

4. Fonctions

Jean-Luc Falcone

Août 2019

Fonction en retour

On peut créer une fonction qui définit et retourne une autre fonction. Celle-ci encapsulera son environnement:

```
function add( i ) {  
  return function( j ) {  
    return i + j;  
  };  
}
```

Fonction en retour

On peut créer une fonction qui définit et retourne une autre fonction. Celle-ci encapsulera son environnement:

```
function add( i ) {  
  return function( j ) {  
    return i + j;  
  };  
}
```

```
let plus2 = add(2);  
> plus2  
[Function]  
> plus2(3);  
5  
> plus2("hello");  
'2hello'  
> add(100)(2);  
102
```

Fermeture (*closure*): 1

Une fonction qui compte le nombre d'appels:

```
let cnt = 0;  
function count() {  
  cnt += 1;  
  return cnt;  
}
```

Fermeture (*closure*): 1

Une fonction qui compte le nombre d'appels:

```
let cnt = 0;  
function count() {  
  cnt += 1;  
  return cnt;  
}
```

```
> count()  
1  
> count()  
2  
> cnt = 100;  
100  
> count()  
101
```

Fermeture (*closure*): 2

On peut faire mieux en utilisant le fait qu'une fonction est une fermeture:

```
function initCounter() {  
  let cnt = 0;  
  return function() {  
    cnt += 1;  
    return cnt;  
  };  
}
```

Fermeture (*closure*): 2

On peut faire mieux en utilisant le fait qu'une fonction est une fermeture:

```
function initCounter() {  
  let cnt = 0;  
  return function() {  
    cnt += 1;  
    return cnt;  
  };  
}
```

```
> let count = initCounter();  
> count()  
1  
> count()  
2  
> let count2 = initCounter();  
> count2()  
1  
> count()  
3
```

Fonction en paramètre

On peut écrire une fonction qui en prend une autre en paramètre:

```
function prettyPrint( x, format ) {  
  console.log( format(x) );  
}  
  
> let point = { x: 2.5, y: -1.0 }  
> let pointFormat = function( p ) {  
  return "Point(" + p.x + ", " + p.y + ")";  
};  
> prettyPrint( point, pointFormat );  
Point(2.5, -1)  
undefined
```


Fonctions anonymes

On peut utiliser les fonctions anonymes pour créer une fonction à usage unique:

```
> prettyPrint( x, function( p ) {  
    return "Point(" + p.x + ", " + p.y + ")";  
});  
Point(2.5, -1)  
undefined
```

Notation flèche (ES6)

Pour les fonctions courtes, on peut utiliser une notation abrégée:

```
> prettyPrint( x, (p) => p )  
{ x: 2.5, y: -1 }  
undefined
```

```
> prettyPrint( x, (p) => "X=" + p.x + ";Y=" +p.y );  
X=2.5;Y=-1  
undefined
```

Combiner des fonctions

On peut définir une fonction qui combine des fonctions:

```
function compose( f, g ) {  
  return function(x) {  
    return f( g( x ) );  
  };  
}  
  
let absSqrt = compose( Math.sqrt, Math.abs );  
> absSqrt( 100 );  
10  
> absSqrt( -100 );  
10
```

Foreach

On peut utiliser la méthode `forEach` pour effectuer une action sur chaque élément:

```
function sum( xs ) { //BAD !
  let s = 0;
  for( let i=0; i < xs.length; i++ ) {
    s += xs[i];
  }
  return s;
}
```

```
function sum( xs ) { //GOOD !
  let s = 0;
  xs.forEach( (x) => s += x );
  return s;
}
```

Map

Souvent on veut créer un second tableau en appliquant une fonction sur chaque élément d'un premier tableau. On utilisera pour cela la fonction `map`:

```
let points = [ {x:1,y:0}, {x:0,y:2}, {x:-1, y:1} ];  
let xs = points.map( (p) => p.x );  
> xs  
[ 1, 0, -1 ]  
> points  
[ { x: 1, y: 0 }, { x: 0, y: 2 }, { x: -1, y: 1 } ]
```

Filter

Souvent, on veut créer un second tableau en filtrant un premier:

```
let xs = [1,2,3,4,5,6,7,8,9,10];  
function isEven(x) {  
  return x % 2 == 0;  
}  
let even = xs.filter( isEven );  
let odd = xs.filter( x => ! isEven(x) );  
> even  
[ 2, 4, 6, 8, 10 ]  
> odd  
[ 1, 3, 5, 7, 9 ]
```

Reduce

On peut réduire un tableau en une seule valeur grâce à reduce:

```
let xs = [1,2,3,4,5,6,7,8,9];  
let sum = xs.reduce( (x,y) => x+y );  
let prod = xs.reduce( (x,y) => x*y );
```

```
> sum  
45  
> prod  
362880
```

Arguments

On peut appeler n'importe quelle fonction avec un nombre arbitraire d'arguments. Cette "fonctionnalité" génère une nouvelle famille de bugs...

```
function f(x) {  
  return x + 1;  
}
```

```
> f(2)
```

```
3
```

```
> f()
```

```
NaN
```

```
> f(100,10)
```

```
101
```


Liste d'arguments

Chaque fonction reçoit une référence vers un objet contenant la liste des arguments effectivement passés à la fonction. On peut le convertir en tableau pour pouvoir l'utiliser plus facilement

```
function average() {  
  let n = arguments.length;  
  if( n === 0 ) {  
    return undefined;  
  } else {  
    let xs = Array.from(arguments);  
    let sum = xs.reduce( (x,y)=> x+y );  
    return sum / n;  
  }  
}
```

```
> average()  
undefined  
> average(100);  
100  
> average(1,2,3,4,5);  
3  
> average(1,2,3,4);  
2.5
```