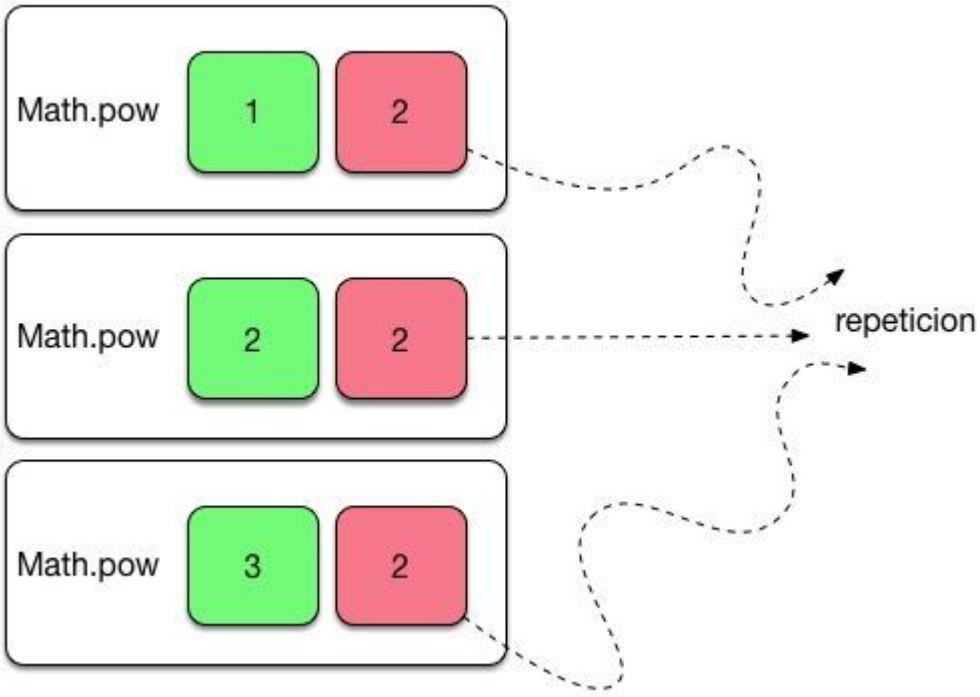
El concepto de JavaScript currying siempre es complicado de entender al principio. Sin embargo una gran parte de la programación funcional se apoya en él . Vamos a abordar este concepto con los típicos ejemplos de matemáticas. Supongamos por un momento que tenemos que usar JavaScript para elevar un numero al cuadrado. Se trata de una operación elemental es suficiente con usar Math.pow.

```
console.log(Math.pow(1,2));
console.log(Math.pow(2,2));
console.log(Math.pow(3,2));
console.log(Math.pow(4,2));
console.log(Math.pow(5,2));
```

Nos imprime el cuadrado de los primeros 5 números por pantalla.

¿Es el código correcto? , por supuesto que sí no tiene nada de especial. Sin embargo si lo miramos un poco más a detalle nos daremos cuenta de que es muy repetitivo ya que continuamente calculamos el cuadrado un número cualesquiera.



Una simplificación rápida es construir la función cuadrado:

```
var cuadrado= function(numero) {

return Math.pow(numero,2);

}

console.log(cuadrado(1));
console.log(cuadrado(2));
console.log(cuadrado(3));
console.log(cuadrado(4));
console.log(cuadrado(5));
```

El resultado será idéntico pero mejor:

JavaScript Currying

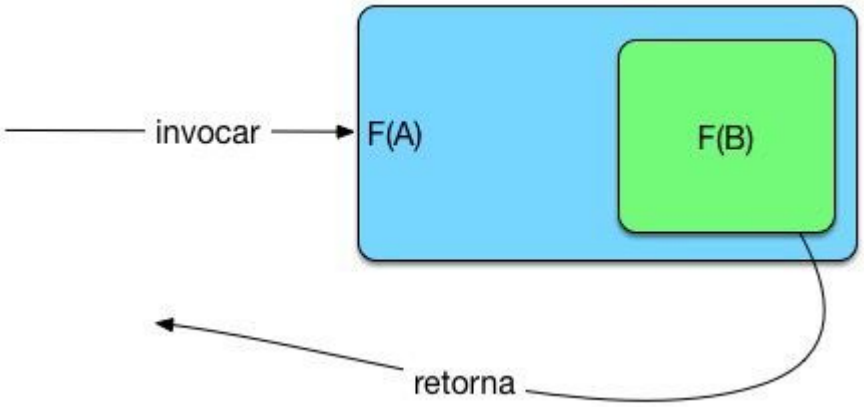
Lamentablemente nos quedamos con una sensación de que no lo hemos hecho todo lo bien que debieramos ya que la función cuadrado parece poco flexible y la de Math.pow lo era mucho más. ¿ Qué podemos hacer? . Podemos usar el concepto de JavaScript currying y diseñar una función que contenga otra y encargue de almacenar una parte del código . A esta función parcial la denominaremos exponencial.

```
var exponencial = function(numero2) {

return function(numero1) {

return Math.pow(numero1, numero2);
 }
}
```

Se trata de una función que devuelve otra función (high order function) .



A partir de ahora será muy sencillo calcular los cuadrados sin repetir código ,bastará con:

```
var cuadrado= exponencial(2);
```

Invocamos a la función exponencial con un 2 y nos construirá una función cuadrado que podremos usar :

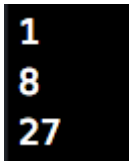
```
console.log(cuadrado(1))
console.log(cuadrado(2))
```

Parece que no hemos avanzado nada y estamos en la misma situación que antes. Sin embargo ahora habiendo aplicado JavaScript currying tenemos más flexibilidad ya que podemos definir de la misma forma la función cubo.

```
var cubo= exponencial(3);

console.log(cubo(1))
console.log(cubo(2))
console.log(cubo(3))
```

La consola nos muestra los cubos de los primeros 3 números:



Acabamos de utilizar JavaScript currying para simplificar nuestros problemas y añadir flexibilidad a las funciones.

Otros artículos relacionados: El concepto de JavaScript Spread operator JavaScript pure functions y su uso ,El uso de JavaScript for in vs for of