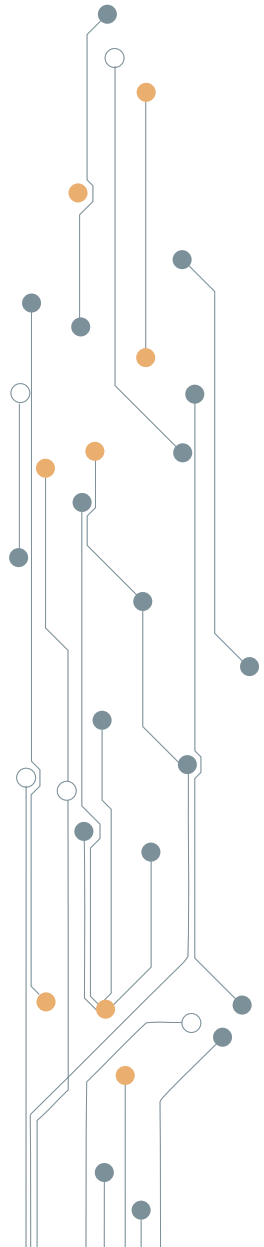
Completado
08/06/2019, 12:31:50

Control de flujo

Telefónica
EDUCACIÓN DIGITAL

Índice



1 | Control de flujo	3
1.1 | Estructura if	3
1.2 | Estructura if...else	4
1.3 | Estructura for	5
1.4 | Estructura for...in	6
2 | Funciones	7
2.1 | Funciones útiles para cadenas de texto	7
2.2 | Funciones útiles para arrays	9
2.3 | Funciones útiles para números	11
3 | El ámbito de las variables	12

1. Control de flujo

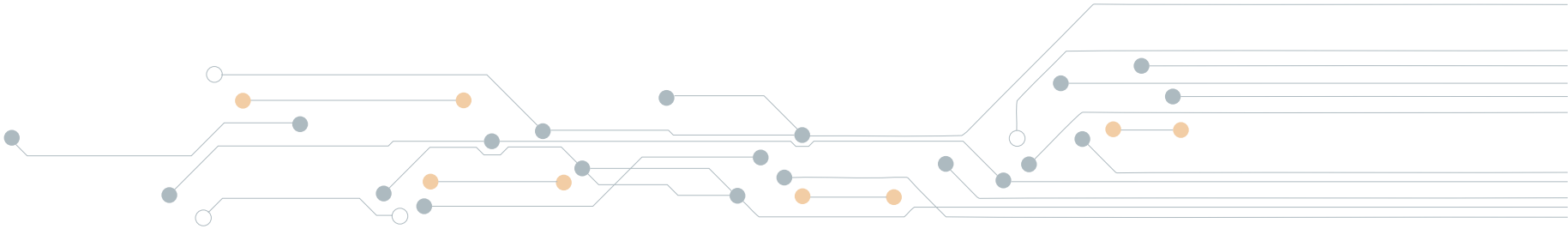
Los códigos que se pueden escribir usando solo variables y operadores, son una sucesión de instrucciones básicas.

Hay programas complejos como recorrer un array o establecer una condición que no pueden ser realizadas simplemente con una sucesión de instrucciones básicas, es por ello que necesitamos instrucciones de control de flujo que nos pueden elegir líneas para ejecutar dentro de nuestro código o repetir una serie de líneas un número de veces según una condición.

1.1 | Estructura if

Es una estructura de condición, si se comprueba el valor true de esa condición se entra dentro del bloque de código encerrado entre {...}, si no lo cumple no entra y, por tanto, no ejecuta esas líneas.

```
if(condicion) {...}
```



