Chuleta: 5 Funciones "Anónimas. Autoejecutables. Especiales. Mágicas"

Las funciones son consideradas como variables.

```
var f1, f2;
f1 = miFuncion;     → La variable f1 tendrá una referencia a la función miFuncion. Puedo llamarla con f1()
f2 = miFuncion();   → La variable f2 almacenará el valor devuelto por la función miFuncion.
```

Con las funciones anónimas auto-ejecutadas conseguimos ejecutar código y definir funciones y propiedades aisladas del resto de los elementos de script de la página, evitando posibles interferencias.

También nos permite entre otras cosas precargar valores para ejecución tardía, aislar variables y métodos para uso interno de nuestro código o definir bien las variables de contexto (*this*) sobre las que se va a ejecutar el mismo.

De dónde nacen las funciones Autoejecutables ó Anónimas

```
this → Operador que hace referencia al objeto de la clase con la que se esté trabajando.
this → Operador que hace referencia al contexto (en JavaScript)

function(){
   this   → Aquí tenemos un contexto (sus variables, funciones…)

   if(){
        this → Aquí tenemos otro contexto (sus variables, funciones…)
  }
}
```

De ahí nace la idea de las funciones autoejecutables o anónimas, con el fin de encapsular el contexto y no se propague al resto del programa. Así evitamos conflictos entre varias librerías (frameworks) que ejecuten la misma parte de código.

Tenemos varias formas de Escribir las funciones Anónimas:

Versión Clásica:

```
(...)()              → El primer paréntesis contiene a la función. (
(function() {})()    → El segundo paréntesis implica ejecución inmediata, del primer paréntesis.
```

```
(function() {
   console.log('versión Clásica');
})();
```

Versión más Encapsulada:

```
((function() {})())    →Un paréntesis más para todo (Autor: Crockford)
```

```
((function() {
   console.log('versión Crockford');
})());
```

Versión Unaria:

```
+function() {}()    → No agrupa entre paréntesis y utiliza el operador +
```

```
+function() {
   console.log('versión Unaria');
}();
```

Versión de Facebook:

```
!function() {}()    → No agrupa y utiliza el símbolo !
```

```
!function() {
   console.log('versión Facebook');
}();
```

```
<script>
    function saludo(){
          alert('Hola');
    }

    // saludo()    Una manera de ejecutar la función.

   window.onload = saludo;   //Cuando la ventana cargue, ejecuta la función saludo()


   window.onload = function(){     // Cuando la ventana se cargue, ejecuta la función anónima (no tiene nombre)
        alert('Hola');             // En el momento que la estoy declarando, la estoy utilizando.
   }                               // Cómo no tiene nombre, si la quiero utilizar más adelante en la programación
                                   // ya no podría.


    (function saludo(){
          alert('Hola');           // Ejecutando esto vemos que nada más arranca el navegador muestra el alert
    })()  → Este segundo paréntesis ejecuta la función anónima y puede tener parámetros.



    (function saludo(d, w, n){           // Creo unas variables que reciben los objetos.
          alert(d); → Aparecerá Object document, object window… → Por eso mejor uso console para ver los objetos

          console.log(d,w,n);      → Abriendo el Navegador, el inspector de elementos y recargamos vemos …
                                    Vemos los objetos document,window,navigator y si lo extendemos es el mapeo
                                    del objeto documento, window y navigator.
          console.log(d,w,n, n.userAgent); → Vemos también alguna propiedad de uno de los argumentos. (navigator)
                                             Con éste sabemos características del usuario que nos está
                                             visitando: Sistema Operativo, Navegador
    })(document, window, navigator)  → Paso como parámetros los objetos.


</script>
```

Esto es más rápido, porque en el momento que las lee el navegador las va a ejecutar

Otro uso muy común es con jQuery:

```
<script src="jquery.min.js"></script>

<script>
    jQuery('h1').css({backgroundColor:'yellow'})  → Cambio una propiedad de Css directamente con jQuery
                                                    Estamos fuera de la función anónima.

    $('h1').css({backgroundColor:'yellow'})  → $ Es el selector de jQuery (es lo mismo poner uno u otro)

          Podría ocurrir que otra librería diferente a jQuery también utilizase el símbolo del $, para que no
          entraran en conflicto podría encapsularlo.

    (function saludo(d, w, n, $){
         $('h1').css({backgroundColor:'yellow'})
    })(document, window, navigator, jQuery)  → Paso los objetos, incluimos el objeto jQuery


    (function saludo(d, w, n, _){               → He cambiado el $ por un _ en la función anónima
         console.log(d,w,n, n.userAgent, _);    → En el inspector de código podemos ver el _
         _('h1').css({backgroundColor:'red'})
    })(document, window, navigator, jQuery)  → Paso los objetos, incluimos el objeto jQuery

</script>
```

Podemos crear funciones sin asignarle un nombre:

```
function(){                                                    //OJO! No podemos llamar a una función sin nombre.
        console.log('Esta función no tiene un nombre');
}
```

```
(function(){                                                    //Todo el código agrupado.
        console.log('Esta función no tiene un nombre \
                    por tanto se ejecuta automáticamente');
})();               → Con los dos paréntesis finales estamos llamando a la función.
```

```
(function (){                    → Definimos una función anónima y al mismo tiempo de llamarla
    var v1 = 0;                     Se ejecuta y desaparece

    function miFunc1(){             // Todo el código se mantendrá aislado del resto de la página
              v1 = 5;
              alert(v1);
    };

    function miFunc2(){
              v1 = 10;
              alert(v1);
    };
})();
```

```
(function(uno,dos,tres){                //Si no la agrupamos entre paréntesis daría error porque se espera var
        console.log(uno);               // en la definición de la función, así obligamos a que se ejecute.
        console.log(dos);
        console.log(tres);
}) (1,2,3);
```

```
(function (w){                          // Estamos pasando el objeto global como parámetro inicial y
    function miFunc1(){                 //lo usamos desde dentro con un alias "w" que perdurará cuando llamemos
              alert(w.document.title); // a las diferentes funciones internas si las exponemos hacia el exterior
    };

    miFunc1();
})(window);
```

Pasando el objeto ***window*** de esta manera y almacenándolo en "**w**" estamos también ganando rendimiento.
El motivo es que al estar *definido en una variable local* a las funciones que definamos dentro de la función anónima, el intérprete las encuentra antes y es más eficiente en su uso.
De hecho es muy habitual pasar como parámetro de inicialización otros objetos comunes como ***document***.

```
(function (w, d, undefined){
        function miFunc1(){
                  alert(d.title);
        };

        miFunc1();
})(window, document);
```

Antes se incluía un tercer parámetro de nombre *undefined*, ya no se hace, **el cual no se le pasa a la llamada de la función anónima** (fíjate en que solo se le pasan dos). Lo que hace en la práctica es definir una variable local /parámetro de la función que se llama *undefined* y nos aseguramos de que tendrá el valor de *undefined* siempre (salvo que se se haya redefinido antes, undefined=1, por ejemplo).

En las versiones modernas de los navegadores, que implementan **ECMAScript 5** o posterior, la instrucción anterior no tiene efecto. Pero en navegadores antiguos como Internet Explorer 8 o anterior, sí que funciona y puede causar estragos, si a alguien le da por cambiar el valor de undefined.