# estructura de repetición

Ahora supongamos que queremos *pedirle al usuario que ingrese su DNI hasta que sea el correcto*. Resulta que no sabemos cuántas veces le puede llevar ingresarlo correctamente: puede ser una vez como pueden ser ochocientas sesenta y dos veces, es decir, esta cantidad de veces es indefinida y desconocida para nosotros.

Para esto los lenguajes de programación nos presentan las **estructuras de repetición**, en las cuales nuestro código lo escribimos una vez y se ejecutará reiterada cantidad de veces. ¿*Cuántas veces*? Las veces que sean necesarias hasta que se cumpla la **condición** que definamos.

# estructura de repetición: *while*

Ahora supongamos que queremos *pedirle al usuario que ingrese su DNI hasta que sea el correcto*. Resulta que no sabemos cuántas veces le puede llevar ingresarlo correctamente: puede ser una vez como pueden ser ochocientas sesenta y dos veces, es decir, esta cantidad de veces es indefinida y desconocida para nosotros.

Para esto los lenguajes de programación nos presentan las **estructuras de repetición**, en las cuales nuestro código lo escribimos una vez y se ejecutará reiterada cantidad de veces. ¿*Cuántas veces*? Las veces que sean necesarias hasta que se cumpla la **condición** que definamos.

# ¿cómo se define un while?

```
while (condicion) {
  ...
}
```

este código se ejecutará mientras que la condición sea verdadera. **¡Cuidado!** Si la condición es verdadera siempre, este código nunca dejará de ejecutarse y nuestro programa nunca va a terminar, es decir, la computadora se va a colgar.

La estructura **while** tiene dos elementos: una condición y una sección de código a ejecutar mientras la condición sea verdadera.  El funcionamiento será así: la primera vez, evaluará si la condición declarada es verdadera, y en ese caso ejecutará el código contenido. Luego de hacerlo, volverá a evaluar si la condición es verdadera, para en tal caso ejecutar de nuevo el código contenido y así ***infinitamente***. Una vez que la condición no sea verdadera, terminará la repetición.

# ejemplos de while

```
let dniDelUsuario = pedirDni()

while (dniDelUsuario !== '29410248') {
  notificarDniIncorrecto()

  dniDelUsuario = pedirDni()
}

notificarDniCorrecto()
```

inicializamos una variable dniDelUsuario con el resultado de la ejecución de la función ***pedirDni***

definimos que nuestro programa va a ejecutar una sección de código mientras el valor de la variable ***dniDelUsuario*** no sea el texto ***'29410248'***

ejecutamos una serie de código en el cual debemos modificar la variable que va a volver a evaluarse una vez que termine este ***loop*** (repetición). En el caso en el que no lo hagamos, podría ejecutarse infinitamente.

una vez finalizado nuestro ***while***, el programa sigue normalmente su flujo. En este caso, como la condición de que el ***dniDelUsuario*** no es ***'29410248'*** es falsa, estamos en condiciones de afirmar que el ***dniDelUsuario*** es ***'29410248'***, por lo que ejecutamos la función ***notificarDniCorrecto***

# estructura de repetición: *for*

Volvamos a suponer. Esta vez supongamos que *le queremos pedir al usuario sus tres gustos de helado preferidos*.

Resulta que esta vez sí sabemos exactamente cuántas veces queremos que se ejecute una sección de código, en este caso **tres**.

Los lenguajes de programación ofrecen una estructura de control para este caso en particular llamada ***for***, que curiosamente es una manera de escribir un ***while*** de forma más sencilla e intuitiva.

# ¿cómo se define un for?

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

antes de que se ejecute por primera vez esta sección de código, se ejecutará la **inicialización**. Luego se evaluará si la **condición** es verdadera, ahí se ejecutará **esta sección de código**, para al finalizar, ejecutar el **incremento**.

La estructura **for** tiene cuatro elementos: una inicialización, una condición, un incremento y un código a ejecutar en cada *loop* (repetición). Antes de ejecutar cualquier cosa, se ejecutará la inicialización (o definición de una variable), luego se evaluará la verdad de la condición. En tal caso, se ejecutará el contenido del for y luego el incremento (de la variable). Luego se volverá a evaluar la verdad de la condición, y así ***infinitamente***.

# ejemplos de for

```
for (let i = 0; i < 2; i = i + 1) {
  ...
}
```

la inicialización será **definir una variable llamada i** (*así se suelen llamar a las variables usadas en los for*) **cuyo valor será 0**. La condición a evaluar será **que i sea menor a 2**, y el incremento será **sumarle 1 al valor de i**.

