# JS07: bucles

## Estructuras de control:
## las iteraciones o bucles

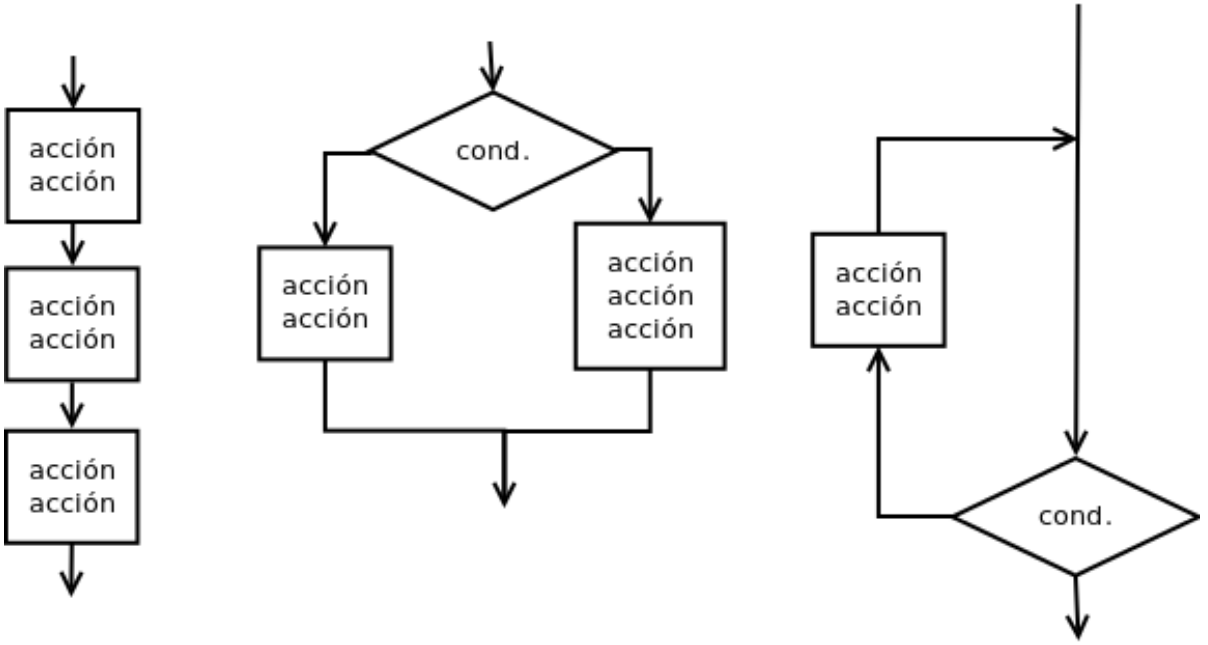
# Índice


- Bucles.
- Tipos de bucles.
  - *while.*
  - *do while.*
  - *for.*
- Bucles infinitos.
- Bucles anidados: bucles dentro de bucles.
  - *break* y *continue*.

- Ejercicios.

# Flujo del programa

- En presentaciones anteriores, se comentó que existen tres tipos de estructuras para el control de flujo en nuestros programas: secuencial, selección (o bifurcación) e iteración (o bucle).



# Bucles


- Un **bucle** es la repetición de una serie de instrucciones mientras se cumpla una determinada condición.

  - Es utilizado para implementar operaciones que se repiten sin tener que escribir varias veces el mismo código.
  - Ahorra tiempo al programador, deja el código más claro y facilita futuras modificaciones.
  - Hay muchos algoritmos que requieren del uso de bucles para poder obtener el resultado deseado.

# Ejemplo imprimir 1000 veces

- Imaginemos que queremos colocar 1000 *divs* con los números del 1 al 1000 sobre el documento.

- Podemos hacer varias cosas:

  - Escribir 1000 veces * numero .*
  - Usar el siguiente bucle de tipo *for*:

```
<script>
    for(let i=1; i<=1000; i++){
        document.write('<div>'+i+'</div>');
    }
</script>
```