1.2 | Estructura if...else

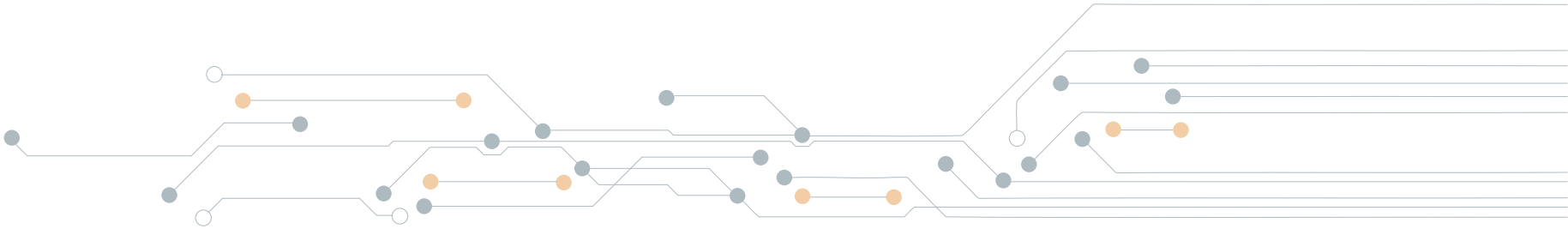
Muchas veces necesitamos ejecutar bloques diferentes dependiendo de una condición. Añadiendo la estructura anterior, se agrega un bloque "else" que permite ejecutarse en el caso que no se cumpla la condición del "if".

```
if(condicion) {...}
else {...}
```

Aparte, se pueden poner otros bloques de código para que se ejecuten si la condición del "if" no se cumple y queremos comprobar que se satisface otra condición. Estos van con la estructura "else if".

```
if(condicion) {...}
else if(condicion2) {...}
else {...}
```

En este caso anterior el bloque "else" solo se ejecuta si no se ejecuta ningún bloque anterior. Se pueden poner tantos bloques "else if" como se quiera, teniendo en cuenta que solo se comprobará su condición si todos los bloques anteriores han dado false en sus respectivas condiciones. Por tanto, el orden en el que incluyamos los bloques "else if" es importante. El bloque "else", por ello, solo se ejecutará al final (y solo se incluirá al final), cuando todos los demás bloques no han satisfecho a su condición.



1.3 | Estructura for

La estructura "for" permite usar repeticiones (denominadas bucles), para que reiteren líneas de código mientras se satisfaga una condición.

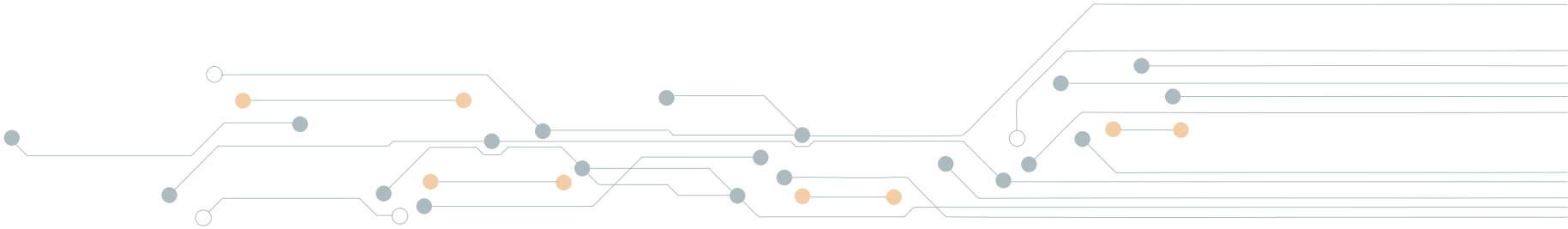
```
for(inicializacion; condicion;
actualizacion) {...}
```

El funcionamiento de la estructura "for" es: repite las líneas del trozo de código, encerrado entre {...}, mientras se cumpla la condición, actualizando con cada repetición los valores cambiantes de la condición.

La "inicialización" → los valores iniciales de las variables que controlan la repetición.

La "condición" → decide si continua o se detiene la repetición.

La "actualización" → valor que se asigna después de cada repetición.



