Tema 4 Estructuras de datos

Practica 04-03 Funciones

Nombre		Curso	
Apellidos		Fecha	

1. Creación de funciones.

En unidades anteriores ya has visto y utilizado alguna vez funciones. **Una función es la definición de un conjunto de acciones pre-programadas**. Las funciones se llaman a través de eventos o bien mediante comandos desde nuestro script.

Siempre que sea posible, tienes que diseñar funciones que puedas reutilizar en otras aplicaciones, de esta forma, tus funciones se convertirán en pequeños bloques constructivos que te permitirán ir más rápido en el desarrollo de nuevos programas.

Si conoces otros lenguajes de programación, quizás te suene el término de *subrutina o procedimiento*. En JavaScript no vamos a distinguir entre **procedimientos** (que ejecutan acciones), o **funciones** (que ejecutan acciones y devuelven valores). **En JavaScript siempre se llamarán funciones**.

Una función es capaz de devolver un valor a la instrucción que la invocó, pero ésto no es un requisito obligatorio en JavaScript. Cuando una función devuelve un valor, la instrucción que llamó a esa función, la tratará como si fuera una expresión.
Te mostraré algunos ejemplos en un momento, pero antes de nada, vamos a ver la sintaxis formal de una función:

```
function nombreFunción ( [parámetro1]....[parámetroN] ){
// instrucciones
}
```

Si nuestra función va a **devolver algún valor** emplearemos la palabra reservada **return**, para hacerlo.
Ejemplo:

```
function nombreFunción ( [parámetro1]....[parámetroN] ){
// instrucciones
return valor;
}
```

Los nombres que puedes asignar a una función, tendrán las mismas restricciones que tienen los elementos HTML y las variables en JavaScript. Deberías asignarle un nombre que realmente la identifique, o que indique qué tipo de acción realiza. Puedes usar palabras compuestas como chequearMail o calcularFecha, y fíjate que las funciones suelen llevar un verbo, puesto que las funciones son elementos que realizan acciones.
Una recomendación que te hacemos, es la de que las funciones sean muy específicas, es decir que no realicen tareas adicionales a las inicialmente propuestas en esa función.
Para realizar una llamada a una función lo podemos hacer con:

```
nombreFuncion();// Esta llamada ejecutaría las instrucciones programadas dentro de la función.
```

Otro ejemplo de uso de una función en una asignación:

```
variable=nombreFuncion(); // En este caso la función devolvería un valor que se asigna a la variable.
```

Las funciones en JavaScript también son objetos, y como tal tienen métodos y propiedades. Un método, aplicable a cualquier función puede ser **toString()**, el cual nos devolverá el código fuente de esa función.

2. Parámetros.

Cuando se realiza una llamada a una función, muchas veces es necesario pasar parámetros (también conocidos como argumentos). Este mecanismo nos va a permitir enviar datos entre instrucciones. Para pasar parámetros a una función, tendremos que escribir dichos parámetros entre paréntesis y separados por comas.
A la hora de definir una función que recibe parámetros, lo que haremos es, escribir los nombres de las variables que recibirán esos parámetros entre los paréntesis de la función.
Veamos el siguiente ejemplo:

```
function saludar(a,b){
alert("Hola " + a + " y "+ b +".");
}
```

Si llamamos a esa función desde el código:

```
saludar("Martin","Silvia"); //Mostraría una alerta con el texto: Hola Martin y Silvia.
```

Los parámetros que usamos en la definición de la función a y b, no usan la palabra reservada **var** para inicializar dichas variables. Esos *parámetros a y b serán variables locales a la función*, y se inicializarán automáticamente en el momento de llamar a la función, con los valores que le pasemos en la llamada. En el siguiente apartado entraremos más en profundidad en lo que son las variables locales y globales.

Otro ejemplo de función que devuelve un valor:

```
function devolverMayor(a,b){
if (a > b) then
return a;
else
return b;
}
```

Ejemplo de utilización de la función anterior:

```
document.write ("El número mayor entre 35 y 21 es el: " + devolverMayor(35,21) + ".");
```

Por valor vs por referencia en JavaScript

En JavaScript, cuando asignamos un valor a una variable, o pasamos un argumento a una función, este proceso siempre se hace “por valor" (by value en inglés). Estrictamente hablando, JavaScript no nos ofrece la opción de pasar o asignar “por referencia" (by reference en inglés), como en otros lenguajes. Lo interesante en nuestro caso, es que cuando una variable hace referencia a un objeto (Object, Array o Function), el “valor" es la referencia en sí.