# Bucles

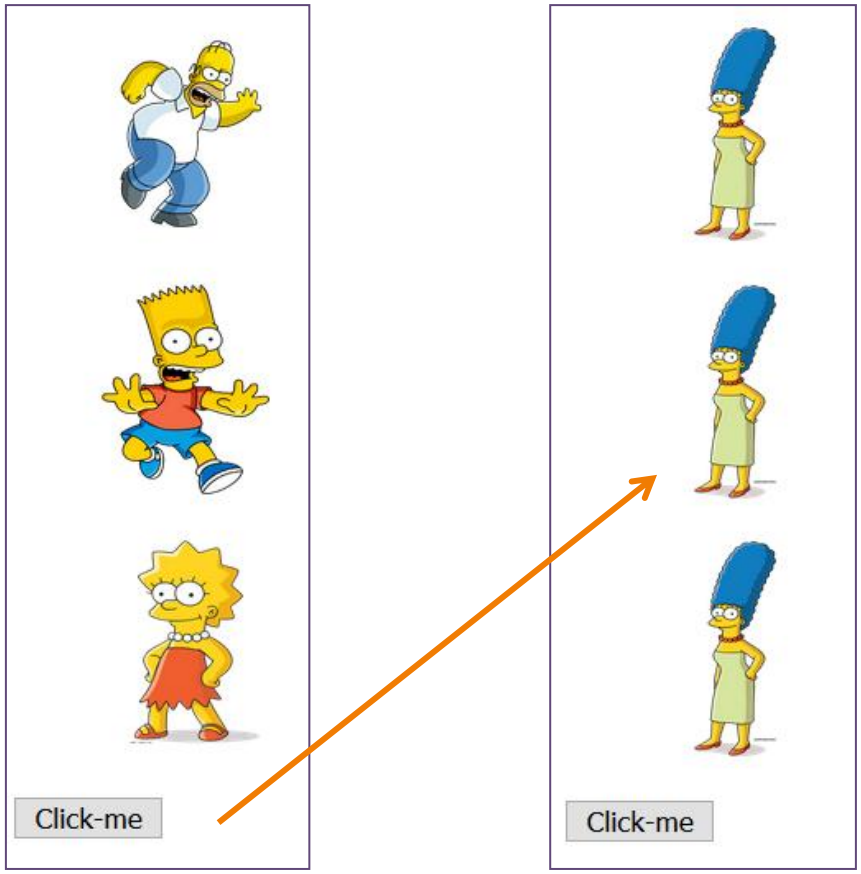

- Los bucles son una de las estructuras básicas en programación.
  - Existen **infinidad de algoritmos que incorporan bucles para poder cumplir sus objetivos**: desde cálculos matemáticos a operaciones con bases de datos o ficheros.
  - En *JavaScript*, con el *DOM*, los usaremos mucho para recorrer listados de elementos *HTML* y realizar con ellos algún tipo de operación.
  - Cuando estemos en el módulo de programación en el lado del servidor, veremos que los utilizaremos para procesar y presentar la información recuperada de la base de datos.

# Ejemplo manipulación de elementos *HTML*

```
<figure>
    <img src="imagenes/homer.png" alt="Homer">
</figure>
<figure>
    <img src="imagenes/bart.png" alt="Bart">
</figure>
<figure>
    <img src="imagenes/lisa.png" alt="Lisa">
</figure>
<button onclick="cambiar()">Click-me</button>

<script>
    function cambiar(){
        // para cada imagen del documento...
        for(let imagen of document.images){
            imagen.src='imagenes/marge.png'; // cambia la foto
            imagen.alt='Marge'; // cambia el texto alternativo
        }
    }
</script>
```

Bucle de tipo *for of*

# Ejemplo manipulación de elementos *HTML*




Probad a ejecutar paso a paso este programa en el depurador del navegador (F12 → pestaña depurador) y observad los cambios que se van produciendo.

Probadlo también con más imágenes. ¿Cambian todas?

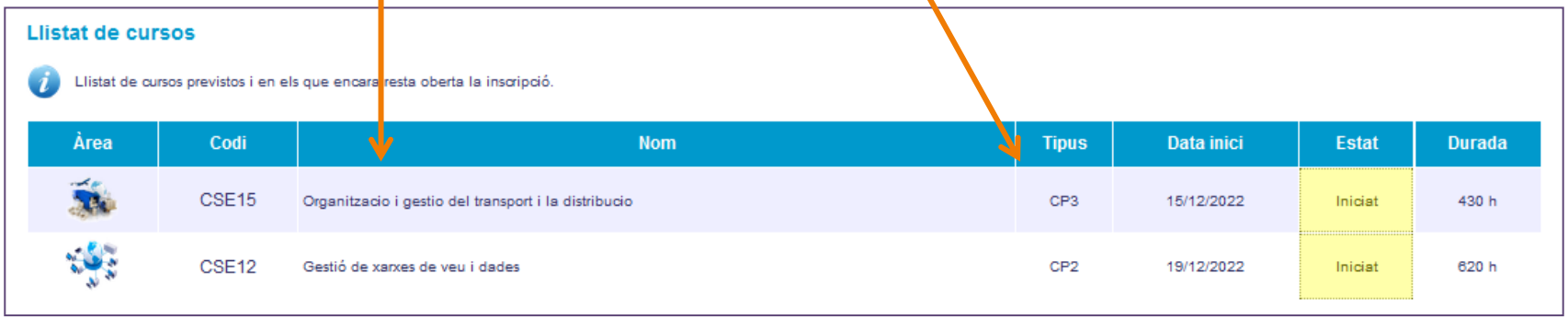
# Bucles


- El ejemplo siguiente no lo podemos probar todavía, puesto que está implementado en *PHP.*

  - Solamente es para explicar la idea de cómo podemos mostrar los datos recuperados desde una base de datos en una tabla *HTML*.
  - Usaremos un bucle en *PHP* para construir la tabla de resultados fila a fila.
  - En el ejemplo se usa un bucle *foreach* de *PHP* que no existe como tal en *JavaScript* pero que tiene como equivalente el *for of*.

# Ejemplo presentación de datos (*PHP*)

```php
<?php
    // coloca la lista de cursos
    foreach($cursos as $curso){ ?>
        <tr class="touchable" onclick="location.href='Curso/ver/<?= $curso->id ?>'">
        <td class="centrado">
            <img class="preview" src="<?= $curso->imatge ?>" alt="<?= $curso->id_area?>" >
        </td>

        <td class="centrado grande mayusculas"><?= $curso->codi ?></td>
        <td><?= $curso->nom ?></td>
        <td class="centrado mayusculas"><?= $curso->tipus ?></td>

        ...
```



**Llistat de cursos**

Llistat de cursos previstos i en els que encara resta oberta la inscripció.

| Àrea | Codi | Nom | Tipus | Data inici | Estat | Durada |
|---|---|---|---|---|---|---|
| | CSE15 | Organitzacio i gestio del transport i la distribucio | CP3 | 15/12/2022 | Iniciat | 430 h |
| | CSE12 | Gestió de xarxes de veu i dades | CP2 | 19/12/2022 | Iniciat | 620 h |

