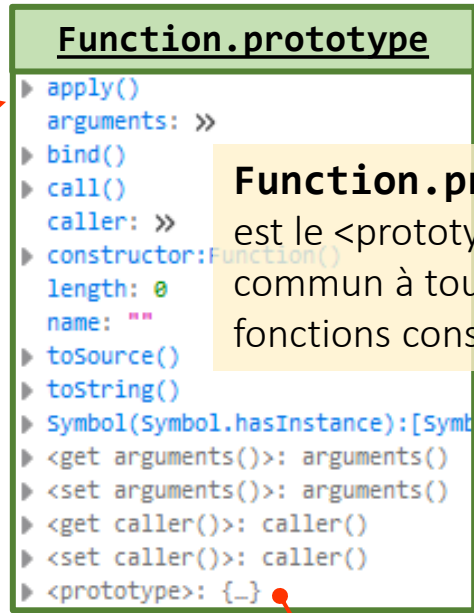
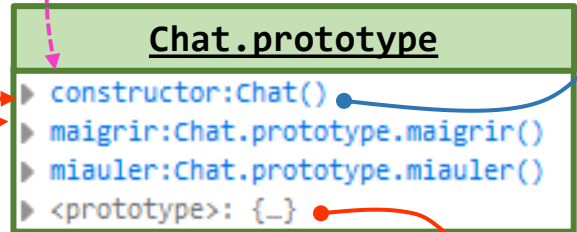
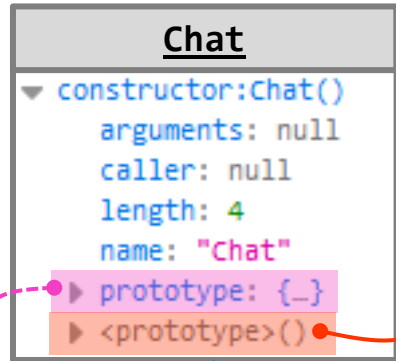
<prototype> du <prototype>  
de l'objet **felicite**

```
▼ __defineGetter__()  
  __defineSetter__()  
  __lookupGetter__()  
  __lookupSetter__()  
  __proto__: >>  
  constructor:Object()  
  hasOwnProperty()  
  isPrototypeOf()  
  propertyIsEnumerable()  
  toLocaleString()  
  toSource()  
  toString()  
  valueOf()
```

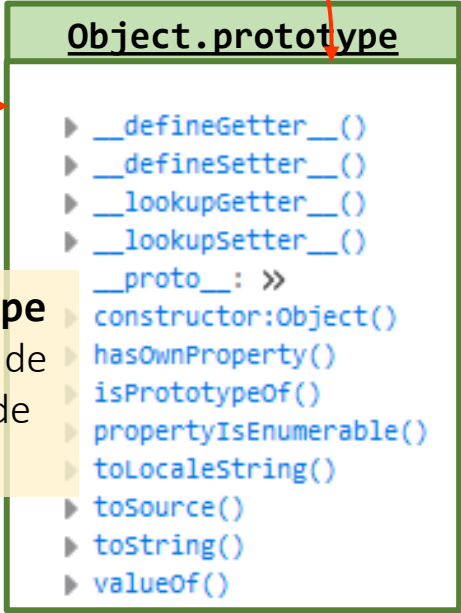
# Built-in objects

**Object** et **Function** sont des objets (fonctions constructeur) prédéfinis (*built-in*)

```
protosObjetsConstructeur1_1.js X
1 function Chat(nom,age,race,poids) {
2   this.nom = nom ;
3   this.age = age ;
4   this.race = race ;
5   this.poids = poids ;
6 }
7
8 Chat.prototype.miauler = function () {
9   console.log("Miaou ! Miaou !");
10 };
11
12 Chat.prototype.maignrir = function(deltaPoids) {
13   this.poids -= deltaPoids;
14 };
15
16 let felicite = new Chat("Félicité",10,"chat de gouttière", 3.5);
17 let felix = new Chat("Felix",6,"siamois",3);
18
19 console.log(felicite);
20
```



**Function.prototype** est le <prototype> commun à toutes les fonctions constructeur



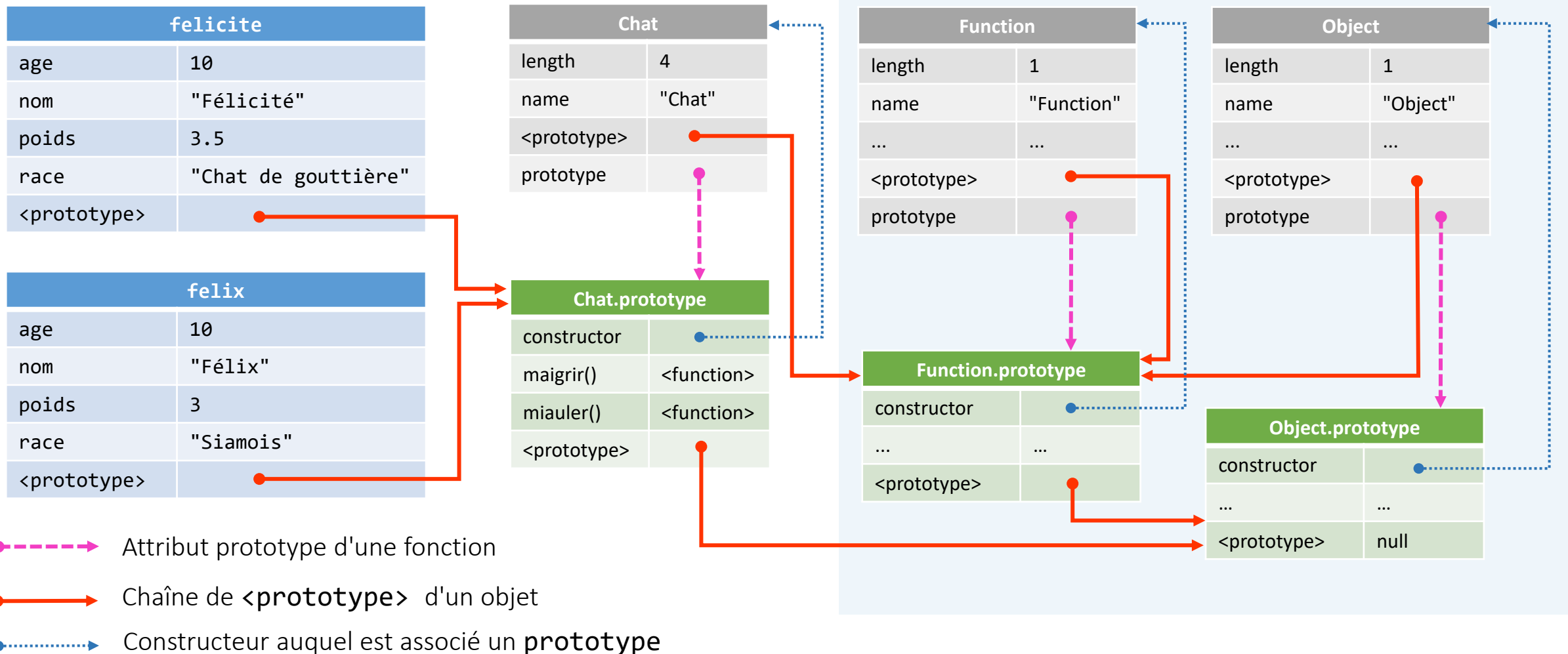
**Object.prototype** est l'objet terminal de toutes les chaînes de <prototypes>

[Voir le code](#)



# Built-in objects

- Les fonctions `Function()` et `Object()` et leur chaîne de `<prototype>`



# <prototype> objet Littéral

- Le <prototype> d'un objet littéral est `Object.prototype`

```
1 function Chat(nom, age, race, poids) {
2   this.nom = nom;
3   this.age = age;
4   this.race = race;
5   this.poids = poids;
6 }
7
8 Chat.prototype.miauler = function () {
9   console.log("Miaou ! Miaou !");
10 };
11
12 Chat.prototype.maignrir = function (deltaPoids) {
13   this.poids -= deltaPoids;
14 };
15
16 let felicite = new Chat("Félicité", 10, "chat de gouttière");
17 let felix = new Chat("Felix", 6, "siamois");
18
19 let fritz = {
20   nom: "Fritz le chat",
21   age: 14,
22   race: "chat tigré",
23   poids: 4.0
24 };
```

