Programación Avanzada
Grado en Ingeniería Informática en Sistemas de Información - Curso 2018/2019

EPD 7: JavaScript Básico

La entrega del trabajo se hará a través de la tarea correspondiente en el Campus Virtual. Pasado el límite de entrega se aceptará el envío del trabajo, con una penalización de 2 puntos de la calificación por cada hora o fracción de retraso. La entrega consistirá en un único fichero comprimido en formato ZIP cuyo nombre deberá ser de la forma *equipoXX.zip*, donde *XX* serán dos cifras que indicará el número del equipo. Por ejemplo, equipo07.zip. Este fichero contendrá una serie de carpetas cuyo nombre deberá ser de la forma *ejY* o *pZ*, donde *Y* y *Z* representan, respectivamente, el número de cada ejercicio o problema del presente guión. Dentro de dichas carpetas se incluirán exclusivamente los archivos necesarios para la resolución del correspondiente ejercicio o problema. Las rutas de los ficheros empleados serán relativas, a fin de que las resoluciones a los ejercicios y problemas puedan ser examinadas en cualquier equipo. Cualquier entrega que no cumpla las reglas de nombrado, el formato de compresión del archivo o el contenido de los archivos del mismo, será penalizada con 2 puntos sobre 10 por cada incumplimiento.

Objetivos

- Conocer los elementos básicos del lenguaje JavaScript.
- Crear páginas web interactivas con *scripts* básicos.

Conceptos

1. Elementos básicos de JavaScript:

JavaScript es un lenguaje de programación interpretado y orientado a prototipos que permite crear interfaces web interactivas de forma sencilla. Una de las ventajas más importantes que ofrece es que su código se ejecuta directamente en el propio navegador del cliente (esto es, empleando los recursos de la máquina del usuario) con la consecuente descarga de trabajo del servidor web.

Los programas escritos en JavaScript se incrustan dentro de las páginas HTML y se envían con éstas a la máquina del cliente cuando éste solicita cada página (o posteriormente, como archivos asociados a la página, de forma similar a como ocurre con las imágenes). El navegador del usuario será el encargado de interpretar y ejecutar los scripts incluidos en la página cuando sea necesario.

2. Incrustación de un script en un documento HTML:

El código de un programa escrito en JavaScript se puede incluir dentro de una página HTML de dos formas:

1. Entre las etiquetas <script type="text/javascript"> y </script>
2. Incluyendo un archivo que contiene un script, de la forma <script type="text/javascript" src = "nombreScript.js"></script> , donde “nombreScript.js” es el nombre del archivo que contiene el script. Debido a la definición del estándar HTML por parte de W3C, y de su implementación estricta en los navegadores Firefox e Internet Explorer, se deberá usar la forma no abreviada aunque el bloque dentro de la etiqueta script no tenga contenido. En los navegadores que no implementan estrictamente el estándar, la forma abreviada <script type="text/javascript" src = "nombreScript.js" /> puede emplearse (Como, por ejemplo, Chrome).

En el caso de que empleemos la segunda opción para incluir código JavaScript tendremos que crear un archivo, aparte de la página web, que contenga el código a incluir.

Podemos crear un archivo para un script en un proyecto web de Netbeans de una forma sencilla. Pulse el botón derecho del ratón sobre el nodo del proyecto en el que desea crear el nuevo *script* y seleccione las opciones del menú contextual *New→Other…*

Realizada la selección, se le mostrará una ventana que le permite indicar el tipo de elemento que desea crear. Seleccione el nodo *JavaScript* de la carpeta Other y pulse en *Next*.

