UNIVERSIDAD
DE MÁLAGA

Iniciación a JavaScript

Flujo de control y Arrays

Daniel Garrido

dgm@uma.es

UNIVERSIDAD
DE MÁLAGA

La sentencia if

Usamos **if** para decirle a JS qué sentencias ejecutar según una condición.

```
if (condition) {
  // sentencias a ejecutar
}
```

```
var x = 5;

if (x > 0) {
  console.log('x is a positive number!');
}
```

Operadores de Comparación

Operadores para comprobar igualdad, desigualdad o diferencias.

```
var myFavoriteNumber = 28;
```

Operador	Significado	Expresiones Verdaderas
==	Igualdad	myFavoriteNumber == 28 myFavoriteNumber == '28' 28 == '28'
===	Igualdad Estricta	myFavoriteNumber === 28
!=	Desigualdad	myFavoriteNumber != 29
!==	Desigualdad Estricta	myFavoriteNumber !== '28' 28 !== '28'
>	Mayor que	myFavoriteNumber > 25 '28' > 25
>=	Mayor que o igual	myFavoriteNumber >= 28 '28' >= 25
<	Menor que	myFavoriteNumber < 30 '28' < 30
<=	Menor que o igual	myFavoriteNumber <= 28 '28' <= 28

Operadores Lógicos

Usados con los operadores de comparación:

```
var posNum = 4;
var negNum = -2;
```

Operador	Significado	Expresiones Verdaderas
&&	Y	posNum > 0 &&negNum < 0 4 > 0 &&-2 < 0
||	O	posNum > 0 || negNum > 0 4 > 0 || -2 > 0
!	NO	!(posNum === negNum) !(posNum < 0)

Usar paréntesis para agrupar múltiples condiciones:

```
var myAge = 28;
if ((myAge >= 0 && myAge < 3)  || myAge > 90) {
  console.log('You\'re not quite in your peak.');
}
```

Cierto vs Falso

JS intentará deducir si un valor es "cierto".

```
var catsRule = true;
if (catsRule)
  console.log('Yay cats!');
```

Valores "falsos": `false`, `""`, `0`, `undefined`, `null`.

```
var name = '';
if (name) {
  console.log('Hello, ' + name);
}
var points = 0;
if (points) {
  console.log('You have ' + points + ' points');
}
var firstName;
if (firstName) {
  console.log('Your name is ' + firstName);
}
```

Evaluación en Cortocircuito

Tan pronto como JS sepa si una operación es cierta o falsa (evaluando de izquierda a derecha), se para su evaluación.

(false &&anything) =>false

(true || anything) =>true

```
var nominator = 5;
var denominator = 0;
if (denominator != 0 && (nominator/denominator > 0)) {
  console.log('Thats a valid, positive fraction');
}
```

La sentencia if/else

Usamos **else** para dar a JS una sentencia alternativa para ejecutar.

```
var age = 28;  if
(age >= 16) {
  console.log('Yay, you can drive!');
} else {
  console.log('Sorry, but you have ' + (16 - age) + ' to wait');
}
```

La sentencia if/elseif/else

Si tenemos múltiles condiciones, podemos usar else if:

```
var age = 20;
if (age >= 35) {
  console.log('You can vote AND hold any place in
  government!'
} else if (age >= 25) {
  console.log('You can vote AND run for the Senate!');
} else if (age >= 18) {
  console.log('You can vote!');
} else {
  console.log('You have no voice in government!');
}
```

La sentencia switch

Usar `switch` cuando tenemos múltiples alternativas.