**felicite**

```
▼ felicite: {}
  age: 10
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {}
```

**felix**

```
▼ felix: {}
  age: 6
  nom: "Felix"
  poids: 3
  race: "siamois"
  <prototype>: {}
```

**fritz**

```
▼ fritz: {}
  age: 14
  nom: "Fritz le chat"
  poids: 4
  race: "chat tigré"
  <prototype>: {}
```

**Chat**

```
▼ constructor:Chat()
  arguments: null
  caller: null
  length: 4
  name: "Chat"
  prototype: {}
  <prototype>()
```

**Chat.prototype**

```
▶ constructor:Chat()
▶ maigrir:Chat.prototype.maignrir()
▶ miauler:Chat.prototype.miauler()
▶ <prototype>: {}
```

**Function.prototype**

```
▶ apply()
  arguments: >>
▶ bind()
▶ call()
  caller: >>
▶ constructor:Function()
  length: 0
  name: ""
▶ toSource()
▶ toString()
▶ Symbol(Symbol.hasInstance):[Symbol]
▶ <get arguments(): arguments()
▶ <set arguments(): arguments()
▶ <get caller(): caller()
▶ <set caller(): caller()
▶ <prototype>: {}
```

**Object.prototype**

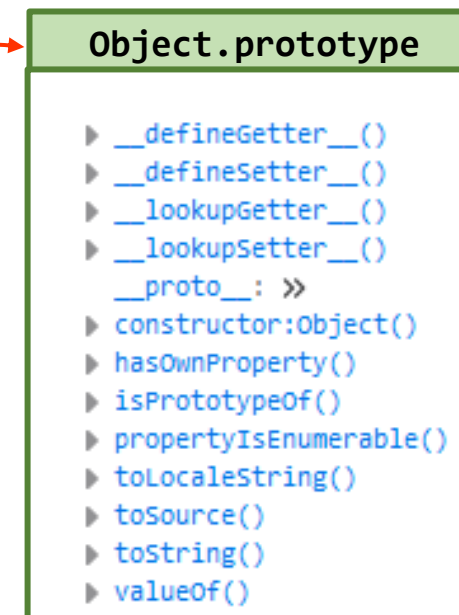
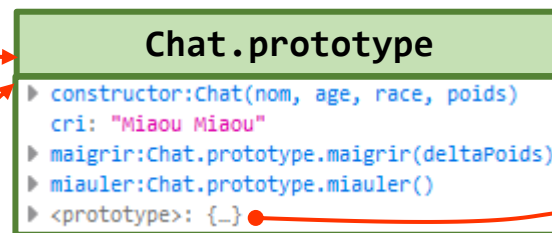
```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: >>
▶ constructor:Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

 [Voir le code](#)

# Fallback de propriétés

- Une propriété définie au niveau du prototype d'un constructeur est 'héritée' par tous les objets ayant ce <prototype>

```
1 function Chat(nom, age, race, poids) {
2   this.nom = nom;
3   this.age = age;
4   this.race = race;
5   this.poids = poids;
6 }
7
8 Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11   console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maignrir = function (deltaPoids) {
15   this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19   console.log("cri de félicité " + felicite.cri);
20   console.log("cri de félix " + felix.cri);
21   felicite.miauler();
22   felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```



[Voir le code](#)

???

# Fallback de propriétés

- Une propriété définie au niveau du prototype d'un constructeur est 'héritée' par tous les objets ayant ce <prototype>

```
1 function Chat(nom, age, race, poids) {
2   this.nom = nom;
3   this.age = age;
4   this.race = race;
5   this.poids = poids;
6 }
7
8 Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11   console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maignrir = function (deltaPoids) {
15   this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19   console.log("cri de félicité " + felicite.cri);
20   console.log("cri de félix " + felix.cri);
21   felicite.miauler();
22   felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

**felicite**

```
▼ felicite: {}
  age: 10
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {}
```

**felix**

```
▼ felix: {}
  age: 6
  nom: "Felix"
  poids: 3
  race: "siamois"
  <prototype>: {}
```

propriété cri ?

**Chat.prototype**

```
▶ constructor: Chat(nom, age, race, poids)
  cri: "Miaou Miaou"
  maigrir: Chat.prototype.maigrir(deltaPoids)
  miauler: Chat.prototype.miauler()
  <prototype>: {}
```

**Object.prototype**

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: >>
▶ constructor: Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

évaluation de `felicite.cri`

recherche de la propriété `cri` dans l'objet `felicite` (propriété propre)

elle n'est pas trouvée

recherche de la `cri` dans le <prototype> de `felicite`

→ "Miaou Miaou"



[Voir le code](#)

# Fallback de propriétés

- Une propriété définie au niveau du prototype d'un constructeur est 'héritée' par tous les objets ayant ce <prototype>

```
1 function Chat(nom, age, race, poids) {
2   this.nom = nom;
3   this.age = age;
4   this.race = race;
5   this.poids = poids;
6 }
7
8 Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11   console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maignrir = function (deltaPoids) {
15   this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19   console.log("cri de félicité " + felicite.cri);
20   console.log("cri de félix " + felix.cri);
21   felicite.miauler();
22   felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

**felicite**

```
▼ felicite: {}
  age: 10
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {}
```

**felix**

```
▼ felix: {}
  age: 6
  nom: "Felix"
  poids: 3
  race: "siamois"
  <prototype>: {}
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```

les cris des objets sont 'hérités' de leur objet <prototype>

propriété cri ?

**Chat.prototype**

```
▶ constructor: Chat(nom, age, race, poids)
  cri: "Miaou Miaou"
  maigrir: Chat.prototype.maigrir(deltaPoids)
  miauler: Chat.prototype.miauler()
  <prototype>: {}
```

**Object.prototype**

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: >>
▶ constructor: Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toString()
▶ valueOf()
```

évaluation de `felicite.cri`

- recherche de la propriété `cri` dans l'objet `felicite` (propriété propre)
- elle n'est pas trouvée
- recherche de la `cri` dans le <prototype> de `felicite`
- "Miaou Miaou"

idem pour `felix.cri` et `this.cri` dans la méthode `miauler`



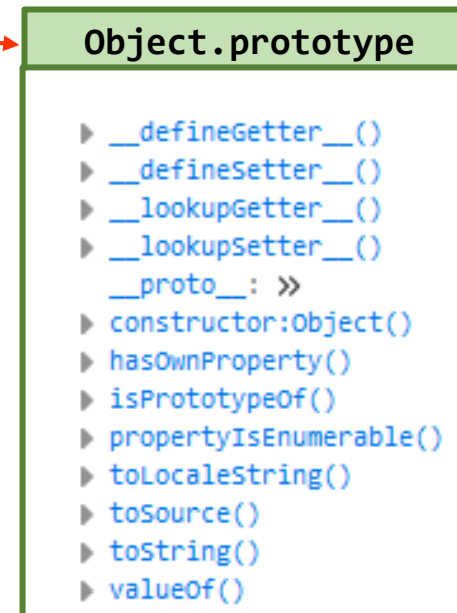
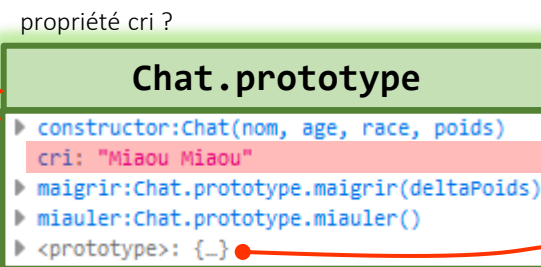
[Voir le code](#)

# Fallback de propriétés

- Une propriété définie au niveau du prototype d'un constructeur est 'héritée' par tous les objets ayant ce <prototype>



les cris des objets sont 'hérités' de leur objet <prototype>



de manière générale  
évaluation de `obj.prop`

- recherche de la propriété `prop` dans l'objet `obj` (propriété propre)
- si elle n'est pas trouvée
  - recherche de la `prop` dans la chaîne de <prototype> de `obj`
  - si elle n'est pas trouvé → `undefined`

