Call, apply y bind

Módulo: **Desarrollo Web en entorno cliente**

CFGS Desarrollo de Aplicaciones Web

¿Qué son las funciones?

Las funciones son objetos, por lo tanto, pueden tener propiedades y métodos:

```
function muestraMensaje(mensaje) {
   console.log(mensaje);
}

console.log(muestraMensaje.name);          // muestraMensaje
console.log(muestraMensaje.toString());    // código de la función
muestraMensaje("hola");                    // hola

var msg = muestraMensaje;
console.log(msg);                          // código de la función
msg("hola");                               // hola
console.log(msg.name);                     // muestraMensaje
console.log(msg.toString());               // código de la función
```

¿This en funciones?

Partidiendo de la siguientes definiciones:

```
var persona = {
        nombre: 'Ana',
        muestraMensaje: function() {
            console.log('Hola '+ this.nombre);
        }
};
persona.muestraMensaje();       // Hola Ana (contexto persona)
var pruebaMensaje = persona.muestraMensaje;
pruebaMensaje();      // undefined (contexto global – this = window)
var persona2 = {
        nombre: 'Pedro',
        saludar: persona.muestraMensaje
};
persona2.saludar();                  //???????????
```

¿This en funciones?

This *es dinámica, su valor es igual al objeto que se usa para invocar a una función o método (contexto).*

Algunas veces nos interesa cambiar este comportamiento y nos interesa cambiar el valor de **this**. Para ello se usan los métodos call, apply y bind.

Consideramos la siguiente función:

```
function multiplicar(num=2){
    console.log(this.valor*num)
}
```

Call: invoca a la función con el valor de this (contexo) que se ponga como primera argumento.

```
var objNum1 = { valor: 10 };
multiplicar.call(objNum1, 2);        // 20

var objNum2 = { valor: 5 };
multiplicar.call(objNum2, 2);        // 10
```

Si hubiera que pasarle más argumentos estos se pasan separados pro comas. Por ejemplo: multiplicar.call(objNum, 4, 5, 6)

Apply: es exactamente igual que call pero los argumentos se pasan como vector. En el ejemplo anterior:

```
multiplicar.apply(objNum1,[2]);        // 20
```

Bind: Este método no invoca directamente a la función (como los dos anteriores) sino que devuelve una copia, que hace referencia al objeto this que se especifique, además de los argumentos que le pasemos. Es decir, nos devuelve una referencia a la función con ese nuevo contexto.

En el ejemplo anterior:

```
var objNum3 = { valor: 3};
var multiplicacion = multiplicar.bind(objNum3, 5);
mutiplicacion();          // 15
```

En este caso, la función está asignada a un variable (copia, nuevo contexto) y la podemos invocar más adelante, e incluso podemos omitir los argumentos:

```
var multiplicacion2 = multiplicar.bind(objNum3);
multiplicacion2();        // 6
```

Si la función tuviera más argumentos, se le pasarían como al método call, separados por comas.

Otro ejemplo:

```
var obj = {
  delta: 2,
  test: function(num1, num2){
     return (num1 + num2) * this.delta;
  }
};

var cambio = { delta: 5 };

var funcionConCambio = obj.test.bind(cambio);
var resultado = funcionConCambio(2, 2);
console.log(resultado);        // (2 + 2) * 5 = 20
```

- http://blog.amatiasq.com/2012/01/javascript-conceptos-basicos-this-call-y-apply/
- https://msdn.microsoft.com/es-es/library/ff841995(v=vs.94).aspx
- https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Function/bind