UNIDAD 3 - ESTRUCTURAS DE CONTROL DE FLUJO

3.1 - INTRODUCCIÓN

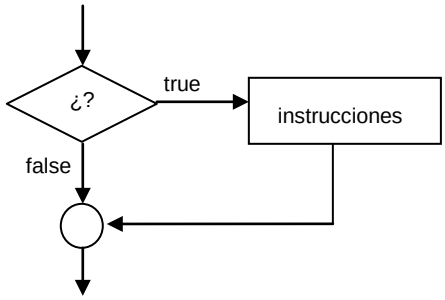
Aunque hasta este momento no se ha indicado nada en relación con el flujo de ejecución de un programa, parece natural que se entienda que las instrucciones se ejecutan en el orden en el que aparecen escritas al igual que un texto las líneas se leen de arriba hacia abajo. Sin embargo este flujo de ejecución hace que todas las instrucciones se tengan que ejecutar y siempre se hagan en el mismo orden. Esta característica de ejecución ordenada y lineal de un programa recibe el nombre de estructura de ejecución secuencial. Sin embargo, los lenguajes de programación incluyen otras estructuras de ejecución adicionales como son la estructura alternativa (con diferentes variantes: simple, doble o múltiple) y la estructura de ejecución cíclica o repetitiva.

En definitiva, el objetivo es disponer de mecanismos para que el flujo de ejecución de un programa pueda alterar el esquema secuencial natural, incorporando fragmentos de código que se ejecutan si se cumplen determinadas condiciones o bien haciendo que un conjunto de instrucciones se ejecuten un número determinado de veces antes de continuar con la ejecución del resto. Estas estructuras de control de flujo han dado lugar al paradigma de programación estructurado. Básico en el ámbito de gran parte de lenguajes de programación de carácter procedimental.

3.2 - ESTRUCTURA ALTERNATIVA SIMPLE: IF

La estructura alternativa simple permite establecer que un conjunto de instrucciones se ejecuten si se cumple una condición lógica. La sintaxis es la siguiente:

```
if(condicion) {
instrucciones;
}
```



Si la condición se cumple (es decir, si su valor es `true`) se ejecutan las instrucciones que se encuentran dentro del bloque delimitado por llaves `{...}` y a continuación se sigue con el flujo normal, ejecutando las instrucciones posteriores a la llave de cierre. Si la condición es false entonces el flujo de ejecución "salta" las instrucciones del bloque encerrado entre llaves y continua con las que se encuentran tras la llave de cierre. En el ejemplo, la variable saludar es creada e inicializada con el valor true. Cuando el flujo de ejecución llega a la instrucción if, se evalúa su valor y al ser igual a true se ejecuta la función alert mostrando en pantalla una ventana con el mensaje. Si se modifica la primera línea asignado false a la variable saludar se observará que el mensaje "Hola Mundo" no aparece. En ambos casos, se mostrará al final la ventana de alerta con el mensaje "Fin".

```
var saludar = true;
if(saludar == true)
{
        alert("Hola Mundo");
}
alert ("Fin");
```

```
<HTML>
<HEAD><TITLE>Alternativa simple</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
// Ejemplo de estructura alternativa simple.
var Nota;
Nota = prompt("Introduce la nota del alumno: ", 0);
if (Nota >= 5)
{
        alert("¡APROBADO!");
}
alert("Fin del programa");
// Fin del programa.
</SCRIPT>
</BODY>
```

Ejemplo 13 – Alternativa simple

Los operadores lógicos `AND (&&), OR (||) y NOT (!)` permiten encadenar varias condiciones simples para construir condiciones complejas.