`valeur(obj, prop)`

```
tantque (obj !== null && ! prop in obj)
  obj = <prototype> de obj
fin tantque
```

```
si (obj === null)
  → undefined
```

```
sinon
  → valeur de prop pour obj
```



[Voir le code](#)

# Fallback de propriétés

- Modification d'une propriété du prototype d'un constructeur s'applique à toutes les instances

```
1 function Chat(nom, age, race, poids) {
2   this.nom = nom;
3   this.age = age;
4   this.race = race;
5   this.poids = poids;
6 }
7
8 Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11   console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maignrir = function (deltaPoids) {
15   this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19   console.log("cri de félicité " + felicite.cri);
20   console.log("cri de félix " + felix.cri);
21   felicite.miauler();
22   felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Félix", 6, "siamois", 3);
27 afficherCris();
```

**felicite**

```
▼ felicite: {...}
  age: 10
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {...}
```

**felix**

```
▼ felix: {...}
  age: 6
  nom: "Félix"
  poids: 3
  race: "siamois"
  <prototype>: {...}
```

**Chat.prototype**

```
▶ constructor: Chat(nom, age, race, poids)
  cri: "Meow ! Meow !"
  maigrir: Chat.prototype.maignrir(deltaPoids)
  miauler: Chat.prototype.miauler()
  <prototype>: {...}
```

**Object.prototype**

```
▶ __defineGetter__()
▶ __defineSetter__()
▶ __lookupGetter__()
▶ __lookupSetter__()
  __proto__: >>
▶ constructor: Object()
▶ hasOwnProperty()
▶ isPrototypeOf()
▶ propertyIsEnumerable()
▶ toLocaleString()
▶ toSource()
▶ toString()
▶ valueOf()
```

```
28
29 Chat.prototype.cri = "Meow ! Meow !";
30 afficherCris();
31
32 felicite.cri = "Miaou ! Miaou !";
33 afficherCris();
34
35 delete felicite.cri;
36 afficherCris();
37
38 delete felix.cri;
39 afficherCris();
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Félix : Miaou Miaou
```

```
cri de félicité Meow ! Meow !
cri de félix Meow ! Meow !
Félicité : Meow ! Meow !
Félix : Meow ! Meow !
```

les cris des objets sont 'hérités' de leur objet <prototype>

les cris des objets sont 'hérités' de leur objet <prototype>

[Voir le code](#)

# Fallback de propriétés

- L'affectation d'une propriété héritée en passant par une instance modifie uniquement l'objet instance

```
1 function Chat(nom, age, race, poids) {
2   this.nom = nom;
3   this.age = age;
4   this.race = race;
5   this.poids = poids;
6 }
7
8 Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11   console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maignrir = function (deltaPoids) {
15   this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19   console.log("cri de félicité " + felicite.cri);
20   console.log("cri de félix " + felix.cri);
21   felicite.miauler();
22   felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

**felicite**

```
felicite: {...}
  age: 10
  cri: "Miaou ! Miaou !"
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {...}
```

**felix**

```
felix: {...}
  age: 6
  nom: "Felix"
  poids: 3
  race: "siamois"
  <prototype>: {...}
```

**Chat.prototype**

```
constructor: Chat(nom, age, race, poids)
cri: "Meow ! Meow !"
maigrir: Chat.prototype.maignrir(deltaPoids)
miauler: Chat.prototype.miauler()
<prototype>: {...}
```

**Object.prototype**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: >>
constructor: Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toString()
valueOf()
```

la propriété `cri` est redéfinie au niveau de l'instance `felicite` (ajout de la propriété si elle n'existe pas, modification si elle existe déjà)

```
28
29 Chat.prototype.cri = "Meow ! Meow !";
30 afficherCris();
31
32 felicite.cri = "Miaou ! Miaou !";
33 afficherCris();
34
35 delete felicite.cri;
36 afficherCris();
37
38 delete felix.cri;
39 afficherCris();
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```

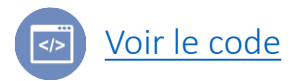
les cris des objets sont 'hérités' de leur objet <prototype>

```
cri de félicité Meow ! Meow !
cri de félix Meow ! Meow !
Félicité : Meow ! Meow !
Felix : Meow ! Meow !
```

les cris des objets sont 'hérités' de leur objet <prototype>

```
cri de félicité Miaou ! Miaou !
cri de félix Meow ! Meow !
Félicité : Miaou ! Miaou !
Felix : Meow ! Meow !
```

Les cris sont différents  
felicite : propriété propre (own property)  
felix : propriété héritée (fallback)





# Fallback de propriétés

- La suppression d'une propriété en passant par une instance modifie uniquement l'objet instance

```
1 function Chat(nom, age, race, poids) {
2   this.nom = nom;
3   this.age = age;
4   this.race = race;
5   this.poids = poids;
6 }
7
8 Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11   console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maignrir = function (deltaPoids) {
15   this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19   console.log("cri de félicité " + felicite.cri);
20   console.log("cri de félix " + felix.cri);
21   felicite.miauler();
22   felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

**felicite**

```
felicite: {...}
  age: 10
  cri: "Miaou ! Miaou !"
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {...}
```

**felix**

```
felix: {...}
  age: 6
  nom: "Felix"
  poids: 3
  race: "siamois"
  <prototype>: {...}
```

**Chat.prototype**

```
constructor: Chat(nom, age, race, poids)
cri: "Miaou Miaou"
maigrir: Chat.prototype.maignrir(deltaPoids)
miauler: Chat.prototype.miauler()
<prototype>: {...}
```

**Object.prototype**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: >>
constructor: Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toString()
valueOf()
```

retrait de la propriété cri dans l'instance felicite

```
28
29 Chat.prototype.cri = "Meow ! Meow !";
30 afficherCris();
31
32 felicite.cri = "Miaou ! Miaou !";
33 afficherCris();
34
35 delete felicite.cri;
36 afficherCris();
37
38 delete felix.cri;
39 afficherCris();
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```

les cris des objets sont 'hérités' de leur objet <prototype>

```
cri de félicité Meow ! Meow !
cri de félix Meow ! Meow !
Félicité : Meow ! Meow !
Felix : Meow ! Meow !
```

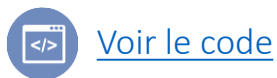
les cris des objets sont 'hérités' de leur objet <prototype>

```
cri de félicité Miaou ! Miaou !
cri de félix Meow ! Meow !
Félicité : Miaou ! Miaou !
Felix : Meow ! Meow !
```

Les cris sont différents  
felicite : propriété propre (own property)  
felix : propriété héritée (fallback)

```
cri de félicité Meow ! Meow !
cri de félix Meow ! Meow !
Félicité : Meow ! Meow !
Felix : Meow ! Meow !
```

La valeur de la propriété cri pour felicite est à nouveau la propriété héritée (fallback)



# Chaîne de prototypes

- La suppression d'une propriété en passant par une instance modifie uniquement l'objet instance

```
1 function Chat(nom, age, race, poids) {
2   this.nom = nom;
3   this.age = age;
4   this.race = race;
5   this.poids = poids;
6 }
7
8 Chat.prototype.cri = "Miaou Miaou";
9
10 Chat.prototype.miauler = function () {
11   console.log(this.nom + " : " + this.cri);
12 };
13
14 Chat.prototype.maignrir = function (deltaPoids) {
15   this.poids -= deltaPoids;
16 };
17
18 function afficherCris() {
19   console.log("cri de félicité " + felicite.cri);
20   console.log("cri de félix " + felix.cri);
21   felicite.miauler();
22   felix.miauler();
23 }
24
25 let felicite = new Chat("Félicité", 10, "chat de gouttière", 3.5);
26 let felix = new Chat("Felix", 6, "siamois", 3);
27 afficherCris();
```

**felicite**

```
felicite: {...}
  age: 10
  cri: "Miaou ! Miaou !"
  nom: "Félicité"
  poids: 3.5
  race: "chat de gouttière"
  <prototype>: {...}
```

**felix**

```
felix: {...}
  age: 6
  nom: "Felix"
  poids: 3
  race: "siamois"
  <prototype>: {...}
```

**Chat.prototype**

```
constructor: Chat(nom, age, race, poids)
cri: "Miaou Miaou"
maigrir: Chat.prototype.maignrir(deltaPoids)
miauler: Chat.prototype.miauler()
<prototype>: {...}
```