1.4 | Estructura for...in

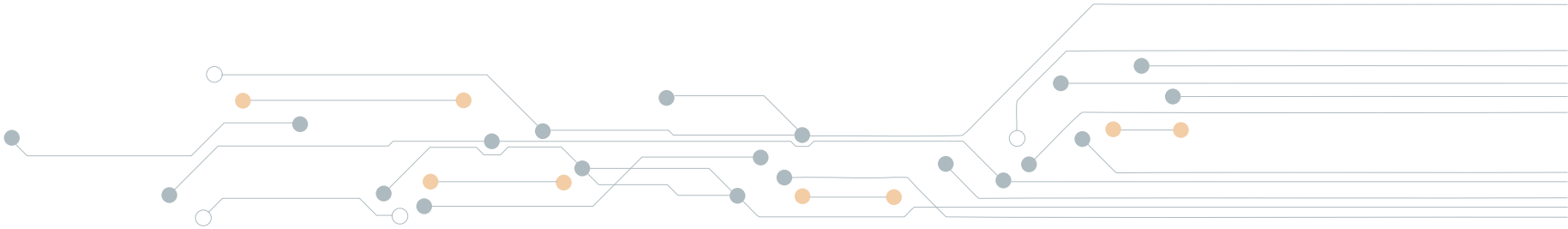
Es una variante de la estructura for y sirve para objetos y arrays dentro del lenguaje JavaScript.

```
for(indice in array) {...}
```

En este caso anterior, nos serviría para recorrer todos los elementos contenidos dentro del array.

```
var numbers =[0,1,2];
for(i in numbers) {var a =numbers[i];}
```

Daríamos 3 iteraciones dentro de nuestro bucle, una por cada elemento de nuestro array. Esta estructura es la óptima para recorrer un array o un objeto en JavaScript ya que funciona sea cual sea el número de elementos de nuestro array.



2. Funciones

Para manejar nuestras diferentes variables JavaScript hace uso de funciones y propiedades, que ya se encuentran en el propio lenguaje. A continuación veremos las funciones según su utilidad.

2.1 | Funciones útiles para cadenas de texto

- **length,** halla la longitud de una cadena de texto (el número de caracteres que la forman)

```
var men = “Hola Mundo”;
var numLetras = men.length; // numLetras = 10
```

- **+,** se emplea para concatenar varias cadenas de texto

```
var men1 = “Hola”;
var men2 = “ Mundo”;
var men = men1 + men2; // men = “Hola Mundo”
```

Aparte del operador +, tiene el mismo funcionamiento **concat()**

```
var men1 = “Hola”;
var men2 = men1.concat(“ Mundo”); // men2 = “Hola Mundo”
```

Y también con variables numéricas:

```
var var1 = “Hola “; var var2 = 3;
var men = var1 + var2; // men = “Hola 3”
```

- **toUpperCase()**, convierte los caracteres a mayúsculas.

```
var men1 = “Hola”;
var men2 = men1.toUpperCase(); // men2 = “HOLA”
```

- **toLowerCase()**, convierte los caracteres a minúsculas.

```
var men1 = “HolA”;
var men2 = men1.toLowerCase(); // men2 = “hola”
```

- **charAt(posicion)**, halla el carácter de la posición.

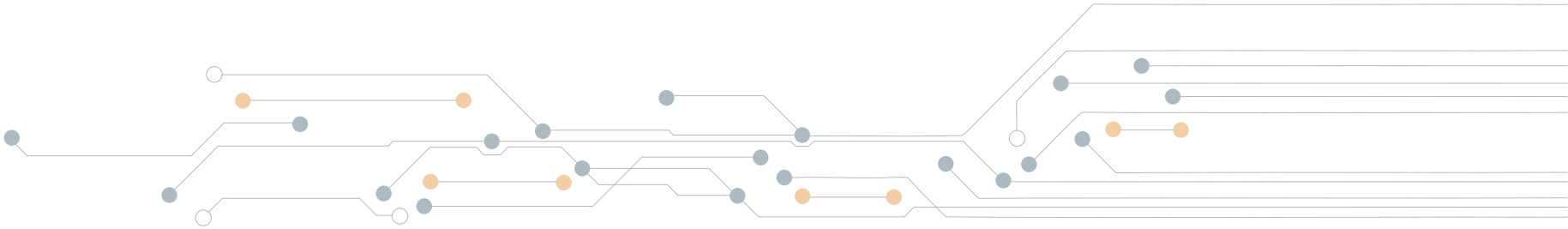
```
var men = “Hola”;
var l = men.charAt(0); // letra = H
l = men.charAt(3);     // letra = a
```

