Front End Web Developer

Módulo 2 :: Javascript (ES6)
Profesor: Saverio Trioni
Convocatoria de 2022 de los programas de formación profesional para el empleo, de especialidades de la oferta de formación no formal, para personas trabajadoras ocupadas, que promueve el Consorcio para la Formación Continua de Cataluña (ref. BDNS 648938.).

Índice

1. Introducción al Módulo 2
2. Objetivos de las sesiones sobre JavaScript
3. Historia y Evolución de JavaScript
4. Fundamentos de JS – Variables
5. Fundamentos de JS - Tipos de Datos
6. Objetos y Valores Especiales
7. Fundamentos de JS – Operadores
8. Fundamentos de JS – Expresiones
9. Estructuras de Control de Flujo
10. Alcance
11. Cierre y Conclusiones

1. **Introducción al Módulo 2**
2. Objetivos de las sesiones sobre JavaScript
3. Historia y Evolución de JavaScript
4. Fundamentos de JS – Variables
5. Fundamentos de JS - Tipos de Datos
6. Objetos y Valores Especiales
7. Fundamentos de JS – Operadores
8. Fundamentos de JS – Expresiones
9. Estructuras de Control de Flujo
10. Alcance
11. Cierre y Conclusiones

5h

3h
30' 1h 30’
1h
Contenido
Introducción
Estructura de la clase
Concepto Ejemplo Ejercicio
1h
Test / Repaso Práctica / Entregable
Práctica / Entregable

Saverio Trioni

saverio.trioni@salle.url.edu

- Matemático (SNS & UAB)
- Innovador y creador de soluciones Web
- Emprendedor

Introducción

En 20 años de experiencia creando software para proyectos de investigación, startups, agencias gubernamentales y grandes empresas, he podido ver cuales son los aspectos más importantes de un proyecto de ingeniería de software.

Mi objetivo en mi trabajo y en este curso es mantener la complejidad bajo control, para asegurar que las soluciones que propongáis aquí y en vuestros futuros empleos sean durables y bien mantenibles.

Nombre

- Estudios y experiencia

.....
Introducción
Alumno

1. Introducción al Módulo 2
2. **Objetivos de las sesiones sobre JavaScript**
3. Historia y Evolución de JavaScript
4. Fundamentos de JS – Variables
5. Fundamentos de JS - Tipos de Datos
6. Objetos y Valores Especiales
7. Fundamentos de JS – Operadores
8. Fundamentos de JS – Expresiones
9. Estructuras de Control de Flujo
10. Alcance
11. Cierre y Conclusiones

Objetivos de las sesiones sobre Javascript

Así como en HTML codificamos el contenido y la semántica, en CSS generamos la presentación, con Javascript agregaremos la lógica de comportamiento de una página o aplicación web

Entre las referencias que nos pueden ayudar a cumplir estos objetivos tenemos:

- MDN JS Guide: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
- MDN JS Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
- Compatibility Check - Can I use: https://caniuse.com/

1. Introducción al Módulo 2
2. Objetivos de las sesiones sobre JavaScript
3. **Historia y Evolución de JavaScript**
4. Fundamentos de JS – Variables
5. Fundamentos de JS - Tipos de Datos
6. Objetos y Valores Especiales
7. Fundamentos de JS – Operadores
8. Fundamentos de JS – Expresiones
9. Estructuras de Control de Flujo
10. Alcance
11. Cierre y Conclusiones

Introducción a JS

JavaScript es un lenguaje de scripting multiplataforma. Es un lenguaje compilado en tiempo de ejecución, por lo que no es necesario compilar los programas antes de ejecutarlos. Se trabaja igual que en un lenguaje interpretado.

Tiene elementos de muchos paradigmas: imperativo, funcional, orientado a objetos. Es un lenguaje débilmente tipado y dinámico.

Está basado en ECMAScript, un lenguaje de programación estandarizado por Ecma International, la asociación europea para la creación de estándares para la comunicación y la información.