**Object.prototype**

```
__defineGetter__()
__defineSetter__()
__lookupGetter__()
__lookupSetter__()
__proto__: >>
constructor: Object()
hasOwnProperty()
isPrototypeOf()
propertyIsEnumerable()
toLocaleString()
toString()
valueOf()
```

```
28
29 Chat.prototype.cri = "Meow ! Meow !";
30 afficherCris();
31
32 felicite.cri = "Miaou ! Miaou !";
33 afficherCris();
34
35 delete felicite.cri;
36 afficherCris();
37
38 delete felix.cri;
39 afficherCris();
```

```
cri de félicité Miaou Miaou
cri de félix Miaou Miaou
Félicité : Miaou Miaou
Felix : Miaou Miaou
```

```
cri de félicité Meow ! Meow !
cri de félix Meow ! Meow !
Félicité : Meow ! Meow !
Felix : Meow ! Meow !
```

```
cri de félicité Miaou ! Miaou !
cri de félix Meow ! Meow !
Félicité : Miaou ! Miaou !
Felix : Meow ! Meow !
```

```
cri de félicité Meow ! Meow !
cri de félix Meow ! Meow !
Félicité : Meow ! Meow !
Felix : Meow ! Meow !
```

```
cri de félicité Meow ! Meow !
cri de félix Meow ! Meow !
Félicité : Meow ! Meow !
Felix : Meow ! Meow !
```

suppression sans effet si l'instance n'a pas directement cette propriété (own property)



# Objets et ES5/ES6

- La création d'objets avec une fonction constructeur et l'opérateur `new` est historique, elle masque le mécanisme de prototypage souvent mal compris par les programmeurs.
- ES5 puis ES6 proposent de nouvelles manières de définir des objets (même si en interne le principe du prototypage est inchangé) afin de rendre l'utilisation des objets plus accessible aux développeurs:
  - `Object.create()` (ES5)
  - Classes (ES6)

# Objets et ES5 : Object

- Depuis la version ES5 de JavaScript, l'objet prédéfini (*built-in*) **Object** propose un certain nombre de méthodes pour créer et manipuler les prototypes :
  - **Object.create(proto:Object) : Object**  
crée un nouvel objet ayant pour <prototype> l'objet **proto** passé en paramètre
  - **Object.getPrototypeOf(obj : Object) : Object**  
renvoie l'objet <prototype> de l'objet **obj** passé en paramètre
  - **Object.setPrototypeOf(obj: Object, proto: Object)**  
fixe le <prototype> de **obj** avec l'objet **proto**
  - ... (voir la doc [🔗 MDN](#) )

# Objets et ES6 : Classes

- Avec l'introduction de **Class**, ES6 offre une syntaxe compacte pour définir des chaînes de prototypes et qui se rapproche de l'approche plus couramment utilisée dans les langages orientés objets (langages de classes comme Java)

avant ES6

```
function Chat(nom,age,race,poids) {  
  this.nom = nom ;  
  this.age = age ;  
  this.race = race ;  
  this.poids = poids;  
}  
  
Chat.prototype.miauler = function () {  
  console.log(this.nom + "-> Miaou ! Miaou !");  
};  
  
Chat.prototype.maignir = function(deltaPoids) {  
  this.poids -= deltaPoids;  
};
```

constructeur



méthodes

ES6 +

```
class Chat {  
  constructor(nom,age,race,poids) {  
    this.nom = nom ;  
    this.age = age ;  
    this.race = race ;  
    this.poids = poids;  
  }  
  miauler() {  
    console.log(this.nom + "-> Miaou ! Miaou !");  
  }  
  maigrir(deltaPoids) {  
    this.poids -= deltaPoids;  
  }  
}
```

mot réservé pour identifier la fonction constructeur

Une classe ne peut avoir qu'un seul constructeur



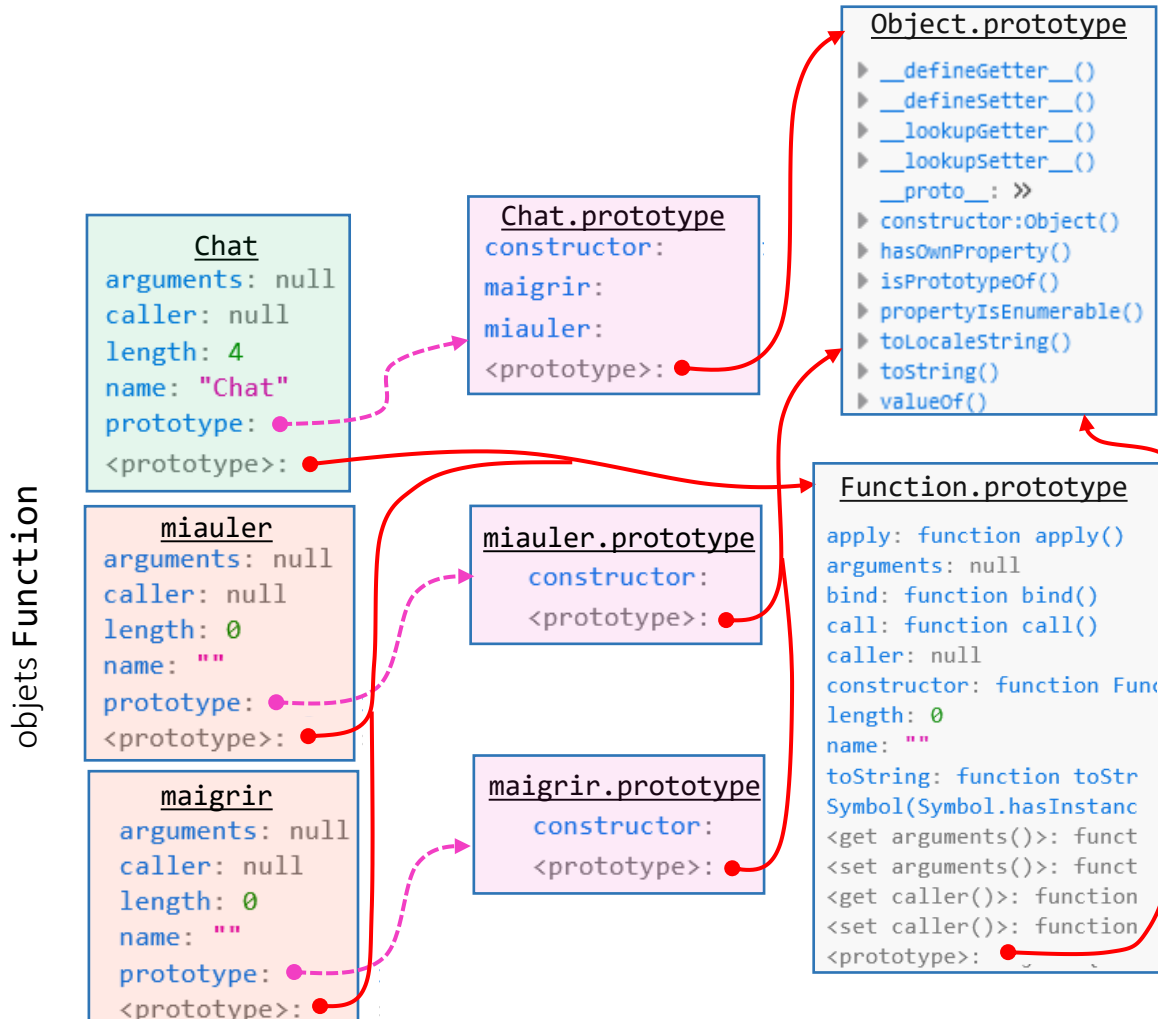
pas de ; entre les déclarations de méthodes




[Voir le code](#)

# Objets et ES6 : Classes

- Les objets créés par une déclaration de classe sont des objets fonction correspondant respectivement au constructeur et aux méthodes de la classe ainsi que les **prototypes** associés



```
class Chat {
  constructor(nom,age,race,poids) {
    this.nom = nom ;
    this.age = age ;
    this.race = race ;
    this.poids = poids;
  }
  miauler() {
    console.log(this.nom + "-> Miaou ! Miaou !");
  }
  maigrir(deltaPoids) {
    this.poids -= deltaPoids;
  }
}
```

 [Voir le code](#)