Cuando asignamos valores primitivos (Boolean, Null, Undefined, Number, String y Symbol), el valor asignado es una copia del valor que estamos asignando. Pero cuando asignamos valores NO primitivos o complejos (Object, Array y Function), JavaScript copia “la referencia", lo que implica que no se copia el valor en sí, si no una referencia a través de la cual accedemos al valor original.

3. Ámbito de las variables.

Ha llegado la hora de distinguir entre las variables que se definen fuera de una función, y las que se definen dentro de las funciones.

Las variables que se definen fuera de las funciones se llaman **variables globales.** Las **variables** que se definen dentro de las funciones, con la palabra reservada **var**, se llaman variables locales.

Una **variable global** en JavaScript tiene una connotación un poco diferente, comparando con otros lenguajes de programación. Para un script de JavaScript, el alcance de una variable global, se limita al documento actual que está cargado en la ventana del navegador o en un frame.

Sin embargo cuando inicializas una variable como variable global, quiere decir que todas las instrucciones de tu script (incluidas las instrucciones que están dentro de las funciones), tendrán acceso directo al valor de esa variable. Todas las instrucciones podrán leer y modificar el valor de esa variable global.

En el momento que una página se cierra, todas las variables definidas en esa página se eliminarán de la memoria para siempre. Si necesitas que el valor de una variable persista de una página a otra, tendrás que utilizar técnicas que te permitan almacenar esa variable (como las cookies, o bien poner esa variable en el documento frameset, etc.).

Aunque el uso de la palabra reservada **var**, para inicializar variables es opcional, te recomiendo que la uses ya que así te protegerás de futuros cambios en las próximas versiones de JavaScript.

En contraste a las variables globales, una **variable local será definida dentro de una función**. Antes viste que podemos definir variables en los parámetros de una función (sin usar **var**), pero también podrás definir nuevas variables dentro del código de la función. En este caso, **si que se requiere el uso de la palabra reservada var cuando definimos una variable local**, ya que de otro modo, esta variable será reconocida como una variable global.

El alcance de una variable local está solamente dentro del ámbito de la función. Ninguna otra función o instrucciones fuera de la función podrán acceder al valor de esa variable.

Reutilizar el nombre de una variable global como local es uno de los bugs (*fallo en programación. Generalmente se refiere a fallos o errores de tipo lógico, que suelen ser más difíciles de detectar que los errores sintácticos, los cuáles si que son mostrados por el analizador sintáctico del lenguaje*) más sutiles y por consiguiente más difíciles de encontrar en el código de JavaScript.

La variable local en momentos puntuales ocultará el valor de la variable global, sin avisarnos de ello. Como recomendación, no reutilices un nombre de variable global como local en una función, y tampoco declares una variable global dentro de una función, ya que podrás crear fallos que te resultarán difíciles de solucionar.

Ejemplo de variables locales y globales:

```
/// Uso de variables locales y globales no muy recomendable, ya que estamos empleando el mismo
nombre de variable en global y en local.
var chica = "Aurora"; // variable global
var perros = "Lacky, Samba y Ronda"; // variable global
function demo(){
// Definimos una variable local (fíjate que es obligatorio para las variables locales usar
var)
```


```
// Esta variable local tendrá el mismo nombre que otra variable global pero con distinto
contenido.
// Si no usáramos var estaríamos modificando la variable global chica.
var chica = "Raquel"; // variable local
document.write( "<br/>"+ perros +" no pertenecen a "+ chica +".");
}
// Llamamos a la función para que use las variables locales.
demo();
// Utilizamos las variables globales definidas al comienzo.
document.write( "<br/>"+ perros +" pertenecen a "+ chica +".");
```

Como resultado obtenemos:
Lacky, Samba y Ronda no pertenecen a Raquel.
Lacky, Samba y Ronda pertenecen a Aurora.

