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**.



Appeler une fonction... "comme une fonction".

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

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).



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.

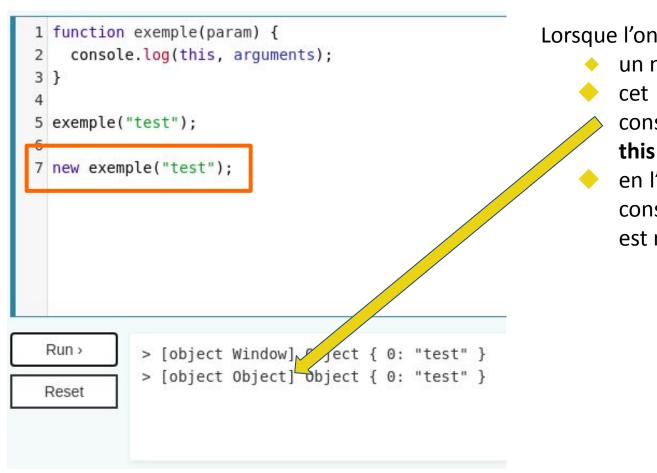
```
1 var robot = {
    prenom: "Henri"
3 };
5 robot.direBonjour = function() {
     console.log(this);
     console.log("Bonjour");
8
10 robot.direBonjour();
  Run
             > Object { prenom: "Henri", direBonjour: function() {
               console.log(this);
  Reset
               console.log("Bonjour");
             } }
             > "Boniour"
```

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.



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



Lorsque l'on utilise new :

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



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() {
     console.log("Bonjour");
3 };
  var robot = {
     prenom: "Henri",
    direBonjour : direBonjour
8
  };
10 var robot2 = {
     prenom: "Jacques",
11
     direBonjour : direBonjour
12
13 };
```

```
1 function Robot(prenom) {
    this.prenom = prenom;
    this.direBonjour = function() {
       console.log("Bonjour");
    };
8 var henri = new Robot("Henri");
9 var jacques = new Robot("Jacques");
10
11 henri.direBonjour();
12 jacques.direBonjour();
  Run
            > "Bonjour"
            > "Bonjour"
  Reset
```



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");
```