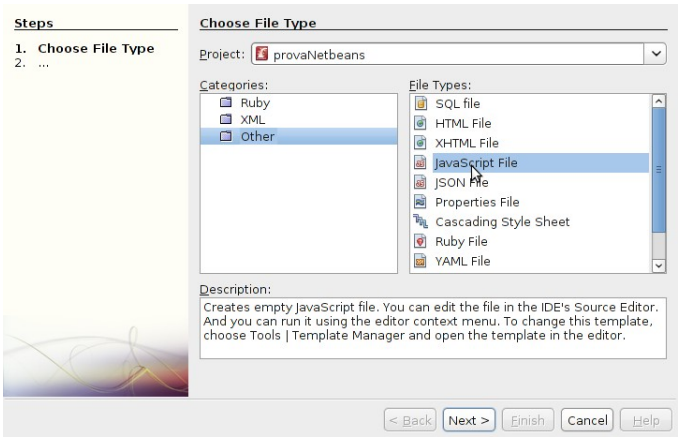

Figura 1: Selección del tipo de elemento

En la ventana que se le muestra a continuación deberá dar un nombre al *script*. Escriba el nombre y pulse en finalizar.

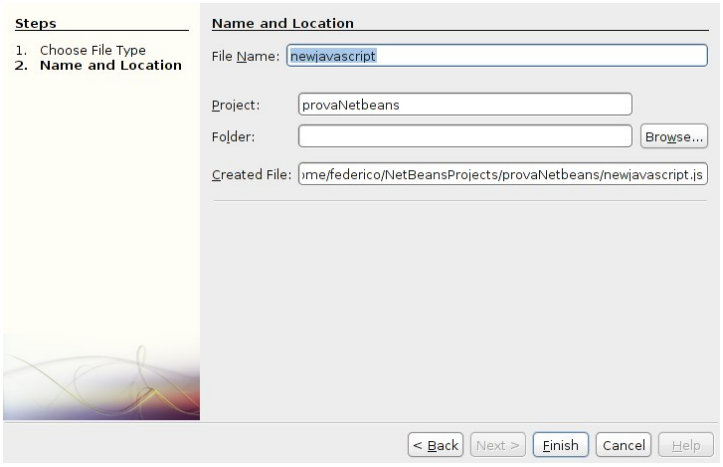

Figura 2: Asignación de nombre al script

Seguidos los pasos anteriores, aparecerá una nueva pestaña que permitirá editar el archivo que contiene el nuevo *script* creado. Este archivo se incorporará al nodo correspondiente en el proyecto para el cual se ha realizado la creación del *script*.

Para poder depurar *scripts* podemos utilizar la consola de error de Firefox. Ésta se abrirá seleccionándola desde la opción del menú *Tools* → *Error Console*, y nos aparecerá una ventana similar a la siguiente:

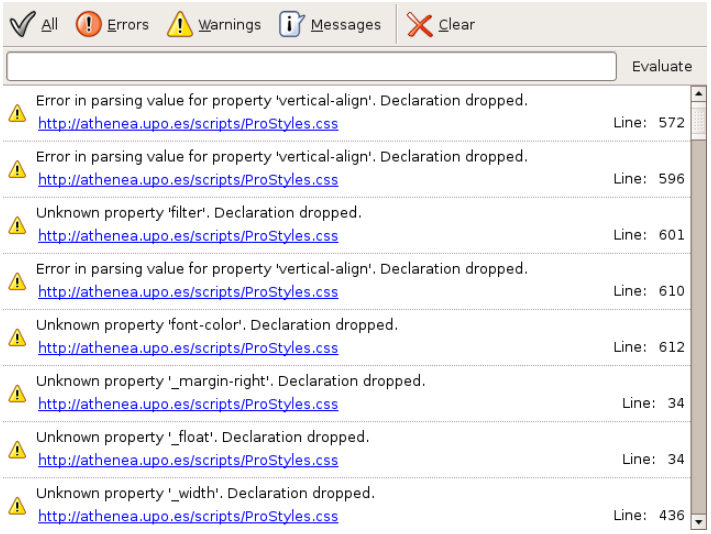

Figura 3: Consola de Error de Firefox

Tenga en cuenta que, en las nuevas versiones del navegador Firefox, la consola web donde poder ver y depurar errores se abre desde el menú Desarrollador → Consola Web.