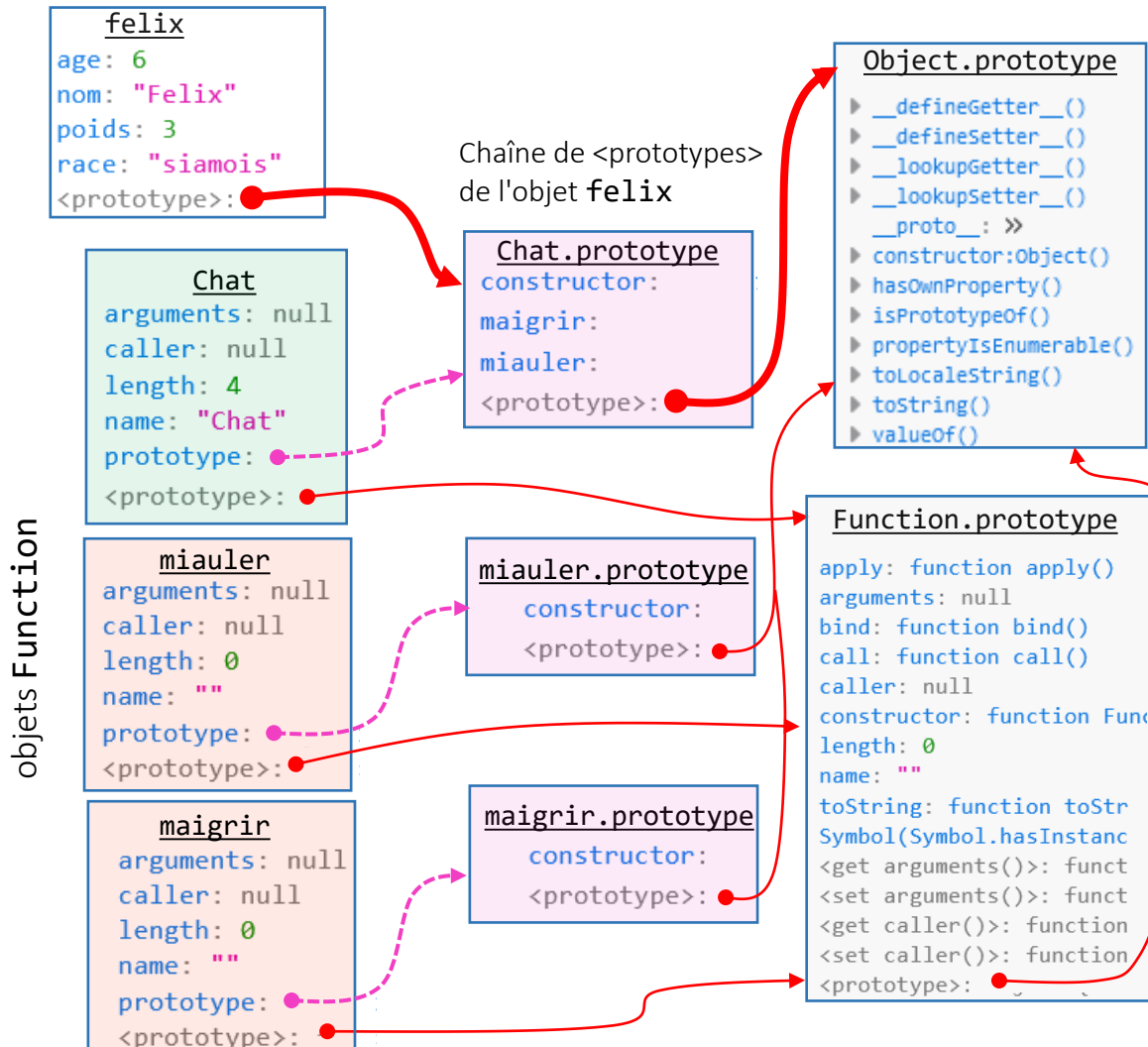
# Objets et ES6 : Classes

- La création des objets se fait comme précédemment avec l'opérateur `new`

allocation mémoire

exécution de la fonction constructeur

```
let felix = new Chat("Felix",6,"siamois",3);
```



```
class Chat {
```

```
  constructor(nom,age,race,poids) {
```

```
    this.nom = nom ;
```

appelée immédiatement après la création d'une nouvelle instance pour l'initialiser. Cette fonction reçoit les arguments passés après le nom de la classe suivant l'opérateur `new`.

```
    this.age = age ;
```

```
    this.race = race ;
```

```
    this.poids = poids;
```

```
  }
```

```
  miauler() {
```

```
    console.log(this.nom + "-> Miaou ! Miaou !");
```

```
  }
```

```
  maigrir(deltaPoids) {
```

```
    this.poids -= deltaPoids;
```

```
  }
```

```
}
```

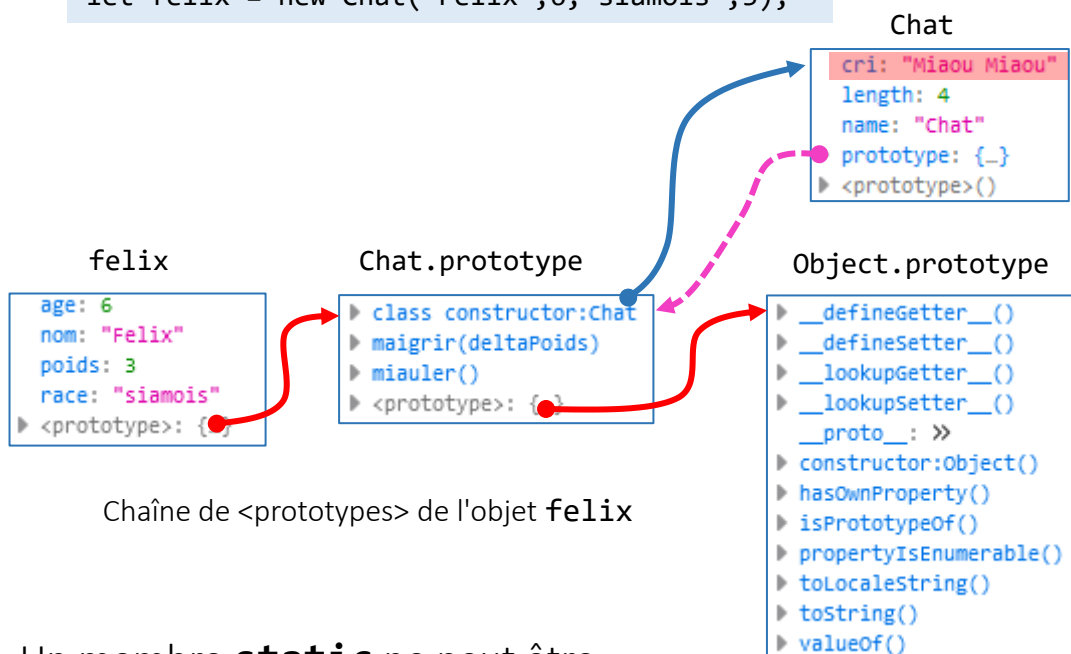


[Voir le code](#)

# Objets et ES6 : Classes – membres statiques

- Propriétés **statiques**
  - Possibilité d'associer des **propriétés** à la classe directement
  - Déclaration de variable préfixée par le mot clé **static**
  - Rattaché à la fonction Constructeur et non pas au **prototype** de celle-ci

```
let felix = new Chat("Felix",6,"siamois",3);
```



Chaîne de <prototypes> de l'objet felix

```
class Chat {  
  constructor(nom,age,race,poids) {  
    this.nom = nom ;  
    this.age = age ;  
    this.race = race ;  
    this.poids = poids;  
  }  
  static cri = "Miaou ! Miaou !";  
  miauler() {  
    console.log(this.nom + "-> " + Chat.cri);  
  }  
  maigrir(deltaPoids) {  
    this.poids -= deltaPoids;  
  }  
}
```

 [Voir le code](#)

- Un membre **static** ne peut être accédé via une instance de la classe (pas dans la chaîne de <prototypes>)
- Un membre **static** est accédé via l'identifiant de la classe

```
felix.cri → undefined
```