4. La declaración de variables con let vs var

La instrucción let declara una variable de alcance local con ámbito de bloque(blockscope), la cual, opcionalmente, puede ser inicializada con algún valor.

Sintaxis

```
let var1 [= valor1] [, var2 [= valor2]] [, ..., varN [= valorN]];
```

Parámetros

var1, var2, …, varN
: Los nombres de la variable o las variables a declarar. Cada una de ellas debe ser un identificador legal de JavaScript

value1, value2, …, valueN
: Por cada una de las variables declaradas puedes, opcionalmente, especificar su valor inicial como una expresión legal JavaScript.

Descripción

let te permite declarar variables limitando su alcance (*scope*) al bloque, declaración, o expresión donde se está usando. a diferencia de la palabra clave var la cual define una variable global o local en una función sin importar el ámbito del bloque. La otra diferencia entre var y let es que este último se inicializa a un valor sólo cuando un analizador lo evalúa (ver abajo).

```
if (x > y) {
  let gamma = 12.7 + y;
  i = gamma * x;
}
```

Diferencia LET con VAR

La redeclaración de la misma variable bajo un mismo ámbito léxico terminaría en un error de tipo SyntaxError. Esto también es extensible si usamos var dentro del ámbito léxico. Esto nos salvaguarda de redeclarar una variable accidentalmente y que no era posible solo con var.

5. Funciones anidadas.

Los navegadores más modernos nos proporcionan la opción de anidar unas funciones dentro de otras. Es decir podemos programar una función dentro de otra función.

Cuando no tenemos funciones anidadas, cada función que definamos será accesible por todo el código, es decir serán funciones globales. Con las funciones anidadas, podemos encapsular la accesibilidad de una función dentro de otra y hacer que esa función sea privada o local a la función principal. Tampoco te recomiendo el reutilizar nombres de funciones con esta técnica, para evitar problemas o confusiones posteriores.

La estructura de las funciones anidadas será algo así:

```
function principalA(){
        function internaA1(){   }
                // instrucciones
        }

function principalB(){
        // instrucciones principalB
        function internaB1(){ // instrucciones internaB1
                        }
        function internaB2(){  //  internaB2
        }
        // instruccionesprincipalB
        }
```

Una buena opción para aplicar las funciones anidadas, es cuando tenemos una secuencia de instrucciones que necesitan ser llamadas desde múltiples sitios dentro de una función, y esas instrucciones sólo tienen significado dentro del contexto de esa función principal. En otras palabras, en lugar de romper la secuencia de una función muy larga en varias funciones globales, haremos lo mismo pero utilizando funciones locales.

Ejemplo de una función anidada:

```
function hipotenusa(a, b){
        function cuadrado(x){ return x*x; }
        return Math.sqrt(cuadrado(a) + cuadrado(b));
}
        document.write("<br/>La hipotenusa de 1 y 2 es: "+hipotenusa(1,2));
        // Imprimirá: La hipotenusa de 1 y 2 es: 2.23606797749979
```

6. Funciones predefinidas del lenguaje.

Has visto en unidades anteriores un montón de objetos con sus propiedades y métodos. Si te das cuenta todos los métodos en realidad son funciones (llevan siempre paréntesis con o sin parámetros). Pues bien en JavaScript, disponemos de algunos elementos que necesitan ser tratados a escala global y que no pertenecen a ningún objeto en particular (o que se pueden aplicar a cualquier objeto).

Propiedades globales en JavaScript:

- **Infinity** Un valor numérico que representa el infinito positivo/negativo.
- **NaN** Valor que no es numérico "Not a Number".
- **undefined** Indica que a esa variable no le ha sido asignado un valor.

Te mostraremos aquí una lista de funciones predefinidas, que se pueden utilizar a nivel global en cualquier parte de tu código de JavaScript. Estas funciones no están asociadas a ningún objeto en particular. Típicamente, estas funciones te permiten convertir datos de un tipo a otro tipo.