La especificación ECMAScript es un conjunto de requisitos

Introducción a JS – Breve Historia

La primera versión de JavaScript (diseñada en 10 días) aparece en Netscape 2.0. En 1995 SUN Microsystems y Netscape decidieron darle el nombre JavaScript (antes se llamaba Mocha y LiveScript)

En 1996, Internet Explorer 3.0 incluye una versión propia del ECMAScript llamada JScript, implementando alguna de las características del estándar definido, y añadiendo características adicionales propietarias

En 1997 se propuso este lenguaje como estándar y la European Computer Manufacturers Association (ECMA) lo adopta como tal, de ahí que también se le llame ECMAScript

Introducción a JS – Breve Historia

En 1998 y a raíz de las diferencias en las características e implementación de ECMAScript en cada navegador, además de la lentitud en la interpretación del JS, la W3C (World Wide Web Consortium) diseñó el estándar DOM, que es un interfaz (una API) para acceder y modificar el contenido estructurado del documento. Normalizando y estandarizando así el uso de este lenguaje de scripting en todos los navegadores

En 2015 se publica la versión ES6 (ECMAScript 2015) con grandes cambios, pasando a actualizaciones anuales.

Introducción a JS – Diferentes Motores

Como ya se ha mencionado, la implementación de ECMAScript en cada navegador es distinta, y cada uno utiliza su propio motor de JavaScript.

La versión ES6 (2015) es soportada a partir de las siguientes versiones de los siguientes navegadores:

- Google Chrome (motor Chromium V8) v.51 (Mayo 2016)
- Mozilla Firefox (motor SpiderMonkey) v.52 (Marzo 2017)
- Safari (motor JavaScriptCore) v.10 (Septiembre 2016)
- Microsoft Edge (motor basado en Chromium, antiguamente IE usaba ChakraCore) v.14 (Agosto 2016)

1. Introducción al Módulo 2
2. Objetivos de las sesiones sobre JavaScript
3. Historia y Evolución de JavaScript
4. **Fundamentos de JS – Variables**
5. Fundamentos de JS - Tipos de Datos
6. Objetos y Valores Especiales
7. Fundamentos de JS – Operadores
8. Fundamentos de JS – Expresiones
9. Estructuras de Control de Flujo
10. Alcance
11. Cierre y Conclusiones

- Diréctamente en consola de navegador
- Embebido en el código html:

```
<script type="text/javascript">
  aquí
</script>
```

- Incluir un fichero

```
<script type="text/javascript" src="main.js"></script>
```

- Sandboxes
  - Codepen.io
  - JSFiddle
  - etc
  - Node

Consola (Chrome)

Ejemplo y Ejercicio

■ Botón derecho -> Inspect Element

■ Usamos console.log o console.dir para mostrar resultados, aunque
el resultado también se evalúa.

■ Ejercicio: ¿Cuál es la diferencia entre console.log y console.dir?

- Case-sensitive
- Los espacios en blanco (space, tab) no aportan significado
- Poner un `;` al final de sentencia es opcional
- No hacerlo puede afectar la interpretación
- Sed consistentes! (y recomendamos usar ESLint para automatizar esta consistencia)
- Comentarios:
  - `/* de bloque */`
  - `// de línea`
  - `/** de bloque que se usa como documentación */`

Características básicas

Ejemplo y Ejercicio

Como todo lenguaje de programación, las variables se utilizan para almacenar datos.

Distintos tipos de “variables”:

- `var` : variables que por definición cambian su valor y el alcance es la función en la que están definidas. Forma antígua de definir variables que ha sido sustituida por las siguientes.
- `const` : constantes que no cambian su valor una vez definidas y su alcance es el bloque donde están definidas
- `let` : variable accesible en el bloque en el que está definida.

No tienen un tipo predefinido, pueden guardar lo que sea (e incluso cambiar de tipo).
Ejemplo

Recomendaciones

- Usar nombres semánticos
- Nombres de constantes siempre en mayúsculas separadas por infra
  guión
- Nombres de variables camelCase
- Definir constantes primero y al principio del bloque
- Definir variables inmediatamente después de las constantes (al principio del bloque)
- Todos los valores que no cambian a lo largo del código definirlos como constantes