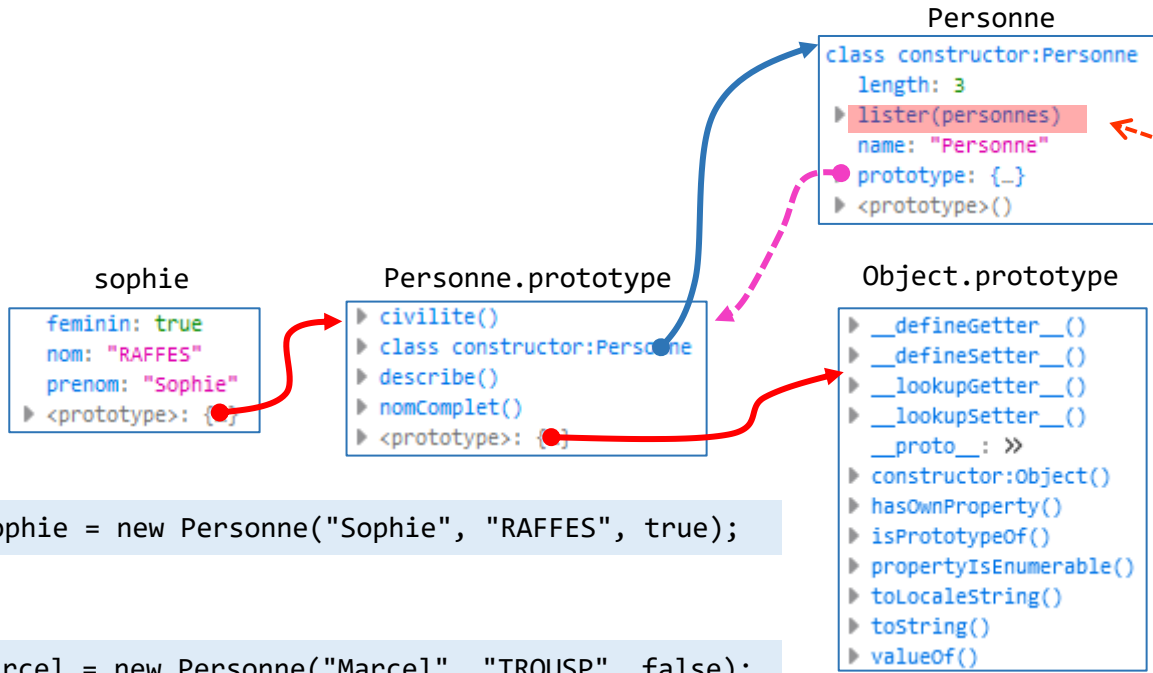
```
Chat.cri → "Miaou ! Miaou !"
```



# Objets et ES6 : Classes – membres statiques

- Méthodes statiques

- déclaration de méthode préfixée par le mot clé **static**
- fonction rattaché à la fonction constructeur et non pas au **prototype**



```
let sophie = new Personne("Sophie", "RAFFES", true);
```

```
let marcel = new Personne("Marcel", "TROUSP", false);
```

```
Personne.lister([sophie, marcel]);
```


→ Personne

Mme RAFFES Sophie  
M. TROUSP Marcel

Pour invoquer la méthode statique il faut passer par l'objet classe ( la fonction constructeur)

```
1
2 class Personne {
3   constructor(prenom, nom, feminin) {
4     this.prenom = prenom;
5     this.nom = nom;
6     this.feminin = feminin; // true si une femme, false sinon
7   }
8
9   nomComplet() {
10    return `${this.nom} ${this.prenom}`;
11  }
12
13  civilite() {
14    return (this.feminin) ? "Mme" : "M.";
15  }
16
17  describe() {
18    return `${this.civilite()} ${this.nomComplet()}`;
19  }
20
21  /**
22   * affiche la description des personnes contenues dans un tableau
23   * @param {Array[Personne]} personnes le tableau des personnes
24   */
25  static lister(personnes) {
26    console.log(this.name);
27    console.log("-----");
28    for (let i = 0; i < personnes.length; i++) {
29      console.log(personnes[i].describe());
30    }
31  }
32 }
```

Méthode statique se situe non pas au niveau du prototype pour les Personnes mais au niveau de la fonction constructeur Personne

 [Voir le code](#)

# Objets et ES6 : Classes - *getters* et *setters*

- Accesseurs : fonctions *getters*

- **get** permet de lier une propriété d'un objet à une fonction qui sera appelée lorsqu'on accédera à la propriété.

```
1
2 class Personne {
3   constructor(prenom, nom, feminin) {
4     this.prenom = prenom;
5     this.nom = nom;
6     this.feminin = feminin; // true si une femme, false sinon
7   }
8
9   nomComplet() {
10    return `${this.nom} ${this.prenom}`;
11  }
12
13  civilite() {
14    return (this.feminin) ? "Mme" : "M.";
15  }
16
17  describe() {
18    return `${this.civilite()} ${this.nomComplet}`;
19  }
20
21  /**
22   * affiche la description des personnes contenues dans un tableau
23   * @param {Array[Personne]} personnes le tableau des personnes
24   */
25  static lister(personnes) {
26    console.log(this.name);
27    console.log("-----");
28    for (let i = 0; i < personnes.length; i++) {
29      console.log(personnes[i].describe());
30    }
31  }
32 }
```

let sophie = new Personne("Sophie", "RAFFES", true);

console.log(sophie.civilite()); → 'Mme'


console.log(sophie.nomComplet()); → 'RAFFES Sophie'

Méthodes transformées en getters

```
1
2 class Personne {
3   constructor(prenom, nom, feminin) {
4     this.prenom = prenom;
5     this.nom = nom;
6     this.feminin = feminin;
7   }
8
9   get nomComplet() {
10    return `${this.nom} ${this.prenom}`;
11  }
12
13  get civilite() {
14    return (this.feminin) ? "Mme" : "M.";
15  }
16
17  describe() {
18    return `${this.civilite} ${this.nomComplet}`;
19  }
20
21  /**
22   * affiche la description des personnes contenues dans un tableau
23   * @param {Array[Personne]} personnes le tableau des personnes
24   */
25  static lister(personnes) {
26    console.log(this.name);
27    console.log("-----");
28    for (let i = 0; i < personnes.length; i++) {
29      console.log(personnes[i].describe());
30    }
31  }
32 }
```

console.log(sophie.civilite); → 'Mme'

console.log(sophie.nomComplet); → 'RAFFES Sophie'

 [Voir le code](#)

Les getters peuvent être utilisés comme des propriétés

# Objets et ES6 : Classes - *getters* et *setters*

- Mutateurs (fonctions *setters*)

- **set** permet de lier une propriété d'un objet à une fonction qui sera appelée lorsqu'on affectera la propriété.

```
let p1 = new Personne("Sophie", "RAFFES", true);
```

```
console.log(p1.nomComplet);    → 'RAFFES Sophie'
```

```
p1.nomComplet = "  Maeva  MERLIN";
```

Appel du setter

```
console.log(p1.prenom);    → 'Maeva'
```

```
console.log(p1.nom);    → 'MERLIN'
```

```
let date = {
  jour : '17',
  mois : 'Novembre',
  annee : '2020',
  get dateComplete() {
    return `${this.jour} ${this.mois} ${this.annee}`;
  },
  set dateComplete(dateString) {
    const tokens = dateString.split(/\b\s+(?!$)/);
    this.jour = parseInt(tokens[0]);
    this.mois = tokens[1];
    this.annee = parseInt(tokens[2]);
  }
};
```

On peut aussi définir getters et setters dans des objets littéraux

 [Voir le code](#)