# Tipos de bucles

Los tipos de bucles básicos que podemos usar en *JavaScript*

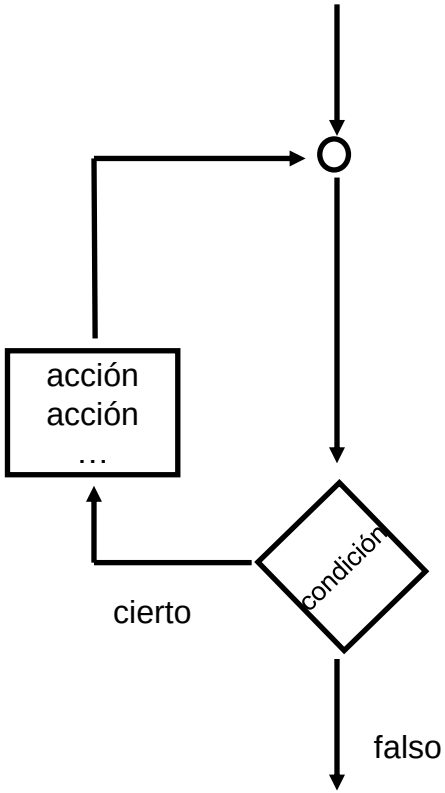

Robert Sallent
www.robertsallent.com

SOC Servei d'Ocupació de Catalunya
CIFO Sabadell- Mancomunitat (Vallès)
Generalitat de Catalunya
Unió Europea
Fons social europeu
L'FSE inverteix en el teu futur

# Tipos de bucles


- Existen tres tipos básicos de bucle, aunque es posible intercambiar uno por otro añadiendo o quitando algunas instrucciones:
  - El bucle ***while.***
  - El bucle ***do… while.***
  - El bucle ***for.***
- En muchos lenguajes existen también variantes del *for*, como el *foreach*, *for…in*, *for…as*, etc.
  - En *JS* tenemos ***for…in*** y ***for…of***, que veremos más adelante.

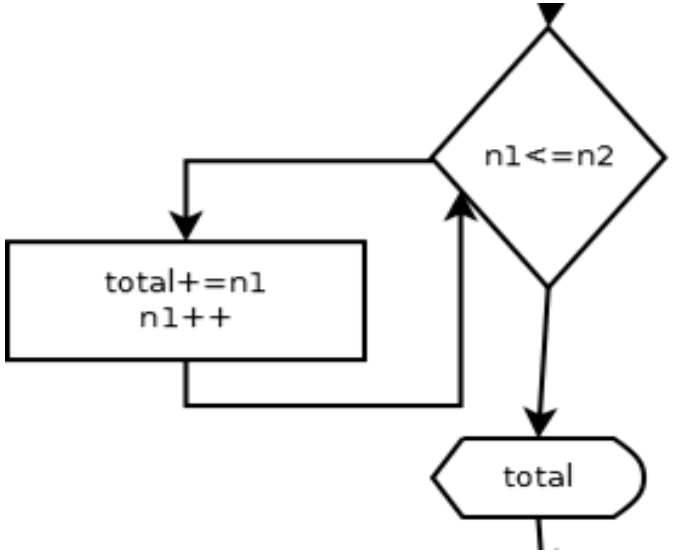
# El bucle *while*




```
while(condición){

      acción;
      acción;
        ...
}
```

# El bucle *while*

- Al dibujar diagramas de flujo, la notación mostrada en la diapositiva anterior puede resultar confusa, especialmente con bucles anidados.
  - Podemos variar la notación, siempre y cuando se siga comprendiendo el diagrama.



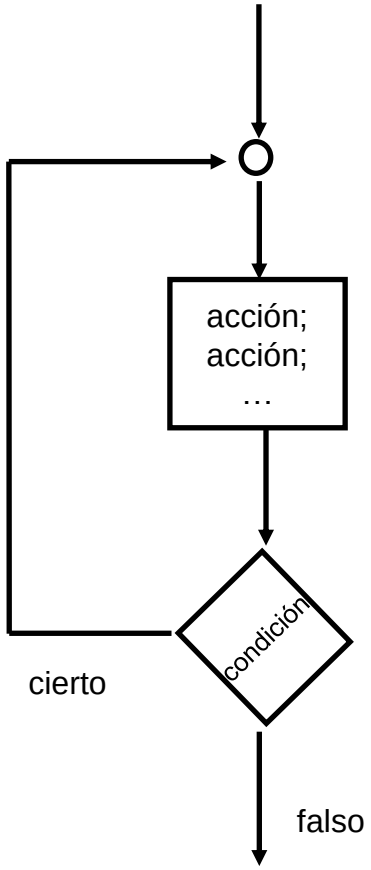
# Ejemplo el bucle *while*

```
<h1>Cuenta de 1 a 10</h1>
<script>
     var i = 1;
     while(i<=10){
          document.write(i+', ');
          i++;
     }
</script>

<h1>Cuenta de 10 a 1</h1>
<script>
     var i = 10;
     while(i>0){
          document.write(i+', ');
          i--;
     }
</script>
```

Observa cómo se **inicializa** la variable inductora del bucle (`i`).

Se **modifica** `i` en cada iteración, de forma que en algún momento dejará de cumplirse la **condición del bucle**, evitando así los bucles infinitos.