El resultado de esto será que ***el código a ejecutar se ejecutará dos veces***. ¿Por qué? Lo primero que va a ejecutarse será definir una variable ***i*** con valor 0. Luego se evaluará si el valor de ***i*** es menor a 2, y como 0 es menor a 2, será *verdadero*. Se ejecutará el código para luego incrementar en 1 a ***i*** (es decir, pasará a valer 1). Luego se evaluará si el valor de ***i*** (1) es menor a 2, lo que será cierto, para volver a ejecutar el código y sumarle 1 a ***i*** (para pasar a valer 2). Por último, se evaluará si el valor de ***i*** (2) es menor a 2, lo cual será falso, por lo que terminará la ejecución del for.

# ejercicio de for ✍️

1. Definir una función llamada ***imprimirNumeros*** que reciba como parámetro *un número* e imprima en consola todos los números desde el 1 hastsa el número del parámetro.

2. Definir una función llamada ***imprimirOtrosNumeros*** que reciba dos parámetros: un número desde y un número hasta. En el caso que el número desde sea menor al número hasta, debe imprimir en consola todos los números que están entre el primer parámetro y el segundo.

# ¿qué hacer con un array?

Bien. Ya aprendimos a definir una variable, a escribir una función y a controlar el flujo de nuestros programas.

Nos faltan aprender algunas cosas más del mundo Javascript. Por ejemplo saber qué hacer con un array.

Al definir un array, **Javascript lo tratará como un objeto** al cual le agregará propiedades que nos permitan interactuar con el array. ¿Qué podemos saber? Por ejemplo podemos *saber el tamaño del array* (la cantidad de elementos que tiene), *agregarle un elemento nuevo*, *borrarle un elemento*, entre otras cosas.

# ¿qué propiedades tiene un array?

Una vez definido nuestro array, este tendrá una serie de propiedades para que podamos interactuar con él. Estos son algunas de esas propiedades:

- **length**: tamaño del array, es decir, la cantidad de elementos que contiene.
- **push**: función que recibe como parámetro un elemento a agregar al array.
- **pop**: función que elimina el último elemento del array.
- **shift**: función que elimina el primer elemento del array.
- **splice**: función para borrar elementos del array que recibe *dos parámetros*: *índice a partir del cual se quiere borrar elementos*; y *cantidad de elementos a borra*r.
- **includes**: función para buscar si un elemento existe en el array.

# ¿cómo interactuar con un array?

```
let artistas = ['Miguel Abuelo', 'Fito Paez']
console.log(artistas.length)
artistas.push('Adrián Dargelos')
artistas.pop()
artistas.push('Pity Álvarez')
artistas.splice(2, 1)
console.log(artistas.length)
console.log(artistas)
```

inicializamos un array ***artistas*** con *dos elementos*

*imprimimos en pantalla* la cantidad de elementos del array ***artistas***, es decir, **2**

agregamos *un elemento* a ***artistas***, que ahora tiene *tres elementos*

borramos el último elemento de ***artistas***, "*Adrián Dárgelos*"

agregamos *un nuevo elemento* a ***artistas***

borramos *un elemento* a partir del *elemento con índice 2* (*"Pity Alvarez"*). Es decir, borramos el elemento *"Pity Alvarez"*

*imprimimos en pantalla* el tamaño de ***artistas***, es decir, 2

*imprimimos en pantalla* ***artistas***, cuyos elementos ahora son *"Miguel Abuelo" y "Fito Paez"*

# ¿qué es recorrer un array?

Dijimos que los arrays son listas de elementos. Es muy probable que una vez que hayamos definido un array queramos saber cuáles son sus elementos y usarlos de algún modo. A esta acción la llamamos ***recorrer/iterar un array***. Recorrer un array será acceder al contenido de cada uno de sus elementos. Lo cual, de alguna manera, supone repetir la acción de acceder a un elemento particular las veces que sea necesario.

*¿Cuántas veces tenemos que repetir la acción de acceder a un elemento particular de un array para acceder a todos sus elementos?* Eso va a depender, precisamente, de la cantidad de elementos que tenga este array.