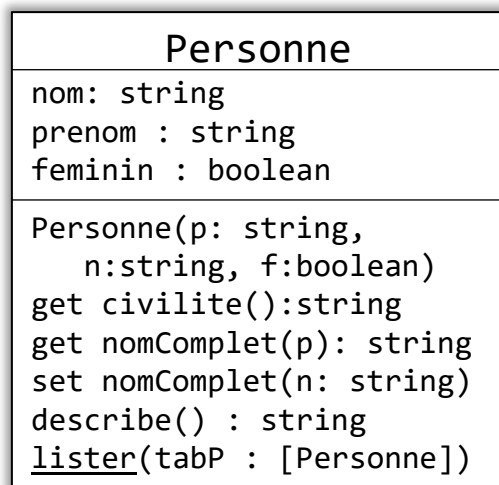
```
1
2 class Personne {
3   constructor(prenom, nom, feminin) {
4     this.prenom = prenom;
5     this.nom = nom;
6     this.feminin = feminin;
7   }
8
9   get nomComplet() {
10    return `${this.nom} ${this.prenom}`;
11  }
12
13  set nomComplet(prenomNom) {
14    const tokens = prenomNom.split(/\b\s+(?!$)/); // expression régulière*
15    this.prenom = tokens[0];
16    this.nom = tokens[1];
17  }
18
19  get civilite() {
20    return (this.feminin) ? "Mme" : "M.";
21  }
22
23  describe() {
24    return `${this.civilite} ${this.nomComplet}`;
25  }
26
27  /**
28   * affiche la description des personnes contenues dans un tableau
29   * @param {Array[Personne]} personnes le tableau des personnes
30   */
31  static lister(personnes) {
32    console.log(this.name);
33    console.log("-----");
34    for (let i = 0; i < personnes.length; i++) {
35      console.log(personnes[i].describe());
36    }
37  }
38 }
```

\* <https://blog.abelotech.com/posts/split-string-into-tokens-javascript/>

 [Voir le code](#)

# Objets et ES6 : Classes - Héritage

- Héritage – possibilité de définir une classe comme étendant une classe existante (sous classe)



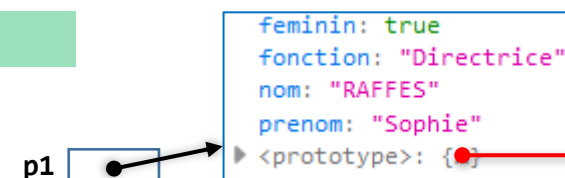
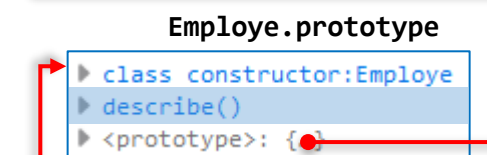
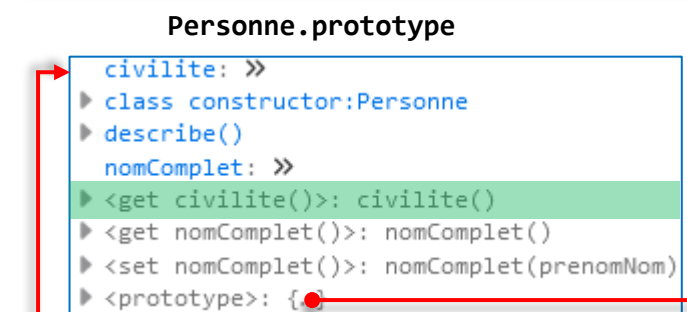
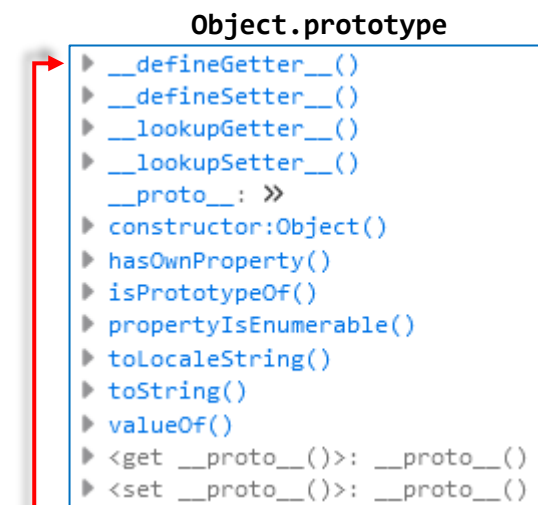
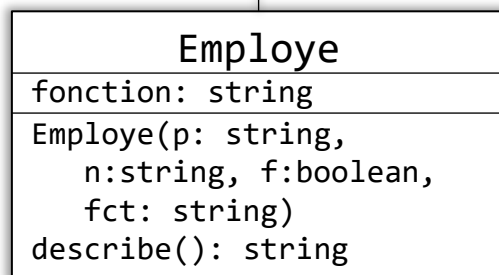
On veut exprimer le fait qu'un employé c'est une personne mais qui en plus a une fonction.

Possibilité d'obtenir ce comportement avec les chaînes de prototypes en JavaScript

```
let p1 = new Employe("Sophie", "RAFFES", true, "Directrice");
```

```
console.log(p1.describe()); → 'Mme RAFFES Sophie (Directrice)'
```

```
console.log(p1.civilite); → 'Mme'
```



# Objets et ES6 : Classes - Héritage

- Héritage – `extends` permet d'exprimer cette relation d'héritage (et de mettre en place la chaîne de prototypes)

```
class Personne {
  constructor(prenom, nom, feminin) {
    this.prenom = prenom;
    this.nom = nom;
    this.feminin = feminin;
  }

  get nomComplet() {
    return `${this.nom} ${this.prenom}`;
  }

  set nomComplet(prenomNom) {
    const tokens = prenomNom.split(/\b\s+(?!$)/);
    this.prenom = tokens[0];
    this.nom = tokens[1];
  }

  get civilite() {
    return (this.feminin) ? "Mme" : "M.";
  }

  describe() {
    return `${this.civilite} ${this.nomComplet}`;
  }

  static lister(personnes) {
    console.log(this.name);
    console.log("-----");
    for (let i = 0; i < personnes.length; i++) {
      console.log(personnes[i].describe());
    }
  }
}
```

`super()` appelle le constructeur de la super classe. Cet appel doit être fait avant d'accéder à `this`

```
class Employee extends Personne {
  constructor(prenom, nom, feminin, fonction) {
    super(prenom, nom, feminin);
    this.fonction = fonction;
  }

  describe() {
    return super.describe() + ` (${this.fonction})`;
  }
}
```

`super.nomMethode()` invoque une méthode héritée

```
let p1 = new Employee("Sophie", "RAFFES", true, "Directrice");
```

```
console.log(p1.describe()); → 'Mme RAFFES Sophie (Directrice)'
```

```
console.log(p1.civilite); → 'Mme'
```

 [Voir le code](#)

```
Object.prototype
  ▶ __defineGetter__()
  ▶ __defineSetter__()
  ▶ __lookupGetter__()
  ▶ __lookupSetter__()
  ▶ __proto__: >>
  ▶ constructor: Object()
  ▶ hasOwnProperty()
  ▶ isPrototypeOf()
  ▶ propertyIsEnumerable()
  ▶ toLocaleString()
  ▶ toString()
  ▶ valueOf()
  ▶ <get __proto__(): __proto__()
  ▶ <set __proto__(): __proto__()
```

```
Personne.prototype
  ▶ civilite: >>
  ▶ class constructor: Personne
  ▶ describe()
  ▶ nomComplet: >>
  ▶ <get civilite(): civilite()
  ▶ <get nomComplet(): nomComplet()
  ▶ <set nomComplet(): nomComplet(prenomNom)
  ▶ <prototype>: {}
```

```
Employee.prototype
  ▶ class constructor: Employee
  ▶ describe()
  ▶ <prototype>: {}
```

```
p1
  ▶ feminin: true
  ▶ fonction: "Directrice"
  ▶ nom: "RAFFES"
  ▶ prenom: "Sophie"
  ▶ <prototype>: {}
```

