

# This et fonctions

Il existe plusieurs manières d'invoquer une fonction en JavaScript :

- ◆ comme une fonction (oui... )
- ◆ comme une méthode
- ◆ comme un constructeur
- ◆ avec `apply()` ou `call()`

Les fonctions sont un concept clef du langage et certaines spécificités sont souvent mal comprises lorsque l'on vient d'autres langages (comme JAVA !).

Lorsque l'on appelle une fonction (quelque soit la méthode) ; le moteur JavaScript ajoute automatiquement (et implicitement) 2 paramètres : **this** et **arguments**.

# This et fonctions

Appeler une fonction... “comme une fonction”.

C’est le premier type d’appels, le plus simple.

```
1 function exemple(param) {  
2   console.log(this, arguments);  
3 }  
4  
5 exemple("test");
```

Run >  
Reset

> [object Window] Object { 0: "test" }

“Contexte” de la fonction

Paramètres de la fonction

Lorsque l’on appelle une fonction de cette manière, la variable **this** correspond à l’objet **window** (dans un contexte web) ou **global** (contexte node).

# This et fonctions

Seconde manière d'appeler une fonction : comme une méthode.

L'idée est d'attacher une fonction à un objet ; et d'appeler cette fonction sur l'objet.

C'est de cette manière que l'on peut faire de la programmation orientée objet en JavaScript.

```
1 var robot = {  
2   prenom: "Henri"  
3 };  
4  
5 robot.direBonjour = function() {  
6   console.log(this);  
7   console.log("Bonjour");  
8 }  
9  
10 robot.direBonjour();
```

Run >

Reset

```
> Object { prenom: "Henri", direBonjour: function() {  
  console.log(this);  
  console.log("Bonjour");  
} }  
> "Bonjour"
```

Dans ce cas, la variable `this` correspond à l'objet "robot".

On retrouve le sens classique de la variable "this" tel qu'on peut le rencontrer dans des langages comme JAVA.

# This et fonctions

Troisième manière d'appeler une fonction : comme un constructeur avec le mot clef **new**.

```
1 function exemple(param) {  
2   console.log(this, arguments);  
3 }  
4  
5 exemple("test");  
6  
7 new exemple("test");
```

Lorsque l'on utilise new :

- ◆ un nouvel objet (vide) est créé
- ◆ cet objet est passé au constructeur et correspond au **this**
- ◆ en l'absence d'un return dans le constructeur, ce nouvel objet est renvoyé implicitement.

Run >

Reset

```
> [object Window] Object { 0: "test" }  
> [object Object] Object { 0: "test" }
```

# This et fonctions

L'invocation par constructeur est pratique pour définir des objets en POO.

Sans utiliser cette méthode, pour créer deux objets de la même classe, on devrait écrire :

```
1 function direBonjour() {  
2   console.log("Bonjour");  
3 };  
4  
5 var robot = {  
6   prenom: "Henri",  
7   direBonjour : direBonjour  
8 };  
9  
10 var robot2 = {  
11   prenom: "Jacques",  
12   direBonjour : direBonjour  
13 };
```



```
1 function Robot(prenom) {  
2   this.prenom = prenom;  
3   this.direBonjour = function() {  
4     console.log("Bonjour");  
5   };  
6 }  
7  
8 var henri = new Robot("Henri");  
9 var jacques = new Robot("Jacques");  
10  
11 henri.direBonjour();  
12 jacques.direBonjour();
```

Run >

Reset

> "Bonjour"

> "Bonjour"

# This et fonctions

Une fonction dispose de ses propres méthodes. En effet, toute fonction en JavaScript est un objet Function. Cet objet permet d'utiliser certaines méthodes dont `apply()` et `call()`.

La dernière manière d'invoquer une fonction est de faire appel à `apply()` et `call()`.

`Apply` comme `call` permettent d'appeler une fonction tout en surchargeant son contexte, c'est à dire la valeur de `this`.

La différence résulte dans la signature de la fonction au niveau du passage des arguments (via un tableau ou non).

```
1 function direBonjour(nomInterlocuteur) {  
2   console.log(this);  
3   console.log("Bonjour " + nomInterlocuteur);  
4 }  
5  
6 direBonjour.apply({}, ["Jean"]);  
7 direBonjour.call({}, "Jean");
```

Run >

Reset

```
> Object { }  
> "Bonjour Jean"  
> Object { }  
> "Bonjour Jean"
```