Funciones globales o predefinidas en JavaScript:

- **decodeURI()** Decodifica los caracteres especiales de una URL excepto: , / ? : @ & = + $ #
- **encodeURI()** Codifica los caracteres especiales de una URL excepto: , / ? : @ & = + $ #
- **decodeURIComponent()** Decodifica todos los caracteres especiales de una URL.
- **encodeURIComponent()** Codifica todos los caracteres especiales de una URL.
- **escape()** Codifica caracteres especiales en una cadena, excepto: * @ - _ + . /
- **unescape()** Decodifica caracteres especiales en una cadena, excepto: * @ - _ + . /

- **eval()** Evalúa una cadena y la ejecuta si contiene código u operaciones.

- **isFinite()** Determina si un valor es un número finito válido.
- **isNaN()** Determina cuando un valor no es un número.
- **Number()** Convierte el valor de un objeto a un número.
- **parseFloat()** Convierte una cadena a un número real.
- **parseInt()** Convierte una cadena a un entero.

Ejemplo de la **función** **eval()**:

```
<script type="text/javascript">
eval("x=50;y=30;document.write(x*y)");  // Imprime 1500
document.write("<br />" + eval("8+6")); // Imprime 14
document.write("<br />" + eval(x+30)); // Imprime 80
</script>
```

7. FUNCIONES ANÓNIMAS DE JAVASCRIPT

La función anónima es una función que no tiene ningún nombre asociado. Normalmente usamos la palabra clave de *función* antes del nombre de la función para definir una función en JavaScript, sin embargo, en funciones anónimas en JavaScript, usamos solo la palabra clave de *función* sin el nombre de la función.
Una función anónima no es accesible después de su creación inicial, solo se puede acceder a ella mediante una variable en la que está almacenada como una *función como un valor* . Una función anónima también puede tener varios argumentos, pero solo una expresión.

Sintaxis:

```
function() {
  // Function Body
}
```

Los siguientes ejemplos demuestran funciones anónimas.
Ejemplo 1: En este ejemplo, definimos una función anónima que imprime un mensaje en la consola. Luego, la función se almacena en la variable *saludar* . Podemos llamar a la función invocando *greet().*


```
<script>
var greet = function () {
  console.log("Welcome to Frikis Anonimos!");
};

greet();
</script>
```

Producción:

```
Welcome to Frikis Anonimos!
```

Ejemplo 2: En este ejemplo, pasamos argumentos a la función anónima.

```
<script>
var greet = function (platform) {
  console.log("Welcome to ", platform);
};

greet("Frikis Anonimos!");
</script>
```

Producción:

```
Welcome to Frikis Anonimos!
```

Como JavaScript admite funciones de orden superior, también podemos pasar funciones anónimas como parámetros a otra función.

Ejemplo 3: En este ejemplo, pasamos una función anónima como función de devolución de llamada al método *setTimeout()* . Esto ejecuta esta función anónima 2000ms después.

```
<script>
setTimeout(function () {
  console.log("Welcome to Frikis Anonimos!");
}, 2000);
</script>
```

Producción:

```
Welcome to Frikis Anonimos!
```

Otro caso de uso de funciones anónimas es invocar la función inmediatamente después de la inicialización, esto también se conoce como *función de ejecución automática* . Esto se puede hacer agregando paréntesis, podemos ejecutar inmediatamente la función anónima.

Ejemplo 4: En este ejemplo, hemos creado una función autoejecutable.

```
<script>

( function () {
  console.log("Welcome to Frikis Anonimos!");
}) ();

</script>
```

Producción:

```
Welcome to Frikis Anonimos!
```

Funciones de flecha

ES6 introdujo una forma nueva y más corta de declarar una función anónima, que se conoce como funciones de flecha. En una función de flecha, todo sigue igual, excepto que aquí no necesitamos la palabra clave de *función* también. Aquí, definimos la función con un solo paréntesis y luego '=>' seguido del cuerpo de la función.

Esto lía mucho pero básicamente es convertir la sintaxis

function Nombre () { … } **var nombre = () => { … }**

Ejemplo 5:

```
<script>
var greet = () =>
{
  console.log("Welcome to Frikis Anonimos!");
}

greet();
</script>
```

Producción:

```
Welcome to Frikis Anonimos!
```

Si solo tenemos una declaración en el cuerpo de la función, incluso podemos eliminar las llaves.

