I.E.S. PV
PALOMERAS VALLECAS

SINTAXIS DEL LENGUAJE (II)

UT2.- INTRODUCCIÓN A JAVASCRIPT

Susana López Luengo

Estructuras de control. SENTENCIAS CONDICIONALES

if … else

- Ejecuta una sentencia si una condición especificada es evaluada como verdadera. 3 sintaxis:

SINTAXIS 1

- if (expresion){
  instrucciones
  }

```
if (hora < 10) {
        saludo = "Buenos días";
}
```

if … else

SINTAXIS 2

- if (expresion) {
  instrucciones si true}
  else{
  instrucciones si false}

```
if (hora < 10) {
        saludo = "Buenos días";
    } else {
        saludo = "Buenas tardes";
    }
```

if … else

SINTAXIS 3

- if (expresion1){
  instrucciones_1}
  else if (expresion2){
  instrucciones_2}
  else {
  instrucciones_3}

```
if (hora < 10) {
    saludo = "Buenos días";
} else if (hora < 20) {
    saludo = "Buenas tardes";
} else {
   saludo = "Buenas noches";
}
```

OPERADOR CONDICIONAL TERNARIO ?:

- El operador condicional: **?:**
  - devuelve un valor en función de una condición lógica
    - Es una **versión más funcional** del operador **if/else**
- Sintaxis: **condición ? <v1> : <v2>**
  - devuelve **<v1>** -> si **condición** es equivalente a **true**
  - devuelve **<v2>** -> en caso contrario

```
true  ? 1 : 7      => 1
false ? 1 : 7      => 7
```

```
7   ? 1 : 7        => 1
""  ? 1 : 7        => 7
```

EJEMPLO:

```
resultado = (edad>=18) ? "Mayor de edad" : "Menor de edad";
```

UTILIZANDO LA SENTENCIA IF SERÍA:

```
if (edad >= 18) {
  resultado = “Mayor de edad”;
}
else {
  resultado = “Menor de edad”;
}
```

Ver ejemplo opcondicional.html

Estructuras de control

Sentencia switch

- La sentencia switch evalúa una expresión, comparando la expresión con un conjunto de valores predefinidos, y ejecuta comandos según el caso

```
switch (fruta) {
  case "Naranjas":
    alert("Las naranjas están a 2.59 el kilo.");
    break;
  case "Mangos": alert (“Los mangos están a 1.99 el kilo”);
    break;
  case "Papayas":
    alert("Mangos y papayas están a 5,9 el kilo.");
    break;
  default:
    alert("Lo siento, no queda nada de " + fruta + ".");
}
```

Estructuras de control

Sentencia switch

- La sentencia if-else equivalente al switch anterior:

```
if (fruta === “Naranjas”) {
    alert("Las naranjas están a 2.59 el kilo.");}
else if (fruta === "Mangos"){
    alert (“Los mangos estan a 1.99 el kilo”);}
else if (fruta === "Papayas"){
    alert("Mangos y papayas están a 5,9 el kilo.");}
else{
    alert("Lo siento, no queda nada de " + fruta + ".");
}
```

Estructuras de control

Sentencia switch

- Si tenemos varias opciones a las que aplicar las mismas instrucciones...

```
switch (fruta) {
case "naranja" :
case "limon" :
case "pomelo" :
    alert ('CITRICOS');
     break;
case "melocoton" :
    alert ('FRUTA DE VERANO');
     break;
default :
    alert ('BUSCA INFORMACION SOBRE LA FRUTA');
     break;
}
```

Estructuras de control

Sentencia switch

- Si tenemos varias opciones a las que aplicar las mismas instrucciones y queremos usar operadores lógicos (&& ||), tenemos que adaptar el código….

```
switch (true) {
case (fruta === "naranja" || fruta ==="limon") :
    alert ('CITRICOS');
    break;
case (fruta === "melocoton") :
    alert ('FRUTA DE VERANO');
    break;
case (fruta === "manzana") :
    alert ('UNA AL DIA');
    break;
default :
    alert ('BUSCA INFORMACION SOBRE LA FRUTA');
    break;}
```

Enlace de referencia

Funciones document.write() y print()

- Con document.write se escribe código HTML en la página actual

```
document.write("<H1>HOLA, estoy escribiendo en el documento web </H1>");
```

HOLA, estoy escribiendo en el documento web”

- Con print() se imprime la página actual

```
document.print();
```

Ver HolaMundoDocumentwrite.htmla

Estructuras de control

Bucle while

- Crea un bucle que ejecuta una sentencia especificada mientras cierta condición se evalúe como verdadera
- Dicha condición es evaluada antes de ejecutar la sentencia

```
var n = 0;
var x = 0;
while (n < 3) {
   n ++;
   x += n;
}
```

Estructuras de control

Bucle do ... while

- Crea un bucle que ejecuta una sentencia especificada, hasta que la condición de comprobación se evalúa como falsa
- La condición se evalúa después de ejecutar la sentencia, dando como resultado que la sentencia especificada se ejecute al menos una vez

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

Estructuras de control

Bucle for

- Crea un bucle que consiste en tres expresiones opcionales, encerradas en paréntesis y separadas por puntos y comas, seguidas de una sentencia ejecutada en un bucle.

```
for (let i = 0; i < 9; i++) {
    n += i;
    miFuncion(n);
}
```

