iesperemariaorts

Desarrollo web en entorno cliente

Tema 2

Javascript

Fundamentos de JavaScript

iesperemariaorts

Características lenguajes "Scripts"

- Son interpretados (por los navegadores), no compilados. No generan ejecutables.
- Javascript es en realidad una implementación de ECMAScript.
- La versión mas actual es ES2018, en ES2015 (ES6) muchos cambios.
- Se interpretan por el propio navegador.
- Se incrustan y ejecutan dentro de otros programas (HTML).
- Se interpreta línea a línea cada vez.
- Fáciles de usar y programar.

iesperemariaorts

Ventajas e incovenientes

ventajas

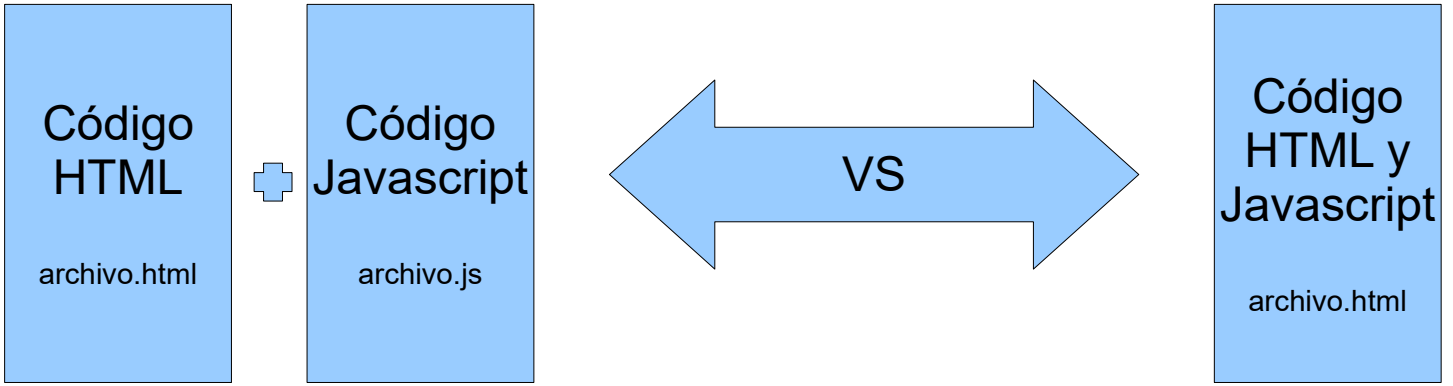
- *El código es cómodo para depurar, ya que no es necesario volver a compilar tras un cambio.*
- *No es necesario disponer de un compilador, ya que el intérprete ejecuta el script.*
- *El mantenimiento es fácil y rápido, por parte del autor o de otro programador.*

inconvenientes

- *La ejecución se ralentiza, al ser necesario la interpretación línea a línea cada vez.*
- *El código es visible y puede ser objeto de plagio por parte de otras personas.*
- *El usuario tiene acceso al código y puede modificarlo, estropeando alguna operación.*

iesperemariaorts

Integrar Javascript dentro de HTML



iesperemariaorts

Integrar Javascript dentro de HTML

- Dentro de las etiquetas <script> </script> (no recomendable).

```
<!DOCTYPE>
<html>
    <head>
        <title>Ejemplo JS</title>
    </head>
    <body>
        <p>Hola Mundo!</p>
        <script>
            console.log("Hola Mundo!");
        </script>
    </body>
</html>
```

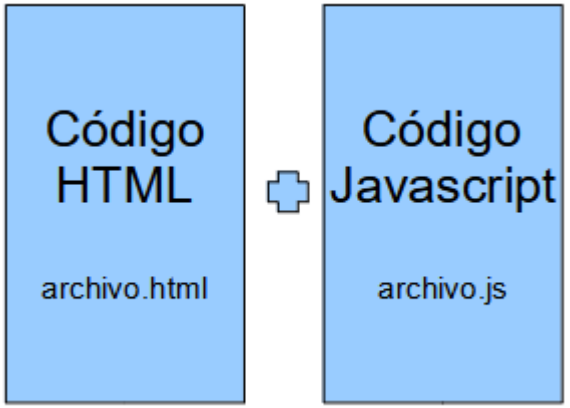
Código HTML y Javascript

archivo.html

iesperemariaorts

Integrar Javascript dentro de HTML

- En un archivo separado (recomendable).