1. Introducción al Módulo 2
2. Objetivos de las sesiones sobre JavaScript
3. Historia y Evolución de JavaScript
4. Fundamentos de JS – Variables
5. **Fundamentos de JS - Tipos de Datos**
6. Objetos y Valores Especiales
7. Fundamentos de JS – Operadores
8. Fundamentos de JS – Expresiones
9. Estructuras de Control de Flujo
10. Alcance
11. Cierre y Conclusiones

Sintaxis básica de Javascript – Tipos de Datos

Concepto y Ejemplo

■ Cualquier valor que se utiliza en JavaScript es de un cierto tipo. En JavaScript existen los siguientes tipos de datos primitivos:

■ number:

■ Puede contener números enteros (integer), decimales (float), hexadecimales,
octales, exponentes y

■ los números especiales NaN e Infinity.

■ string: Cualquier número de caracteres entre comillas simples, dobles o tildes

■ boolean: puede ser TRUE or FALSE

■ undefined: Es un tipo de datos con un solo valor: undefined

1. Introducción al Módulo 2
2. Objetivos de las sesiones sobre JavaScript
3. Historia y Evolución de JavaScript
4. Fundamentos de JS – Variables
5. Fundamentos de JS - Tipos de Datos
6. **Objetos y Valores Especiales**
7. Fundamentos de JS – Operadores
8. Fundamentos de JS – Expresiones
9. Estructuras de Control de Flujo
10. Alcance
11. Cierre y Conclusiones

■ La clase Object representa uno de los tipos de datos en Javascript. Es usado para guardar una colección de datos definidos y entidades más complejas.
■ Si queremos definir un objeto, utilizamos llaves {}
■ Los elementos de un objeto (propiedades) los separamos por coma
■ El par clave/valor (key/value) lo dividimos con dos puntos :
■ Las claves (keys, nombres de las propiedades) pueden ir entre comillas

```
let hero = {
  name: "Ludovico", class: "Warlock", level: 10
};
```

Hay 2 maneras de acceder a la propiedad de un objeto:

- Con la notación de corchetes: `hero['name']`
- Con la notación de puntos: `hero.name`

Los objetos pueden contener otros objetos:

```
let book = {
  name: 'Catch-22',
  published: 1961,
  author: {
    firstname: 'Joseph',
    lastname: 'Heller'
  }
};
```

■ Los objetos pueden contener como propiedad una función, en ese caso decimos que esta propiedad es un método del objeto:

```
let dog = {
  name: 'Benji',
  talk: function() { alert('Woof, woof!'); }
};
```

Podemos definir un objeto vacío y luego añadirle (y quitarle) propiedades y métodos.

```
let hero = {};
typeof hero.breed // "undefined"
hero.breed = 'turtle';
hero.name = 'Leonardo';
hero.sayName = function() { return hero.name; };
hero.sayName(); // "Leonardo"
delete hero.name; // true
hero.sayName(); // Reference to undefined property hero.name
```

Otra forma de crear objetos es mediante funciones constructoras.

Para crear objetos con estas funciones hay que usar el operador new. La ventaja que tiene utilizar estas funciones constructoras es que aceptan parámetros al crear objetos.

```
function Hero(name) {
  this.name = name;
  this.occupation = 'Ninja';
  this.whoAreYou = function() { return "I'm " + this.name + " and I'm a " + this.occupation; }
}

const michelangelo = new Hero('Michelangelo');
const donatello = new Hero('Donatello');
michelangelo.whoAreYou(); // "I'm Michelangelo and I'm a Ninja"
donatello.whoAreYou(); // "I'm Donatello and I'm a Ninja"
```

Un objeto contiene las siguientes propiedades o métodos:

- La propiedad `constructor` que identifica a la función constructora del objeto.
- El método `toString()` que devuelve el objeto en formato texto.
- El método `valueOf()` que devuelve el valor del objeto

- Son diccionarios clave-valor
- Podemos guardar objetos dentro de objetos
- Son extremadamente dinámicos y redefinibles
- Podemos añadir y quitar propiedades
- Podemos usar propiedades dinámicamente
- Si una propiedad no existe, undefined

Arrays

Un array es un conjunto de valores con un único nombre de variable.

Para asignar valores a un array encerramos los elementos entre corchetes [] (array literal notation)

Los elementos de un array son indexados con números consecutivos a partir de 0.