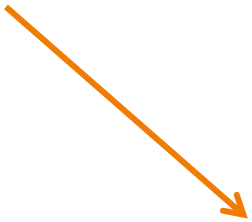
# El bucle *do…while*




```
do {

      acción;
      acción;
        ...

}while(condición);
```

# Ejemplo el bucle *do…while*

```
<h1>Bucle do while</h1>
<script>
    var i=1;

    do{
        document.write('Esta es la iteración '+i+'<br>');
        i++;

    }while(i<=10);
</script>
```

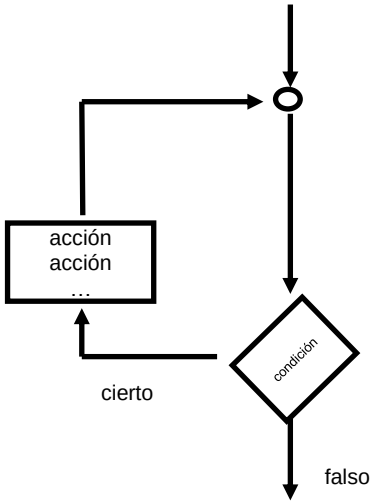


**Bucle do while**

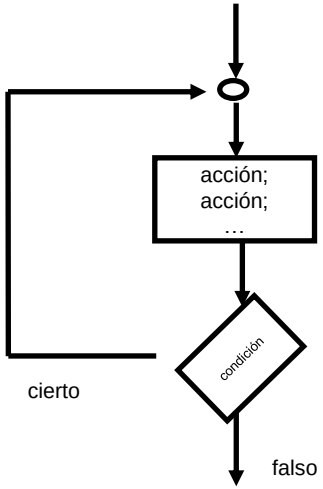
Esta es la iteración 1
Esta es la iteración 2
Esta es la iteración 3
Esta es la iteración 4
Esta es la iteración 5
Esta es la iteración 6
Esta es la iteración 7
Esta es la iteración 8
Esta es la iteración 9
Esta es la iteración 10

# Diferencia *while* / *do…while*

- La diferencia entre `while` y `do while` es que, en el segundo, siempre se ejecutan las acciones al menos una vez, puesto que la comprobación se realiza al final.




while

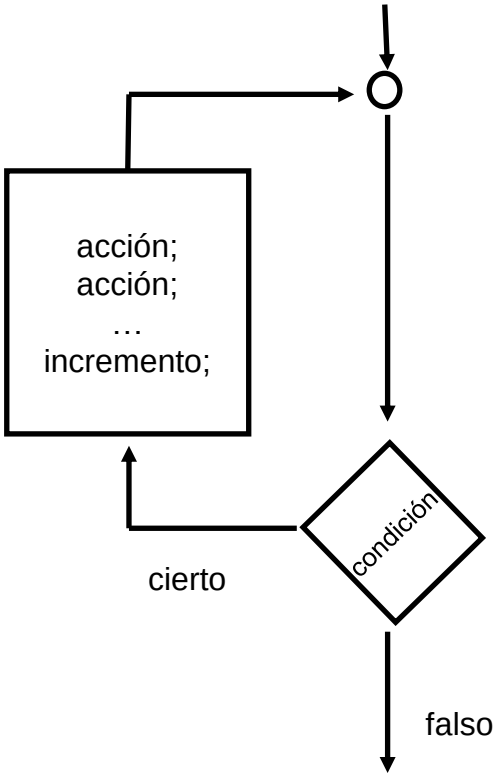



do…while

# El bucle *for*


- El bucle de tipo ***for*** es similar al de tipo ***while***, pero permite indicar la inicialización, condición e incremento en una sola línea.
  - Es un tipo de bucle muy usado cuando se conoce a priori el número de iteraciones.
  - Es muy útil para recorrer listas.
  - Dispone de algunas variantes, como el `for…in` o el `for…of`.

# El bucle *for*




```
for(init; cond; inc){

        acción;
        acción;
        ...

}
```

# El bucle *for*

- Podemos interpretar las tres partes que componen el `for` de esta forma:

una vez al inicio

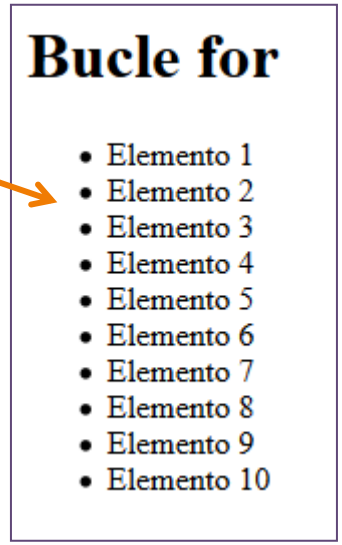
antes de cada iteración

al final de cada iteración

```
for(inicialización; condición; incremento){
        acción;
        acción;
        ...
}
```

# Ejemplo el bucle *for*

```
<h1>Bucle for</h1>
<script>
    document.write('<ul>');

    for(let i=1; i<=10; i++){
        document.write('<li>Elemento '+i+'</li>');
    }

    document.write('</ul>');
</script>
```



# El bucle *for*


- Por lo general, los bucles de tipo `for` suelen ser sencillos, del estilo del que se ha mostrado en el ejemplo anterior.

- Sin embargo, la sintaxis del `for` permite mucho juego (si se quiere):

  - Se pueden inicializar varias variables simultáneamente.
  - Se pueden realizar múltiples incrementos en distintas variables.
  - Se pueden poner condiciones complejas mediante conectivas lógicas e incluso llamadas a función.
  - Se pueden omitir partes del `for`.

# La sintaxis del *for*

- Podemos contar hacia atrás:

```
for(i=10; i>=0; i--){
   document.write(i+', ');
}
```

- También se permite inicializar o modificar varias variables (usando una coma):

```
for(i=10, j=0; i!=j; i--, j++){
   document.write('('+i+','+j+')');
}
```

# La sintaxis del *for*

- Se pueden omitir algunas de las partes (pero siempre tiene que haber tres, así que deben estar los dos punto y coma).

```
var n1 = 5;

for( ;n1<10; n1++){
    alert('faltan '+(10-n1)+' iteraciones');
}
```

O bien:

```
for( ;n1<10;){
    alert('faltan '+(10-n1++)+' iteraciones');
}
```

# La sintaxis del *for*

- Y también se pueden expresar condiciones complejas usando los operadores lógicos:

```
var n1 = 5;

for( ;!confirm('salir') && n1<10;){
    alert('faltan '+(10-n1++)+' iteraciones');
}
```

# For of


- Como he comentado antes, en *JavaScript* existe una variante del bucle *for* que es útil para recorrer listas o conjuntos.

