Fonction constructeur

Une fonction classique peut définir un constructeur, pas une fonction flèchée. Par convention le nom de la fonction commencera par une majuscule :

```
function User(name){
  // constructeur
  this.name = name;

  console.log(this.name);
}

const u1 = new user("Alan");
const u2 = new user("Alan");

// Le code qui suit produira une erreur
// pas de constructeur dans ce cas
/*const userArrow = name => {
  this.name = name;

  console.log(this.name);
}

const uA1 = new userArrow("Alan");
const uA2 = new userArrow("Alan");
*/
```

Remarque : si vous appelez la fonction constructeur comme une fonction classique, alors le this sera de type "undefined" en mode strict.

```
'use strict';

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

User('Alan'); // this undefined
```

Lorsque vous définissez une méthode dans un objet littéral, le this est l'objet littéral lui-même.

```
const Model = {
    table : "Model",

    subModel:function(){
        console.log(this); // Objet model
    },

    // de manière totalement équivalente vous pour écrire ceci
    // pour définir une méthode/fonction
    subModel2(){
      console.log(this); // Objet model
    }
}

Model.subModel(); // this objet Model
```

Introduction à la notion de prototype pour une fonction

```
const Student = {
  name : '',
  average : 17.5,
  situation: function(){
    console.log(`Name ${this.name} average : ${this.average}`);
  }
}
```

Cet objet possède une propriété **prototype**, elle listera l'ensemble des propriétés héritées depuis l'objet Student. La quasi-totalité des objets JS héritent de l'objet **Object** de JS.

```
Student.__proto__
```

Vous pouvez dès lors appeler des méthodes, qui ne sont pas directement héritées dans l'objet Student.

(Application) Ajouter une propriété sur un constructeur

Reprenons l'exemple précédent, nous allons voir comment ajouter une propriété au constructeur User qui sera partagée par toutes ses instances :

```
'use strict';

function User(name, lastname){
  this.name = name;
  this.lastname = lastname;
}

let u1 = new User('Alan', 'Phi');

// On ajoute sur le constructeur lui-même la propriété
User.prototype.fullName = function (){

  return this.name + ' ' + this.lastname;
}

console.log(u1.fullName()); // Alan Phi
```

01 Exercice prototype average pour la fonction User

Ajoutez la possibilité de définir l'âge dans la fonction constructeur User. Modifiez pour toutes les instances de User la fonction fullName pour qu'elle affiche le name, le fullName et l'âge d'un user.

Créez maintenant les 4 users suivants :

```
- Alan Phi age 45 ans notes : 15, 17, 13
- Bernad Lu age 78 ans notes : 11, 12, 9
- Sophie Boo age 56 ans notes : 10, 15, 11
- Alice Car age 45 ans notes : 5, 18, 20
```

Créez un nouveau prototype average dans la fonction constructeur User, qui calculera la moyenne des notes de chaque user.

Quand JS appelle cette méthode il ne la trouvera pas dans l'instance de User mais dans son prototype. Cette technique permet donc de créer des méthodes partagées par toutes les instances. Notez que vous pouvez tout à fait définir la méthode fullName après avoir fait son instance.

JS possède depuis **ES6** un mot clé class pour définir une classe, nous verrons qu'en fait ce mot clé permet de définir, comme dans l'exemple précédent, un constructeur.

Modifié le: vendredi 22 septembre 2023, 13:07