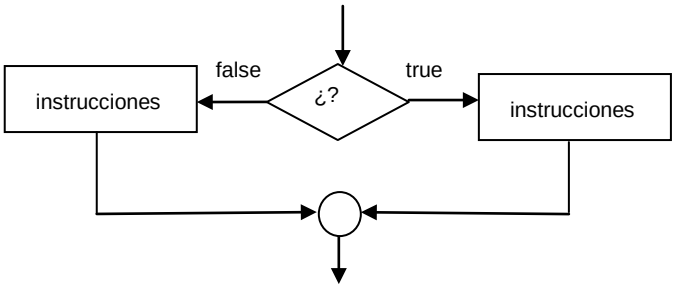
```
var llueve = true;
var calor = false;
If (!llueve && calor==false)
{
        alert (“Día de tenis”);
}
```

El fragmento anterior muestra dos expresiones lógicas unidas por el operador AND. El resultado será verdadero si ambas expresiones son verdaderas. La primera expresión está formada por la negación del valor de la variable llueve. Dado que la variable llueve vale true, la negación devolverá false. La segunda expresión compara el valor de calor con false como es cierta esa comparación, el resultado será true. Por tanto false AND true será false y la alerta no se mostrará en pantalla. Modifique el valor de la inicialización de llueve a a false y la alerta se mostrará.

3.3 - ESTRUCTURA ALTERNATIVA DOBLE: IF ELSE

Cuando en un programa se desea que se ejecuten un bloque de instrucciones si se cumple una condición y otro bloque diferente si la condición no se cumple, la estructura anterior se debe completar con una cláusula else que establece a través de un bloque encerrado entre llaves, las instrucciones correspondientes al caso de que la condición no se cumpla.

```
var llueve = true;
var calor = false;
If (!llueve && calor==false){
        alert (“Día de tenis”);
}
else
{
        Alert (“Día de lectura”);
}
```




```
<HTML>
<HEAD><TITLE>Alternativa doble</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
// Ejemplo de estructura alternativa doble.
var Nota;
Nota = prompt("Introduce la nota del alumno: ", 0);
if (Nota >= 5)
{
        alert("¡APROBADO!");
}
else
{
        alert("¡suspenso!");
}
// Fin del programa.
</SCRIPT>
</BODY>
</HTML>
```

Ejemplo 14 – Alternativa doble

3.4 - ANIDAR ESTRUCTURAS ALTERNATIVAS

Todas las estructuras de control se pueden anidar, es decir, dentro de una puede aparecer otra y dentro de esta otra a su vez construyendo estructuras alternativas más complejas. No obstante conviene ser cuidadoso con este mecanismo porque puede complicar el código en exceso y hacer que no todas las alternativas estén incluidas o que incluso aparezcan algunas no previstas por el programador.

```
<HTML>
<HEAD><TITLE>Anidamiento de alternativas</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
// Ejemplo de anidamiento de estructuras alternativas.
var Nota;
Nota = prompt("Introduce la nota del alumno: ", 0);
if (Nota >= 5 && Nota <= 10)
{
        alert("¡APROBADO!");
}
else
{
        if (Nota >= 0 && Nota < 5)
        {
                alert("¡suspenso!");
        }
        else
        {
                alert("Nota errónea");
        }
}
// Fin del programa.
</SCRIPT>
</BODY>
</HTML>
```

Ejemplo 15 –Anidamiento de Alternativas

3.5 - EL OPERADOR CONDICIONAL ?

En JavaScript está disponible el operador condicional ?. Se puede utilizar en algunas ocasiones en lugar de los la alternativa doble cuando las instrucciones de la parte true como las de la parte false son muy sencillas. Su formato genérico es:

(condición) ? ExpresionSiVerdadera : ExpresionSiFalsa

Donde ***Condición*** es cualquier expresión que devuelve un valor *true* o *false*. ***ExpresionSiVerdadera*** es una expresión que se ejecutará en el caso de que la condición tenga el valor ***true***. ***ExpresionSiFalsa*** igual que la anterior pero solo se ejecutará cuando la condición es falsa. Por ejemplo, la alternativa doble del ejemplo anterior puede sustituirse por la siguiente expresión:

```
(Nota >= 0 && Nota < 5) ? alert("¡suspenso!") : alert("Nota erronea");
```

3.6 - ALTERNATIVA MÚLTIPLE

Esta estructura está disponible a partir de la versión JavaScript1.3. Tiene el siguiente formato:

```
switch (expresión)
{ case valor1:
                instrucciones1;
                break;
case valor2:
                instrucciones1;
                break;
case valor3:
                instrucciones1;
                break;
...
case valorn:
                instruccionesn;
                break;
[default:
                instrucciones;]
};
```

Donde: ***expresión*** es cualquier expresión válida con la que se compararán los valores que acompañan a la cláusula case. ***valor1,..valorn*** son valores que suponemos puede tomar la expresión a los que les siguen la instrucción o instrucciones que queremos que

se ejecute en cada caso. ***default*** es una cláusula opcional, se ejecutarán las instrucciones que la siguen en el caso de que el valor no coincidiese con ninguno de los casos contemplados. El funcionamiento de esta estructura es el siguiente:

1. - calcula el valor de ***expresión***.