Bibliografía Básica

1. JavaScript programming for the absolute beginner: the fun way to learn programming.
Harris, Andrew, 1964-Roseville, CA : Prima Tech, 2001.
http://site.ebrary.com/lib/bupo/Doc?id=10067190

2. Introducción a JavaScript. Javier Eguíluz Pérez
http://librosweb.es/libro/javascript

Experimentos

Ex1. (15 mins.) Observe el comportamiento del navegador Firefox ante códigos JavaScript que contienen errores de diferentes tipos:

a) Errores en tiempo de carga: Se producen cuando el navegador puede determinar que el *script* tiene un error al procesar el código. Normalmente proceden de errores de sintaxis, como los que contiene este código:

```
<html>
	<head>
		<script type="text/javascript">
		<!--
			function hola(){
				var cadena = ¡Hola mundo!
				document.write(cadena);
			}
		//-->
		</script>
	</head>
	<body>
		<script type="text/javascript">
		<!--
			hola();
		//-->
		</script>
	</body>
</html>
```

b) Errores en tiempo de ejecución: Este tipo de errores se produce cuando el navegador detecta un error al ejecutar una sentencia del código:

```
<html>
	<head>
		<script type="text/javascript">
		<!--
			function hola(nombre){
				document.write("¡Hola " + Nombre + "!");
			}
		//-->
		</script>
	</head>
	<body>
		<script type="text/javascript">
		<!--
			var nombre = prompt("Dime tu nombre","");
			hola(nombre);
		//-->
		</script>
	</body>
</html>
```

c) Errores lógicos: Aunque el programa no contenga ningún error de los anteriores y, por tanto, esté perfectamente escrito en función de las reglas del lenguaje del programación, puede contener un error lógico. Este tipo de error supone que el programa no se comporte como el programador esperaba. En esto casos, la función *alert( )* nos permitirá ver qué está ocurriendo en el código. Observe qué ocurre en el siguiente código:

```
<html>
	<head>
		<script type="text/javascript">
		<!--
			function power(numero,p){
				//alert("Calculando potencia de "+p+ " de " + numero);
				return numero * power(numero,p-1);
				}
		//-->
		</script>
	</head>
	<body>
		<script type="text/javascript">
		<!--
			var numero = prompt("¿A que numero le calculamos la potencia?","");
			var potencia = prompt("la potencia ");
			var resultado = power(numero,potencia);
			alert("La potencia de " + potencia + " de " +numero +
				" es " + resultado);
		//-->
		</script>
	</body>
</html>
```

Ex2. (15 mins.) Empleando el código anterior, pruebe las funcionalidades del depurador de Firefox (con Firebug) o Chrome.

Ejercicios

EJ1. (15 mins.) Escriba una función para calcular el área de un triángulo. Para ello, integre esta función en una página web de tal forma que ésta pida dos lados del triángulo y el ángulo que forman entre ellos, en grados sexagesimales, y devuelva el área del mismo reduciendo la salida a 2 decimales si fuera necesario. Puede obtener más información de la forma de realizar este cálculo en la página https://www.universoformulas.com/matematicas/trigonometria/area-triangulo-razones-trigonometricas/.

EJ2. (45 mins.) Realice la implementación de la función *toFixed*, que permite redondear un número. El programa pedirá al usuario 2 cadenas:

- Cadena principal: Será un número, que podrá ser entero o decimal. En caso de que sea decimal, se usará como carácter separador entre la parte entera y la parte decimal el '.' Ej: "9.656"

- Cadena con el número decimal al que redondear. Ej: "2"

Y mostrará el resultado de hacer el redondeo a la unidad que se le ha pasado como segunda cadena. Para el ejemplo dado, el resultado que se mostraría por pantalla sería:

```
El resultado de redondear "9.656" a "2" es 9.66
```

Tenga en cuenta que si el redondeo se efectúa a más decimales de los que tenga el número original, se rellenará con 0. Ejemplo: el redondeo de 9.656 a 6 será 9.656000.

Además, deberá comprobar que la cadena principal introducida por el usuario contenga únicamente un símbolo de punto decimal y que el resto de caracteres sean numéricos. Para ello, deberá iterar carácter a carácter sobre dicha cadena mediante el método charCodeAt de las cadenas de caracteres (más información en https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/charCodeAt).