- Lo veremos en más profundidad cuando hablemos de *arrays*, de momento os dejo un ejemplo:

```
<script>
    var lista = ['patatas', 'queso', 'bacon'];

    for(let producto of lista){
        document.write('<p>Recuerda comprar '+producto+'</p>');
    }
</script>
```

# Ámbito o alcance

- La variable usada para llevar el control de las iteraciones (variable inductora) muchas veces suele tener ámbito de bloque.

- Sin embargo, podemos extender el ámbito o usar variables que ya existían en ámbitos de orden superior.

```
<h1>Ámbitos</h1>

<h2>Ámbito global</h2>
<script>
   for(i=1; i<=10; i++)
        document.write(i+', ');
</script>

<h2>Ámbito local (o global)</h2>
<script>
   for(var i=1; i<=10; i++)
        document.write(i+', ');
</script>

<h2>Ámbito de bloque</h2>
<script>
   for(let i=1; i<=10; i++)
        document.write(i+', ');
</script>
```

# Algunas consideraciones

Estrategias para resolver algunos problemas y ejercicios.


Robert Sallent
www.robertsallent.com

SOC Servei d'Ocupació de Catalunya
CIFO Sabadell- Mancomunitat (Vallès)
Generalitat de Catalunya
Unió Europea
Fons social europeu
L'FSE inverteix en el teu futur

# Concatenando texto


- A veces es interesante concatenar texto en una variable para luego presentarlo de golpe en un *output*.
  - Para hacer esto, preparamos una variable de texto inicialmente vacía y luego le vamos añadiendo texto con +=.
  - Al final tendremos la variable con la concatenación de todo el texto resultante, que mostraremos en el output.
  - También podemos concatenar texto directamente en el output, pero esto es más lento.
- A continuación tenemos dos ejemplos que debes probar y ejecutar paso a paso desde el depurador para ver cómo funcionan.

# Ejemplo concatenando texto en el *output*

```
<script>
    function contar(){
        var numero = parseInt(inNumero.value);
        outResultado.innerHTML = ''; // borra el output

        for(let i=1; i<=numero; i++)
            // va concatenando directamente en el output (lento)
            outResultado.innerHTML += i+', ';
    }
</script>

<label>Número</label>
<input type="number" value="5" id="inNumero">
<button onclick="contar()">Contar</button>
<br>
<output id="outResultado"></output>
```

# Ejemplo concatenando texto en una variable


```
<script>
    function contar(){
        var numero = parseInt(inNumero.value);
        var resultado = ''; // variable para guardar el resultado

        for(let i=1; i<=numero; i++)
            resultado += i+', '; // va concatenando

        outResultado.innerHTML = resultado; //escribe el resultado de golpe
    }
</script>

<label>Número</label>
<input type="number" value="5" id="inNumero">
<button onclick="contar()">Contar</button>
<br>
<output id="outResultado"></output>
```

# Variable para el total


- Otros algoritmos implican ir haciendo cálculos aritméticos parciales en bucle hasta obtener el resultado final que mostraremos.

- En estas situaciones también se suele usar una variable adicional, esta vez de tipo numérico, para ir calculando el resultado.

  - Esta variable se suele inicializar a 0 o a 1, dependiendo de si la operación requiere sumas o multiplicaciones.

- Recordemos que podemos crear todas las variables que necesitemos.

# Ejemplo cálculos parciales


```
<script>
    function cubo(){
        var numero = parseInt(inNumero.value);
        var resultado = 1; // variable para guardar el resultado

        for(let i=0; i<3; i++)
            resultado *= numero; // va calculando

        outResultado.innerHTML = resultado; //escribe el resultado
    }
</script>

<label>Número</label>
<input type="number" value="5" id="inNumero">
<button onclick="cubo()">Cubo</button>
<br>
<output id="outResultado"></output>
```

# Bucles infinitos

Los errores de programación en bucles causan problemas

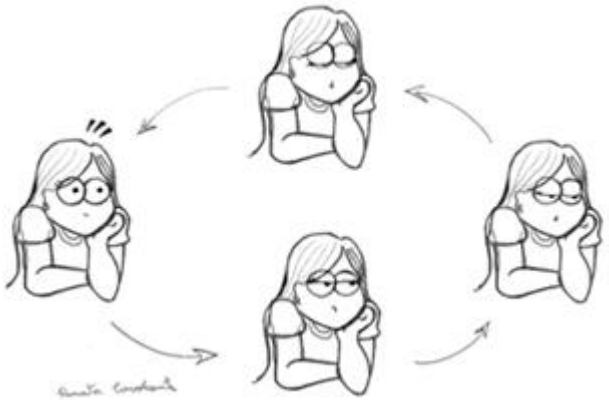

Robert Sallent
www.robertsallent.com

SOC Servei d'Ocupació de Catalunya
CIFO Sabadell- Mancomunitat (Vallès)
Generalitat de Catalunya
Unió Europea
Fons social europeu
L'FSE inverteix en el teu futur

# Bucle infinito

- En los bucles hay que tener cuidado en **modificar la variable inductora** y no provocar accidentalmente **bucles infinitos.**

- Un bucle infinito es el que no acaba nunca, la mayoría de veces son debidos a errores de programación.