# ¿cómo se recorre un array?

```
const artistas = ['Miguel Abuelo', 'Fito Paez']

for (let i = 0; i <  artistas.length; i++) {
  console.log(artistas[i])
}
```

definimos un array con dos elementos

declaramos un ***for*** inicializando una variable ***i*** en 0, que se repetirá mientras que esta variable ***i*** sea menor o igual a la *cantidad de elementos que tiene* ***artistas***

dentro de *cada iteración*, ***i*** tendrá distintos valores, que serán los **distintos índices** que tengan los elementos de ***artistas***

El array tiene dos elementos, por lo que el ***for*** hará dos iteraciones: en la primera, ***i*** tendrá como valor 0, y en la segunda, 1. De este modo, dentro del contenido del for, accediendo a ***artistas[i]*** estaremos accediendo al *elemento con índice i del array artistas*. En el primer caso será ***artistas[0]***, es decir, el primer elemento de artistas ("*Miguel Abuelo*"), y en el segundo, ***artistas[1]***, el segundo elemento ("*Fito Paez*").

# ejercicio de array ✍️

1. Definir una función ***duplicarNumeros*** que reciba como parámetro *una lista de números* y devuelva la lista con esos números duplicados.

2. Definir una función ***multiplicar*** que reciba como parámetro *una lista de números* y devuelva el resultado de la multiplicación de estos números.

3. Definir una función ***calcularPromedio*** que reciba como parámetro *una lista de números* y devuelva el promedio de estos números.

# ¿qué hacer con un string?

Analicemos a una palabra. Podemos pensar a una palabra como un conjunto de letras. Javascript las piense del mismo modo.

**Los strings son justamente *arrays* de caracteres** (*letras, números, espacios, símbolos*).

Esto nos permite, por ejemplo, saber cuántos caracteres tiene un string del mismo modo que obtenemos el tamaño de un array, o acceder a un caracter en particular igual que accedemos a un elemento en particular de una lista.

# ¿qué propiedades tiene un string?

Al igual que los arrays, los strings tienen ciertas propiedades mediante la cual podemos acceder a cierta información de ellos. Algunas son:

- **length**: tamaño del string, es decir, la cantidad de caracteres que tiene.
- **includes**: función para buscar si un string existe dentro de este string.
- **replace**: reemplazar un string por otro dentro de este string.
- **substring**: obtener un nuevo string a partir del original *desde el caracter que está en el índice de primer parámetro hasta el caracter que está en el índice del segundo*.
- **toLowerCase**: obtener el string todo en minúscula.
- **toUpperCase**: obtener el string todo en mayúscula.

# ¿cómo interactuar con un string?

```
const cancion = 'El vino entibia sueños al jadear'
console.log(cancion[4])
console.log(cancion.length)
```

definimos una variable ***canción*** con *un string*

accedemos al elemento con índice 4 de ***canción***, es decir, al quinto caracter (la letra ***i***)

consultamos la cantidad de caracteres que tiene el string ***cancion***, en este caso, 32

Es así que a un ***string*** lo podemos pensar como un array.

Podemos contar sus elementos, acceder a un elemento en particular, e incluso podemos recorrerlo, accediendo a cada uno de sus caracteres.

# operaciones entre strings

```
const primeraEstrofa = 'Mi libertad es corta'
const segundaEstrofa = 'puedo salir a algún lado'
const cancion = primerEstrofa + ', ' + segundaEstrofa
```

definimos una variable ***primeraEstrofa*** con *un string*

definimos una variable ***segundaEstrofa*** con *un string*

definimos una variable ***cancion*** cuyo valor es la *suma* (o *concatenación*) entre ***primeraEstrofa***, el texto ***", "*** y ***segundaEstrofa***, dando como resultado: ***"Mi libertad es corta, puedo salir a algún lado"***

***Concatenar*** dos strings es juntar cada uno de sus caracteres con otros caracteres.

En este caso, juntamos ***una primer estrofa*** con ***una coma y un espacio***, y luego con ***una segunda estrofa*** dando como resultado ***una canción***.

# ejercicio de strings ✍️

1. Definir una función ***contarEspacios*** que reciba *un texto* y devuelva la cantidad de espacios que tenga.

2. Definir una función ***buscarTweets*** que reciba *una lista de tweets (mensajes)* y *un texto a buscar*, y devuelva todos los tweets que contengan ese texto.

3. Definir una función ***censurarTexto*** que reciba *una frase* y *una lista de expresiones censuradas* y devuelva la frase reemplazando esas expresiones por astericos que tengan su mismo tamaño.

# ejercicio para investigar 🔬

1. Definir ***calcularVuelto*** que reciba *un precio a pagar* y *un monto con el que se paga* y debe expresar el vuelto en **billetes** (cuántos y cuáles), buscando que se entregue la menor cantidad de billetes posible.

2. Definir ***dibujarTriangulo*** que reciba *una altura del triángulo* e imprima en consola un triángulo con esa altura hecho con asteriscos.