Para acceder a un elemento del array especificamos el índice entre corchetes.

```
let students = ['Albert', 'Xavier'];
```

Hay dos maneras de crear un array:

- Utilizando la notación literal: `let a = [];`
- Utilizando la función constructora Array():
  `let b = Array();`

Si usamos el constructor podemos pasarle parámetros:

- Varios parámetros: serán asignados como elementos del array
- Un número: se considerará el tamaño del array

```
let a = new Array(1,2,3,'four'); a; // [1, 2, 3, "four"]
let a2 = new Array(5);
a2; // [undefined, undefined, undefined, undefined, undefined]
```

Además los arrays disponen de la propiedad length:

Que nos devuelve el tamaño del array (número de elementos)

Podemos modificar y cambiar el tamaño del array

```
a[0] = 1;
a.length; // 1
a.length = 5;
a; // [1, undefined, undefined, undefined, undefined]
a.length = 2; // 2
a; // [1, undefined]
```

Los arrays definidos como objetos, disponen de unos cuantos métodos muy útiles:

- push()
- pop()
- sort()
- join()
- slice()
- splice()

Ejercicio: busca (en MDN) la definición de cada uno de los métodos listados arriba y luego haz un ejemplo para cada uno de ellos (se hará presentación en clase).

Los arrays:

- Son listas dinámicas de valores
- Podemos consular y modificar su tamaño con la propiedad length
- Técnicamente son objetos (las claves se autogeneran en orden numérico).
- Tienen métodos: `push`, `pop`, `sort`, `join`, `slice`, `splice`, `includes`...

  Ejemplo

1. Introducción al Módulo 2
2. Objetivos de las sesiones sobre JavaScript
3. Historia y Evolución de JavaScript
4. Fundamentos de JS – Variables
5. Fundamentos de JS - Tipos de Datos
6. Objetos y Valores Especiales
7. **Fundamentos de JS – Operadores**
8. Fundamentos de JS – Expresiones
9. Estructuras de Control de Flujo
10. Alcance
11. Cierre y Conclusiones

Sintaxis básica de Javascript – Operadores

Clasificamos a los operadores en:

- Operadores aritméticos
- Operadores de asignación
- Operadores lógicos
- Operadores de comparación
- Operadores de cadena
- Operador condicional (ternario)
- Operador coma

- Operadores unarios
- Operadores relacionales
- Operadores bit a bit

Los Operadores toman uno o dos valores (o variables), realizan una operación, y devuelven un valor

Conversiones de Tipo al realizar una operación

Introducción

Operadores aritméticos

- Addition: `1 + 2; // 3`
- Subtraction: `2 - 1; // 1`
- Multiplication: `2 * 3; // 6`
- Division: `6 / 4; // 1.5` (Division by 0 returns `Infinity`)
- Remainder: `5 % 3; // 2`
- Power: `2 ** 3; // 8`
- Increment/decrement value in 1:

```
let base = 100;
let baseMasUno = base++;
let baseMenosUno = base--;
```

Operadores de asignación

El operador simple de asignación es `=`.

A la izquierda del operador hay una variable (`let` o `var`), a la derecha una expresión que se evalúa y se asigna a la variable.

El valor de la expresión es igual al valor asignado a la variable.

```
a = 3 + 2; // 5
```

Los operadores aritméticos, binarios y de bits pueden combinarse con el de asignación. Ej:

```
let base = 1;
base += 5; // 6
```

Operadores Lógicos

- `&&` (AND lógico). `expr1 && expr2` devuelve `expr1` si se puede convertir a `false`; de lo contrario, devuelve `expr2`. Por lo tanto, cuando se usa con valores booleanos, `&&` devuelve `true` si ambos operandos son `true`; de lo contrario, devuelve `false`.
- `||` (OR lógico). `expr1 || expr2` devuelve `expr1` si se puede convertir a `true`; de lo contrario, devuelve `expr2`. Por lo tanto, cuando se usa con valores booleanos, `||` devuelve `true` si alguno de los operandos es `true`; si ambos son falsos, devuelve `false`.
- `!` (NOT lógico). `!expr` devuelve `false` si su único operando se puede convertir a `true`; de lo contrario, devuelve `true`.