# Bucle infinito

- Cuando se produce un bucle infinito, el navegador se queda "colgado" puesto que está realizando miles de operaciones inútiles.
  - Tras un tiempo, mostrará un mensaje sugiriendo al usuario detener el *script*. Si confirmamos, finalizará la ejecución.


  - En el peor de los casos, el navegador se quedará totalmente colgado y habrá que reiniciarlo o incluso reiniciar el equipo…

# Ejemplo error de programación

Parece que se pretendía un bucle de 0 a 99

```
var i = 0;

while(i<100){
  document.write('Hola mundo');
}
```

# Bucle infinito

- Podemos provocar bucles infinitos de forma intencionada :

```
while(true){
    //acciones
}
```

```
do{
    //acciones
}while(true);
```

```
for(;true;){
    //acciones
}
```

# Bucles anidados

## Bucles dentro de bucles…

# Bucles anidados

- Podemos tener un **bucle dentro de otro bucle**. El bucle interno se ejecutará tantas veces como iteraciones tenga el bucle externo.

- Las instrucciones dentro del bucle interno se ejecutarán tantas veces como iteraciones haga el interno multiplicadas por las iteraciones del externo.

```
for(let i=0; i<3; i++){
  for(let j=0; j<2; j++){
       document.write(i+' - '+j);
  }
}
```

3 iteraciones

2 iteraciones

3 x 2 = 6 ejecuciones

# Ejercicio bucles anidados


- Observa el siguiente ejemplo de bucles anidados:

```
for(let i=1; i<=5; i++){
    for(let j=1; j<=10; j++){
        document.write(i+' x '+j+' = '+i*j+'<br>');
    }
    document.write('<br>');
}
```

- ¿Qué hace este código? Intenta analizarlo a mano y escribe en papel lo que va sucediendo en cada paso.

# Bucles anidados


- Como habrás podido observar, el ejemplo anterior crea las tablas de multiplicar para los números del 1 al 5.
  - Él bucle exterior crea la tabla de multiplicar para cada número.
  - El bucle interior crea cada una de las filas.
- En el ejemplo siguiente se ha mejorado, permitiendo elegir qué tablas se quieren visualizar, combinando el *JavaScript* con el *HTML* y añadiendo un poquito de *CSS*.
  - Observa su funcionamiento y el resultado final.

# Ejemplo las tablas de multiplicar (1/3)

```
<head>
    <meta charset="utf-8">
    <title>Tablas multiplicar</title>
    <style>
        table{
            font-family: arial;
            margin: 10px;
            float: left;
        }
        table th{
            background-color: lightgrey;
        }
        table td, table th{
            padding: 5px 20px;
            border: solid 1px grey;
        }
        table td:last-of-type{
            font-weight: bold;
        }
    </style>
</head>
```

# Ejemplo las tablas de multiplicar (2/3)

```
<h1>Ejemplo tablas multiplicar</h1>

<input id="inInicial" type="number" value="1">
<input id="inFinal" type="number" value="10">
<button onclick="calcular()">Click-me</button>
<br>
<output id="outResultado"></output>
<br>
```

# Ejemplo las tablas de multiplicar (3/3)

```
<script>
    function calcular(){
        var inicial = parseInt(inInicial.value); // tabla inicial
        var final = parseInt(inFinal.value);     // tabla final

        var resultado = '';

        for(let i=inicial; i<=final; i++){
            resultado += '<table>';
            resultado += '<tr>';
            resultado += '<th colspan="2">Tabla del '+i+'</th>';
            resultado += '</tr>';

            for(let j=0; j<=10; j++){
                resultado += '<tr>';
                resultado += '<td>'+i+' x '+j+'</td><td>'+i*j+'</td>';
                resultado += '</tr>';
            }
            resultado += '</table>';
        }
        outResultado.innerHTML = resultado;
    }
</script>
```

# Ejemplo las tablas de multiplicar


1 5 Click-me

| Tabla del 1 | |
|---|---|
| 1 x 0 | **0** |
| 1 x 1 | **1** |
| 1 x 2 | **2** |
| 1 x 3 | **3** |
| 1 x 4 | **4** |
| 1 x 5 | **5** |
| 1 x 6 | **6** |
| 1 x 7 | **7** |
| 1 x 8 | **8** |
| 1 x 9 | **9** |
| 1 x 10 | **10** |

| Tabla del 2 | |
|---|---|
| 2 x 0 | **0** |
| 2 x 1 | **2** |
| 2 x 2 | **4** |
| 2 x 3 | **6** |
| 2 x 4 | **8** |
| 2 x 5 | **10** |
| 2 x 6 | **12** |
| 2 x 7 | **14** |
| 2 x 8 | **16** |
| 2 x 9 | **18** |
| 2 x 10 | **20** |

| Tabla del 3 | |
|---|---|
| 3 x 0 | **0** |
| 3 x 1 | **3** |
| 3 x 2 | **6** |
| 3 x 3 | **9** |
| 3 x 4 | **12** |
| 3 x 5 | **15** |
| 3 x 6 | **18** |
| 3 x 7 | **21** |
| 3 x 8 | **24** |
| 3 x 9 | **27** |
| 3 x 10 | **30** |

| Tabla del 4 | |
|---|---|
| 4 x 0 | **0** |
| 4 x 1 | **4** |
| 4 x 2 | **8** |
| 4 x 3 | **12** |
| 4 x 4 | **16** |
| 4 x 5 | **20** |
| 4 x 6 | **24** |
| 4 x 7 | **28** |
| 4 x 8 | **32** |
| 4 x 9 | **36** |
| 4 x 10 | **40** |

| Tabla del 5 | |
|---|---|
| 5 x 0 | **0** |
| 5 x 1 | **5** |
| 5 x 2 | **10** |
| 5 x 3 | **15** |
| 5 x 4 | **20** |
| 5 x 5 | **25** |
| 5 x 6 | **30** |
| 5 x 7 | **35** |
| 5 x 8 | **40** |
| 5 x 9 | **45** |
| 5 x 10 | **50** |