Archivo: ejemplo1.html

```
<!DOCTYPE>
<html>
    <head>
        <title>Ejemplo JS</title>
    </head>
    <body>
        <p>Hola Mundo!</p>
        <script src="ejemplo1.js"></script>
    </body>
</html>
```

Archivo: ejemplo1.js

```
console.log("Hola Mundo!");
```

iesperemariaorts

Integrar Javascript dentro de HTML

Código HTML

archivo.html

```html
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
<body>
  <script src="ejemplo1.js">

  </script>
</body>
</html>
```

```javascript
function saludo()
{
    document.write("Hola a todos");
}

saludo();
```

Código Javascript

archivo.js

iesperemariaorts

Tipos de variables

- Se declaran usando la palabra "let"
- "var" desde ES2015 no se recomienda
- No tienen un tipo de dato explícito
- Formato camelCase
- Si no asignamos valor --> "undefined" (no es "null")

```
let a = 15;
console.log(a);

console.log(typeof a); //Imprimirá "number"

a = "Cadena";
console.log(typeof a); //Imprimirá "string"

a = [12, 23, 4];
console.log(typeof a); //Imprimirá "object"
console.log(a instanceof Array); //Imprimirá "true"

let b;
console.log(typeof b); //Imprimirá "undefined"
```

iespereMariaorts

Tipos de variables

- Si olvidamos "let" o "var": declarada como global
  - Para evitar olvidar declarar variable: 'use strict'

```
'use strict';
v1 = "Hola Mundo";
```

```
▶ Uncaught ReferenceError: v1 is not defined   example1.js:2
>
```

- Constantes "const"

```
'use strict';
const MY_CONST=10;
MY_CONST=200;  → Uncaught TypeError: Assignment to constant variable.
```

iesperemariaorts

Funciones Javascript

- Palabra reservada "function"
- Argumentos pasados dentro del paréntesis
- Entre llaves {} la función
- Formato camelCase

```
function sayHello(name) {
    console.log("Hello " + name);
}

let sayHello2 = sayHello; // Los 2 nombres referencian misma función

sayHello("World!"); //Retorna "Hello World!"
sayHello(); //Retorna "Hello undefined"
console.log(typeof sayHello); //Retorna"function"
sayHello2("Peter"); //Retorna "Hello Peter"
```

iesperemariaorts

Funciones Javascript

- "return" retorna valores en una función

```
function sum(n1, n2) {
    return n1 + n2;
}

let r = sum(3, 6);
console.log(r); //Retorna "9"
```

iesperemariaorts

Funciones Javascript

- Funciones anónimas: No se le asigna nombre, se puede asignar directamente como valor a una variable

```
let totalPrice = function(priceUnit, units) {
   return priceUnit * units;
}

console.log(typeof totalPrice); // Imprime "function" (tipo de la variable totalPrice)

console.log(totalPrice(5.95, 6)); // Imprime 35.7
let getTotal = totalPrice; // Referenciamos a la misma función desde la variable getTotal
console.log(getTotal(5.95, 6));  // Imprime 35.7. También funciona
```

iesperemariaorts

Funciones Javascript

- Valores por defecto

```
function Persona(nombre) {
   this.nombre = nombre || "Anónimo";
   ...
}
```

```
function Persona( nombre = "Anónimo") {
   this.nombre = nombre;
   ...
}
```

iesperemariaorts

Estructuras condicionales

- if-else <> if-else if-else

```
let price = 65;

if(price < 50) {
    console.log("Esto es barato!");
} else if (price < 100) {
    console.log("Esto no es barato...");
} else {
    console.log("Esto es caro!");
}
```

iespereMariaorts

Estructuras condicionales

- switch: evalua variable y ejecuta bloque correspondiente

- "break": finaliza el bloque correspondiente

```
let userType = 1;

switch(userType) {
   case 1:
   case 2: // Tipos 1 y 2 entran aquí
      console.log("Puedes acceder a esta zona");
      break;
   case 3:
      console.log("No tienes permisos para acceder aquí");
      break;
   default: // Ninguno de los anteriores
      console.error("Tipo de usuario erróneo!");
}
```

iesperemariaorts

Estructuras condicionales

- En JavaScript "switch" como un "if"

```
let age = 12;

switch(true) {
   case age < 18:
      console.log("Eres muy joven para entrar");
      break;
   case age < 65:
      console.log("Puedes entrar");
      break;
   default:
      console.log("Eres muy mayor para entrar");
}
```

iesperemariaorts

Bucles

- while: evalua condición y ejecuta hasta que sea falsa
- Puede no ejecutarse nunca

```
let value = 1;

while (value <= 5) { // Imprime 1 2 3 4 5
   console.log(value++);
}
```