2.- comprueba desde el principio cada valor que acompaña a las cláusulas *case* hasta encontrar alguno que coincida con el valor de ***expresión***.

3.- cuando encuentra un valor que coincida con ***expresión*** ejecuta las instrucciones correspondientes hasta que encuentra la cláusula ***break***.

4.- si no encuentra ningún valor que coincida con ***expresion*** ejecuta las instrucciones correspondientes a la cáusula ***default*** (si existe, en caso contrario no hará nada).

Observaciones:

- La cláusula ***break*** que aparece en este formato no es obligatoria pero si no se utiliza cambiará el funcionamiento de la estructura ***switch***. Esta cláusula es la responsable la salida del bloque una vez que se ejecuten las instrucciones de una de las cláusulas ***case***. Si no se pone, se seguirán ejecutando todas las instrucciones hasta llegar al final. Es decir las cláusulas ***case*** sin ***break*** actuarían como puntos de entrada a la estructura, siendo la salida la misma en todos los casos.
- Para una misma cláusula ***case*** NO se pueden especificar diversos valores en lugar de un único valor. Si se desea que varios valores tengan la misma acción se deben poner claúsulas case por cada uno y dejar la acción en el último sacerlo:

```
case valor1:
case valor2:
case valor3: acciones;break;
```

```
<HTML>
<HEAD><TITLE>Anidamiento de estructuras</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
// Ejemplo de estructura alternativa múltiple
// (Anidada dentro de una alternativa doble).
var Nota;
var Calificacion = " ";
Nota = prompt("Introduce la nota del alumno: ", 0);
Nota = Math.round(Nota);
if (Nota >= 0 && Nota <= 10)
{
switch (Math.round(Nota)) {
case 5: Calificacion="aprobado"; break;
case 6: Calificacion="bien"; break;
case 7:
case 8: Calificacion="notable"; break;
case 9:
case 10: Calificacion="sobresaliente"; break;
default: Calificacion="suspenso"; break;
}; //end switch
}
else
{
Calificacion = "Nota erronea";
}
alert(Calificacion);
// Fin del programa.
</SCRIPT>
</BODY>
</HTML>
```

Ejemplo 16 – Anidamiento de estructuras alternativas

El ejemplo anterior muestra la utilización de una estructura alternativa múltiple anidada dentro de una estructura alternativa doble. En esta ocasión, en lugar de ejecutar una alerta para cada caso, hemos optado por guardar el texto a mostrar en una variable (Calificacion) para, al final, mostrar el contenido de dicha variable. También podemos observar que la variable Nota se redondea antes de comprobar su contenido (***Nota = Math.round(Nota);***) ya que los valores de las cláusulas case deben ser exactos (5, 6, etcétera).

3.7 - ESTRUCTURAS REPETITIVAS

A la hora de elaborar un programa es muy habitual que un conjunto de instrucciones tengan que ejecutarse de forma repetida. Para ello la mayor parte de los lenguajes de programación incorporan una familia de estructuras repetitivas que permiten establecer que un bloque de instrucciones se ejecute varias veces seguidas. El lenguaje JavaScript proporciona las siguientes: ***for, while, do while***

3.8 - ESTRUCTURA FOR

La existencia de esta estructura es muy habitual aunque presenta ciertos matices entre unos lenguajes y otros. Originalmente se diseñó en otros lenguajes de programación anteriores con objeto de poder ejecutar de forma repetida un bloque de instrucciones, pero sabiendo exactamente y con carácter previo el número de repeticiones. Por ejemplo, para calcular la suma de los diez primeros números naturales se podía utilizar una estructura for que repitiese diez veces una operación de suma en la que cada número natural se sumase al valor calculado previamente. Sin embargo, eran muchas las estrategias que permitían alterar el funcionamiento ya que a pesar de estar concebido para repetirse un número de veces, se podía alterar el valor del contador que detectaba la iteración en curso y por tanto modificar el valor previamente estimado.