- **indexOf(caracter)**, halla la posición en la que se encuentra el carácter indicado. Si no está devuelve -1, y si está varias veces su primera aparición.

```
var men = “Hola”;
var pos = men.indexOf(‘a’); // pos = 3
pos = men.indexOf(‘z’);     // pos = -1
```

- **lastIndexOf(caracter)**, halla la última posición en la que se encuentra el carácter. Si no está devuelve -1.

```
var men = “Hola”;
var pos = men.lastIndexOf(‘a’); // pos = 3
pos = men.lastIndexOf(‘z’);     // pos = -1
```



- **substring(inicio, final)**, saca un trozo de una cadena de texto. El parámetro 'final' no es obligatorio. Si no se pone corta la cadena de texto hasta el final del string.

```
var men = "Hola Mundo";
var por = men.substring(2); // por = "la Mundo"
por = men.substring(5);     // por = "Mundo"
por = men.substring(1, 8); // por = "ola Mun"
por = men.substring(3, 4);     // por = "a"
por = men.substring(7);     // por = "ndo"
```

- **split(separador)**, divide la cadena de texto en diferentes trozos, definiendo un separador para dividir esa cadena, y mete las porciones dentro de un array.

```
var men = "Hello World, be a string!";
var p = men.split(" ");
// p = ["Hello ", " World,", " be ", " a ", " string!"];
```

2.2 | Funciones útiles para arrays

- **length**, halla el número de elementos dentro de un array.

```
var v = ["a", "e", "i", "o", "u"];
var num = v.length; // num = 5
```

- **concat()**, concatena los elementos de varios arrays.

```
var a1 = [1, 2, 3];
a2 = a1.concat(4, 5, 6);   // a2 = [1, 2, 3, 4, 5, 6]
a3 = a1.concat([4, 5, 6]); // a3 = [1, 2, 3, 4, 5, 6]
```

- **join(separador)**, une los elementos de un array para formar una cadena de texto. Es lo contrario al "split". Se índice un separador para unir los elementos de la cadena.

```
var a = ["hola", "mundo"];
var men = a.join(""); // men = "holamundo"
men = a.join(" ");    // men = "hola mundo"
```

- **pop()**, suprime el último elemento del array y lo mete en la variable seleccionada.

```
var a = [1, 2, 3];
var u = a.pop(); // ahora a = [1, 2], u = 3
```

- **push()**, agrega un elemento (o varios) a nuestro array.

```
var a = [1, 2, 3];
a.push(4); // ahora a = [1, 2, 3, 4]
```

cambio

- **shift()**, suprime el primer elemento de nuestro array y lo mete en la variable seleccionada.

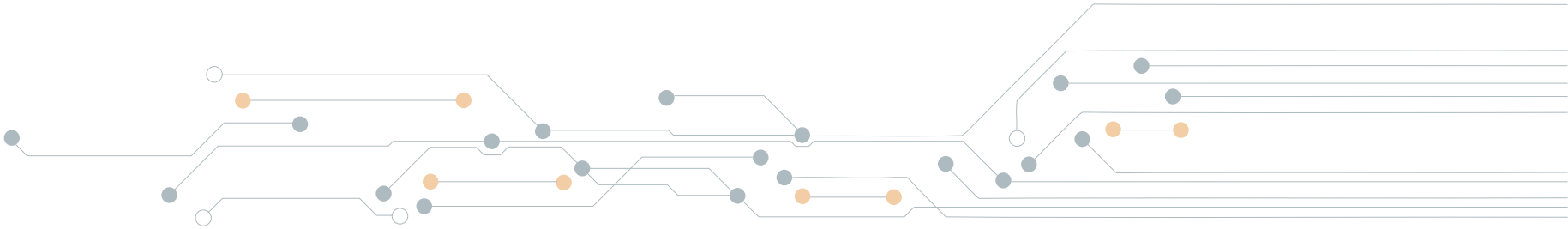
```
var a = [1, 2, 3];
var p = a.shift(); // ahora a = [2, 3], p = 1
```