Los operadores lógicos se utilizan normalmente con valores booleanos (lógicos); cuando lo son, devuelven un valor booleano.

Sin embargo, los operadores `&&` y `||` en realidad devuelven el valor de uno de los operandos especificados, por lo que si estos operadores se utilizan con valores no booleanos, pueden devolver un valor no booleano.

Operadores de Comparación

Un operador de comparación compara sus operandos y devuelve un valor lógico en función de si la comparación es verdadera (true) o falsa (false).

Los operandos pueden ser valores numéricos, de cadena, lógicos u objetos.

Las cadenas se comparan según el orden lexicográfico estándar, utilizando valores Unicode.

En la mayoría de los casos, si los dos operandos no son del mismo tipo, JavaScript intenta convertirlos a un tipo apropiado para la comparación.

Este comportamiento generalmente resulta en comparar los operandos numéricamente.

Las únicas excepciones a la conversión de tipos dentro de las comparaciones involucran a los operadores `===` y `!==`, que realizan comparaciones estrictas de igualdad y desigualdad. Estos operadores no intentan convertir los operandos a tipos compatibles antes de verificar la igualdad.

- `==` Comparación de igualdad: Devuelve `true` cuando ambos operandos son iguales. Los operandos se convierten en el mismo tipo antes de ser comparados
- `===` Identidad: Devuelve `true` si ambos operandos son iguales y del mismo tipo. Por lo general es mejor y más seguro si se compara de esta manera, porque evitamos conversiones de tipos
- `!=` La comparación de no igualdad: Devuelve `true` si los operandos no son iguales entre sí (después de una conversión de tipo)
- `!==` La comparación de no igualdad sin conversión de tipo: Devuelve `true` si los operandos no son iguales o son diferentes tipos.
- `<`, `<=`, `>` , `>=`

Busca información sobre los siguientes operadores y haz una breve presentación sobre cada uno de ellos (cada grupo presentará uno de los tipos de operadores):

- Operadores de Cadena
- Operador Condicional Ternario
- Operador Coma (no se usa)
- Operadores Unarios
- Operadores Relacionales
- Operadores Bit a Bit (infrecuentemente usados)
- 15’ búsqueda de información y preparación de presentación
- 15’ presentación (3’ por equipo)

Conversiones de Tipo

Si utilizamos un numero entre comillas (string) en una operación aritmética JavaScript lo convierte en número

- ¡Cuidado! `undefined` y `null` devuelven cosas diferentes al convertirlas a numero.
  (Hacer: `1 * undefined;` y hacer: `1 * null;`)

Si utilizamos true o false entre comillas JavaScript lo convierte a string

Tradicionalmente la doble negación se ha usado para convertir cualquier valor en su Booleano correspondiente. Actualmente es más elegante y limpio usar `Boolean(objeto)`. Aplicándolo podemos comprobar como cualquier valor convertido a Booleano es true excepto:

- `""`
- `null`
- `undefined`
- `0`
- `NaN`
- `false`

Ejemplo

Precedencia de Operadores

La precedencia de los operadores determina el orden en que se aplican al evaluar una expresión. Puedes redefinir la precedencia de los operadores mediante el uso de paréntesis y en caso de duda es mejor hacerlo (mejor explícito que implícito).

La siguiente tabla describe la precedencia de los operadores, de mayor a menor.

Ejercicio

Ejecuta las siguientes sentencias e indica el resultado:

```
let s = '1n'; s++; // Resultado:
!!"false" ; // Resultado:
!!undefined; // Resultado:
undefined === null; // Resultado:
false == ""; // Resultado:
0 == ""; // Resultado:
typeof 2e3; // Resultado:
typeof "2e3"; // Resultado:
```

Ejercicio

¿Cuál es el valor de x después de las siguientes sentencias?

```
let x;
x = x || 10;
x = 5; x = x || 10;
x = 20; x = x || 10;
x = null; x = x || 10; //Resultado:
```

Ejercicio

Ejecuta las siguientes sentencias e indica el resultado:

```
parseInt(1e1) ; // Resultado:
parseInt('1e1') ; // Resultado:
parseFloat('1e1') ; // Resultado:
isFinite(0 / 10) ; // Resultado:
isFinite(20 / 0) ; // Resultado:
isNaN(0 / 0) ; // Resultado:
isNaN(parseInt('0/0')) ; // Resultado:
isNaN(parseInt(NaN)) ; // Resultado:
```

¿Qué hacen las siguienes funciones?