Actualmente, la mayor parte de los lenguajes, incluyendo JavaScript proporcionan un modelo amplio que aporta mucha flexibilidad y potencia a esta estructura de control. La sintaxis general es la siguiente:

```
For (inicializacion; condicion; actualizacion)
{
        Instrucciones_que_se_repetirán
}
```

El funcionamiento de una estructura ***for*** es sencillo. En primer lugar y exclusivamente la primera vez que se entra en el bucle, se evalúa la expresión que ocupa la zona

denominada inicialización. Generalmente en esta zona se pone una expresión que utilice el operador de asignación para inicializar una variable denominada de control, el valor de inicio. A continuación se evalúa la expresión que ocupa la segunda parte y que se denomina condición ya que lo normal es que se trate de una condición cuyo resultado devuelva un valor true o false. Si la condición es true, se ejecutan las instrucciones que se encuentran dentro del bloque encerrado entre llaves y finalmente se evalúa la expresión que aparece en la sintaxis bajo la denominación de actualización. Tras esta actualización se vuelve a evaluar la condición y se vuelven a ejecutar las instrucciones del bloque, y así sucesivamente hasta que la evaluación de la condición devuelva el valor false, en cuyo caso el programa continúa en la siguiente instrucción a la llave de cierre del bucle.

Por ejemplo, el siguiente fragmento de programa visualiza en pantalla los cinco primeros números naturales y para ello utiliza un bucle cuya variable de control se ha denominado número:

```
for (numero=1; numero<6;numero++)
{
        alert (numero);
}
```

El proceso que sigue el intérprete consiste, en primer lugar y por una única vez, en asignar a la variable ***numero*** el valor 1. A continuación evalúa la condición ***numero < 6*** y dado que la misma actualmente es true ejecuta el bloque de instrucciones (***alert***). El siguiente paso consiste en ejecutar la expresión de actualización que en este caso utiliza el operador de post-incremento para que la variable numero pase al siguiente valor (en este caso 2) y de nuevo vuelve a evaluar al condición numero<6 que al seguir siendo true permite de nuevo la ejecución del ***alert***. Este proceso se repite hasta que la actualización hace que la variable ***numero*** pase a valer 6 de manera que en ese momento la condición ***numero < 6*** deja de ser ***true*** para ser ***false*** y por tanto el bucle se termina.

Hay que prestar mucha atención en las expresiones que se utilizan en los bucles con especial cuidado a la expresión denominada condición ya que el programador debe tener garantía de que ese condición será false en algún momento porque de no ser así el bucle

se ejecutaría indefinidamente, motivo por el cual muchos programas se quedan bloqueados, incapaces de avanzar y generando inestabilidad en el sistema.

Las características del bucle ***for*** hacen que se utilice mucho a la hora de manipular arrays ya que los elementos de una array se identifican por el índice que ocupan y por tanto recorrer un array con un bucle es muy sencillo. Por ejemplo:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado","Domingo"];
for(var i=0; i<7; i++)
{
        alert(dias[i]);
}
```

Es importante recordar que los arrays en Javascript se organizan de forma que el primer elemento ocupa la posición 0, motivo por el cual la variable de control del bucle del ejemplo anterior se inicializa a cero y la condición es i<7 ya que el último elemento del array ocupa la posición 6. Esto, lejos de ser un inconveniente, es una ventaja, ya que la condición en estos casos es variable_de_control < número_de_elementos_del_array.

3.9 - ESTRUCTURA FOR...IN

Una variante de la estructura de control repetiva for es la denominada for..in cuya aplicación está muy relacionada con la programación orientada a objetos. En este punto, dado que aún no se ha señalado nada en relación con ese paradigma de programación, se explica exclusivamente su uso con arrays y a través del mismo ejemplo utilizado antes para mostrar la sencillez de su uso.

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
for(i in dias)
{
        alert(dias[i]);
}
```

