CFGS DAW

Mòdul 6: Desenvolupament web en entorn client

INS Joan d'Àustria

Funciones predefinidas

- Funciones predefinidas
  - Javascript incorpora una serie de funciones globales o predefinidas que podemos usar
  - Entre estas funciones destacan:

Funciones predefinidas		
decodeURI()	eval()	parseInt()
decodeURIComponent()	isFinite()	String()
encodeURI()	isNaN()	unescape()
encodeURIComponent()	Number()	
escape()	parseFloat()	

Funciones predefinidas

- eval
  - Esta función recibe un único parámetro de tipo string y lo ejecuta como si fuera código javascript

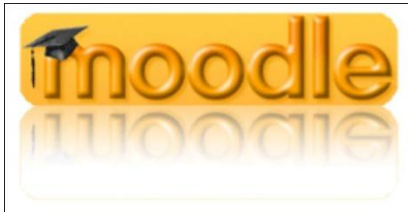
```
var x = prompt('Introduce una expresión:');
alert('resultado: ' + eval(x));
```

- isNaN
  - Esta función comprueba que el número que le pasamos como argumento es de tipo numérico

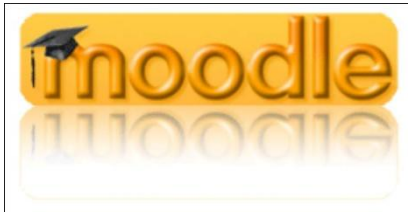
```
var x = prompt("Introduce un valor numérico: ");
if (isNaN(input)){
        alert("Eeeeyyy no hagas trampas.");
}
else{
        alert("Muy bien.");
}
```

Funciones predefinidas

¿Qué diferencias existen entre las funciones escape, encodeURI y encodeURIComponent? Puedes consultar por ejemplo la web: http://xkr.us/articles/javascript/encode-compare/
Añade algún ejemplo de uso de estas funciones



¿Por qué se dice la expresión “eval is evil”?



Funciones (declaración)

- Podemos definir nuestras propias funciones
- Las funciones javascript son objetos y por tanto, en realidad, un nombre de función no es más que un puntero a este objeto
- La sintaxis es:

```
function nombre (parametros) {
        … Instrucciones
}
```

Funciones (declaración)

- En la declaración de las funciones javascript no indicamos si devuelven o no algún valor ni el tipo de este valor

¿Retorna algún valor?
¿De qué tipo?

```
function nombre (parametros) {
        … Instrucciones
}
```

- Las funciones pueden devolver el valor que quieran, cuando quieran o no devolver nada. No hay diferencia entre procedimientos y funciones

Funciones (return)

- Cuando una función quiera devolver un valor, usará la palabra return seguida del valor

```
function sum(num1, num2) {
return num1 + num2;
}
```

- No obstante, debemos tener presente que una función no se continuará ejecutando después de la instrucción return

```
function sum(num1, num2) {
return num1 + num2;
alert("¿Llega aquí?"); //no se ejecutará!!
}
```

Funciones (parámetros)

- Los parámetros en javascript no se comportan de la forma “tradicional”.
- Javascript no va a comprobar que se llame a una función con el mismo número o no de parámetros según está definida, es decir, sólo por haber declarado una función con dos parámetros, no significa que la tengamos que llamar con dos parámetros. Podemos llamarla con uno, tres… y nadie se va a quejar.

Funciones (parámetros)

- En realidad, esto es debido a que los parámetros se pasan a las funciones como un array (no es un array, sino un objeto, pero podemos simplificarlo como un array). Este array siempre se pasa, pero la función no comprueba que dentro tenga alguna cosa o no.
- Este array se denomina arguments y podemos acceder a él con arguments[0], arguments[1], etc.

Funciones (parámetros)

- Así, son equivalentes:

```
function saluda(name, message) {
alert("Hola " + name + ", " + message);
}
```

```
function saluda() {
alert("Hola" + arguments[0] + ", " + arguments[1]);
}
```

Funciones (parámetros)

- Ejemplo:

```
function flexisum(a) {
var total = 0;
for(var i = 0; i < arguments.length; i++) {
                var element = arguments[i];
                if (!element) continue; // Ignore null and undefined arguments
                var n;
                switch(typeof element) {
                case "number":
                                n = element; // No conversion needed here
                                break;
                case "string":
                                n = parseFloat(element); // Try to parse strings
                                break;
                case "boolean":
                                n = NaN; // Can't convert boolean
                                break;
                }
                // If we got a valid number, add it to the total.
                if (typeof n == "number" && !isNaN(n)) total += n;
}
return total;
}
```

Funciones (internamente)

- Como una función es un puntero, no puede existir sobrecarga. La última definición sobrescribirá la anterior

```
function incrementa(num){
return num + 100;
}
function incrementa(num) {
return num + 200;
}
var result = incrementa(50); //250
```

Funciones (internamente)

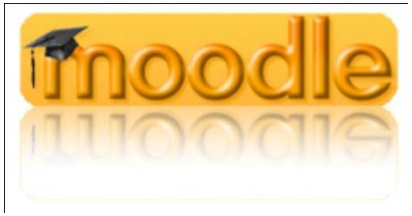
- En realidad, dada su condición de puntero, podemos definir las funciones de la siguiente manera:

```
var incrementa = function (num){
return num + 100;
};
```

Es decir, tratando el nombre de la función como un puntero que contiene una variable tipo function.

Funciones (internamente)

Declarar funciones o utilizar funciones como expresiones, pueden resultar parecidas, pero presentan alguna diferencia. Investiga donde está la diferencia y pon algún ejemplo.




```
function incrementa(num){
return num + 100;
}
```

VS

```
var incrementa = function (num){
return num + 100;
};
```

Llamadas a funciones

- Podemos llamar a una función javascript desde cualquier parte del código

```
<html>
< head>
<script type="text/javascript">
function miFuncion() {
alert("Hola");
}
< /script>
</head>
< body>
…
<script>
miFuncion()
</script>
…
```

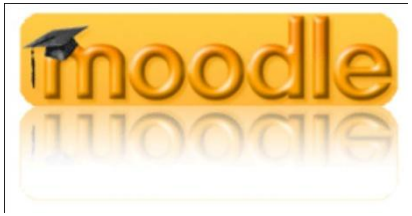
Llamadas a funciones

- También directamente como respuesta a un evento, como por ejemplo cuando se haga click en un botón HTML

```
<html>
< head>
<script type="text/javascript">
function miFuncion(){
alert("Hola");
}
< /script>
</head>
< body>
…
<form>
< input type="button"  onclick="miFuncion()" value="Prueba">
< /form>
…
```

Llamadas a funciones

Si una función no recibe ningún parámetro, ¿es necesario llamarla con paréntesis?




```
...
miFuncion();
...
```

VS

```
...
miFuncion;
...
```