EJ3. (15 mins.) Implemente un *script* que imprima el siguiente contenido en una página web . Para ello, use dos estructuras *while*.

123456789
2468
369
48
5

Problemas

P1. (60 mins.) Cree una página web que permita obtener el resultado de una serie de operaciones de cadenas de caracteres aplicadas en serie. La web pedirá al usuario un texto conteniendo una primera cadena y a continuación las operaciones y sus operandos (todos los elementos separados por ";"). Las operaciones permitidas serán estas tres: sub(*ini*, *fin*), cat(*cadena*) y rep(*buscar*,*reemplazar*). La operación sub(*ini*,*fin*) devolverá la subcadena desde la posición ini hasta la posición fin (incluidas). La operación cat(*cadena*) concatenará una cadena al final. La operación rep(*buscar*,*reemplazar*) devolverá una cadena donde se ha sustituido la cadena *buscar* por *reemplazar*. Por ejemplo, el texto de entrada puede ser "pepe;rep(e,a);cat(da);sub(3,6)". La página imprimirá el resultado parcial y total de la operaciones ejecutando éstas de izquierda a derecha. Para la cadena de entrada anterior, el resultado total sería "pada" y los resultados parciales sería pepe, papa, papada, pada. Observe que cada operación usa como cadena base el resultado de la operación anterior, salvo la primera que es la cadena de comienzo. Por cada operación soportada emplee una función distinta para ejecutar la operación, aceptando éstas como argumento los operandos que necesite y devolviendo el resultado de la operación.

P2. (30 mins.) Consulte la página https://css-tricks.com/snippets/javascript/ y realice tres páginas distintas que incorporen cada una de ellas una funcionalidad implementada en JavaScript usando al menos tres de los snippets que se ofrecen gratuitamente en esta página web. Indique la página original de la que ha incorporado el código.

P3. (60 mins.) Vamos a crear una web que tendrá un área de texto donde se deberá introducir un fragmento de texto cualquiera. Del éste, una vez procesado, se deberá mostrar en la página resultante la siguiente información: número de palabras del texto, primera palabra del texto y última palabra del texto. Además, en una tabla se mostrará la siguiente información: las palabras del texto, el número de veces que se repite una palabra en el texto y número de vocales y de consonantes de cada palabra, esta tabla será ordenada por el número de veces que se repiten las palabras. A las palabras del texto se tendrán que considerar sin diferenciar mayúsculas y minúsculas y eliminando los posibles signos de puntuación. Solo se considerarán los siguientes signos de puntuación: la coma, el punto, los signos de exclamación y de interrogación. Para ello, haga uso del método *split* propio de los objetos que representan cadenas (puede encontrar información detallada sobre éste método en https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/String/split), del método charAt para recuperar los caracteres

de una cadena (puede encontrar más información en https://www.w3schools.com/jsref/jsref_charat.asp) y del método *sort* (puede encontrar información detallada en https://www.w3schools.com/jsref/jsref_sort.asp).

P4. (30 mins.) Cree una página web que recibirá como entrada un número y comprobará si el número pertenece a la sucesión de Fibonacci. La aplicación seguirá solicitando números hasta que el usuario introduzca un número negativo. La página mostrará el número total de números pertenecientes a la sucesión y los que no. El que sea mayor, se mostrará con un borde sólido de 5px. Más información sobre la sucesión de Fibonacci en https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci

P5. (45 mins.) Cree una página que imprima una tabla de temperaturas aleatorias para pacientes de un hospital. Los nombres de los pacientes se leerán de un vector. El programa pedirá un valor de temperatura mínimo y otro máximo, con la restricción de que el valor mínimo sea menor que el valor máximo. Cree una función que devuelva un vector de temperaturas aleatorias a partir de los valores mínimo y máximo de temperatura especificados y un número que indique cuántos valores aleatorios generar. Cree una segunda función que reciba dos vectores (uno de nombres de pacientes y otro de temperaturas) y devuelva una tabla HTML que muestre cada paciente y la temperatura que tiene (emparejando uno a uno los elementos de ambos vectores).