La variable que se indica como indice es la que se puede utilizar dentro del bucle ***for...in*** para acceder a los elementos del array. Como se puede observar el valor inicial, la condición y la actualización no hay que especificarlos, simplificando mucho la manipulación de estas estructuras de datos. Hay que pensar que todo aquello que el intérprete conoce durante la ejecución debería aprovecharse para que no quede en manos del programador controles y decisiones que puedan provocar errores. El intérprete conoce perfectamente la longitud de un array en cada momento, por lo tanto es mejor que sea el propio intérprete el que determine cuántos elementos hay y por tanto cuántas veces es necesario repetir el bucle. Evidentemente este bucle recorre todos los elementos sin excepción.

```
<HTML>
<HEAD><TITLE>Bucle  for con operador in</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
var Alumnos = new Array ("MIGUEL", "ELENA", "JORGE",
"CRISTINA","IGNACIO");
var Apellidos = new Array ("GIL", "SANCHEZ", "RODRIGUEZ",
"ALONSO","VEGA");
for(i in Alumnos)
{
document.write(Alumnos[i]+" ** ");
document.write(Apellidos[i]+"<br>");
}
</SCRIPT></BODY></HTML>
```

Ejemplo 17 – Bucle con operador in

Tal y como se ha señalado en el capítulo 2, los arrays son estructuras de datos del mismo tipo, identificadas por un único nombre y que recurren al indexado para distinguir unos elementos de otros. Su naturaleza hace que la gestión de los mismos esté fuertemente vinculada a las estructuras de control repetitivo. A continuación veremos una representación gráfica del fragmento de memoria que contiene un Array que almacena los nombres de 5 alumnos:

Alumnos

MIGUEL	ELENA	JORGE	CRISTINA	IGNACIO
Alumnos[0]	Alumnos[1]	Alumnos[2]	Alumnos[3]	Alumnos[4]

Observamos que hay cinco elementos de información (el Array tiene cinco elementos). Todos ellos comparten un mismo identificador (*Alumnos*) pero cada uno tiene, además, un índice que hace referencia al elemento en particular. Podemos apreciar también que el primer elemento es el 0, el segundo el 1, y así sucesivamente. Así, cuando hagamos referencia a Alumnos[3] estaremos refiriéndonos al elemento que ocupa la posición 4 (recuerde que la numeración comienza en cero) cuyo contenido en este caso es "CRISTINA".

3.10 - ESTRUCTURAS WHILE Y DO WHILE

Las estructuras *while* y *do…while* nos permiten obtener un control de flujo. Son similares a un ciclo *for*, con la diferencia de que no se cumplen un número determinado de veces, sino que se ejecutan mientras que la condición expresada sea verdad.

While

while (condición){
//sentencias a ejecutar
}

```
<html>
<head>
<title>Ejemplo de ciclo while</title>
<script type="text/javascript">
var dejarBlucle = "";
while (dejarBucle != "salir"){
dejarBucle = prompt("Escribe 'salir' para dejar el bucle, o cualquier otra palabra para continuar","")
}
</script>
</head>
<body> </body>
</html>
```

Ejemplo 18 – Estructura WHILE

Este es un ejemplo del bucle while. Lo que hace es pedir que el usuario introduzca palabras repetidas veces, mientras que la palabra introducida no sea “salir” el bucle se repetirá preguntando de nuevo.

do ... While

Si hay que ejecutar el bloque de instrucciones, por lo menos una vez, utilizaremos do…while que seguirá ejecutando la expresión más veces siempre que la condición sea verdad.

do {
//sentencias del bucle
} **while** (condición)

```
<html>
<head>
<title>Ejemplo de ciclo do…while</title>
<script type="text/javascript">
var dejarBlucle = "";
do {
 dejarBucle= prompt("Escribe ‘salir’ para dejar el bucle, o cualquier otra palabra para continuar","")
}
while (dejarBucle != "salir")
</script>
</head>
<body> </body>
</html>
```

Ejemplo 19 – Estructura DO WHILE