Estructuras de control

Bucle forEach

- Itera sobre los **elementos del array**

```
var miarray = [7,'hi','adios'];
miarray.forEach(p =>
document.writeln("Elemento del array: " + p +
"<br>" ));
```

Resultado:

Elemento del array: 7
Elemento del array: hi
Elemento del array: adios

Estructuras de control

Bucle for ... in

- La sentencia for..in itera sobre todas las **propiedades de un objeto**, en un orden arbitrario
- Para cada una de las propiedades, se ejecuta la sentencia especificada

```
var obj = { a:7,  b:'hi', c:'adiós'};
for (var p in obj) {
   document.write("Propiedad " + p + " = " +
obj[p] + "<br>"); }
```

Resultado:

Propiedad a = 7
Propiedad b = hi
Propiedad c = adiós

Buenas prácticas de programación en JavaScript

Consideraciones generales

- El código debe ser mantenible
- Debemos pensar que, seguramente, no seremos el único desarrollador que trabaje con él, por tanto, trataremos de que nuestro código sea:
  - Intuitivo
  - Comprensible
  - Adaptable
  - Extensible
  - Depurable
  - Testeable
- En JavaScript Garden ofrecen información sobre los detalles escabrosos de JavaScript

Buenas prácticas de programación en JavaScript

Consideraciones generales

- No mezclar código JavaScript con otros lenguajes de programación
  - No usar HTML en JS

```
document.getElementyId("contenedor").innerHTML =
"<div class=\"datos\"></div>";
```

  - No usar JS en CSS

```
.borde {
    width:expression(document.body.clientWidth
        < 955 ? "955px": "100%" );
}
```

  - No usar CSS en JS

```
document.getElementyId("main").style.color = "red";
```

Buenas prácticas de programación en JavaScript

Sangrado y espacios

- Cuánto menos código tengamos en una línea, menor probabilidad tendremos de tener un conflicto al mezclarlo (merge) con los cambios de otros desarrolladores
- Adaptarse al modo de sangrado (indentation) acordado en el proyecto en el que se participe. En caso de que no haya ninguno definido, está bastante extendido el uso de cuatro espacios para el sangrado
- Utilizar el espacio en blanco para mejorar la legibilidad del código
- Toda palabra reservada seguida de paréntesis “(“ se debe separar por un espacio. Ej. while (

Buenas prácticas de programación en JavaScript

Sangrado y espacios

- No se debe usar espacio en blanco en la llamada a una función entre el nombre de la función y el (Esto ayudará a distinguir las palabras clave
- Todos los operadores binarios se deben separar de sus operandos por un espacio en blanco
- Escribir siempre un espacio después de cada coma
- Evitar líneas más de más 80 caracteres
- Si hay que romper una línea, hacerlo después de un operador, idealmente después de una coma

Buenas prácticas de programación en JavaScript

Funciones y variables

- Evitar la declaración de variables y funciones en ámbito global, incluirlas en un espacio de nombres.
- Declarar todas las variables al comienzo del bloque de código en el que se vayan a utilizar
- Cuidado con la sentencia **return** en funciones, escribir una sola, como última sentencia y terminar con ;
- No comparar variables con **null**. Si se utiliza **instanceof** para comprobar tipos de objetos y **typeof** para tipos del lenguaje atención a cómo funcionan estos operadores
- No se recomienda el uso de **eval**. solo en casos justificados y cuando se han validado correctamente los datos a evaluar

- Las cadenas de caracteres suelen cambiarse a lo largo del desarrollo
- No escribirlas literalmente en el código, usar constantes en su lugar
- En particular urls, cadenas mostradas al usuario, ajustes, etc.
- Esto es aplicable a otros tipos de datos como los numéricos, por ejemplo:

```
const GOODBYE_MESSAGE = "¡Hasta luego!";
const NUMBER_OF_CARS = 10;
```

Buenas prácticas de programación en JavaScript

Otras

- Usar === en lugar de == y !== en lugar de !=
- Para facilitar la legibilidad del código no se recomienda utilizar los operadores ++ -- ?:
- Cuidado con **switch/case**, si se utilizan, se debe utilizar break
- No se recomienda el uso de las siguientes palabras reservadas: **break**, **continue**, **void**, **with**
- Cada punto y coma en la parte de control de una sentencia for se debe seguir de un espacio en blanco

Linting

- Linting es el proceso de ejecutar un programa para analizar código en busca de errores potenciales
- Un Lint o Linter es un programa que soporta linting (verificación de calidad del código). Están disponibles para la mayoría de los lenguajes de programación y marcas: JavaScript, CSS, HTML, Python, etc..
- Algunos de los linters más útiles son JSLint, CSSLint, JSHint, Pylint
- Muchos editores como lo tienen integrado de forma nativa como sublime, VSCode, brackets, etc.

WEBGRAFÍA Y ENLACES DE INTERÉS

- https://www.w3schools.com/js/
- https://developer.mozilla.org/es/docs/Web/JavaScript
- Curso Desarrollo de aplicaciones con HTML,node.js y Javascript. UPM. Miriadax


ESCUELA PÚBLICA:
DE TOD@S
PARA TOD@S
Vallecas