# Objets et ES6 : Classes - Héritage

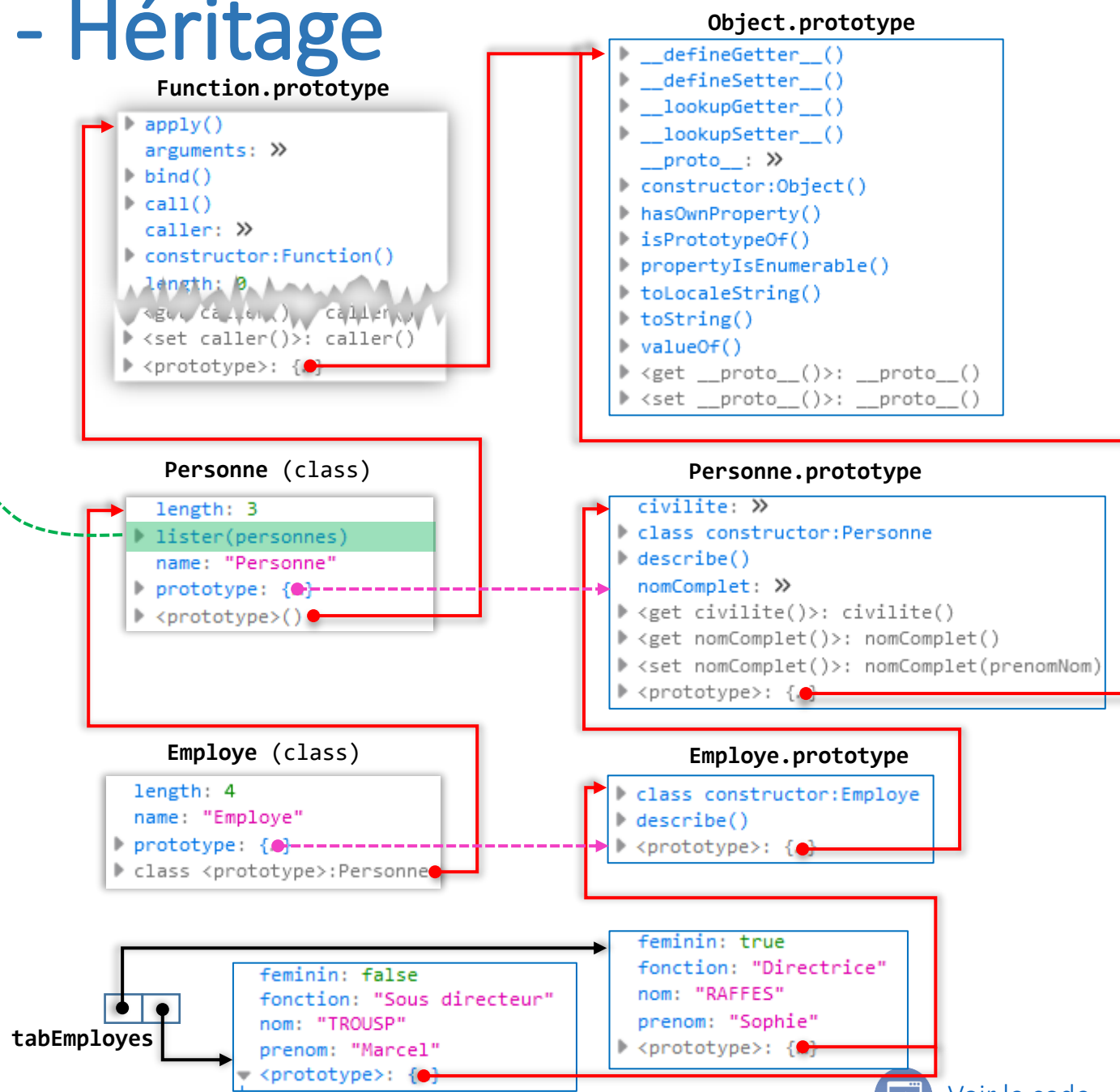
- Héritage – extends défini aussi une chaîne de <prototype> au niveau des classes (fonctions constructeurs) → héritage des membres statiques

```
class Personne {  
  // ...  
  static lister(personnes) {  
    console.log(this.name);  
    console.log("-----");  
    for (let i = 0; i < personnes.length; i++) {  
      console.log(personnes[i].describe());  
    }  
  }  
}  
  
class Employee extends Personne {  
  // ...  
}
```

```
let tabEmployes = [  
  new Employee("Sophie", "RAFFES", true, "Directrice"),  
  new Employee("Marcel", "TROUSP", false, "Sous directeur")  
];
```

```
Personne.lister(tabEmployes); → Personne  
-----  
Mme RAFFES Sophie (Directrice)  
M. TROUSP Marcel (Sous directeur)
```

```
Employee.lister(tabEmployes); → Employee  
-----  
Mme RAFFES Sophie (Directrice)  
M. TROUSP Marcel (Sous directeur)
```



# Objets : opérateur instanceof

- `x instanceof C` renvoie `true` si l'objet référencé par `x` est instance de la classe `C` ou d'une sous classe de `C` (plus précisément si `C.prototype` se trouve dans la chaîne de `<prototypes>` de `x`), `false` sinon

```
let p1 = new Employe("Sophie", "RAFFES", true, "Directrice");
```

```
p1 instanceof Employe → true
```

```
p1 instanceof Personne → true
```

```
p1 instanceof Object → true
```

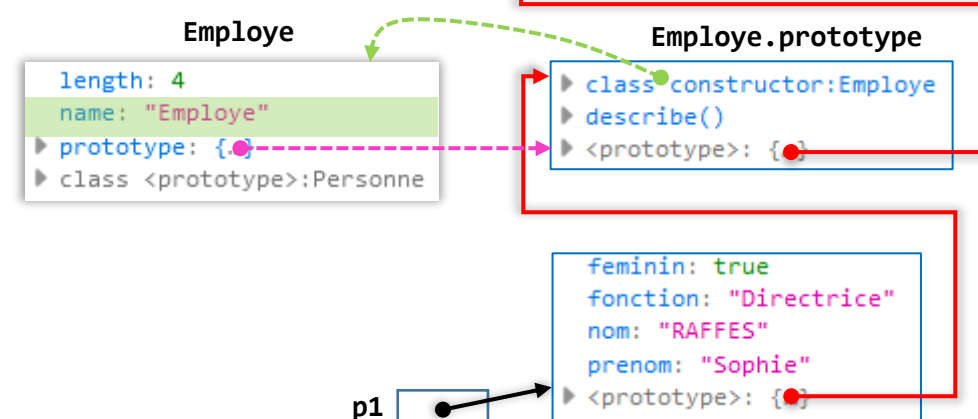
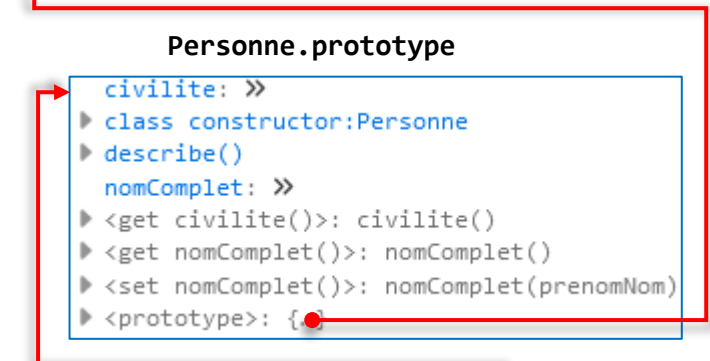
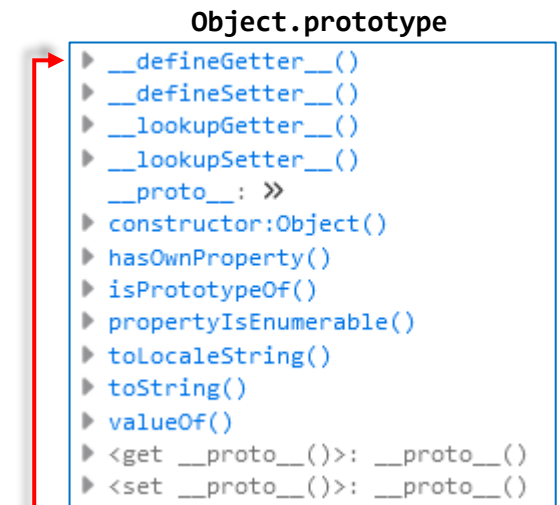
```
let p2 = new Personne("Maeva", "FASFER", true);
```

```
p2 instanceof Employe → false
```

```
typeof p1 → object
```

Nom de la classe qui a permis de créer p1

```
p1.constructor.name → Employe
```



# Objets JavaScript

- Quelques liens
  - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details of the Object Model](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model)
  - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>
  - [https://www.w3schools.com/js/js\\_object\\_definition.asp](https://www.w3schools.com/js/js_object_definition.asp)
  - [https://www.w3schools.com/js/js\\_class\\_intro.asp](https://www.w3schools.com/js/js_class_intro.asp)
  - <http://blog.xebia.fr/2013/06/10/javascript-retour-aux-bases-constructeur-prototype-et-heritage/>