- **unshift()**, agrega un elemento (o varios) al principio de nuestro array.

```
var a = [1, 2, 3];
a.unshift(0); // ahora a = [0, 1, 2, 3]
```

- **reverse()**, coloca los elementos de un array en su orden inverso.

```
var a = [1, 2, 3];
a.reverse(); // ahora a = [3, 2, 1]
```



2.3 | Funciones útiles para números

- **NaN, (del inglés, "Not a Number")** el lenguaje JavaScript devuelve esto si la variable con la que estamos trabajando o el resultado de una operación no es un número.

```
var num1 = 0; var num2 = 0;
alert(num1/num2); // muestra el valor NaN
```

- **isNaN()**, protege nuestro código de valores no numéricos.

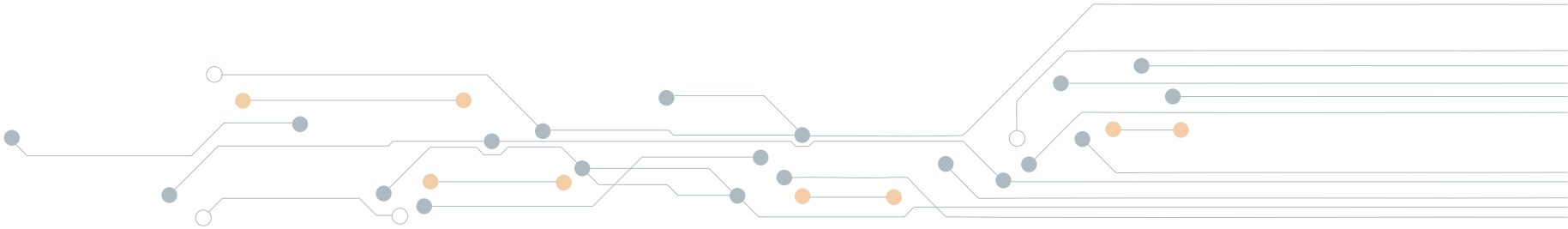
```
var num1 = 0; var num2 = 0;
if(isNaN(num1/num2)) {...} else {...}
```

- **Infinity**, es el valor de infinito cuando las operaciones dan dicho resultado.

```
var num1 = 10; var num2 = 0;
alert(num1/num2); // valor Infinity
```

- **toFixed(digitos)**, fija el número de decimales que tiene que resolver una operación y redondea si es necesario.

```
var num1 = 4564.34567;
num1.toFixed(2); // 4564.35
num1.toFixed(6); // 4564.345670
num1.toFixed(); // 4564
```



3. El ámbito de las variables

El ámbito de una variable ("scope"): ubicación en el código en la que se define la variable. Hay dos tipos de ámbitos: global y local.

```
function crea() { var m = "Mensaje";}
crea(); alert(m);
```

Este mensaje no muestra ningún mensaje ya que la variable ha sido inicializada dentro de una función por lo que se dice que es una variable local de esa función.

```
function crea() {var m = "Mensaje";alert(m);}
crea();
```

En este caso anterior el mensaje si se muestra ya que el "alert" ha sido ejecutado dentro de la misma función y tiene por tanto acceso a esa variable local. Ahora vamos a hablar de las variables globales:

```
var m = "Mensaje";
function muestra() {alert(m);}
```

La variable "m" la hemos definido fuera de cualquier función por lo que se convierte en una variable global inicializada en el momento en el que se ejecuta dicha sentencia. Por tanto, todas las funciones tienen acceso a esa variable.

Un problema habitual es llamar a una variable global y otra local de la misma manera. Si esto sucede, dentro de la función donde esté definida esa local, la local es la que toma prevalencia frente a la global.

```
var m = "global";
function muestra() {var m = "local"; alert(mensaje);}
alert(m);muestra();alert(m);
```

Si ejecutamos dicho código obtenemos:

```
global  local       global
```

Telefónica

EDUCACIÓN DIGITAL