# *Break y continue*

## Finalizando el bucle y saltando iteraciones


*Robert Sallent*
*www.robertsallent.com*

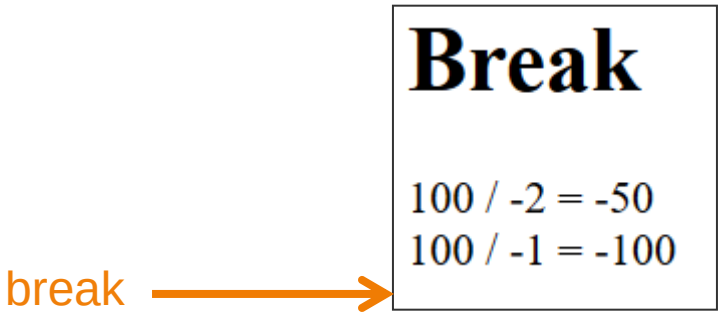
SOC Servei d'Ocupació de Catalunya
CIFO Sabadell- Mancomunitat (Vallès)
Generalitat de Catalunya
Unió Europea
Fons social europeu
L'FSE inverteix en el teu futur

# Break y continue


- ***Break*** detiene la ejecución del bucle o bloque en el que se ejecuta.
  - Podemos usarlo para forzar salidas de bucles en caso de que se cumplan unas condiciones concretas.
  - Si el *break* se encuentra en un bucle anidado, se detiene la ejecución del bucle interno, pero no la del externo.
- ***Continue***, detiene la actual iteración del bucle y prosigue con la siguiente.
  - A efectos prácticos, podemos decir que se salta el código que tiene por debajo en la iteración actual.

# Ejemplo saliendo del bucle con *break*

```
<h1>Break</h1>

<script>
  var num = 100;

  for(let den=-2; true; den++){

    //finalizará la ejecución del bucle si "den" es 0
    if(den == 0) break;

    let resultado= num/den;
    document.write(num+' / '+den+" = "+resultado+'<br>');
  }
</script>
```



# Ejemplo saltando una iteración con *continue*

```
<h1>Continue</h1>

<script>
  var num = 100;

  for(let den=-2; den<=2; den++){

    //se saltará los pasos siguientes cuando 'den' sea 0
    //pero seguirá por la siguiente iteración
    if(den == 0) continue;

    let resultado= num/den;
    document.write(num+' / '+den+" = "+resultado+'<br>');
  }
</script>
```

continue →

**Continue**

100 / -2 = -50
100 / -1 = -100
100 / 1 = 100
100 / 2 = 50

# Ejercicios

## Bucles


Robert Sallent
www.robertsallent.com

SOC Servei d'Ocupació de Catalunya
CIFO Sabadell- Mancomunitat (Vallès)
Generalitat de Catalunya
Unió Europea
Fons social europeu
L'FSE inverteix en el teu futur

# Ejercicios


1. Implementa estos bucles, primero con un `for`, luego con un `while` y finalmente con un `do-while`:
   a) Imprime 10 veces “Hola mundo”.
   b) Imprime los números de 1 a 10 en orden ascendente.
   c) Imprime los números pares del 2 al 100.
   d) Imprime los múltiplos de 3 del -27 al +27.
   e) Imprime los múltiplos de 5 del 100 al 5 (orden descendente).
   f) Genera e imprime las 10 primeras potencias de 2 mediante un bucle.
      Son: 2,4,8,16,32,64,128,256,512,1024.

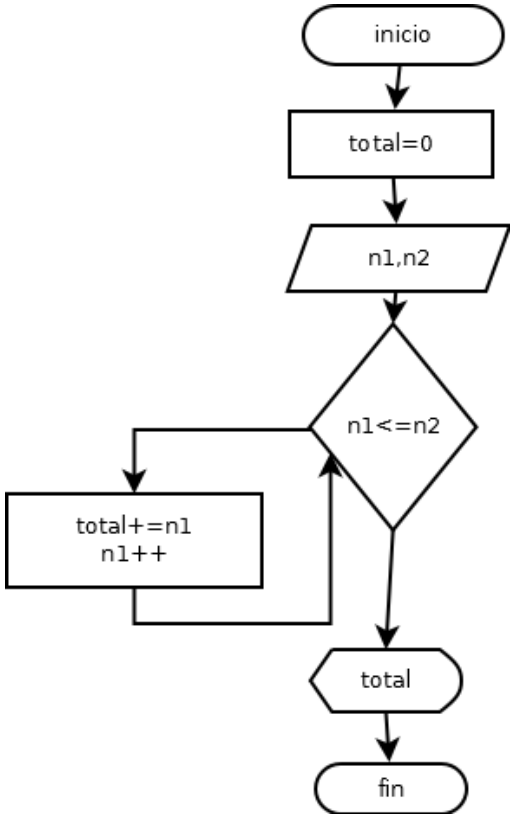
# Ejercicios

2. Implementa un programa que sume los números de *n1* a *n2* mediante un bucle.

   Tanto *n1* como *n2* serán indicados mediante `<input>` y el resultado será escrito en un `<output>`.

   Por ejemplo, para *n1=1* y *n2=5* hará 1+2+3+4+5 y mostrará el resultado: **15**.

