El concepto de JavaScript Function Composition es uno de los conceptos clásicos de la programación funcional. En muchas ocasiones cuando construimos funciones nos puede ser útil crear nuevas funciones que combinen la funcionalidad de otras previamente construidas.

Vamos a ver un par de ejemplos:

```
function cuadrado(numero) {

  return numero*numero;

}
function incremento(numero) {

  return ++numero;
}
numero=1;
console.log(cuadrado(incremento(numero)));
```

En este caso tenemos dos funciones una que eleva al cuadrado un numero y otra función que incrementa un numero en uno. Aplicamos ambas funciones sobre un valor e imprimimos el resultado:

```
    cuadrado(incremento(numero))
```

¿Podemos enfocar de otra manera a la hora de definir las funciones? . La realidad es que sí. Para ello vamos a convertir nuestras funciones en named functions con lo cual podamos acceder a ellas como variables.

```
var cuadrado= function (numero) {

  return numero*numero;

}
var incremento= function(numero) {

  return ++numero;
}
numero=1;
console.log(cuadrado(incremento(numero)));
```

El resultado será idéntido:

Una vez hecho esto podemos usar el concepto de JavaScript Function Composition y crear

una función que se encargue de componer otras funciones.

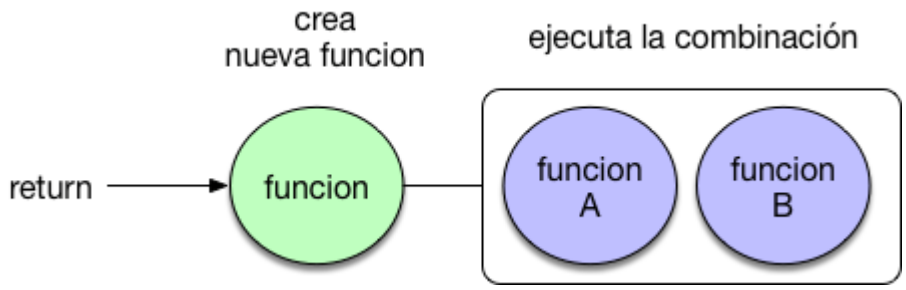
```
var composicion= function(f1,f2) {

            return function(x) {

              return f2(f1(x));
            }

}
```

Acabamos de construir una high order function que recibe como parámetros dos funciones y nos devuelve una nueva función que combina la funcionalidad de ambas. El código es complejo de entender. Básicamente lo que hacemos es retornar una nueva función que combina la funcionalidad de las dos anteriores.



Vamos a verlo en ejecución.

```
var incrementoCuadrado=composicion(incremento,cuadrado);
```

```
console.log(incrementoCuadrado(1));
```

El resultado se mantiene:

Acabamos de construir una nueva función que combina la funcionalidad de las dos anteriores. Esta combinación nos es muy útil para abordar nuestro ejemplo . Pero existen ocasiones en las cuales se necesita mayor flexibilidad y que la función de partida pueda disponer de n parámetros.

```
var composicion= function(f1,f2) {

return function(...args) {

return f2(f1(...args));
}

}
```

En este caso hemos usado el concepto de rest parameters para añadir flexibilidad lo cual nos permite trabajar con n parámetros de entrada y otras funciones.

```
var suma= function (numero1,numero2) {
      return numero1+numero2;
}
```

Acabamos de crear una nueva función que realiza la operación de suma. Podemos utilizar

JavaScript Function Composition y generar una nueva función.

```
var sumaCuadrado=composicion(suma,cuadrado);

console.log(sumaCuadrado(2,2));
```

Hemos realizado la operación de suma de dos números y elevado al cuadrado el número con una función que combina las dos anteriores.

1. Utilizando JavaScript template String
2. JavaScript Console log y el manejo de Objetos
3. Usando Rx Observables en JavaScript
4. JavaScript Rest Parameters