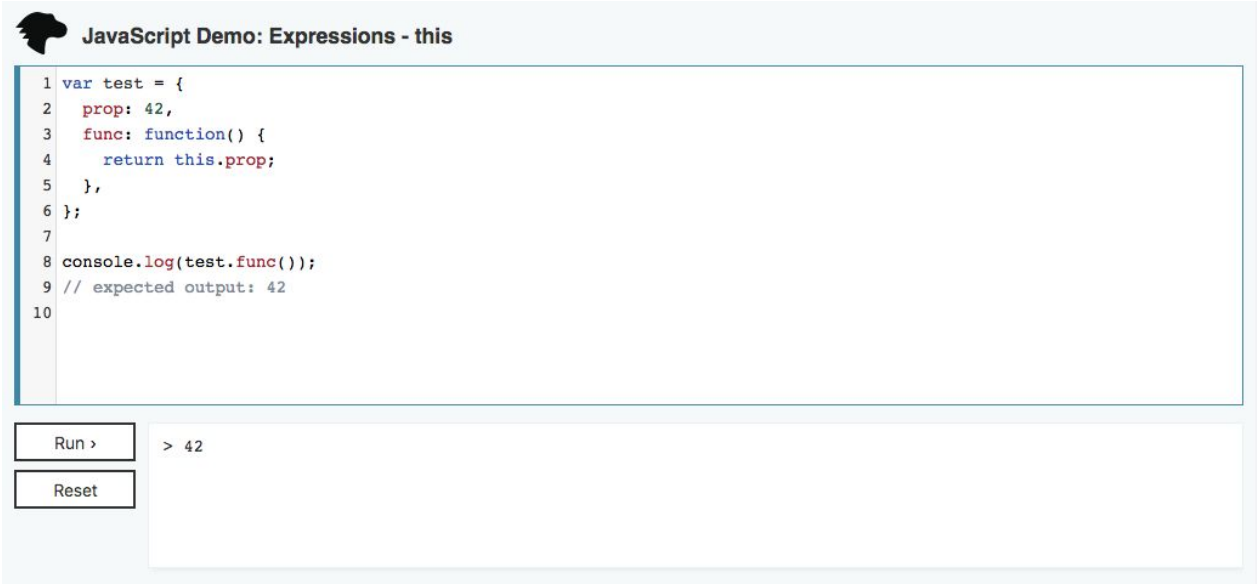
L’opérateur This.

En JavaScript, **le mot-clé this** se comporte légèrement différemment des autres langages de programmation. Son comportement variera également légèrement selon qu'on utilise le mode strict ou le mode non-strict. Dans la plupart des cas, la valeur de this sera déterminée à partir de la façon dont une fonction est appelée. Il n'est pas possible de lui affecter une valeur lors de l'exécution et sa valeur peut être différente à chaque fois que la fonction est appelée. La méthode bind a été introduite avec ECMAScript 5 pour définir la valeur de this pour une fonction, indépendamment de la façon dont elle est appelée. ECMAScript 2015 (ES6) a ajouté les fonctions fléchées dans lesquelles this correspond à la valeur du contexte englobant.



Dans le contexte global

Dans le contexte global d'exécution (c'est-à-dire, celui en dehors de toute fonction), this fait référence à l'objet global (qu'on utilise ou non le mode strict).

```
console.log(this.document === document); // true

// Si l'environnement de script est un navigateur,
// l'objet window sera l'objet global
console.log(this === window); // true

this.a = 37;
console.log(window.a); // 37
```

Dans le contexte d'une fonction

S'il est utilisé dans une fonction, la valeur de this dépendra de la façon dont la fonction a été appelée

```
function f1(){
  return this;
}

// Dans un navigateur
f1() === window; // true (objet global)

// Côté serveur (ex. Node)
f1() === global; // true
```

Dans cet exemple, la valeur de `this` n'est pas définie lors de l'appel. Le code n'étant pas en mode strict, `this` doit toujours être un objet et ce sera donc l'objet global.

```
function f2(){
  "use strict"; // on utilise le mode strict
  return this;
}

f2() === undefined; // true
```

En mode strict, la valeur de `this` est conservée (il reste le même) entre le moment de sa définition et l'entrée dans le contexte d'exécution. S'il n'est pas défini, il reste `undefined`. Il pourrait être défini avec n'importe quelle autre valeur, telle que `null` ou `42` ou `"Je ne suis pas this"`.