- `parseInt`
- `parseFloat`
- `isFinite`
- `isNaN`

Índice

1. IntroducciónalMódulo2
2. ObjetivosdelassesionessobreJavaScript 3. HistoriayEvoluciónde JavaScript
3. FundamentosdeJS–Variables
4. FundamentosdeJS-TiposdeDatos
5. ObjetosyValoresEspeciales
6. FundamentosdeJS–Operadores
7. FundamentosdeJS–Expresiones
8. EstructurasdeControldeFlujo 10. Alcance 11.Cierre y Conclusiones

Sintaxis básica de Javascript – Expresiones

■ Una expresión es cualquier unidad de código válida que se resuelve en un valor.
■ Toda expresión sintácticamente válida se resuelve en algún valor, pero conceptualmente, hay dos tipos de expresiones:
■ con efectos secundarios (por ejemplo: las que asignan valor a una variable). Ej.: x = 7; (las llamamos statements)
■ las que en algún sentido evalúan y por lo tanto se resuelven en un valor. Ej.: 3 + 4; (no asigna el resultado a ninguna variable). (las llamamos expresiones)

Sintaxis básica de Javascript – Expresiones

■ JavaScript tiene las siguientes categorías de expresión:
■ Aritméticas: se evalúa como un número, por ejemplo 3.14159.
(Generalmente usa operadores aritméticos).
■ Cadenas: se evalúa como una cadena de caracteres, por ejemplo, "Fred" o "234". (Generalmente usa operadores de cadena).
■ Lógicas: se evalúan como true o false. (A menudo implica operadores lógicos).
■ Expresiones primarias: palabras clave básicas y expresiones generales en JavaScript.
■ Expresiones del lado izquierdo: los valores del lado izquierdo son el destino de una asignación

Sintaxis básica de Javascript – Expresiones

■ Expresiones primarias: palabras clave básicas y expresiones generales en JavaScript.
■this (lo veremos más adelante en el temario). Utiliza la palabra clave this para hacer referencia al objeto actual. En general, this se refiere al objeto que llama en un método. Usa this con la notación de punto o entre corchetes:
■ this['propertyName']
■ this.propertyName
■ Operador de agrupación (). El operador de agrupación () controla la precedencia de la evaluación en las expresiones.

Sintaxis básica de Javascript – Expresiones

■ Expresiones del lado izquierdo: los valores del lado izquierdo son el destino de una asignación.
■new (lo veremos más adelante en el temario). Puedes utilizar el operador new para crear una instancia de un tipo de objeto definido por el usuario o de uno de los tipos de objeto integrados. Utiliza new de la siguiente manera:
■ let objectName = new objectType([param1,..., paramN]);
■super (lo veremos más adelante en el temario). La palabra clave super se utiliza para llamar a funciones en el padre de un objeto. Es útil con clases llamar al constructor padre, por ejemplo.
■ super([arguments]); // llama al constructor padre.
■ super.functionOnParent([arguments]);

Índice

1. IntroducciónalMódulo2
2. ObjetivosdelassesionessobreJavaScript 3. HistoriayEvoluciónde JavaScript
3. FundamentosdeJS–Variables
4. FundamentosdeJS-TiposdeDatos
5. ObjetosyValoresEspeciales
6. FundamentosdeJS–Operadores
7. FundamentosdeJS–Expresiones
8. EstructurasdeControldeFlujo
9. Alcance

11.Cierre y Conclusiones

Estructuras de Control de Flujo – Bloque

■ JavaScript admite un compacto conjunto de declaraciones, específicamente declaraciones de control de flujo, que puedes utilizar para incorporar una gran cantidad de interactividad en tu aplicación.