Ampliación de Bibliografía

1. The book of JavaScript a practical guide to interactive Web pages Thau.San Francisco : No Starch Press, 2006. http://site.ebrary.com/lib/bupo/Doc?id=10158196

2. Learn JavaScript in a weekend Ford, Jerry Lee.Boston, Mass. : Premier Press, 2004. http://site.ebrary.com/lib/bupo/Doc?id=10054331

Datos de la Práctica

Autor del documento: Federico Divina (Octubre 2007).

Revisiones:

1. Carlos D. Barranco González (Noviembre 2008).
2. Federico Divina (Abril 2010). Utilizo de Eclipse substituido por utilizo de Netbeans. Experimento 1b y 1c modificados.
3. Carlos D. Barranco González (Abril 2010). Revisión de texto y retoques de formato.
4. Francisco A. Gómez Vela (Marzo 2011). Revisión de texto, retoques en el formato. Cambios en los ejercicios 1 y 3. En el problema 1 se ha cambiado la fuente de las funciones javascript y en el problema 3 y 4 se ha cambiado el enunciado.
5. Carlos D. Barranco González (Marzo 2012): Cambios en el texto. Cambio del sitio web referenciado en el problema 1, ya que el anterior dejó de funcionar. Nuevo enunciado para los ejercicios 2, 4 y 5, así como para el problema 2.
6. Carlos D. Barranco González (Diciembre 2012): Adaptación del formato a Programación Avanzada. Mejora del texto en la sección "Conceptos". Adecuación del formato del código del Exp. 1.
7. Carlos D. Barranco González (Noviembre 2013): Mejora de la redacción de la parte de conceptos y corrección de formato de la referencia 1 de la bibliografía básica. Nuevo enunciado para los ejercicios 1, 3 y los problemas 1 (por renovación del sitio de referencia), 3 y 4.
8. Carlos D. Barranco González (Noviembre 2014): Mejora del texto de la sección de conceptos. Inclusión del experimento 2, Replanificación de tiempos en ejercicios y problemas.
9. Pablo Soler Regli (Noviembre 2015): Cambio de sitio web referenciado en el punto 2 de la bibliografía. Añadida referencia a la apertura de la consola web en las nuevas versiones Firefox. Nuevo enunciado para el problema 5.
10. Carlos D. Barranco González (Noviembre 2015): Revisión de objetivos y textos de la sección de conceptos.
11. Pablo Soler Regli (Noviembre 2015): Cambio del enunciado del ejercicio 2 y del problema 4. Cambio de la cabecera de las páginas. Recálculo de tiempos totales.
12. Carlos D. Barranco González (Noviembre 2015): Introducidas aclaraciones en EJ2 y ampliación en P4.
13. Ricardo – León Talavera Llames (Noviembre 2016). Ejercicio 1, Ejercicio 2, Problema 3 y Problema 4 modificados.
14. Carlos D. Barranco González (Noviembre 2016): Mejora en texto de la sección de conceptos y en el experimento 1.
15. Ricardo – León Talavera Llames (Noviembre 2016): Revisados ejercicio 1, Ejercicio 2, Problema 3 y Problema 4.
16. Gualberto Asencio Cortés (Noviembre 2017): Modificados ejercicios 2, 3 y problemas 1, 2 y 5. Eliminado ejercicio 4 por sobrepasar la dedicación temporal máxima de la EPD. Corregidas las estimaciones temporales, pues estaban erróneas. Corregidas erratas en el texto del guión de prácticas.
17. Carlos D. Barranco González (Noviembre 2017): Correcciones de formato y mejora del enunciado del ejercicio 2.
18. Daniel Prieto Tagua (Noviembre 2018): Modificaciones en el ejercicio 1 y los problemas 3 y 4.
19. Carlos D. Barranco (Noviembre 2018): Mejoras de redacción y formato.

Estimación temporal:

- Parte presencial: 120 minutos.
    - Explicación inicial: 5 minutos.
    - Experimentos: 30 minutos.
    - Ejercicios: 85 minutos.
- Parte no presencial: 270 minutos.
    - Lectura y estudio del guión y bibliografía básica: 45 minutos
    - Problemas: 225 minutos