iesperemariaorts

Bucles

- do-while: evalua condición al final del bloque de instrucciones
- Se ejecuta almenos una vez

```
let value = 1;

do { // Imprime 1 2 3 4 5
   console.log(value++);
} while (value <= 5);
```

iesperemariaorts

Bucles

- for: puedes inicializar una o mas variables y también ejecutar diversas instrucciones en cada iteración separandolas con comas.

```
let limit = 5;

for (let i = 1, j = limit; i <= limit && j > 0; i++, j--) {
   console.log(i + " - " + j);
}
/* Imprime
1 - 5
2 - 4
3 - 3
4 - 2
5 - 1
*/
```

iesperemariaorts

Tipos de datos básicos: Números

- No hay diferencia entre enterosy decimales (float, double)

```
console.log(typeof 3); // Imprime number
console.log(typeof 3.56); // Imprime number
```

- Son objetos y podemos acceder a métodos y propiedades

```
console.log(3.32924325.toFixed(2)); // Imrpime 3.33
console.log(5435.45.toExponential()); // Imrpime 5.43545e+3
console.log((3).toFixed(2)); // Imrpime 3.00 (Un entero necesita estar dentro de un paréntesis para poder acceder a sus propiedades)
```

iesperemariaorts

Tipos de datos básicos: Números

- Podemos realizar las operaciones típicas (+,-,*,/,%,…)
- Operaciones numéricas con valores no numéricos -> NaN (Not a Number)

```
let a = 3;
let b = "asdf";
let r1 = a * b; // b es “asdf”, y no será transformado a número
console.log(r1); // Imprime NaN

let c;
let r3 = a + c; // c es undefined, no será transformado a número
console.log(r3); // Imprime NaN

let d = "12";
console.log(a * d); // Imprime 36. d puede ser transformado al  número 12
console.log(a + d); // Imprime 312. El operador + concatena si hay un string
console.log(a + +d); // Imprime 15.  El operador ‘+’ delante de un valor lo transforma en numérico
```

iesperemariaorts

Tipos de datos básicos: undefined y null

- undefined: ausencia de valor (avisa de variables no inicializadas)
- null: tipo de valor (referencia vacía)

```
let value; // Value no ha sido asignada (undefined)
console.log(typeof value); // Imprime undefined

value = null;
console.log(typeof value); // Imprime object
```

iespereMariaorts

Tipos de datos básicos: Boolean

- Se representan en minúsculas (true, false)
- Se pueden negar usando el operador **!** delante del valor.

iesperemariaorts

Tipos de datos básicos: Strings

- Se representan dentro de ‘comillas simples’ o “comillas dobles”
  - Podemos utilizar el operador + para concatenar cadenas.

```
let s1 = "Esto es un string";
let s2 = 'Esto es otro string';

console.log(s1 + " - " + s2); // Imrpime: Esto es un string – Esto es otro string


        console.log("Hello 'World'"); // Imprime: Hello 'World'
        console.log('Hello \'World\''); // Imprime: Hello 'World'

        console.log("Hello \"World\""); // Imprime: Hello "World"
        console.log('Hello "World"'); // Imprime: Hello "World"
```

iesperemariaorts

Tipos de datos básicos: Strings

- Son objetos y tienen métodos que podemos utilizar
- Estos métodos no modifican el valor

```
let s1 = "Esto es un string";
// Obtener la longitud del string
console.log(s1.length); // Imprime 17

// Obtener el carácter de una cierta posición del string (Empieza en 0)
console.log(s1.charAt(0)); // Imprime "E"

// Obtiene el índice de la primera ocurrencia
console.log(s1.indexOf("s")); // Imprime 1

// Obtiene el índice de su última ocurrencia
console.log(s1.lastIndexOf("s")); // Imprime 11
```

iesperemariaorts

Tipos de datos básicos: Conversión de tipos

```
let num1 = 32;
let num2 = 14;

// Cuando concatenamos un string, el otro operando es convertido a string
console.log(String(32) + 14); // Imprime 3214
console.log("" + 32 + 14); // Imprime 3214
```

```
let s1 = "32";
let s2 = "14";
console.log(Number(s1) + Number(s2)); // Imprime 46
console.log(+s1 + +s2); // Imprime 46
```

iesperemariaorts

Variables globales

- Variable declarada en bloque principal: global
- Variable declarada sin "let": global, para evitar esto: "strict mode"

```
let  global = "Hello";

function cambiaGlobal() {
   global = "GoodBye";
}

cambiaGlobal();
console.log(global); // Imprime "GoodBye"
console.log(window.global); // Imprime "GoodBye"
```

```
'use strict';

function changeGlobal() {
   global = "GoodBye";
}

changeGlobal(); // Error → Uncaught ReferenceError: global is not defined
```