■ Bloque: La declaración más básica es una declaración de bloque, que se utiliza para agrupar instrucciones. El bloque está delimitado por un par de llaves:

{
statement1; statement2; ⋮ statementN;
}

Ejemplo

Estructuras de Control de Flujo – Condicionales (if...else)

■ Una expresión condicional es un conjunto de instrucciones que se ejecutarán si una condición especificada es verdadera. JavaScript admite dos expresiones condicionales: if...else y switch.

■ If...else: Utiliza la expresión if para ejecutar una instrucción si una condición lógica es true. Utiliza la cláusula opcional else para ejecutar una instrucción si la condición es false:
if (condition) { statement1;
} else { statement2;
}

■ Se recomienda evitar ifs anidados lectura/seguimiento del código).
(dado que no

Estructuras de Control de Flujo – Condicionales (switch)

■ switch: Una instrucción switch permite que un programa evalúe una expresión e intente hacer coincidir el valor de la expresión con una etiqueta case. Si la encuentra, el programa ejecuta la declaración asociada.

switch (expression) { case value_1:
statements_1
[break;] case value_2:
statements_2 [break;]
...
default: statements_def [break;]
}

Ejemplo

Estructuras de Control de Flujo – Bucles

■ Los bucles ofrecen una forma rápida y sencilla de hacer algo repetidamente.
■ Hay diferentes tipos de bucles, pero esencialmente, todos hacen lo mismo: repiten una acción varias veces. (¡Ten en cuenta que es posible que ese número sea cero!).
■ Los diversos mecanismos de bucle ofrecen diferentes formas de determinar los puntos de inicio y terminación del bucle. Hay varias situaciones que son fácilmente atendidas por un tipo de bucle que por otros:
■ Declaración for, for...in, for...of
■ Declaración while, do...while
■ Declaración break, continue
■ Declaración labeled

Estructuras de Control de Flujo – Bucles (for)

■ for: Un ciclo for se repite hasta que una condición especificada se evalúe como false.
for ([expresiónInicial]; [expresiónCondicional]; [expresiónDeActualización]) { instrucciones
}
■ for...in: La instrucción for...in itera una variable especificada sobre todas las propiedades enumerables de un objeto. Para cada propiedad distinta, JavaScript ejecuta las instrucciones especificadas.
for (variable in objeto) { instrucciones
}

Estructuras de Control de Flujo – Bucles (for)

■ for...of: La declaración for...of crea un bucle que se repite sobre objetos iterables (incluidos Array, Map, Set, argumentos de objetos y así sucesivamente), invocando un bucle de iteración personalizado con declaraciones que se ejecutarán para el valor de cada propiedad.
for (variable of objeto) { instrucciones
}

Estructuras de Control de Flujo – Bucles (while)

■ while: Una declaración while ejecuta sus instrucciones siempre que una condición especificada se evalúe como true. Una instrucción while tiene el siguiente aspecto:
while (condición) { instrucciones
}
■ do...while: a instrucción do...while se repite condición especificada se evalúe como falsa:
do { instrucciones
} while (condición);
hasta
que una

Estructuras de Control de Flujo – Bucles (break)

■ break sirve para terminar un bucle, switch o junto con una declaración etiquetada.
■ Cuando usas break sin una etiqueta, inmediatamente termina el while, do-while, for o switch y transfiere el control a la siguiente declaración.
■ Cuando usas break con una etiqueta, termina la declaración etiquetada especificada.
■ La sintaxis de la instrucción break se ve así: break;
break [label];
■ La primera forma de la sintaxis termina el bucle envolvente más interno o el switch.
■ La segunda forma de la sintaxis termina la instrucción etiquetada específica.

Estructuras de Control de Flujo – Bucles (continue)

■ La instrucción continue se puede usar para reiniciar un while, do-while, for, o declaración label.
■ Cuando utilizas continue sin una etiqueta, finaliza la iteración actual del while, do-while o for y continúa la ejecución del bucle con la siguiente iteración.
■A diferencia de la instrucción break, continue no termina la ejecución del bucle por completo. En un bucle while, vuelve a la condición. En un bucle for, salta a la expresión-incremento.
■ Cuando usas continue con una etiqueta, se aplica a la declaración de bucle identificada con esa etiqueta.
■ La sintaxis de la instrucción continue se parece a la siguiente: continue [label];

Estructuras de Control de Flujo – Bucles (label)

■ Una label proporciona una instrucción con un identificador que te permite hacer referencia a ella en otra parte de tu programa. Por este motivo desaconsejamos su uso, dado que oscurece la lectura del código innecesariamente.