CALL ET APPLY

Pour passer ‘this’ d’un context à un autre, on pourra utiliser ‘call’ et ‘apply’ :

Lorsque le mot-clé this est utilisé dans le corps d’une fonction, il est possible d’utiliser les méthodes call() et apply() pour lier ‘this’ à un objet donné. Toutes les fonctions héritent de ces 2 méthodes grâce à function.prototype.

```
1  // Un objet peut être passé en premier argument
2  // de call ou de apply
3  var obj = { a: "Toto" };
4
5  // Ici, on définit une propriété sur l'objet
6  // global
7  var a = "Global";
8
9  function whatsThis(arg) {
10   // La valeur de this ici dépend de la façon
11   // dont la fonction est appelée
12   return this.a;
13 }
14
15 whatsThis();          // "Global"
16 whatsThis.call(obj);  // "Toto"
17 whatsThis.apply(obj); // "Toto"
```

Note : Si la valeur a lier a ‘this’, passée à call() ou apply(), n’est pas un objet, le moteur JavaScript tentera de la convertir en un objet grâce à l’opération interne ToObject. Si la valeur est d’un type primitif autre qu’objet, exemple 7 ou ‘toto’, elle sera convertie en un objet grâce à new Number(7) et la chaîne ‘toto’ convertie en objet grâce à new String(‘toto’).

o est l’objet que l’on souhaite lié à ‘this’ dans la fonction appelé.

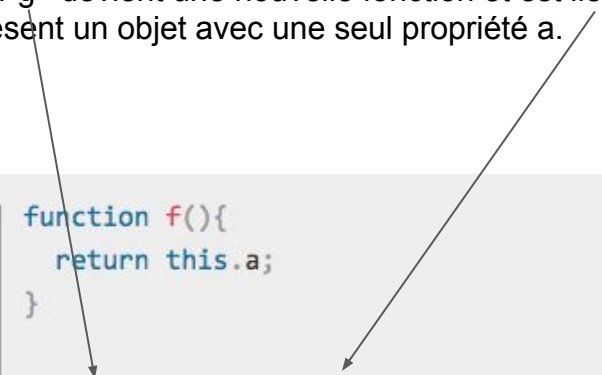
{a:1, b:3} correspond à la définition de l’objet o, ça aurait pus être des fonctions, un constructeur ou toute chose qui constitue notre objet..
C’est donc cela qui est utilisé dans la fonction.

```
1  function ajout(c, d){
2    return this.a + this.b + c + d;
3  }
4
5  var o = {a:1, b:3};
6
7  // Le premier paramètre correspond à l'objet qu'on souhaite
8  // lier à 'this', les paramètres suivants sont les arguments
9  // à utiliser dans l'appel de la fonction
10 ajout.call(o, 5, 7); // 1 + 3 + 5 + 7 = 16
11
12 // Le premier paramètre correspond à l'objet qu'on souhaite
13 // lier à 'this', le second paramètre est le tableau dont les
14 // les éléments sont les arguments à passer à la fonction
15 ajout.apply(o, [10, 20]); // 1 + 3 + 10 + 20 = 34
```

La méthode bind()

Avec ECMAScript 5, une nouvelle fonction fut introduite: Function.prototype.bind, Lorsqu'on appel f.bind(unObjet), on crée une nouvelle fonction qui possède le même corps et la même la même portée que f, mais ou 'this' sera lié, de façon permanente, au premier argument passé à bind(), quelle que soit la façon dont la méthode est utilisée.

'var g ' devient une nouvelle fonction et est lié de façon permanente au premier argument passé à bind() qui est dans le cas présent un objet avec une seul propriété a.



```
function f(){
  return this.a;
}

var g = f.bind({a:"azerty"});
console.log(g()); // azerty

var h = g.bind({a:"coucou"}); // bind ne fonctionne qu'une seule fois
console.log(h()); // azerty

var o = {a:37, f:f, g:g, h:h};
console.log(o.f(), o.g(), o.h()); // 37, azerty, azerty
```