iespereemariaorts

Variables definidas en funciones

- Son variables locales

```
function setPerson() {
   let person = "Peter";
}

let person = "John";
setPerson();
console.log(person); // Imprime John
```

```
function setPerson() {
   let  person = "Peter";
}
setPerson();
console.log(person); // Error → Uncaught ReferenceError: person is not defined
```

iesperemariaorts

Operadores: suma

- Suma números o concatena cadenas variables locales

```
console.log(4 + 6); // Imprime 10
console.log("Hello " + "world!"); // Imprime "Hello world!"
console.log("23" + 12); // Imprime "2312"
console.log("42" + true); // Imprime "42true"
console.log("42" + undefined); // Imprime "42undefined"
console.log("42" + null); // Imprime "42null"
console.log(42 + "hello"); // Imprime "42hello"
console.log(42 + true); // Imprime 43 (true => 1)
console.log(42 + false); // Imprime 42 (false => 0)
console.log(42 + undefined); // Imprime NaN (undefined no puede ser convertido a number)
console.log(42 + null); // Imprime 42 (null => 0)
console.log(13 + 10 + "12"); // Imprime "2312" (13 + 10 = 23, 23 + "12" = "2312")
```

iesperemariaorts

Operadores: Aritméticos (-, *, /)

- Operan siempre con números

```
console.log(4 * 6); // Imprime 24
console.log("Hello " * "world!"); // Imprime NaN
console.log("24" / 12); // Imprime 2 (24 / 12)
console.log("42" * true); // Imprime 42 (42 * 1)
console.log("42" * false); // Imprime 0 (42 * 0)
console.log("42" * undefined); // Imprime NaN
console.log("42" - null); // Imprime 42 (42 - 0)
console.log(12 * "hello"); // Imprime NaN ("hello" no puede ser convertido a número)
```

iespereMariaorts

Operadores: Unarios

- Preincrementar (++variable), postincrementar (variable++)
- Predecrementar (--variable), postdecrementar (variable--)
- Operan siempre con números

```
let a = 1;
let b = 5;
console.log(a++); // Imprime 1 y incrementa a (2)
console.log(++a); // Incrementa a (3), e imprime 3
console.log(++a + ++b); // Incrementa a (4) y b (6). Suma (4+6), e imprime 10
console.log(a-- + --b); // Decrementa b (5). Suma (4+5). Imprime 9. Decrementa a (3)
```

- Transformar a número con "+"

```
let a = "12";
let b = "13";
let c = true;
console.log(a + b); // Imprime "1213"
console.log(+a + +b); // Imprime 25 (12 + 13)
console.log(+b + +c); // Imprime 14 (13 + 1). True -> 1
```

iesperemariaorts

Operadores: Relacionales

- Podemos usar == o === para comparar la igualdad (o el contrario !=, !==)
  - "==" no tienen en ccuenta el tipo de datos, compara valores
  - "===" compara valores y tipo de datos

```
console.log(3 == "3"); // true
console.log(3 === "3"); // false
console.log(3 != "3"); // false
console.log(3 !== "3"); // true
// Equivalente a falso (todo lo demás es equivalente a cierto)
console.log("" == false); // true
console.log(false == null); // false (null no es equivalente a cualquier boolean).
console.log(false == undefined); // false (undefined no es equivalente a cualquier boolean).
console.log(null == undefined); // true (regla especial de JavaScript)
console.log(0 == false); // true
console.log({} >= false); // Object vacío -> false
console.log([] >= false); // Array vacío -> true
```

iesperemariaorts

Operadores: Relacionales

- Operadores menor que (<), mayor que (>), menor o igual que (<=), y mayor o igual que (>=)
- Compara codificación Unicode dígito a dígito

```
console.log(6 >= 6); // true
console.log(3 < "5"); // true ("5" → 5)
console.log("adiós" < "bye"); // true
console.log("Bye" > "Adiós"); // true
console.log("Bye" > "adiós"); // false. Las letras mayúsculas van siempre antes
console.log("ad" < "adiós"); // true
```

iesperemariaorts

Operadores: Booleanos

- Negación (!), y (&&), o (||).

```
console.log(!true); // Imprime false
console.log(!(5 < 3)); // Imprime true (!false)

console.log(4 < 5 && 4 < 2); // Imprime false (ambas condiciones deben ser ciertas)
console.log(4 < 5 || 4 < 2); // Imprime true (en cuanto una condición sea cierta, devuelve cierta y deja de comparar)
```

iesperemariaorts

FIN