■ Por ejemplo, puedes usar una etiqueta para identificar un bucle y luego usar las declaraciones break o continue para indicar si un programa debe interrumpir el bucle o continuar su ejecución. La sintaxis de la instrucción etiquetada es similar a la siguiente:
label : instrucción

■ El valor de label puede ser cualquier identificador de JavaScript que no sea una palabra reservada. La declaración que identifica a una etiqueta puede ser cualquier enunciado

JS Fundamentals – Ejercicio

■ Escribe un bloque de código donde se definan dos valores constantes (que iremos modificando para probar el código) numéricos y que genere un array con todos los valores impares que hay entre ellos no incluidos
■ Ese array ha de mostrarse por consola
■ Si el valor de la primer constante es mayor al valor de la segunda
constante se ha de mostrar un error
■ Si no hay resultados ha de mostrar un array vacío

JS Fundamentals – Ejercicio

■ Escribe un bloque de código donde se defina un valore constante (que iremos modificando para probar el código) numérico y que genere un array con todos los múltiplos de 13 menores que dicho valor
■ Este array ha de mostrarse por consola

JS Fundamentals – Ejercicio

■ Escribe un bloque de código que muestre por consola todos los múltiplos de 13 inferiores a 150, la suma de todos ellos y la división de dicha suma entre 13

JS Fundamentals – Ejercicio

■ Escribe un bloque de código que calcule la letra del DNI a partir de su número
■ Escribe un bloque de código que dado un DNI verifique si la letra es la correcta
■ Ayuda:
let letrasDNI = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E'];
Referencia

Índice

1. IntroducciónalMódulo2
2. ObjetivosdelassesionessobreJavaScript 3. HistoriayEvoluciónde JavaScript
3. FundamentosdeJS–Variables
4. FundamentosdeJS-TiposdeDatos
5. ObjetosyValoresEspeciales
6. FundamentosdeJS–Operadores
7. FundamentosdeJS–Expresiones
8. EstructurasdeControldeFlujo
9. Alcance

11.Cierre y Conclusiones

Alcance

Definición

■ El contexto actual de ejecución.

■ El contexto en el que los valores y las expresiones son "visibles" o
pueden ser referenciados.

■ Si una variable u otra expresión no está "en el Scope- alcance actual", entonces no está disponible para su uso.

■ Los Scope también se pueden superponer en una jerarquía, de modo que los Scope secundarios tengan acceso a los ámbitos primarios, pero no al revés.

■ Una función sirve como un cierre en JavaScript y, por lo tanto, crea un ámbito, de modo que (por ejemplo) no se puede acceder a una variable definida exclusivamente dentro de la función desde fuera

■ Distintos tipos de “variables”

■ var -> variables que por definición cambian su valor y el alcance es la función en la que están definidas. Forma antígua de definir variables que ha sido sustituida por las siguientes.
■ const -> Constantes que no cambian su valor una vez definidas y su alcance es el bloque donde están definidas
■ let -> variable accesible en el bloque en el que está definida.
Alcance
Repaso
Ejemplo

Índice

1. IntroducciónalMódulo2
2. ObjetivosdelassesionessobreJavaScript 3. HistoriayEvoluciónde JavaScript
3. FundamentosdeJS–Variables
4. FundamentosdeJS-TiposdeDatos
5. ObjetosyValoresEspeciales
6. FundamentosdeJS–Operadores
7. FundamentosdeJS–Expresiones
8. EstructurasdeControldeFlujo
9. Alcance

11.Cierre y Conclusiones

Conclusiones + Q&A

¡Gracias

Saverio Trioni – saverio.trioni@salle.url.edu Alejandro Bremermann– alejandro.bremermann@salle.url.edu
Convocatoria de 2022 de los programas de formación profesional para el empleo, de especialidades de la oferta de formación no formal, para personas trabajadoras ocupadas, que promueve el Consorcio para la Formación Continua de Cataluña (ref. BDNS 648938.).