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

La sentencia switch

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  ...
    }
```

¡TIEMPO DE EJERCICIOS!
if-else

El bucle while

El bucle `while` permite repetir sentencias mientras una condición es cierta:

```
while (expression) {
  // sentencias a repetir
}
```

```
var x = 0;
while (x < 5) {
  console.log(x);
  x = x + 1;
}
```

El bucle do-while

El bucle `do-while` garantiza una iteración al menos:

```
do {
  // sentencias a repetir
}
while (condition);
```

```
var x = 0;
do {
  console.log(x);
  x = x + 1;
} while (x < 5);
```

El bucle for

Para cuando conocemos el número de iteraciones:

```
for (initialize; condition; update) {
  // sentencias a repetir
}
```

```
for (var i = 0; i < 5; i++) {
  console.log(i);
}
```

La sentencia break

Para salir prematuramente de un bucle, utilizamos la sentencia break:

```
for (var current = 100; current < 200; current++)
  {  console.log('Testing ' + current);
  if (current % 7 == 0) {
    console.log('Found it! ' + current);
    break;
  }
}
```

La sentencia continue

Para terminar prematuramente una iteración, utilizamos la sentencia continue:

```
for (var current = 100; current < 200; current++)
  {  if (current % 2 == 0) {
    continue;
  }

  console.log('Odd ' + current);
}
```

¡TIEMPO DE EJERCICIOS!
Bucles

El tipo de datos Array

Un array almacena una colección ordenada de valores de cualquier tipo:

```
var arrayName = [element0, element1, ...];
```

```
var rainbowColors = ['Red', 'Orange', 'Yellow', 'Green', 'Blue
var raceWinners = [33, 72, 64];
var myFavoriteThings = ['Broccoli', 60481, 'Love Actually'];
```

La propiedad length indica el tamaño del array:

```
console.log(rainbowColors.length);
```

Acceso a los elementos

Para acceder a los elementos utilizamos [ ]. El primer índice comienza en 0.

```
var arrayItem = arrayName[indexNum];
```

```
var rainbowColors = ['Red', 'Orange', 'Yellow', 'Green', 'Blue
var firstColor = rainbowColors[0];
var lastColor = rainbowColors[6];
```

Cambiando los elementos

Podemos modificar un elemento de un array con [ ]:

```
var myFavoriteThings = ['Broccoli', 60481, 'Love Actually'];
myFavoriteThings[0] = 'Celery Root';
```

O para añadir elementos:

```
myFavoriteThings[4] = 'Playgrounds';
```

También podemos usar el método push:

```
myFavoriteThings.push('Dancing'); // pop elimina el último ele
```

Bucles for con arrays y strings

Podemos usar un bucle for para procesar fácilmente cada elemento de un array:

```
var rainbowColors = ['Red', 'Orange', 'Yellow', 'Green', 'Blue
for (var i = 0; i < rainbowColors.length; i++) {
  console.log(rainbowColors[i]);
}
```

o string:

```
var rainbowColorsLetters = 'ROYGBIV';
for (var i = 0; i < rainbowColorsLetters.length; i++) {
  console.log(rainbowColorsLetters[i]);
}
```

Ordenar arrays

s o r t ordena los elementos de un array

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
```

Para números, hace falta usar una función de ordenación

```
nums=[3,12,4];
nums.sort((a,b) => a-b);
```

reverse invierte los elementos de un array

¡TIEMPO DE EJERCICIOS!
Arrays

Tratamiento de Errores

Para el tratamiento de errores, JS dispone de bloques try/catch y throw muy similares a los de otros lenguajes

```
try {
  Bloque de código a proteger
}
catch(err) {
  Manejo de errores
}
finally {
  Siempre se ejecuta
}
```

Tratamiento de Errores

```
try {
  nonExistentFunction();
}
catch(error) {
  console.error(error);
  // ReferenceError: nonExistentFunction is not defined
}
```

La sentencia throw

- throw nos permite generar nuestros propios errores
- Se pueden elevar strings, números, booleanos y objetos

```
throw "Too big";      // throw texto
throw 500;            // throw número
```

El objeto Error

JavaScript tiene un objeto Error predefinido con 2 propiedades que se establecen cuando se producen errores del sistema:

1. name: Establece o devuelve un nombre de error
2. message: Establece o devuelve un mensaje descriptivo

```
  try {
    x=y+1;
} catch (err) {
    console.log(err.message);
}
```

EJERCICIOS ADICIONALES

- Strings
- El adivinador de palabras

Acknowledgements / Reconocimientos:

Gran parte del material del curso procede originalmente de la web Teaching Materials Posteriormente, han sido enriquecidos con material adicional de diversas fuentes públicas así como adaptados a las necesidades específicas del curso a impartir.