Ejemplo 6: En este ejemplo, creamos una función autoejecutable.

```
<script>
 let greet = () => console.log("Welcome to Frikis Anonimos!");
 greet();
</script>
```

Producción:

```
Welcome to Frikis Anonimos!
```

Parámetros Rest

La sintaxis de los parámetros rest nos permiten representar un número indefinido de argumentos como un array.

```
function( a, b, ...Args ) {
var parametro1obligatorio=a;
var parametro2obligatorio=b;
var parametro3opcional=Args[0];
var parametro4opcional=Args[1];
numerodeparametrosopcional =  Args.length;
// recorrer los parametros
foreach (parametro in Args) {     }

}
```

Practica 04-03 Funciones

Ejercicio 1. Crea una función **Ej1Trigonometica** que se le pase dos parametros

1) un número que es angulo en grados
2) un valor : cos, sen o tan

   la función debe pasar el valor a radianes y devolver el resultado dependiendo del parámetro la operación correcta

   - cos→ coseno
   - sen → seno
   - tan → tangente
   - Lo tienes que pasar sabiendo las funciones solo admiten Radianes (PI's) y que 180º = PI

Ejercicio 2. Crea una función **Ej2JS** que le pases una lista de valores y un id y te cree una lista ordenada con ellos escrita en el el id

<script >

funcion listaen(“hola,adios, bien,Hasta, mañana”, “algo”)

</script >

Eso debe escribir.

- hola
- adios
- bien
- Hasta mañana

Ejercicio 3. Modifica el anterior para que uses parametros rest con un numero indeterminado de parametros, esto es la función debe ser algo como

</script > listaen(identificador,Hola, adios, bien,Hasta, mañana, algo) </script >

Eso debe escribir.

- Hola
- adios
- bien
- Hasta mañana
- algo

Ejercicio 4. Crea una función anónima que calcule el triple de un numero y asigna lo a una variable y ejecutalo.

Ejercicio 5. Crea una función anónima con flechas que calcule el cuadrado de un numero y asigna lo a una variable y ejecutalo.

Ejercicio 6. Crea una función anónima con flechas que escriba hola mundo y que se ejecute automáticamente.

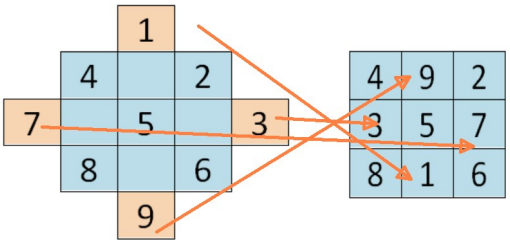
Ejercicio 7. Crea una función anónima con flechas que ponga en negrita el parámetro añadiéndole   y usala con la función map sobre un array como por ejemplo

a = [“ola”,”adios”,”amalia pide muchos detalles”];

Ejercicio 8. Escribe una función que genere al azar 20 números enteros comprendidos entre 0 y 9.

Estos números se deben introducir en un array de 4 filas por 5 columnas y mostrárnoslo como un string que sea una tabla

Ejercicio 9. **Crea un función que te dibuje el cuadrado mágico 3x3 en una tabla con el valor de entrada i**



Un cuadrado mágico es una tabla de grado primario donde se dispone de una serie de números enteros en un cuadrado o matriz de forma tal que la suma de los números por columnas, filas y diagonales principales sea la misma.

Sabiendo que hay que hacer una array asignación

i+3	i+8	i+1
i+2	i+4	i+6
i+7	i	i+5

8	13	6
7	9	11
12	5	10

Ejemplo para i=5

Finalmente Formateamos como una tabla

Ejercicio 10. Haz una función que se le pase un id y un color y cambie su color de texto. Debes usar el siguiente código

```
let obj = document.getElementById(Parametro);
obj.style.color = nombreColor;
```

Debes asociarlo a un combotext y un button para poder ver varios ejemplos.

Ejercicio 11. Observa esta función recursiva

```
function factorial(valor) {
          if (valor==1) return 1;
          else return valor+factorial(valor-1);
    }
```

Ejercicio 12. Haz un ejercicio que calcule el factorial de un numero de manera recursiva.