   Usa el diagrama de flujo que se muestra a la derecha:



# Ejercicios

3. Haz un programa que muestre los 10 primeros múltiplos de un número dado mediante un *input*. Por ejemplo, si nos dan el 5, mostrará: 5, 10, 15, 20, 25, 30, 35, 40, 45, 50.

   Para mostrar correctamente el resultado, usa la siguiente estrategia:

   ```
   output.innerHTML += valor+', ';
   ```

   Con lo que se irán concatenando los distintos números en el `output`, (separados por comas) en lugar de ir machacando el valor en cada iteración.

# Ejercicios

4. Escribe un programa que imprima los números de n a m (introducidos ambos mediante *inputs* de formulario).

   Por ejemplo: para 100 y 103, mostrará: 100,101,102,103

5. Escribe un programa que eleve un número a otro (el exponente debe ser un número entero y positivo siempre). Los dos números se introducirán desde *inputs*.

   Por ejemplo: para 2 y 5, hará: 2*2*2*2*2 y mostrará 32.

   No se puede usar *Math.pow()* ni el operador **.

# Ejercicios

6. Realiza un programa que calcule **el factorial** de un número. Se calcula de la siguiente forma: 5! = 5x4x3x2 = 120.

   ¿Cuánto vale el factorial de 11?

| Número | Factorial | Número | Factorial |
|---|---|---|---|
| 0 | *1* | 6 | *720* |
| 1 | *1* | 7 | *5.040* |
| 2 | *2* | 8 | *40.320* |
| 3 | *6* | 9 | *362.880* |
| 4 | *24* | 10 | *3.628.800* |
| 5 | *120* | 11 | *???* |

# Ejercicios

7. Escribe un programa que nos diga todos los **divisores de un número n**, introducido por el usuario.

   Básicamente, intenta dividir el número n por 2,3,4… y si el resto de la división es 0, imprime ese número. Piensa en usar el operador módulo (%).

| Número | Divisores |
|---|---|
| 4 | 1,2,4 |
| 6 | 1,2,3,6 |
| 8 | 1,2,4,8 |
| 10 | 1,2,5,10 |

# Ejercicios

8. Intenta hacer el ejemplo de las tablas de multiplicar desde n hasta m, que vimos anteriormente, por tu cuenta sin consultar el ejemplo.

| Tabla del 1 | | | | |
|---|---|---|---|---|
| 1 | X | 0 | = | 0 |
| 1 | X | 1 | = | 1 |
| 1 | X | 2 | = | 2 |
| 1 | X | 3 | = | 3 |
| 1 | X | 4 | = | 4 |
| 1 | X | 5 | = | 5 |
| 1 | X | 6 | = | 6 |
| 1 | X | 7 | = | 7 |
| 1 | X | 8 | = | 8 |
| 1 | X | 9 | = | 9 |
| 1 | X | 10 | = | 10 |

| Tabla del 2 | | | | |
|---|---|---|---|---|
| 2 | X | 0 | = | 0 |
| 2 | X | 1 | = | 2 |
| 2 | X | 2 | = | 4 |
| 2 | X | 3 | = | 6 |
| 2 | X | 4 | = | 8 |
| 2 | X | 5 | = | 10 |
| 2 | X | 6 | = | 12 |
| 2 | X | 7 | = | 14 |
| 2 | X | 8 | = | 16 |
| 2 | X | 9 | = | 18 |
| 2 | X | 10 | = | 20 |

| Tabla del 3 | | | | |
|---|---|---|---|---|
| 3 | X | 0 | = | 0 |
| 3 | X | 1 | = | 3 |
| 3 | X | 2 | = | 6 |
| 3 | X | 3 | = | 9 |
| 3 | X | 4 | = | 12 |
| 3 | X | 5 | = | 15 |
| 3 | X | 6 | = | 18 |
| 3 | X | 7 | = | 21 |
| 3 | X | 8 | = | 24 |
| 3 | X | 9 | = | 27 |
| 3 | X | 10 | = | 30 |

# Ejercicios

9. Haz un programa que, a partir de dos números enteros “f” y “c” introducidos por formulario, cree una tabla HTML con “f” filas y “c” columnas como la que se muestra en la imagen:


Opcionalmente, puedes hacer también que las filas pares e impares tengan formatos diferentes e introducir una fila de cabecera.

# Ejercicios

10. Realiza un programa que calcule el resultado de **elevar un número a otro,** pero esta vez **debe funcionar para exponentes positivos y negativos**.

    Cuando el exponente es negativo, el cálculo es diferente.

    Debéis realizar algunas pruebas para validar el correcto funcionamiento. No os olvidéis de las fases de análisis, diseño y prueba.

    **No se puede usar** `Math.pow()` ni el operador ** así que ni lo preguntéis ☺

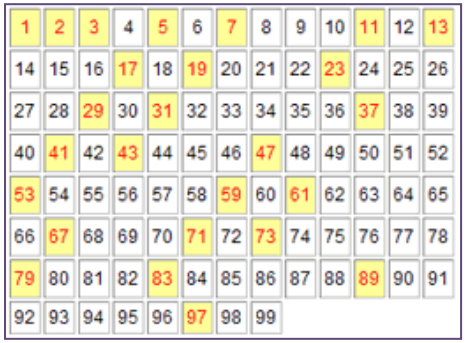
# Ejercicios

11. Realiza un programa que nos diga si un número introducido en un input es primo o no.

    Un número es primo si solamente es divisible por uno y por sí mismo.

    Puedes hacer un bucle que vaya dividiendo el número por 2,3,4,5…
    Si el resto de alguna de las divisiones es 0, el número no es primo.

    Estos son los primos del 1 al 99 para validar:



| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 |
| 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 |
| 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 |
| 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | | | | | |