



Fullstack Developer Software

MATERIAL TÉCNICO DE APOYO

TAREA N° 01

Conoce y aplica prácticas de desarrollo de software colaborativo.

1. DEFINE GIT Y GITHUB.

Todos los desarrolladores utilizarán algún tipo de sistema de control de versiones (VCS), una herramienta que les permita colaborar con otros desarrolladores en un proyecto sin peligro de que sobrescriban el trabajo de los demás, y volver a las versiones anteriores de la base de código si existe un problema descubierto más tarde. El VCS más popular (al menos entre los desarrolladores web) es Git, junto con GitHub, un sitio que proporciona alojamiento para tus repositorios y varias herramientas para trabajar con ellos.



[Git y GitHub.](#)

Los VCS son esenciales para el desarrollo de software:

- Es raro que trabajes en un proyecto completamente por tu cuenta, y tan pronto como comiences a trabajar con otras personas, comenzarás a correr el riesgo de entrar en conflicto con el trabajo del otro, es decir, cuando ambos intentan actualizar simultáneamente la misma pieza de código. Debes tener algún tipo de mecanismo para administrar las ocurrencias y, como resultado, evitar la pérdida de trabajo.
- Cuando trabajes en un proyecto por tu cuenta o con otros, querrás poder hacer una copia de seguridad del código en un lugar central, para que no se pierda si tu computadora se daña.

- También querrás poder volver a versiones anteriores si más tarde descubres un problema. Es posible que hayas empezado a hacer esto en tu propio trabajo mediante la creación de diferentes versiones de un mismo archivo, por ejemplo `myCode.js`, `myCode_v2.js`, `myCode_v3.js`, `myCode_final.js`, `myCode_really_really_final.js`, etc, pero esto es muy propenso a errores y poco fiable.
- Los diferentes miembros del equipo generalmente querrán crear sus propias versiones separadas del código (llamadas ramas en Git), trabajar en una nueva característica en esa versión y luego fusionarla de manera controlada (en GitHub usamos solicitudes de extracción) con la versión maestra cuando hayan terminado con ella.

Los VCS proporcionan herramientas para satisfacer las necesidades anteriores. Git es un ejemplo de VCS, y GitHub es un sitio web + infraestructura que proporciona un servidor Git más una serie de herramientas realmente útiles para trabajar con repositorios git individuales o en equipo, como informar problemas con el código, herramientas de revisión, características de administración de proyectos tal como asignación de tareas, estados de tareas, y más.

Nota: Git en realidad es un sistema de control de versiones distribuido, lo cual significa que se realiza una copia completa del repositorio que contiene la base de código en tu computadora (y en la de todos los demás). Realizas cambios en tu propia copia, y luego empujas esos cambios nuevamente al servidor, donde un administrador decidirá si fusiona tus cambios con la copia maestra.

- **Prerrequisitos**

Para usar Git y GitHub, necesitas:

- Una computadora de escritorio con Git instalado (consulta la página de descargas de Git).
- Una herramienta para usar Git. Dependiendo de cómo te guste trabajar, puedes usar un cliente Git con GUI (te recomendamos GitHub Desktop, SourceTree o Git Kraken) o simplemente usar una ventana de la terminal. De hecho, probablemente sea útil que conozcas al menos los conceptos básicos de los comandos de la terminal git, incluso si tienes la intención de usar una GUI.
- Una cuenta de GitHub. Si aún no tienes una, regístrate ahora usando el enlace provisto.

En términos de conocimiento previo, no necesitas saber nada sobre desarrollo web, Git/GitHub o VCS para iniciar este módulo. Sin embargo,

se recomienda que conozcas algo de codificación para que tengas conocimientos informáticos razonables y algún código para almacenar en tus repositorios.

También es preferible que tengas algunos conocimientos básicos de la terminal, por ejemplo, moverte entre directorios, crear archivos y modificarla variable del sistema PATH.

Nota: Github no es el único sitio/conjunto de herramientas que puedes usar con Git. Hay otras alternativas, como GitLab, que podrías probar, y también podrías intentar configurar tu propio servidor Git y usarlo en lugar de GitHub. Solo nos hemos quedado con GitHub en este curso para proporcionar una forma única que funciona.

2. COMANDOS GIT.

- **Sinopsis**

```
git [-v | --version] [-h | --help] [-C <ruta>] [-c <nombre>=<valor>]
    [--exec-path[=<ruta>]] [--html-path] [--man-path] [--info-path]
    [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--no-lazy-fetch]
    [--no-optional-locks] [--no-advice] [--bare] [--git-dir=<ruta>]
    [--work-tree=<ruta>]      [--namespace=<nombre>]      [--config-
    env=<nombre>=<var_entorno>]
    <comando> [<args>]
```

- **Descripción**

Git es un sistema de control de revisiones distribuido, rápido y escalable con un conjunto de comandos inusualmente rico que proporciona operaciones de alto nivel y acceso completo a los componentes internos.

-v
--versión

Imprime la versión de la suite Git de la que proviene el programa Git. Esta opción se convierte internamente a `git version ...` y acepta las mismas opciones que el comando `git-version[1]--help`. Si también se proporciona, tiene prioridad sobre `--version`.

-h
--ayuda

Imprime la sinopsis y una lista de los comandos más utilizados. Si se proporciona la opción `--all`, `-ase` imprimen todos los comandos disponibles. Si se nombra un comando de Git, esta opción mostrará la página del manual de ese comando.

Hay otras opciones disponibles para controlar cómo se muestra la página del manual. Consulte `git-help[1]` para obtener más información, ya que `git --help ...` se convierte internamente en `git --help`

`-C <ruta>`

Se ejecuta como si Git se hubiera iniciado en `<path>` en lugar del directorio de trabajo actual. Cuando `-C` se dan varias opciones, cada opción no absoluta subsiguiente `-C <path>` se interpreta en relación con la anterior `-C <path>`. Si `<path>` está presente pero está vacío, por ejemplo `-C ""`, el directorio de trabajo actual no se modifica. Esta opción afecta a las opciones que esperan nombres de ruta como `--git-dir` `--work-tree` en el sentido de que sus interpretaciones de los nombres de ruta se harían en relación con el directorio de trabajo causado por la opción. Por ejemplo, las siguientes invocaciones son equivalentes:

```
git --git-dir=a.git --work-tree=b -C c estado
git --git-dir=c/a.git --work-tree=c/b estado
-c <nombre>=<valor>
```

Pase un parámetro de configuración al comando. El valor proporcionado anulará los valores de los archivos de configuración. Se espera que `<name>` tenga el mismo formato que aparece en `git config` (subclaves separadas por puntos). Tenga en cuenta que se permite omitir el `=in` y se establece en el valor booleano `true` (tal como se haría en un archivo de configuración). Incluir `=` pero con un valor vacío como se establece en la cadena vacía que se convertirá en `.git`.

```
-c foo.bar ...foo.bar[foo]bargit -c foo.bar= ...foo.bargit config --
type=boolfalse
--config-env=<nombre>=<variable de entorno>
```

Al igual que `-c <name>=<value>`, asigne un valor a la variable de configuración `<nombre>`, donde `<var_entorno>` es el nombre de una variable de entorno de la que se recuperará el valor. A diferencia de `-c`, no existe un atajo para establecer directamente el valor en una cadena vacía, sino que la variable de entorno en sí debe establecerse en la cadena vacía. Es un error si `<envvar>` no existe en el entorno. `<envvar>` no puede contener un signo igual para evitar la ambigüedad de `<name>` contener uno.

Esto es útil para los casos en los que desea pasar opciones de configuración transitorias a Git, pero lo hace en sistemas operativos donde otros procesos podrían leer su línea de comandos (por ejemplo, `/proc/self/cmdline`), pero no su entorno (por ejemplo, `/proc/self/environ`). Ese comportamiento es el predeterminado en Linux, pero puede que no lo sea en su sistema.

Tenga en cuenta que esto puede agregar seguridad para variables como, por ejemplo `http.extraHeader`, dónde la información confidencial es parte del valor, pero no, por ejemplo, `url.<base>.insteadOf` dónde la información confidencial puede ser parte de la clave.

`--exec-path[=<ruta>]`

Ruta a donde están instalados los programas Git principales. Esto también se puede controlar configurando la variable de entorno `GIT_EXEC_PATH`. Si no se proporciona ninguna ruta, Git imprimirá la configuración actual y luego saldrá.

`--ruta-html`

Imprime la ruta, sin barra final, donde está instalada la documentación HTML de Git y sale.

`--ruta-del-hombre`

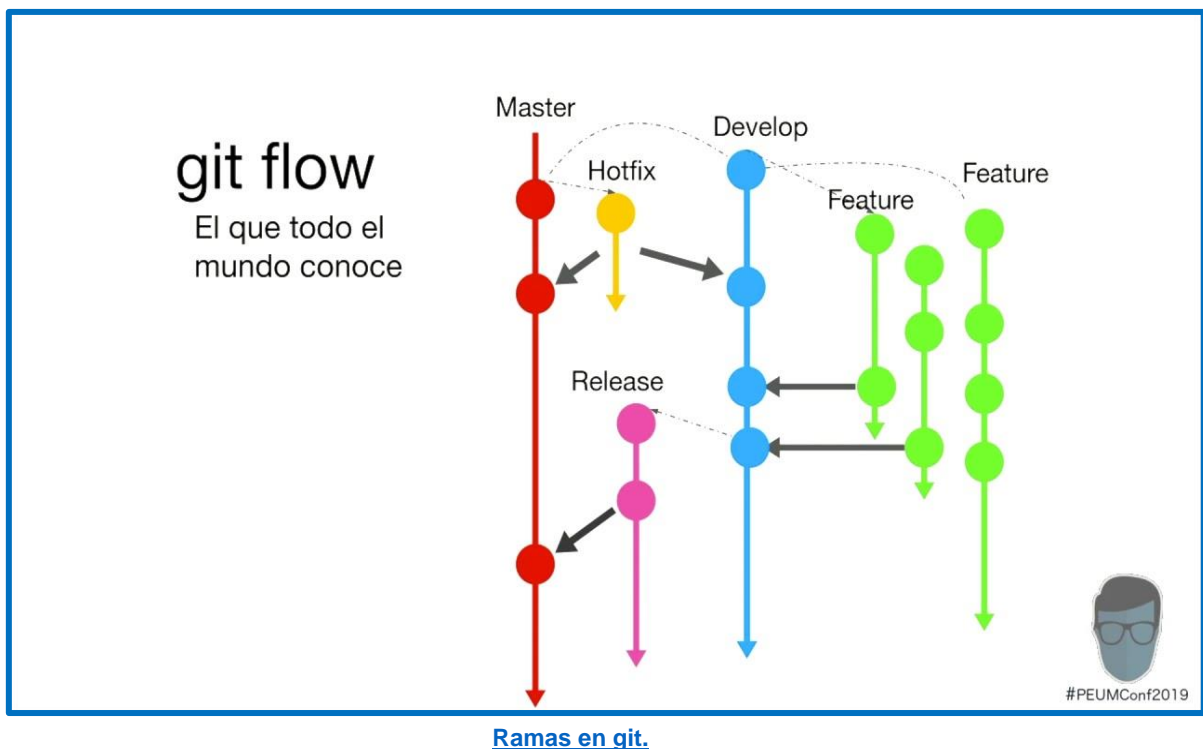


[Comandos GIT](#)

3. DEFINICIÓN DE RAMAS GIT

Git es un sistema de control de versiones distribuido que permite a los desarrolladores trabajar en paralelo y gestionar versiones de código de manera eficiente. Uno de los conceptos clave en Git es el uso de ramas, que permiten trabajar en diferentes versiones de un proyecto simultáneamente.

- **¿Qué es una rama en Git?** Una rama es una línea independiente de desarrollo dentro de un proyecto de Git. Funciona como un "punto de partida" o "copia" que permite hacer cambios sin afectar la rama principal (generalmente main o master). Esto facilita experimentar o trabajar en características nuevas sin comprometer la versión estable del código.
- **¿Por qué usar ramas?** Las ramas son fundamentales en el desarrollo colaborativo y permiten que cada miembro de un equipo trabaje en su tarea sin interferir con el trabajo de otros. Al final, las ramas se pueden fusionar (merge) en la rama principal, integrando los cambios.
- **Flujo de trabajo común con ramas:**
 - Crear una nueva rama para una característica o bug específico.
 - Hacer commits en esta rama para mantener un historial de los cambios realizados.
 - Fusionar la rama con la rama principal una vez completada la tarea y tras pasar por revisiones de código.



Los tipos de ramas en Git son:

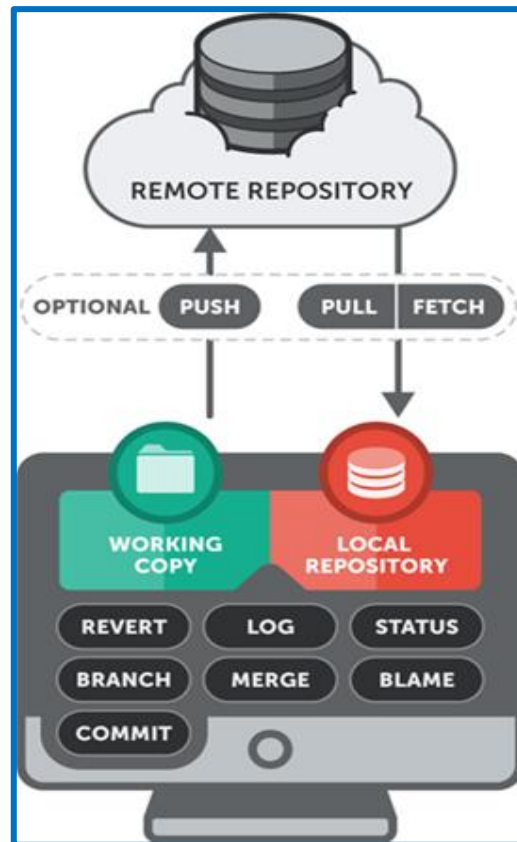
- **Rama principal (main o master):** Es la rama principal que contiene la versión más estable y lista para producción del código. En un flujo de trabajo típico, los cambios se fusionan en esta rama solo después de pasar pruebas y revisiones.
- **Ramas de características (feature):** Se crean para trabajar en nuevas funcionalidades. Son ramas temporales y se eliminan una vez que se fusionan con la rama principal.
- **Ramas de corrección de errores (bugfix):** Sirven para solucionar errores específicos. Una vez solucionado el error, se fusionan con la rama principal.
- **Ramas de desarrollo (develop):** En algunos flujos de trabajo (como Gitflow), se utiliza una rama de desarrollo que sirve de "puente" antes de fusionar los cambios en main. Es útil para consolidar varias características antes de lanzar una nueva versión.
- **Ramas de soporte (hotfix):** Estas ramas se utilizan para solucionar problemas urgentes en producción. Se crean desde la rama principal y, una vez solucionado el problema, se fusionan nuevamente en main y en develop.

4. TIPOS DE REPOSITORIOS.

Los repositorios son espacios donde se almacenan y gestionan los archivos y el historial de cambios de un proyecto. Hay distintos tipos de repositorios en función de su configuración y uso:

- **Repositorios Locales:** Son repositorios que existen en la computadora de un usuario. Cada usuario puede tener su propia copia del repositorio completo con su historial de cambios. Git es un sistema distribuido, por lo que cada copia local es un repositorio en sí mismo.
- **Repositorios Remotos:** Son repositorios ubicados en un servidor y a los que se puede acceder desde diferentes máquinas. Los repositorios remotos facilitan la colaboración, ya que permiten a varios desarrolladores trabajar en el mismo proyecto y sincronizar sus cambios.
- **Repositorios Centralizados:** Aunque Git es distribuido, algunos equipos pueden establecer un "repositorio central" en el que todos los colaboradores deben sincronizar sus cambios. El repositorio central puede estar en plataformas como GitHub, GitLab, o Bitbucket. Aunque todos los usuarios tienen una copia local, el repositorio central actúa como la versión principal compartida del proyecto.

- **Repositorios Públicos:** Son repositorios accesibles para cualquier persona, como en GitHub o GitLab, donde los proyectos de código abierto están disponibles para que otros los revisen, utilicen o contribuyan.
- **Repositorios Privados:** Estos repositorios están restringidos a un grupo selecto de usuarios o equipos. Solo las personas autorizadas pueden acceder, clonar o hacer cambios en estos repositorios. Son comunes en proyectos internos de empresas.



[Ejemplo de repositorio Git](#)

A continuación, veremos los Consejos para el Trabajo con Repositorios y Ramas en Desarrollo Colaborativo:

- **Establecer convenciones para nombres de ramas:** Utilizar prefijos como feature/, bugfix/, o hotfix/ ayuda a organizar el trabajo y hacer que los propósitos de cada rama sean claros.
- **Usar Pull Requests o Merge Requests:** Son solicitudes de fusión que permiten que otros revisen y aprueben el código antes de integrarlo en la rama principal.
- **Mantener una rama main o master estable:** Asegurarse de que esta rama esté siempre lista para ser desplegada en producción sin errores.

TAREA N°02

Usa entorno de ejecución backend con JavaScript.

1. ALGORITMOS Y PROGRAMACIÓN CON JAVASCRIPT.

Los algoritmos son conjuntos de instrucciones o pasos lógicos que resuelven un problema o realizan una tarea específica. En el contexto de la programación, los algoritmos son la base fundamental para resolver problemas de manera eficiente y automatizada.

- **Importancia de utilizar algoritmos eficientes**

La utilización de algoritmos eficientes es crucial para los desarrolladores de JavaScript. Un algoritmo eficiente permite realizar tareas en menos tiempo y con menos recursos, lo que se traduce en una mejor experiencia para los usuarios finales.

Al dominar los algoritmos los desarrolladores pueden optimizar el rendimiento de sus aplicaciones, mejorar la escalabilidad y reducir los tiempos de respuesta. Esto se traduce en aplicaciones más rápidas, eficientes y capaces de manejar de manera efectiva los retos y demandas del mundo real.

- **Algoritmos de búsqueda**

Los algoritmos de búsqueda son fundamentales en el desarrollo de aplicaciones para encontrar elementos específicos en un conjunto de datos. En JavaScript, existen varios algoritmos de búsqueda comunes, dos de los cuales son el Algoritmo de Búsqueda Lineal y el Algoritmo de Búsqueda Binaria.

- **Algoritmo de Búsqueda Lineal**

El Algoritmo de Búsqueda Lineal es el método más simple y directo para buscar un elemento en una lista o arreglo. Comienza desde el primer elemento y recorre secuencialmente cada uno hasta encontrar la coincidencia deseada. En el peor de los casos, si el elemento se encuentra al final de la lista, se recorrerán todos los elementos.

```
function busquedaLineal(arr, elemento) {  
  for (let i = 0; i < arr.length; i++) {  
    if (arr[i] === elemento) {  
      return i; // Retorna el índice del elemento encontrado  
    }  
  }  
}
```

```
    return -1; // Retorna -1 si el elemento no se encuentra en el arreglo
  }
```

```
const arreglo = [10, 5, 3, 8, 2, 6];
const elementoBuscado = 8;
const indice = busquedaLineal(arreglo, elementoBuscado);
console.log(`El elemento ${elementoBuscado} se encuentra en el
índice ${indice}.`);
// Output:
// El elemento 8 se encuentra en el índice 3.
```

➤ Algoritmo de búsqueda binaria

El Algoritmo de Búsqueda Binaria es una técnica más eficiente para buscar elementos en una lista ordenada. Funciona dividiendo repetidamente a la mitad el rango de búsqueda hasta encontrar la coincidencia.

Este enfoque aprovecha la propiedad de que los datos están ordenados, lo que permite descartar la mitad de los elementos en cada paso.

```
function busquedaBinaria(arr, elemento) {
  let inicio = 0;
  let fin = arr.length - 1;
  while (inicio <= fin) {
    let medio = Math.floor((inicio + fin) / 2);
    if (arr[medio] === elemento) {
      return medio; // Retorna el índice del elemento encontrado
    } else if (arr[medio] < elemento) {
      inicio = medio + 1;
    } else {
      fin = medio - 1;
    }
  }
  return -1; // Retorna -1 si el elemento no se encuentra en el arreglo
}
```

```
const arregloOrdenado = [2, 3, 5, 6, 8, 10];
const elementoBuscado = 6;
const indice = busquedaBinaria(arregloOrdenado,
elementoBuscado);
console.log(`El elemento ${elementoBuscado} se encuentra en el
índice ${indice}.`);
```

Sabiendo esto, podemos decir que el Algoritmo de Búsqueda Lineal es útil cuando trabajamos con un conjunto de datos pequeño o que no están ordenados y no se dispone de ninguna información adicional sobre la ubicación del elemento.

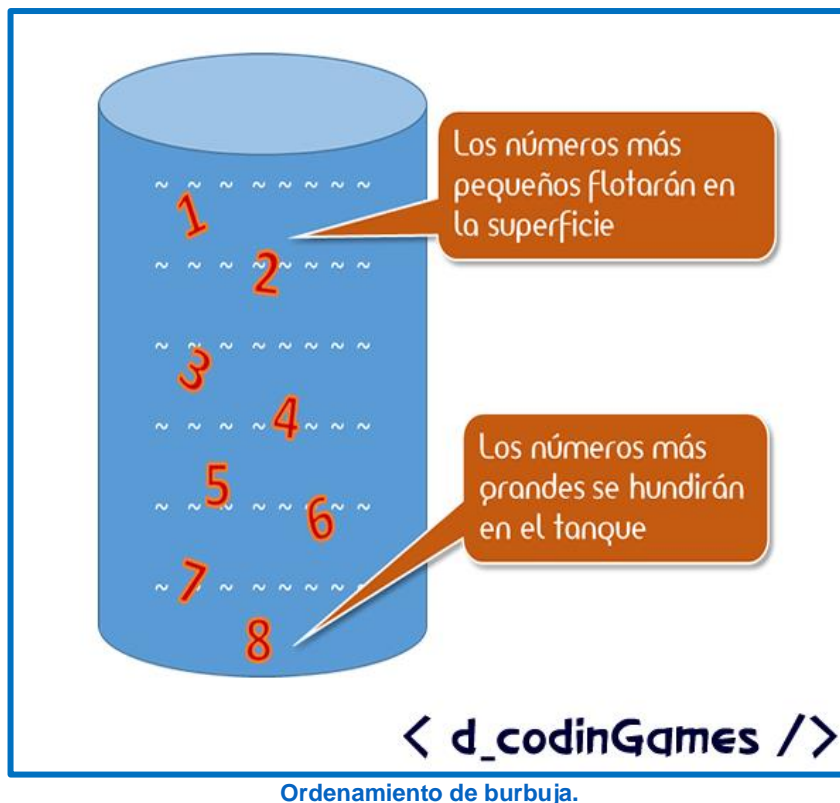
Sin embargo, en casos en los que los datos están ordenados, el Algoritmo de Búsqueda Binaria puede ser mucho más eficiente, ya que reduce el rango de búsqueda a la mitad en cada iteración.

○ Algoritmos de Ordenamiento

Los Algoritmos de Ordenamiento son técnicas utilizadas para organizar los elementos de una lista o arreglo en un orden específico. En JavaScript, existen varios algoritmos de ordenamiento populares y de gran utilidad para los desarrolladores. A continuación, se presentan algunos de ellos:

➤ Algoritmo de Ordenamiento Burbuja

El algoritmo de burbuja es un método que funciona para ordenar algoritmos tanto de forma creciente como decreciente. Con este algoritmo, iteramos e intercambiamos pares hasta lograr mover el elemento más grande (o el más pequeño, dependiendo lo que se quiera lograr) al final de la lista. Una vez logrado esto, se procede a mover el segundo elemento más grande (o segundo más pequeño) al penúltimo puesto de la lista y así iremos “burbujeando” hasta lograr el arreglo ordenado que buscamos.



A continuación, una ilustración que demuestra cómo funciona el algoritmo en su primera iteración, siendo [5, 3, 8, 2, 6] el arreglo original. En este caso nuestro objetivo sería ordenar el algoritmo de forma ascendente. Como se mencionó anteriormente, nuestro

objetivo en la primera iteración es mover el elemento más grande al final. En este caso, nuestro elemento más grande es 8. En la siguiente iteración donde $i=1$ nuestra meta sería mover 6 (el segundo número mayor) al penúltimo puesto, el cual en este caso ya se encuentra ahí.

Ya tenemos una idea de cómo funciona el algoritmo, pero ¿cómo se vería en código JavaScript?

```
function ordenamientoBurbuja(arr) {
  const n = arr.length;
  for (let i = 0; i < n; i++) {
    for (let j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {

        // Intercambio de elementos
        const temp = arr[j]
        arr[j] = arr[j + 1]

        arr[j + 1] = temp
      }
    }
  }
  return arr;
}

const arreglo = [5, 3, 8, 2, 6];
const arregloOrdenado = ordenamientoBurbuja(arreglo);
console.log(arregloOrdenado); // Output: [2, 3, 5, 6, 8]
```

Es importante tener en cuenta que el algoritmo de ordenamiento burbuja no es eficiente para grandes conjuntos de datos, ya que su complejidad es cuadrática ($O(n^2)$). Sin embargo, es un algoritmo útil para aprender y comprender los conceptos básicos de los algoritmos de ordenamiento.

Recuerda que existen otros algoritmos más eficientes, como el Algoritmo de Ordenamiento Rápido (Quicksort), que se utiliza en situaciones donde se requiere un mayor rendimiento en la ordenación de grandes conjuntos de datos.

➤ Algoritmo de ordenamiento rápido (Quicksort)

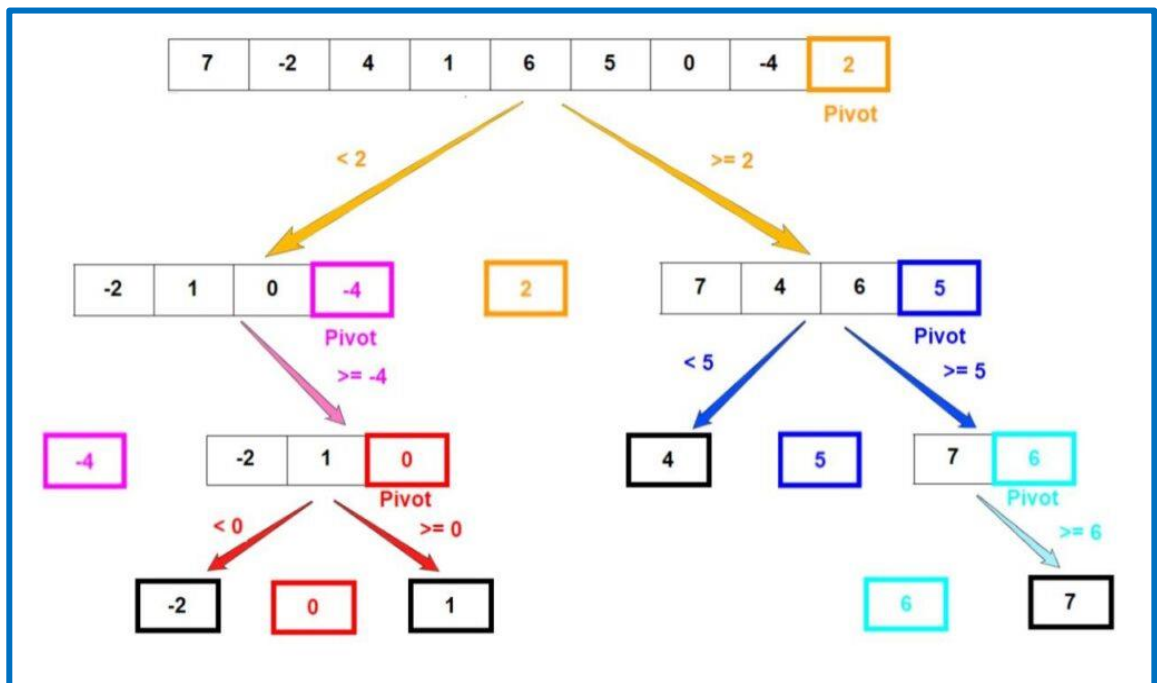
Utiliza el enfoque de "divide y vencerás" para ordenar la lista. Este algoritmo divide la lista en subconjuntos más pequeños y ordena cada subconjunto de manera recursiva. Es uno de los más eficientes y ampliamente utilizados para el ordenamiento. Aquí tienes un ejemplo de implementación:


```

function ordenamientoRapido(arr) {
  if (arr.length <= 1) {
    return arr;
  }
  const pivot = arr[arr.length - 1];
  // En muchos casos también se utiliza pivot = arr[0]
  let menores = [];
  let mayores = [];
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] < pivot) {
      menores.push(arr[i]);
    } else {
      mayores.push(arr[i]);
    }
  }
  return [...ordenamientoRapido(menores), pivot, ...ordenamientoRapido(mayores)];
}

const arreglo = [5, 3, 8, 2, 6];
const arregloOrdenado = ordenamientoRapido(arreglo);
console.log(arregloOrdenado); // Output: [2, 3, 5, 6, 8]

```



Ejemplo de ordenamiento rápido.

➤ Algoritmo de Ordenamiento Notation (Notation Sort)

También conocido como Notation Sort, es un algoritmo de ordenamiento poco convencional. A diferencia de otros algoritmos que se basan en comparaciones directas entre elementos, el Notation Sort utiliza una notación especial para determinar el orden de los elementos en la lista. Aunque no es un algoritmo práctico en términos

de eficiencia y rendimiento, es interesante explorar su funcionamiento y sus posibles usos en situaciones particulares.

El Algoritmo de Ordenamiento Notation se basa en la asignación de un valor numérico o una etiqueta a cada elemento de la lista a ordenar. Estos valores numéricos o etiquetas son generados de alguna manera específica, no necesariamente relacionada con el valor real de los elementos. Luego, se realiza un proceso de clasificación utilizando estos valores asignados, lo que resulta en una nueva ordenación de los elementos.

La idea detrás del algoritmo de ordenamiento Notation es que la asignación de valores o etiquetas no depende de las comparaciones directas entre los elementos, sino de una serie de reglas o criterios establecidos previamente. Estos criterios pueden ser arbitrarios o basados en algún aspecto particular de los elementos. Por ejemplo, se pueden asignar valores según el número de vocales en una cadena de texto o según la longitud de las palabras.

Veamos un ejemplo de implementación del Algoritmo de Ordenamiento Notation en JavaScript utilizando una asignación de valores basada en la longitud de las palabras:

```
function notationSort(arr) {
  const sortedIndices = arr
    .map((value, index) => index)
    .sort((a, b) => {
      const lengthA = arr[a].length;
      const lengthB = arr[b].length;
      return lengthA - lengthB;
    });

  const result = [];
  for (const index of sortedIndices) {
    result.push(arr[index]);
  }

  return result;
}

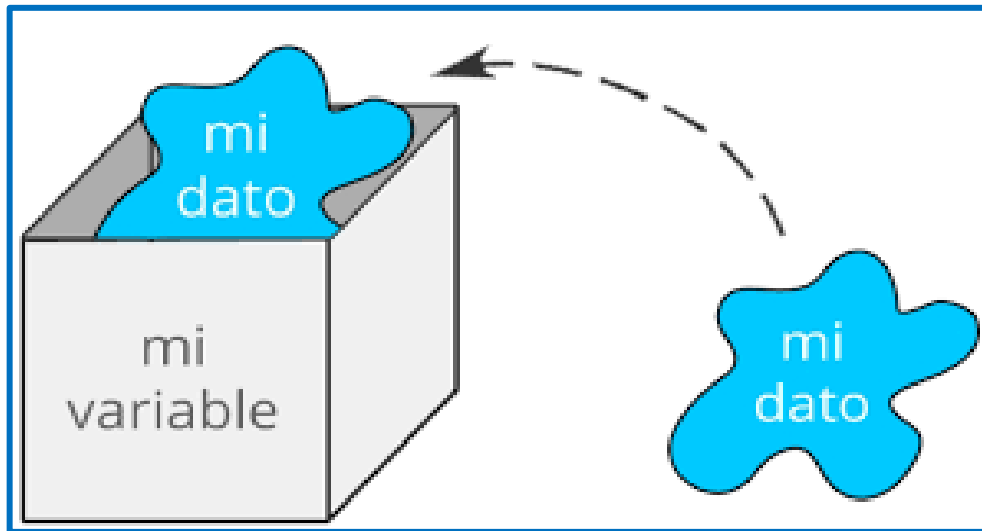
// Ejemplo de uso
const palabras = ["manzana", "perro", "gato", "banana"];
const palabrasOrdenadas = notationSort(palabras);
console.log(palabrasOrdenadas); // Output: ["gato", "perro", "banana", "manzana"]
```

En este ejemplo, tenemos un array de palabras desordenadas ["manzana", "perro", "gato", "banana"]. Aplicando el Algoritmo de

Ordenamiento Notación basado en la longitud de las palabras, obtendremos el array ordenado ["gato", "perro", "banana", "manzana"].

- **Variables**

Las variables son un concepto fundamental en cualquier lenguaje de programación. En JavaScript, puedes declarar variables usando las palabras clave `var`, `const` o `let`.



[Noción de una variable.](#)

- **¿Para qué se usan las variables en JavaScript?**

En el contexto de la programación, los datos son información que usamos en nuestros programas informáticos. Por ejemplo, tu nombre de usuario en Twitter es un dato. Gran parte de la programación se trata de manipular o mostrar datos. Para hacer esto, los programadores necesitan alguna manera de guardar y registrar datos. Vamos a demostrar esto con un ejemplo.

Primero abriremos nuestra consola de JavaScript. Para abrir tu consola de JavaScript en Chrome, puedes usar el atajo `Ctrl + Shift + J` en Windows y Linux. Para Mac, usa `Cmd + Option + J`.

- **Introduciendo variables en JavaScript**

Una analogía útil es pensar en las variables como etiquetas para nuestros valores. Piensa en un contenedor de arándanos con una etiqueta que dice 'arándanos'. En este ejemplo, la variable `arándanos`, señala hacia un valor, que son los mismos arándanos. Declaremos una variable, `edad`, y usemos el operador de asignación (signo igual) para asignar nuestro valor, `4`, a esta variable. Usaremos la palabra clave `var`.

```
var edad = 4
```

Las variables son la manera como los programadores le dan nombre a un valor para poder reusarlo, actualizarlo o simplemente registrarlo. Las variables se pueden usar para guardar cualquier tipo de dato en JavaScript. Ahora que hemos asignado este valor a la variable edad, podremos referirnos a este más adelante. Si escribes ahora la variable edad en tu consola, esta te devolverá el valor de 4.

- **¿Qué es el Scope?**

El ámbito (Scope) se refiere a los lugares dentro de nuestro código en donde las variables están disponibles para su uso. Cuando una variable tiene un ámbito global, significa que está disponible en cualquier lugar de tu programa. Veamos un ejemplo.

Toma el siguiente bloque de código y pégalo en tu consola.

```
var nombre = 'Madison'  
function imprimirNombre() {  
  console.log(nombre)  
}  
imprimirNombre()
```

Aquí hemos creado y llamado una función, imprimirNombre, que imprimirá el valor de la variable nombre, Madison. Verás eso impreso en tu consola.

Debido a que nuestra variable fue creada por fuera de la función, tiene alcance global. Esto significa que está disponible en cualquier parte de tu código, incluyendo dentro de cualquier función. Es por eso que la función, imprimirNombre, tiene acceso a la variable nombre.

Vamos ahora a crear una variable que tenga alcance de función. Esto significa que la variable solo se puede acceder dentro de la función en la que fue creada. Este siguiente ejemplo será muy similar al código anterior, pero con un posicionamiento diferente de la variable.

```
function imprimirAño() {  
  var año = 2020  
}  
console.log(año)
```

Ahora nos saldrá un error en la consola: año is not defined (año no está definida). Esto es porque la variable año tiene alcance de función. Es decir, solo existe dentro de la función dentro de la cuál fue creada. Al ejecutar console.log, hemos intentado acceder a la variable desde afuera de la función, donde no tenemos acceso.

Las variables con alcance de función son útiles a los programadores porque frecuentemente queremos crear variables que sólo sirvan o sean necesarias dentro de cierta función. Crear variables globales también nos puede generar errores o fallos.

- **Condicionales**

Hasta ahora hemos visto código que se ejecuta línea a línea, una detrás de otra. Pero a veces se hace necesario romper esa secuencia y crear ramas que nos permitan tomar diferentes caminos en el código dependiendo de ciertas condiciones.

```
if (condición){
    código a ejecutar si la condición se cumple;
} else if (condición 2){
    código a ejecutar si la condición 2 se cumple;
} else if (condición 3){
    código a ejecutar si la condición 3 se cumple;
} else {
    código a ejecutar si la condición 3 tampoco se cumple;
}
```

Estructura de condicionales con distintas variaciones.

Por ejemplo, imagina cómo podríamos hacer un programa que nos diga si un número es mayor o menor a diez. Si es mayor a 10 debería imprimir una cosa, pero si es menor debería imprimir otra.

A este concepto se le conoce como condicionales y su sintaxis es la siguiente:

```
if (<condición>) {
    // código que se ejecuta si se cumple la condición
}
```

a condición puede ser cualquier expresión que evalúe a verdadero (true) o falso (false). Crea un archivo llamado conditionals.jsy agrega el siguiente contenido:

```
if (true) {
    console.log("Hola Mundo");
}
```

Nota: Las líneas que terminan con un corchete ({o }) no se les agrega punto y coma (;).

Ejecuta el archivo desde la consola, deberías ver lo siguiente:


```
$ node conditionals.js
Hola Mundo
```

No importa cuantas veces ejecutes este archivo, el resultado siempre será el mismo.

Ahora probamos con falso (false) en vez de verdadero (true):

```
if (false) {
  console.log("Hola Mundo");
}
```

Ejecútalo. Esta vez nunca debería imprimir "Hola mundo", no importa cuantas veces lo ejecutes.

En vez de utilizar true o false como condición, podemos utilizar una expresión que evalúe a true o false.

```
if (1 == 1) {
  console.log("Hola Mundo");
}
```

El resultado al ejecutarlo debería ser:

```
$ node conditionals.js
Hola mundo
```

Pruebe ahora con $1 == 2$, $1 < 6$ y $8 < 6$ en la condición y fíjate que tenga sentido.

Ahora que ya sabes cómo funcionan los condicionales (muchos los llamamos los ifs) crea un programa en un archivo llamado number.js que imprime "El número es menor a 10" solo si el número que está almacenado en la variable num es menor a 10:

```
var num = 8;
if (num < 10) {
  console.log("El número es menor a 10");
}
```

Si lo ejecutas te debería aparecer lo siguiente:

```
$ node number.js
El número es menor a 10
```

○ De lo contrario

Lo único que necesitas para hacer condicionales es el if. Pero existen dos atajos que te van a permitir escribir código más corto.

El primer atajo es el else, que significa "de lo contrario" en inglés. El else nos permite definir el código que se debe ejecutar si el if no se cumple, es decir si la condición evalúa a falsa. La sintaxis es la siguiente:

```
if (<condición>) {
  // código que se ejecuta si se cumple la condición
} else {
  // código que se ejecuta si NO se cumple la condición
}
```

Podemos modificar el programa anterior, que nos dice si el número almacenado en la variable num es menor a 10, o si es mayor o igual, con un else.

```
var num = 8;
if (num < 10) {
  console.log("El número es menor a 10");
} else {
  console.log("El número es igual o mayor a 10");
}
```

Más corto y si lo ejecutas debería funcionar igual.

○ Condiciones anidadas

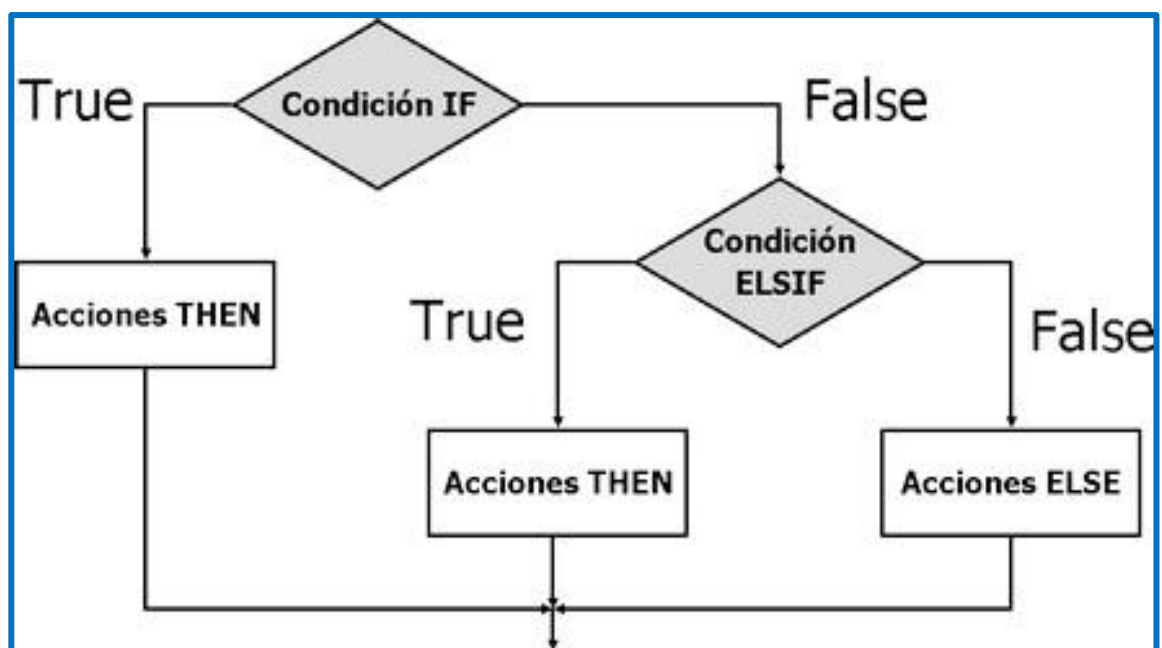


Diagrama de una estructura condicional anidada.

Ahora imagina que queremos modificar este programa para que en vez de imprimir "El número es igual o mayor a 10", imprima "El número es igual a 10" o "El número es mayor a 10" dependiendo si el número es igual a 10 o mayor a 10 respectivamente.

En JavaScript (y en la mayoría de lenguajes de programación) es posible anidar condicionales, así que una posible solución sería la siguiente:

```
var num = 8;
if (num < 10) {
  console.log("El número es menor a 10");
} else {
  if (num > 10) {
    console.log("El número es mayor a 10");
  } else {
    console.log("El número es igual a 10");
  }
}
```

Pruébalo con un número menor a 10, otro mayor a 10 y con 10. Te debería aparecer "El número es menor a 10", "El número es mayor a 10" y "El número es igual a 10" respectivamente.

➤ De lo contrario, si

En general, es preferible no tener que anidar condicionales porque son difíciles de leer y entender. Otro atajo que nos ofrece JavaScript para los condicionales es el `else if`, que significa "De lo contrario, si ..." en inglés. La sintaxis es la siguiente:

```
if (<primera condición>) {
  // código que se ejecuta si <primera condición> se cumple
} else if (<segunda condición>) {
  // código si <primera condición> NO se cumple, pero <segunda condición> se cumple
} else if (<tercera condición>) {
  // código si <primera condición> y <segunda condición> NO se cumplen, pero <tercera condición> sí se cumple
} else {
  // código si ninguna de las condiciones se cumple
}
```

Puedes definir tantos `else if` como desees. El `else` es opcional. Modifiquemos nuestro ejemplo anterior y en vez de utilizar condiciones anidadas, utilicemos **else if**:

```
var num = 8;
```

```

if (num < 10) {
  console.log("El número es menor a 10");
} else if (num > 10) {
  console.log("El número es mayor a 10");
} else {
  console.log("El número es igual a 10");
}

```

Lo más importante de entender en este código es que el programa sólo va a entrar a una de estas ramas. Por ningún motivo va a entrar a dos de ellas. Si la condición del primer if se cumple, el programa ejecuta el código que esté en ese bloque y después salta hasta después del else para continuar con el resto del programa o terminar.

Si la condición del primer if no se cumple, pero la del else if sí se cumple, el programa ejecuta el código de ese bloque y salta hasta después del else para continuar con el resto del programa o terminar.

○ Condiciones compuestas

Imagina que queremos escribir un programa que imprima "El número está entre 10 y 20" si el valor de una variable está efectivamente entre 10 y 20. ¿Cómo te imaginas que lo podríamos solucionar?

Una opción es usar condiciones anidadas, de esta forma:

```

var num = 15;
if (num >= 10) {
  if (num <= 20) {
    console.log("El número está entre 10 y 20");
  }
}

```

Sin embargo, cómo decíamos antes, leer condiciones anidadas es difícil y, en lo posible, es mejor evitarlas. En cambio, podemos utilizar los operadores lógicos y (&&) y ó (||) para crear condiciones compuestas.

El ejemplo anterior lo podemos mejorar con y:

```

var num = 15;
if (num >= 10 && num <= 20) {
  console.log("El número está entre 10 y 20");
}

```

Lo que estamos diciendo con este código es: si el número es mayor o igual a 10 y menor o igual 20 entonces imprima "El número está entre 10 y 20". Fíjate que a cada lado del && hay una expresión que evalúa a

verdadero o falso: `num >= 10` y `num <= 20`.

Imagina ahora que necesitamos escribir un programa que imprima "Excelente elección" cuando el valor de una variable sea "rojo" o "negro" únicamente:

```
var color = "negro";
if (color === "rojo" || color === "negro") {
  console.log("Excelente elección");
}
```

- **Bucles**

En JavaScript los bucles (loops) son utilizados para realizar tareas repetitivas con base en una condición. Las condiciones típicamente devuelven true (verdadero) o false(falso) al ser evaluados. El bucle continuará ejecutándose hasta que la condición devuelva false.

Los tres tipos más comunes de bucles son:

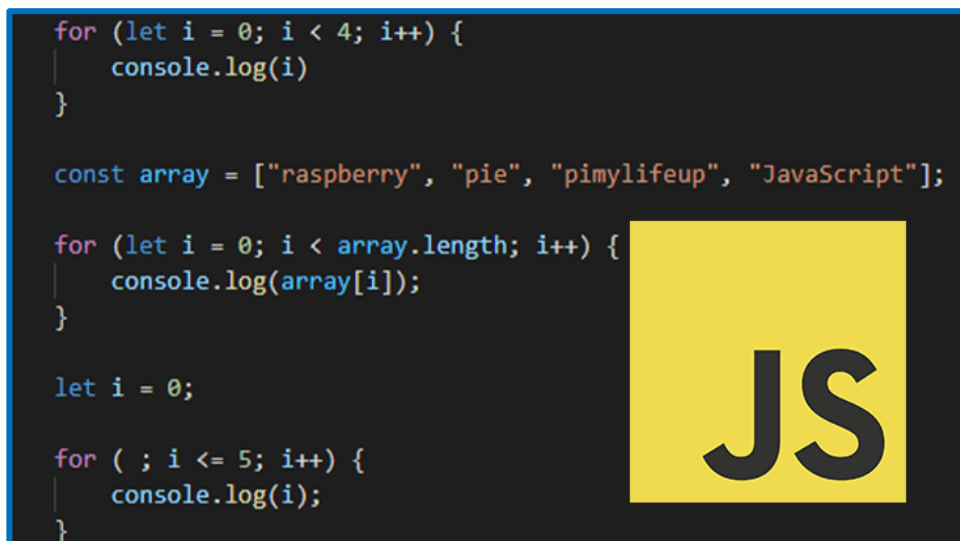
- for
- while
- do while

Veremos algunas variaciones:

- **for loop**

- **Sintaxis**

```
for ([inicializacion]; [condicion]; [expresion-final]) {
  // sentencias
}
```



```
for (let i = 0; i < 4; i++) {
  console.log(i)
}

const array = ["raspberry", "pie", "pimylifeup", "JavaScript"];

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

let i = 0;

for ( ; i <= 5; i++) {
  console.log(i);
}
```

[Ejemplo de for loop en Javascript.](#)

La sentencia for en javascript consiste de tres expresiones y una declaración:

➤ Descripción

- ✓ **Inicialización** - Sucede antes de la primera ejecución del bucle. Esta expresión es comúnmente utilizada para crear contadores. Las variables creadas tienen un alcance (scope) limitado al cuerpo del bucle. Una vez que el bucle ha terminado su ejecución las variables son destruidas.
- ✓ **Condición** - Expresión que es evaluada antes de la ejecución de cada iteración. Si se omite, esta expresión es evaluada como verdadera. Si devuelve true, la sentencia del cuerpo del bucle se ejecuta. Si devuelve false, el bucle se detiene.
- ✓ **Expresión-final** - Expresión que se ejecuta luego de cada iteración. Usualmente es utilizada para incrementar un contador. Pero también puede ser utilizada para decrementar el contador.
- ✓ **Sentencia o declaración** - Código que será ejecutado repetidamente por el bucle.

Cualquiera de estas tres expresiones o la sentencia del cuerpo del bucle pueden ser omitidas. Los bucles For son comúnmente utilizados para contar un cierto número de iteraciones para repetir una sentencia. Puedes utilizar una sentencia break para salir del bucle antes que la expresión de condición devuelva false.

➤ Complicaciones comunes

Exceder los límites del array. Cuando recorremos un array utilizando los índices es fácil exceder los límites (ej. intentar referenciar el 4º elemento de un array de 3 elementos).

```
// Esto ocasionará un error.
// Los límites del array serán excedidos.
var arr = [ 1, 2, 3 ];
for (var i = 0; i <= arr.length; i++) {
  console.log(arr[i]);
}
```

Resultado:

```
1
2
3
```

undefined

Hay dos maneras de arreglar este código. Que la condición sea `i < arr.length` o `i <= arr.length - 1`.

Ejemplos

Iterar a través de los enteros del 0-8

```
for (var i = 0; i < 9; i++) {
  console.log(i);
}
```

Resultado:

```
0
1
2
3
4
5
6
7
8
```

○ for...in loop

La declaración `for...in` itera sobre las propiedades enumerables de un objeto, en orden arbitrario. Para cada propiedad distinta, se pueden ejecutar sentencias.

```
for (variable in object) {
  ...
}
```

➤ Descripción

✓ **Variable:** Un nombre distinto es asignado a la variable en cada iteración.

✓ **Objeto:** Objeto cuyas propiedades enumerables (no de tipo `symbol`) son iteradas.

✓ Ejemplos

// Inicializar objeto.

```
a = { "a": "Athens", "b": "Belgrade", "c": "Cairo" }
```

// Iterar sobre las propiedades.

```
var s = ""
```

```

for (var key in a) {
    s += key + ": " + a[key];
    s += "<br />";
}
document.write (s);

```

// Resultado:

// a: Athens

// b: Belgrade

// c: Cairo

// Inicializar el array (arreglo).

```
var arr = new Array("cero", "uno", "dos");
```

// Agregar algunas propiedades expando al array.

```
arr["naranja"] = "fruta";
```

```
arr["zanahoria"] = "vegetal";
```

// Iterar sobre las propiedades y elementos.

```
var s = "";
```

```
for (var key in arr) {
```

```
    s += key + ": " + arr[key];
```

```
    s += "<br />";
```

```
}
```

```
document.write (s);
```

// Output:

// 0: cero

// 1: uno

// 2: dos

// naranja: fruta

// zanahoria: vegetal

// Forma eficiente de obtener los índices (keys) de un objeto usando la expresión dentro de las condiciones del bucle for-in

```
var myObj = {a: 1, b: 2, c:3}, myKeys = [], i=0;
```

```
for (myKeys[i++] in myObj);
```

```
document.write(myKeys);
```

//Resultado:

// a

// b

// c

- **Funciones**

Por otro lado, si ya conoces el concepto de función en programación, quizás te interese profundizar un poco en cómo se utilizan en el mundo de JavaScript. Las funciones son uno de los tipos de datos más importantes, ya que estamos continuamente utilizándolas a lo largo de nuestro código.

Y no, no me he equivocado ni he escrito mal el texto anterior. En JavaScript, las funciones pueden ser tipos de datos como un número o una cadena de texto.

```
typeof function () {}; // 'function'
```

Como puedes ver, si le pedimos a JavaScript que nos diga el tipo de dato de función, nos devuelve function, por lo tanto, es un tipo de dato reconocido. Vamos a analizar cómo podemos utilizar las funciones en JavaScript, ya que hay varias formas diferentes:

- **Formas de crear funciones**

Hay varias formas principales de crear funciones en JavaScript, aunque probablemente sólo conozcas alguna de ellas:

Ejemplo		Descripción
1	<code>function nombre(p1, p2...){}</code>	Mediante declaración (la más usada por principiantes)
2	<code>let nombre = function(p1, p2...){}</code>	Mediante expresión (la más habitual en programadores con experiencia)
3	<code>new Function(p1, p2..., code);</code>	Mediante un constructor de objeto (no recomendada)

- **Funciones por declaración**

```
// a la función le llegan
function ejecutarOrden(dato1, dato2 ){
  // declaración variables locales
  var datoLocal;

  // instrucciones
  datoLocal = dato1 + dato2;

  return datoLocal;
}
```

[Ejemplo de declaración de funciones.](#)

Probablemente, la forma más popular de estas tres, y a la que estaremos acostumbrados si venimos de otros lenguajes de programación, es la primera, a la creación de funciones por declaración. Esta forma permite declarar una función que existirá a lo largo de todo el código:

```
function saludar() {
  return "Hola";
}

saludar();    // 'Hola'
typeof saludar; // 'function'
```

De hecho, podríamos ejecutar la función saludar() incluso antes de haberla creado y funcionaría correctamente, ya que JavaScript primero busca las declaraciones de funciones y luego procesa el resto del código.

○ Funciones por expresión

Sin embargo, en JavaScript es muy habitual encontrarse códigos donde los programadores «guardan funciones» dentro de variables, para posteriormente «ejecutar dichas variables». Se trata de un enfoque diferente, creación de funciones por expresión, que fundamentalmente, hacen exactamente lo mismo (con algunos matices diferentes):

```
const saludo = function saludar() {
  return "Hola";
};
saludo(); // 'Hola'
```

Con este nuevo enfoque, estamos creando una función en el interior de una variable, lo que nos permitirá posteriormente ejecutar la variable (como si fuera una función, que de hecho lo es, porque es lo que contiene).

Observa también que el nombre de la función (saludar) pasa a ser inútil, ya que si intentamos ejecutar saludar() nos dirá que no existe y si intentamos ejecutar saludo() funciona correctamente. ¿Qué ha pasado? Ahora el nombre de la variable pasa a ser el «nombre de la función», mientras que el anterior nombre de la función desaparece y se omite, creando un concepto llamado funciones anónimas (o funciones lambda), que retomaremos más adelante.

La diferencia fundamental entre las funciones por declaración y las funciones por expresión es que estas últimas sólo están disponibles a partir de la inicialización de la variable. Si «ejecutamos la variable» antes de declararla, nos dará un error.

- **Funciones como objetos**

Como curiosidad, debes saber que se pueden declarar funciones como si fueran objetos. Sin embargo, es un enfoque que no se suele utilizar en el mundo real, ya que es incómodo, poco práctico y muy verboso:

```
const saludar = new Function("return 'Hola';");  
saludar(); // 'Hola'
```

Simplemente es interesante saberlo para darse cuenta que en JavaScript todo pueden ser objetos.

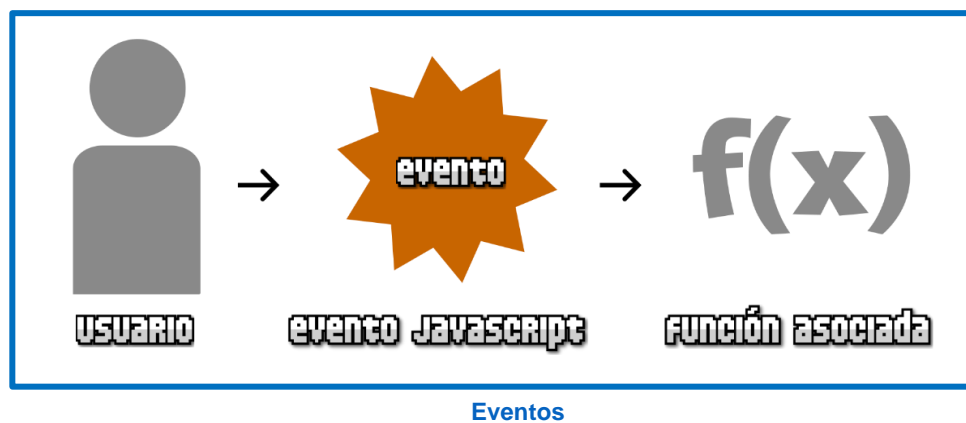
- **Eventos**

En JavaScript existe un concepto llamado evento, que no es más que una notificación de que alguna característica interesante acaba de ocurrir, generalmente relacionada con el usuario que navega por la página.

Dichas características pueden ser muy variadas:

- Click de ratón del usuario sobre un elemento de la página
- Pulsación de una tecla específica del teclado
- Reproducción de un archivo de audio/video
- Scroll de ratón sobre un elemento de la página
- El usuario ha activado la opción «Imprimir página»

Como desarrolladores, nuestro objetivo es preparar nuestro código para que cuando ocurra un determinado evento, se lleve a cabo una funcionalidad asociada. De esta forma, podemos preparar nuestra página o aplicación para que cuando ocurran ciertos eventos (que no podemos predecir de otra forma), reaccionen a ellos:



Uno de los eventos más comunes, es el evento click, que es el que se produce cuando el usuario hace click con el ratón en un elemento de la página. Vamos a utilizar este evento a modo de ejemplo en las siguientes secciones de la página, pero recuerda que hay muchos tipos de eventos diferentes.

○ Formas de manejar eventos

Existen varias formas alternativas de manejar eventos en JavaScript. Vamos a ver cada una de ellas por separado, con sus particularidades, pero antes hagamos un pequeño resumen:

Forma	Ejemplo	Artículo en profundidad
Mediante atributos HTML	<code><button onClick="..."></button></code>	<u>Eventos JS desde atributos HTML</u>
Mediante propiedades JavaScript	<code>.onclick = function() { ... }</code>	<u>Eventos JS desde propiedades JavaScript</u>
Mediante <code>addEventListener()</code>	<code>.addEventListener("click", ...)</code>	<u>Eventos JS desde listeners</u>

Cada una de estas opciones se puede utilizar para gestionar eventos en JavaScript de forma equivalente, pero cada una de ellas tiene sus ventajas y sus desventajas. En los siguientes apartados veremos detalladamente sus características, pero por norma general, lo aconsejable es utilizar la última, los listeners, ya que son las más potentes y flexibles.

2. MANIPULACIÓN DEL DOM EN JAVASCRIPT

El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. En resumen, es la representación de la página web en la memoria del navegador, a la que podemos acceder a través de JavaScript. El DOM es un árbol donde cada nodo es un objeto con todas sus propiedades y métodos que nos permiten modificarlo. Estas son algunas funciones que nos permiten acceder y modificar los elementos del DOM:

- **Acceso a elementos del DOM**

```
// Obtiene un elemento por id
document.getElementById('someid');
```

```
// Obtiene una lista con los elementos que tienen esa clase
document.getElementsByClassName('someclass');
```

```
// Obtiene una HTMLCollection con todos los elementos 'li'
document.getElementsByTagName('li');
```

```
// Devuelve el primer elemento del documento que cumpla la selección
// (la notación es como en CSS)
document.querySelector('.someclass');
```

```
// Devuelve una lista de elementos que cumplen con la selección (notación
como en CSS)
document.querySelectorAll('div.note, div.alert');
```

- **Acceder a hijos/padres de un elemento**

```
// Obtener los hijos de un elemento
var elem = document.getElementById('someid');
var hijos = elem.childNodes;
```

```
// Su nodo padre
var padre = elem.parentNode;
```

- **Crear nuevos elementos en el DOM**

```
// Para crear elementos llamamos a createElement con el nombre del
elemento
var nuevoH1 = document.createElement('h1');
var nuevoParrafo = document.createElement('p');
```

```
// Crear nodos de texto para un elemento
var textoH1 = document.createTextNode('¡Hola mundo!');
var textoParrafo = document.createTextNode('lorem ipsum...');
```

```
// Añadir el texto a los elementos
nuevoH1.appendChild(textoH1);
nuevoParrafo.appendChild(textoParrafo);
```

```
// también podemos asignar directamente el valor a la propiedad innerHTML
nuevoH1.innerHTML = textoH1;
nuevoParrafo.innerHTML = textoParrafo;
```

```
// los elementos estarían listos para añadirlos al DOM, ahora mismo solo
existen en memoria, pero no serán visibles hasta que no los añadamos a
un elemento del DOM
```

- **Añadir elementos al DOM**

```
// seleccionamos un elemento
var cabecera = document.getElementById('cabecera');
```

```
// Añadir elementos hijos a un elemento
cabecera.appendChild(nuevoH1);
```

```
cabecera.appendChild(nuevoParrafo);
```

```
// También podemos añadir elementos ANTES del elemento seleccionado
```

```
// Tomamos el padre
```

```
var padre = cabecera.parentNode;
```

```
// Insertamos el h1 antes de la cabecera
```

```
padre.insertBefore(nuevoH1, cabecera);
```

También podemos añadir directamente un trozo de HTML antes o después de un elemento del DOM, supongamos que tenemos estos elementos en la página:

```
<div id='box1'>
  <p>aquí algo de texto</p>
</div>
```

```
<div id='box2'>
  <p>otro parrafo bla bla bla</p>
</div>
```

Podemos hacer:

```
var box2 = document.getElementById('box2');
box2.insertAdjacentHTML('beforebegin', '<div><p>un parrafo
nuevo.</p></div>');
```

```
// beforebegin - El nuevo HTML es insertado justo antes del elemento, a la
misma altura (hermano).
```

```
// afterbegin - El nuevo HTML se inserta dentro del elemento, antes del
primer hijo.
```

```
// beforeend - El nuevo HTML se inserta dentro del elemento, después del
último hijo.
```

```
// afterend - El nuevo HTML es insertado justo después del elemento, a la
misma altura (hermano).
```

Añadir/eliminar/modificar Clases

```
// Tomamos un elemento
```

```
var cabecera = document.getElementById('cabecera');
```

```
// elimina una clase del elemento
```

```
cabecera.classList.remove('foo');
```

```
// Añade una clase si no existe
```

```
cabecera.classList.add('otra');
```

```
// añade o elimina varias clases a la vez
cabecera.classList.add('foo', 'bar');
cabecera.classList.remove('foo', 'bar');

// Si la clase existe la elimina, si no existe, la crea
cabecera.classList.toggle('visible');

// Devuelve true si el elemento contiene esa clase
cabecera.classList.contains('foo');
```

3. FUNDAMENTOS DE NODE PACKAGE MANAGER (NPM)

NPM se puede considerar como las siglas de Node Package Manager, es decir, gestor de paquetes de NodeJS, un entorno de ejecución multiplataforma para ejecutar JavaScript no sólo en un navegador web (como se concibió originalmente) sino fuera de él, y poder utilizarlo en sistemas de escritorio o servidores web.



[Logo de Node Package Manager.](#)

Este gestor de paquetes (muy similar al concepto de apt-get en GNU/Linux), nos permitirá instalar de forma muy sencilla y automática paquetes JavaScript (tanto de Node como JavaScript para el navegador) para poder utilizarlos y mantenerlos en los proyectos o sistemas que utilicemos.

Para comenzar, necesitaremos instalar NodeJS en su versión LTS (recomendada) o en su última versión (experimental) si quieres tener las últimas novedades. Al instalar Node, se instalará automáticamente su gestor de paquetes y algunas otras utilidades interesantes que necesitaremos. En primer lugar, vamos a comprobar si tenemos NodeJS/NPM instalado y que versión tenemos:

```
# Muestra la versión de Node
$ node --version
# Muestra la versión de NPM
$ npm --version
```

En el caso de que no tengamos node instalado en nuestro sistema, se nos mostrará un mensaje de error como node: command not found o similar, en cuyo caso deberemos proceder a instalarlo (necesitaremos tener privilegios de root o permisos de sudo).

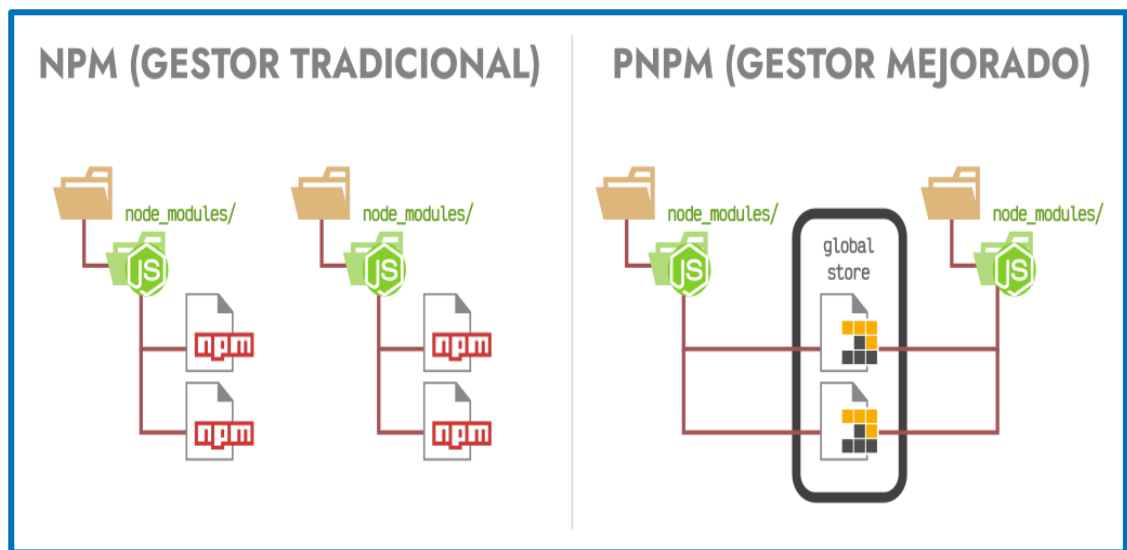
En el caso de tener una versión muy antigua, también podemos realizar los pasos que veremos a continuación y aprovechar para actualizarlo.

- **Instalación de Node/NPM**

Existen varias formas de instalar NodeJS. En las siguientes líneas se explica la instalación oficial de NodeJS. Sin embargo, no la recomiendo ya que suelen surgir problemas de instalación de paquetes y permisos que son complicados de resolver. En su lugar, recomiendo altamente realizar la instalación de NodeJS mediante PNPM o la instalación de NodeJS mediante NVM, ya que simplifican y automatizan la instalación y permite el cambio de versiones de Node de forma cómoda y rápida, algo que no es posible mediante la forma oficial.

- **Instalación de NodeJS mediante PNPM (recomendada)**

Las siglas de pnpm significan performant node package manager, es decir, gestor de paquetes de node de alto rendimiento. Tradicionalmente, npm siempre ha gestionado la instalación de paquetes de una forma particular, mediante la cual gasta demasiado tiempo y espacio en disco.



[NPM](#)

En el caso de pnpm, se utiliza una forma alternativa, donde se gestionan los paquetes mediante un almacén compartido, ahorrando espacio en disco y realizando una instalación de paquetes mucho más rápida.

- **Instalación de PNPM**

Para instalar pnpm, simplemente necesitaremos tener instalado curl en nuestro sistema (en caso contrario, ejecutamos el primer comando de los siguientes) para descargar el script de instalación de pnpm e instalarlo:

```
$ sudo apt install curl  
$ curl -fsSL https://get.pnpm.io/install.sh | sh -
```

Si en la instalación nos da el error `ERR_PNPM_UNKNOWN_SHELL Could not infer shell type` es porque nuestra terminal no tiene establecida la terminal por defecto. Esto se puede solucionar fácilmente, escribiendo el comando `export SHELL=/bin/bash`.

Finalmente, hacemos un `source ~/.bashrc` (o simplemente, cerramos y volvemos a abrir la terminal, para obligar a cargar los cambios). Si todo ha ido bien, deberíamos tener instalada la última versión de pnpm. Lo podemos comprobar con el siguiente comando:

```
$ pnpm --version
```

Sin embargo, aún no tenemos Node instalado. Vamos a ello.

- **Instalación de Node con PNPM**

Una vez instalado pnpm, vamos a por la instalación de Node. Podemos ver que versiones de Node son las actuales, entrando a la web oficial de NodeJS o simplemente, ejecutando el siguiente comando en una terminal:

```
$ pnpm env list --remote
```

```
[...]  
20.8.0  
20.8.1  
20.9.0  
20.10.0  
21.0.0  
21.1.0  
21.2.0  
21.3.0  
21.4.0  
21.5.0
```

Observa que el parámetro `--remote` es necesario para obtener las versiones disponibles desde la web de Node. Si lo omitimos, lo que obtendremos serán las versiones de node instaladas en nuestro sistema:


```
$ pnpm env list
```

Si examinas la página oficial de Node, como podrás ver, normalmente aparecen dos versiones:

- La versión estable (recomendada): También llamada LTS.
- La versión experimental: Con las últimas novedades.

Mi recomendación es instalar la primera de ellas. Esto puede hacerse directamente sin necesidad de ninguna comprobación de versiones, ejecutando el siguiente comando:

```
$ pnpm env use --global lts
```

Esto descargará la versión LTS de Node y la activará como versión por defecto. Puedes comprobar que todo ha ido bien, simplemente ejecutando los siguientes comandos para comprobar que Node ya está disponible en nuestro sistema.

```
$ node --version
```

```
$ npm --version
```

Si todo ha ido bien, debería darnos la versión disponible.

• Cambiar versiones con PNPM

Lo que hemos visto es la versión rápida para instalar Node. Sin embargo, puede que sea posible que queramos ir cambiando de versión de Node según los proyectos en los que estemos trabajando. Para ello, podemos hacer uso de los siguientes comandos.

Por ejemplo, mediante `pnpm env add --global` podemos descargarnos versiones concretas de Node. Sin embargo, sólo las descargará y no las activará para utilizar. Si queremos que también se activen, ejecutaremos el comando `pnpm env use --global`. Ambos comandos, siempre seguidos de la versión a instalar:

```
$ pnpm env add --global 21.5.0
```

```
$ pnpm env use --global 21.5.0
```

Si en algún momento queremos desinstalar alguna versión de nuestro sistema, simplemente utilizaremos `pnpm env remove`:

```
$ pnpm env remove --global 21.5.0
```

Con esto, deberíamos poder trabajar con node y pnpm en las versiones que necesitamos. A medida que transcurra el tiempo, necesitaremos actualizar

pnpm a las nuevas versiones que vayan apareciendo. Para ello, simplemente auto actualizáremos con pnpm:

```
$ pnpm install -g pnpm
```

Esto actualizará la versión de pnpm a la última disponible.

4. FUNDAMENTOS DE PROGRAMACIÓN WEB

- **Frontend**

El frontend o «desarrollo del lado del cliente» se refiere a la práctica de producir HTML, CSS y JavaScript. Estos tres elementos se encargan de dar forma a la parte frontal de un sitio web o aplicación. Esto incluye los fondos, colores, texto, animaciones o efectos. Precisamente de ahí proviene el nombre de «desarrollo del lado del cliente», pues con el frontend se puede construir por completo lo que los usuarios perciben al explorar un sitio y con el que pueden interactuar.

- **Para qué sirve el frontend**

El frontend sirve para realizar la interfaz de un sitio web, desde su estructura hasta los estilos, como pueden ser la definición de los colores, texturas, tipografías, secciones, entre otros. Su uso es determinante para que el usuario tenga una buena experiencia dentro del sitio o aplicación.

- **Elementos del frontend**

- **Estructuras de navegación:** Este elemento se refiere al orden en que se organizan las diferentes páginas de un sitio web y a los componentes que se vinculan entre sí para realizar diferentes funciones dentro del sitio.
 - **Layout:** También nombrado como diseño de página, se refiere a todos los componentes de la página web, por ejemplo: ubicación del menú, botones, footer; todo lo necesario para que un sitio sea útil y fácil de navegar.
 - **Contenido web:** Todo aquello que brinde información relevante o interesante para los usuarios. Es importante destacar que el contenido no tiene que ser necesariamente texto, puede incluir sonido o materiales interactivos.
 - **Imágenes:** Todos los recursos visuales ayudan a aumentar el interés de los usuarios. Esto también puede incluir videos, animaciones, mapas, gráficas, infografías, GIFs, ilustraciones, diagramas, etc.

- **Logotipo:** Para que un sitio web tenga mayor identidad es vital que contenga el logotipo que represente a la marca o empresa.
- **Diseño gráfico:** Este elemento engloba todo lo relacionado con cómo se ve el sitio web y su apariencia: colores, formas, tipografías, tamaños, etc.

- **Ejemplos de aplicación del frontend**

Como ya lo mencionamos, el frontend son todos los elementos y componentes visibles para los usuarios, y utilizan lenguajes de diseño como CSS, HTML y JavaScript. Algunos ejemplos de frontend son los siguientes:

- Optimización de motores de búsqueda (SEO).
- Accesibilidad (reconocimiento de voz, conversión de texto a voz).
- Funcionalidad en todos los navegadores y tamaños de pantalla (computadoras de escritorio, teléfonos móviles y tablets).
- Velocidad (cuanto más rápido cargue el sitio, mejor).
- Rendimiento del sitio web por medio de la limpieza del código.

Ahora que ya conoces qué es el frontend y para qué sirve, te explicaremos el aspecto interior de la construcción de un sitio o aplicación web; nos referimos al backend.

- **Backend**

El backend es el encargado de procesar toda la información que alimenta a un frontend. Se compone de marcos, bases de datos o códigos. Para que un sitio web o aplicación opere efectivamente, se requiere mucha información y datos que se almacenan en «la parte trasera» de un sistema informático. En oposición al frontend, el usuario no puede ver o acceder a esta información.

- **Para qué sirve el backend**

El backend son todos los códigos ocultos que sirven para que una página web o aplicación funcione correctamente. Además, de su estructura y organización depende la experiencia de usuario. De igual forma, el backend se encarga de optimizar otros elementos y recursos como la seguridad y privacidad en un sitio web o aplicación.

- **Elementos del backend**

- El backend se constituye por lenguajes de programación como PHP, Python y C++ y frameworks.
- Los servidores controlan cómo los usuarios acceden a los archivos.
- Las bases de datos son colecciones de datos organizadas y estructuradas.
- La seguridad es uno de los elementos más importantes dentro de un sitio web, pues garantiza que los visitantes y su información estén seguros. Esto también incluye evitar, en lo posible, ciberataques.

- **Ejemplos de aplicación del backend**


Algunos ejemplos para terminar de comprender el término son los siguientes:

- **Inicio de sesión:** Cuando una persona accede a un sitio web o aplicación utiliza un correo electrónico y contraseña, esta información es validada y resguardada por el servidor, que consulta su base de datos y así identifica y permite el acceso al usuario.
- **Carrito de compras:** Este elemento permite la compra de productos en línea y sirve para facilitar la selección de diferentes productos o servicios que algún usuario desee comprar.
- **Cookies:** Muchos sitios utilizan cookies para realizar un seguimiento de aquello que los usuarios vieron anteriormente, lo que les permite sugerir otros contenidos (o productos) de interés.
- **CMS:** Un sistema de gestión de contenidos permite al propietario de un sitio web actualizar la información sin tener que modificar el código HTML.
- **Formularios de contacto:** Si un visitante del sitio web se interesa por recibir más información o ponerse en contacto, se debe contar con un elemento que sea capaz de vincular al usuario con la empresa.

- **Diferencias entre frontend y backend**

Por sus aplicaciones y características podemos afirmar que el frontend comprende todas las acciones relacionadas con el diseño de experiencia que tendrá un visitante a una página web, mientras que el backend se refiere a la estructuración del sitio y la programación de sus funcionalidades principales.

Para que sea más sencillo reconocer las diferencias entre estos dos aspectos de la programación de un sitio web te dejamos la siguiente tabla comparativa:

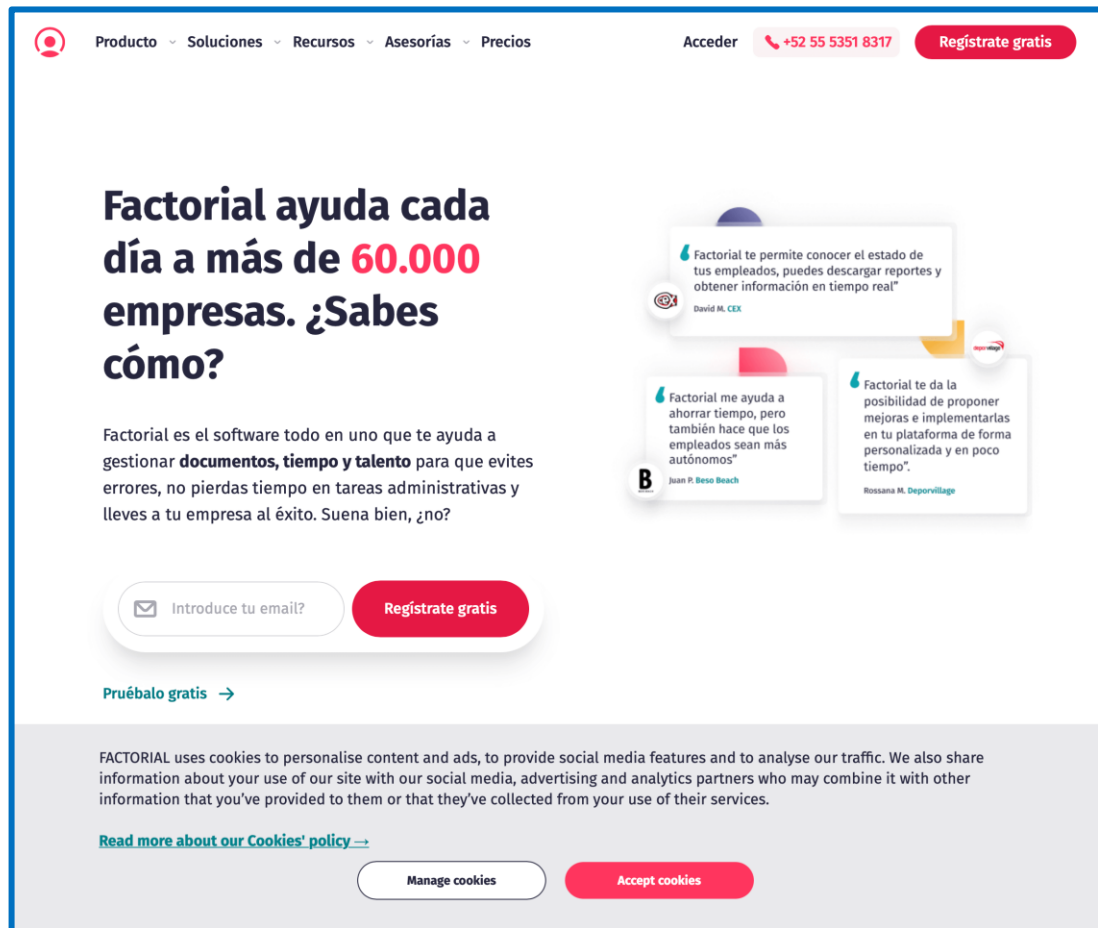


Frontend	Backend
Se diseña en lenguajes como HTML, CSS y JavaScript	Se programa en lenguajes como PHP, Python y C++
Prioriza la experiencia del usuario	Prioriza las necesidades de la marca
Centrado en asegurar una buena experiencia de navegación	Centrado en asegurar un buen desempeño del sitio
Optimiza la experiencia desde los navegadores	Optimiza la experiencia desde los servidores
Sus resultados definen el estilo del sitio y siempre son visibles	Sus resultados dan sostén al sitio, pero no son accesibles al usuario
Recorre al diseño de contenidos como textos, imágenes o multimedia	Requiere el diseño y uso de herramientas como bases de datos y hardware
Requiere conocimientos de diseño	Precisa de conocimientos de lógica
Sus profesionales necesitan una amplia capacidad creativa	Sus profesionales necesitan una gran capacidad analítica

Diferencias entre Frontend y Backend

○ **Cómo funcionan el frontend y backend en una página real**

Para ilustrar mejor los aspectos que hemos listado hasta ahora, veamos un ejemplo de un sitio real. Hemos elegido la página de inicio de Factorial, empresa de software de Recursos Humanos y uno de nuestros clientes.



Funcionamiento Frontend y Backend

Como puedes ver, en esta página hay una importante cantidad de elementos: un menú, iconos, imágenes, botones de acción, formularios y avisos de uso de información. ¿Logras identificar cuáles dependen del desarrollo backend y el frontend?

Esa pregunta quizás es difícil de responder, ya que todos los elementos que ves en un sitio tienen algo de desarrollo backend y algo de desarrollo frontend. Recuerda que el backend es el cerebro de tu página, por lo que no podrías ver los contenidos sin él. Pero veamos los elementos con más detalle.

○ Elementos frontend

- **Colores:** uno de los elementos que mejor definen la identidad de marca de Factorial son sus colores. La elección de un rojo intenso que contrasta con el blanco del fondo depende de un diseñador frontend.
- **Logotipo:** el logotipo de Factorial presenta una minimalista sombra de una persona. Este elemento fue creado por diseñadores e integrado en el sitio por un desarrollador web.
- **Imagen:** en la página podemos ver una imagen con reseñas de algunos clientes satisfechos. La disposición y tamaño de esta imagen

es definida en el frontend.

- **Botones de acción:** en la pantalla hay un botón de registro y uno de envío de formulario. La ubicación, el diseño y las opciones de estos elementos también se definen en frontend.
- **Información:** la inclusión de texto es responsabilidad del diseñador web.
- **Necesitas conocimientos de frontend y backend para crear un sitio web**

En términos simples podemos afirmar que no es necesario tener conocimientos de frontend y backend para crear sitios web. Aunque es común que para desarrollar un sitio o aplicación móvil haya encargados de cada uno de estos aspectos. Lo que sí es necesario es que ambos gestores trabajen en colaboración para generar una plataforma realmente funcional, que se adapte tanto a los intereses de la marca como de los usuarios.

Actualmente existen desarrolladores con experiencia y habilidades para cumplir con ambas partes de la ecuación. Estos expertos, conocidos como desarrolladores full stack, son capaces de gestionar adecuadamente todos los aspectos relevantes a la programación, diseño y estructura del sitio.

Sin embargo, también existen opciones de creación de sitios web realmente impactantes para aquellos que no dominan la programación, el diseño o el desarrollo web. Por ejemplo, el creador de sitios web gratuito de HubSpot permite que cualquier persona diseñe y gestione su propio sitio.

Si cuentas con un programador en tu equipo, podrás aprovechar más esta herramienta para crear mejores experiencias de usuario gracias a sus funciones, por ejemplo: páginas personalizadas, pruebas adaptativas para mantener el mejor desempeño y obtención de informes a fin de analizar el impacto del sitio y hacer los cambios necesarios, según el comportamiento de los visitantes.

Ten en cuenta que siempre es deseable estar informado y conocer los componentes, tanto exteriores como interiores, de un sitio web para lograr un diseño agradable y efectivo.

Ahora ya sabes qué es el frontend y el backend, así como algunos ejemplos de cómo pueden aplicarse para beneficio de tu empresa.

5. BASE DE DATOS Y CONSULTAS SQL

La integración de SQL en JavaScript es posible utilizar gracias a librerías externas como Node-SQLite o MySQL. Estas librerías permiten conectarse a una base de datos y ejecutar consultas SQL desde el código JavaScript.

Por ejemplo, para conectarse a una base de datos en MySQL desde JavaScript, se puede utilizar la librería MySQL:

```
const mysql = require('mysql');
const conexion = mysql.createConnection({
  host: 'localhost',
  user: 'usuario',
  password: 'contraseña',
  database: 'nombre_base_datos'
});

conexion.connect((error) => {
  if (error) throw error;
  console.log('Conexión exitosa a la base de datos');
});
```

Una vez que se establece la conexión, se pueden ejecutar consultas SQL desde el código JavaScript. Por ejemplo, para seleccionar todos los registros de una tabla llamada «usuarios»:

```
conexion.query('SELECT * FROM usuarios', (error, resultados) => {
  if (error) throw error;
  console.log('Registros de la tabla "usuarios":', resultados);
});
```

También es posible realizar operaciones de inserción, actualización y eliminación de datos utilizando sentencias SQL en el código JavaScript. Por ejemplo, para insertar un nuevo registro en la tabla «usuarios»:

```
conexion.query('INSERT INTO usuarios (nombre, correo) VALUES (?, ?)',
['Juan', 'juan@example.com'], (error) => {
  if (error) throw error;
  console.log('Registro insertado correctamente');
});
```

TAREA N° 03

Usa tecnología frontend con JavaScript.

1. NODE JS

Node.js es un entorno de ejecución de JavaScript de código abierto y multiplataforma que permite ejecutar código JavaScript fuera de un navegador web. Construido sobre el motor V8 de Chrome, Node.js utiliza un modelo de E/S no bloqueante y orientado a eventos, lo que lo hace ideal para aplicaciones de alta escalabilidad y en tiempo real.



[Logo de Node.js](https://nodejs.org/)

- **Ventajas de utilizar Node.js**
 - **JavaScript unificado:** Al utilizar JavaScript tanto en el front-end como en el back-end, los desarrolladores pueden compartir código y conocimientos entre ambos niveles, lo que reduce la curva de aprendizaje y aumenta la productividad.
 - **Alto rendimiento:** El modelo de E/S no bloqueante de Node.js permite manejar un gran número de conexiones simultáneas de manera eficiente, lo que lo hace ideal para aplicaciones de alta carga.
 - **Gran ecosistema:** Node.js cuenta con un ecosistema de paquetes (npm) extremadamente amplio y activo, lo que significa que hay una solución para casi cualquier problema de desarrollo.
 - **Fácil de aprender:** Si ya conoces JavaScript, aprender Node.js será relativamente sencillo.

- **Desarrollo full-stack:** Node.js permite a los desarrolladores crear aplicaciones web completas utilizando un único lenguaje, lo que simplifica el desarrollo y la gestión de proyectos.

Aplicaciones en tiempo real: Node.js es ideal para crear aplicaciones en tiempo real, como chat, juegos y aplicaciones de colaboración, gracias a su modelo de E/S no bloqueante y a bibliotecas como Socket.IO.

- **Casos de uso de Node.js**

- **APIs REST:** Node.js es ampliamente utilizado para crear APIs RESTful que sirven como la base de muchas aplicaciones web modernas.
- **Aplicaciones de una sola página (SPA):** Node.js puede servir como el back-end para SPAs, proporcionando los datos necesarios para renderizar la interfaz de usuario en el navegador.
- **Aplicaciones de tiempo real:** Node.js es ideal para aplicaciones que requieren una actualización constante de datos, como chat, juegos y aplicaciones de colaboración.
- **Herramientas de línea de comandos:** Node.js se puede utilizar para crear herramientas de línea de comandos personalizadas.
- **Microservicios:** Node.js es una excelente opción para construir arquitecturas de microservicios, ya que permite crear servicios pequeños y escalables.

- **Comparación con otros frameworks de back-end**

A diferencia de otros frameworks de back-end como Ruby on Rails o Django, Node.js ofrece una mayor flexibilidad y personalización. Sin embargo, esto también significa que los desarrolladores tienen más libertad para tomar decisiones arquitectónicas, lo que puede llevar a un mayor tiempo de desarrollo si no se planifica cuidadosamente.

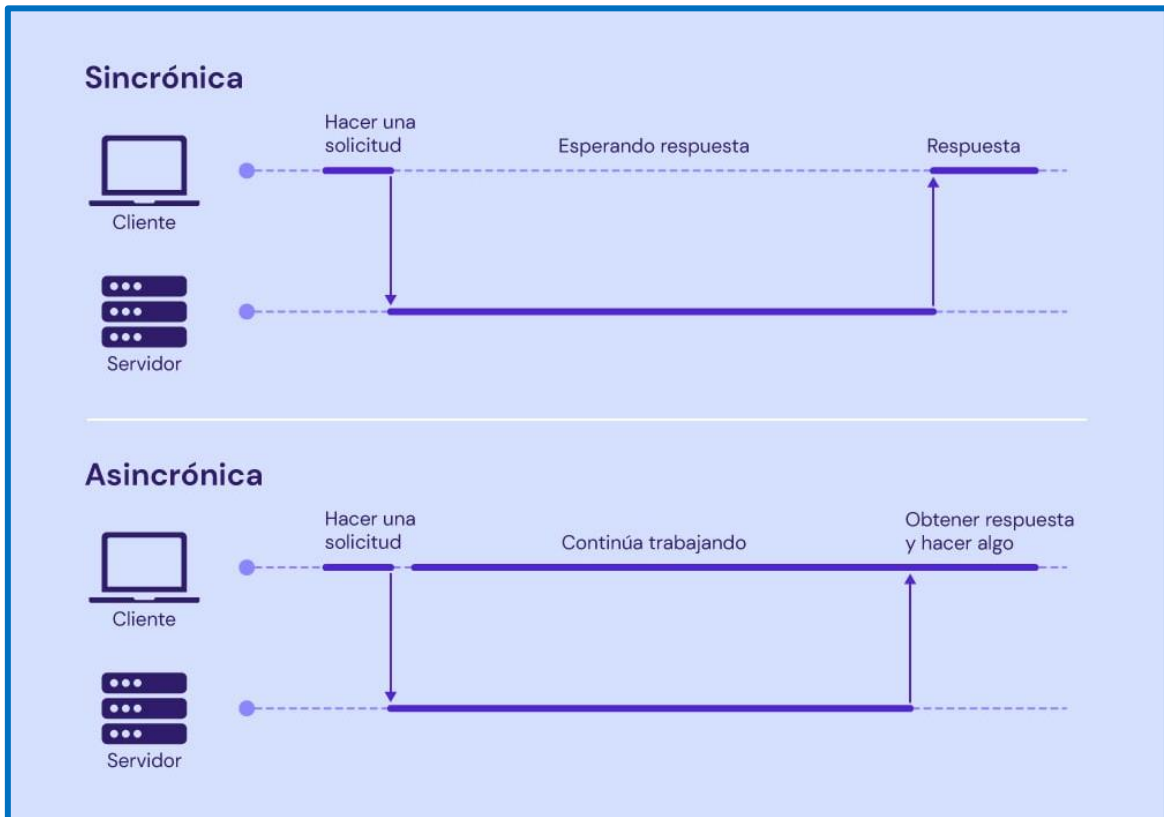
- **Arquitectura asincrónica**

Como se ha comentado anteriormente, el hilo de Node.js no espera una respuesta y pasa a procesar una petición posterior. En la arquitectura asincrónica, el bucle de eventos opera de forma dirigida por eventos. Una vez que ha recibido una respuesta de la llamada anterior a la API, pone la respuesta en la cola de eventos.

El bucle de eventos terminará todas las peticiones anteriores y actuales antes de ejecutar una función de devolución de llamada para enviar la respuesta del servidor al cliente.

Dado que Node.js utiliza un bucle de eventos de un solo hilo, puede atender varias peticiones simultáneamente con tiempos de ejecución más rápidos y un menor uso de recursos.

En comparación, el software de arquitectura sincrónica realiza una tarea a la vez. Por lo tanto, el bucle de eventos sólo pasará a la siguiente si la tarea anterior ha terminado.



Arquitectura asincrónica

- **Casos de uso de Node.js**

Node.js es una buena solución para realizar tareas intensivas de datos o análisis en tiempo real, ya que tiene una arquitectura asincrónica y características de E/S sin bloqueo. Algunos casos de uso populares incluyen:

- **Chat en tiempo real:** Node.js puede trabajar con programas de uso intensivo de datos, como las aplicaciones de chat, ya que maneja las tareas de E/S con eficacia. Utiliza tecnología push sobre sockets web, lo que permite la comunicación bidireccional entre servidores y clientes. Como resultado, el servidor no necesita mantener hilos separados para cada conexión abierta debido a la arquitectura asincrónica de un solo hilo de Node.js.
- **Streaming de datos:** Node.js tiene módulos incorporados que soportan el flujo de datos, lo que permite la creación de flujos de lectura y escritura.

Node.js ayudará a poner en cola los datos y distribuirlos de forma asincrónica sin bloqueos ni interrupciones. Es una buena opción para las empresas que cuentan con funciones de streaming que pueden procesar archivos mientras se cargan.

- **Proxies del lado del servidor:** Node.js puede gestionar una cantidad masiva de conexiones simultáneas utilizando un enfoque de no bloqueo. Puede emplearse como un eficaz proxy del lado del servidor que recoge datos de varios recursos de terceros. En algunos casos, Node.js se utiliza para construir aplicaciones del lado del cliente para gestionar activos y hacer proxies y stubs de solicitudes de API.
- **Tableros de control del sistema:** Gracias a la función de bucle de eventos de Node.js, puedes crear un panel de control basado en la web para comprobar el estado de cualquier servicio de forma asincrónica. Todo ello se puede informar en directo y en tiempo real de los estados de los servicios tanto internos como públicos.
- **API REST:** Node.js cuenta con una serie de paquetes como Express.js y Koa.js que se pueden utilizar para construir aplicaciones web. Puede acelerar el proceso de integración de la API y ser la base de una API REST ligera y rápida.
- **Aplicaciones de una sola página (SPA):** Las SPAs enteras se cargan en una sola página para una experiencia similar a la de una aplicación de escritorio. Como Node.js puede manejar llamadas asincrónicas de manera eficiente entre las operaciones pesadas de E/S, permite que las SPA tengan actualizaciones de datos sin refrescar la página.

También es importante tener en cuenta que Node.js soporta tanto el desarrollo de frontend como de backend. Estas son las razones por las que funciona para ambos:

- **Código reutilizable:** Se pueden reutilizar múltiples componentes de Node.js tanto para el backend como para el frontend.
- **Alta eficiencia:** El uso de Node.js puede reducir el cambio de contexto entre múltiples lenguajes de programación.
- **Node.js vs npm:** Mientras que Node.js es un entorno de ejecución de JavaScript, el Node Package Manager o npm es una gran parte del ecosistema de Node.js.

• Instalación

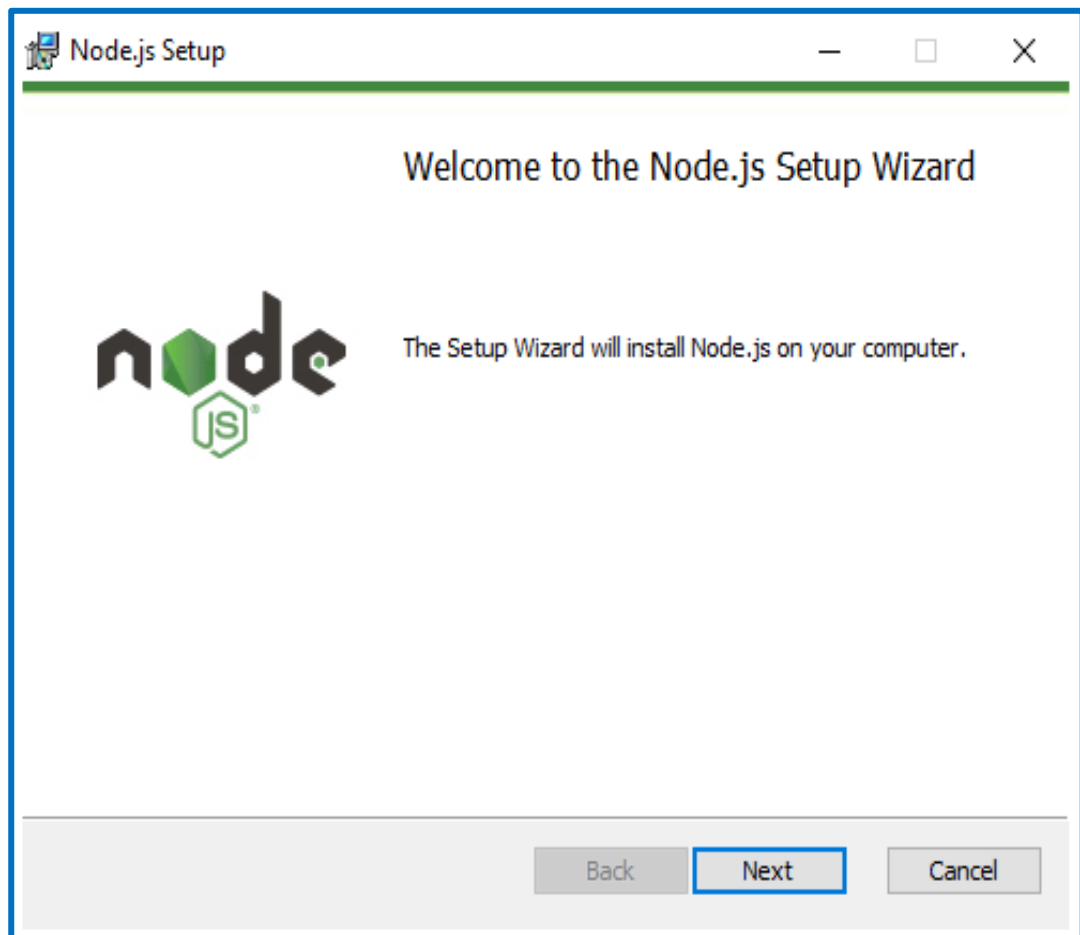
Node.js se puede instalar de diferentes maneras. Estos son los métodos más convenientes para instalarlo dependiendo del sistema operativo,

Windows, macOS y Linux:

- **Windows**

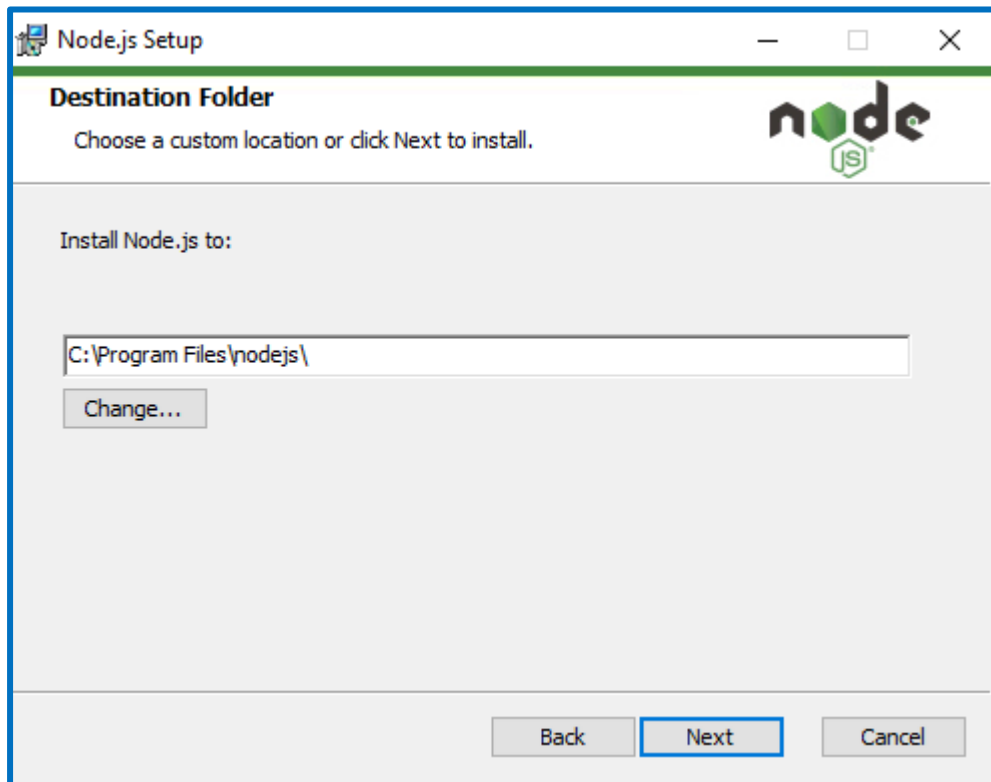
Sigue estas instrucciones para instalar Node.js en un equipo con Windows:

- Descarga el instalador de Node.js directamente desde la web oficial.
- Haz doble clic en el archivo descargado, se abrirá la ventana de instalación de Node.js. Pulsa Next.



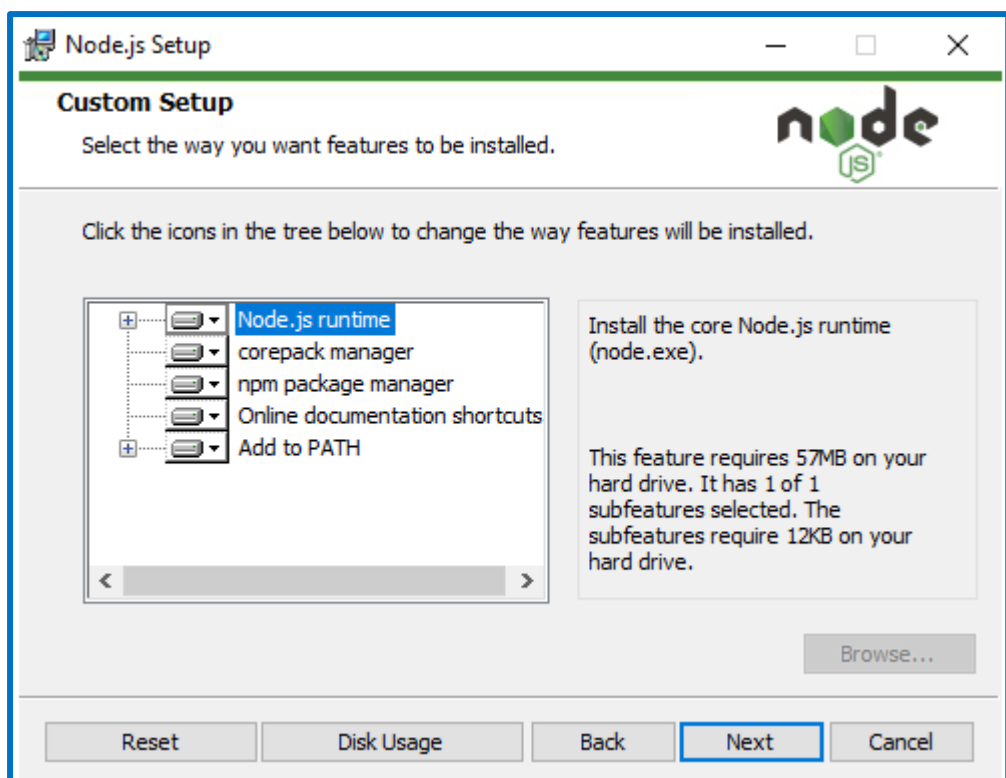
[Instalación en Windows](#)

- Revisa el Acuerdo de Licencia de Usuario Final y marca I accept the terms in the License Agreement. Pulsa Next.
- Elige la carpeta de destino y pulsa Next.



Ubicación de la instalación

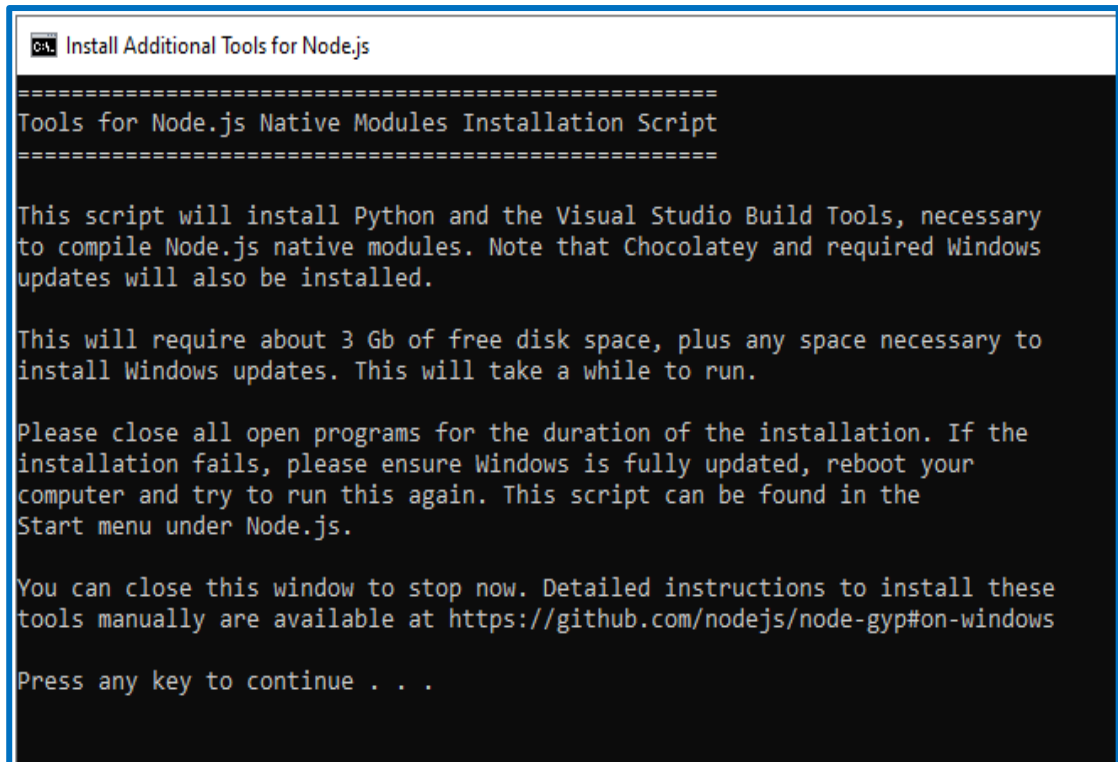
- Selecciona las características que se van a instalar. Si no estás seguro, deja los valores por defecto y pulsa Next.



Herramientas necesarias

- En la página siguiente, marca Automatically install the necessary tools. Haz clic en Next y luego en Install. Es posible que te pregunte si deseas permitir que el programa de instalación realice cambios, elige Yes.

- Una vez finalizada la instalación por defecto, se abrirá una ventana de símbolo del sistema para la configuración de herramientas adicionales. Pulsa dos veces cualquier tecla para continuar.



```

C:\> Install Additional Tools for Node.js

=====
Tools for Node.js Native Modules Installation Script
=====

This script will install Python and the Visual Studio Build Tools, necessary
to compile Node.js native modules. Note that Chocolatey and required Windows
updates will also be installed.

This will require about 3 Gb of free disk space, plus any space necessary to
install Windows updates. This will take a while to run.

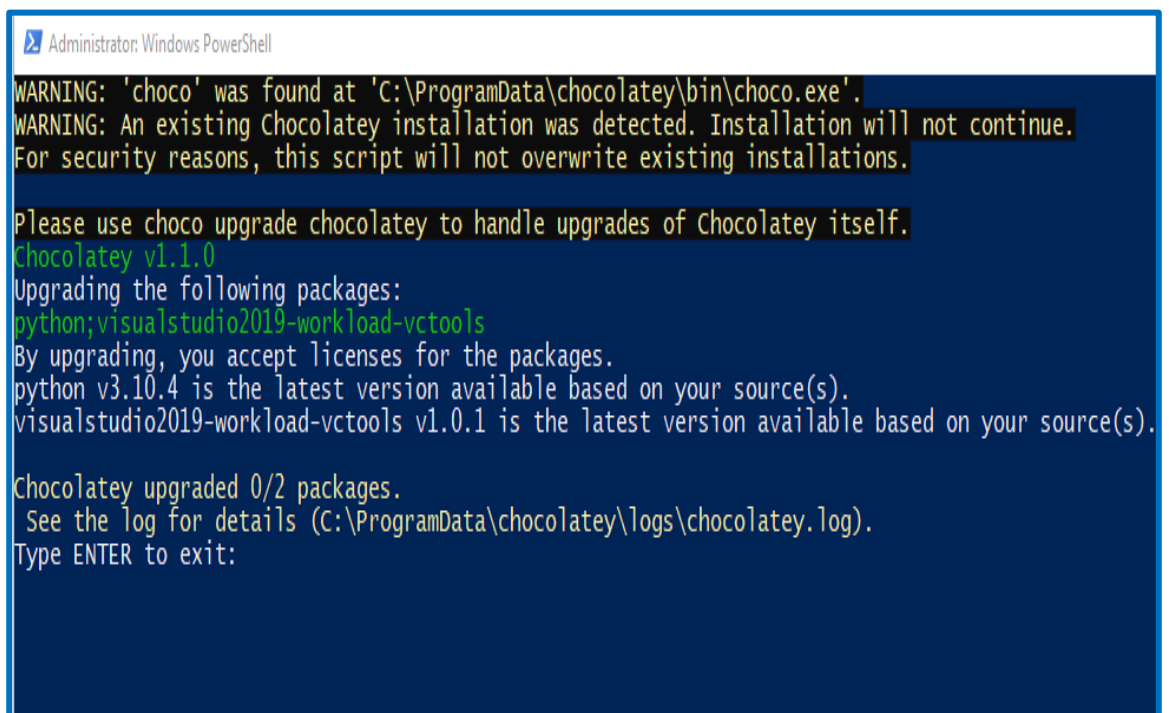
Please close all open programs for the duration of the installation. If the
installation fails, please ensure Windows is fully updated, reboot your
computer and try to run this again. This script can be found in the
Start menu under Node.js.

You can close this window to stop now. Detailed instructions to install these
tools manually are available at https://github.com/nodejs/node-gyp#on-windows

Press any key to continue . . .
  
```

[Instalación adicional para Node.js](#)

- Una vez finalizado el proceso, la interfaz te pedirá que pulses Enter para cerrar la ventana.



```

Administrator: Windows PowerShell

WARNING: 'choco' was found at 'C:\ProgramData\chocolatey\bin\choco.exe'.
WARNING: An existing Chocolatey installation was detected. Installation will not continue.
For security reasons, this script will not overwrite existing installations.

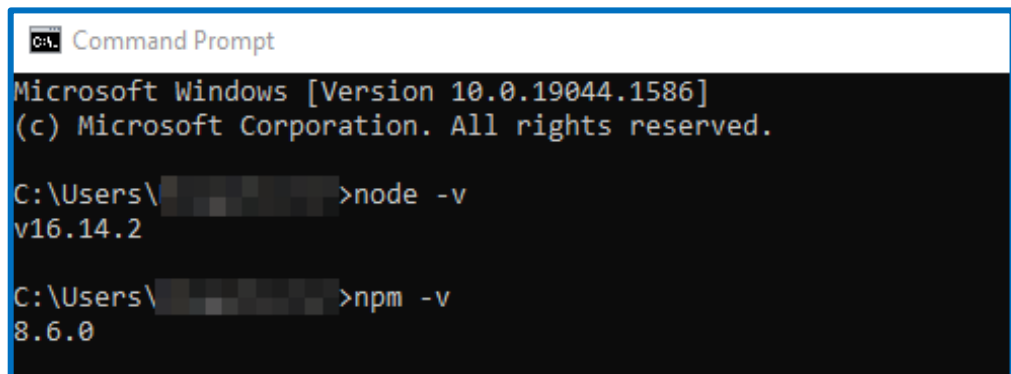
Please use choco upgrade chocolatey to handle upgrades of Chocolatey itself.
Chocolatey v1.1.0
Upgrading the following packages:
python;visualstudio2019-workload-vctools
By upgrading, you accept licenses for the packages.
python v3.10.4 is the latest version available based on your source(s).
visualstudio2019-workload-vctools v1.0.1 is the latest version available based on your source(s).

Chocolatey upgraded 0/2 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
Type ENTER to exit:
  
```

[Finalizado proceso](#)

- Para verificar la versión de Node.js, abre el símbolo del sistema y ejecuta el siguiente comando:

```
node -v  
npm -v
```



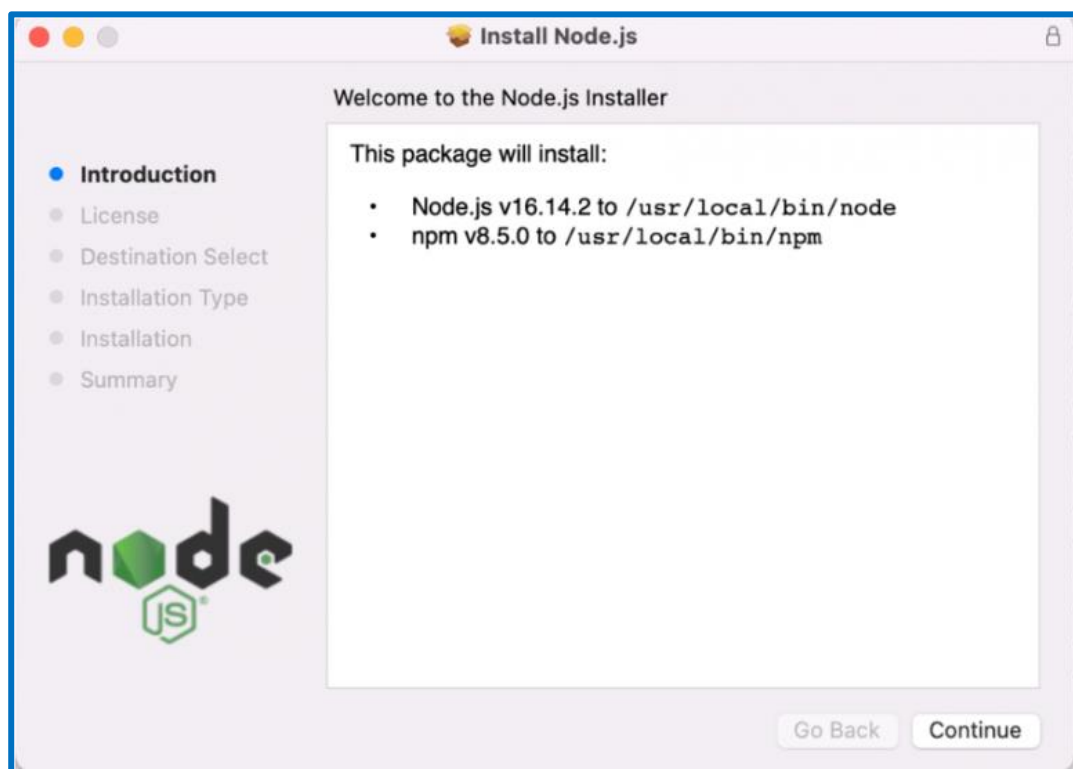
```
cmd Command Prompt  
Microsoft Windows [Version 10.0.19044.1586]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\>node -v  
v16.14.2  
  
C:\Users\>npm -v  
8.6.0
```

[Verificación de Node.js](#)

- **macOS**

A continuación, se explica cómo instalar Node.js en un equipo macOS:

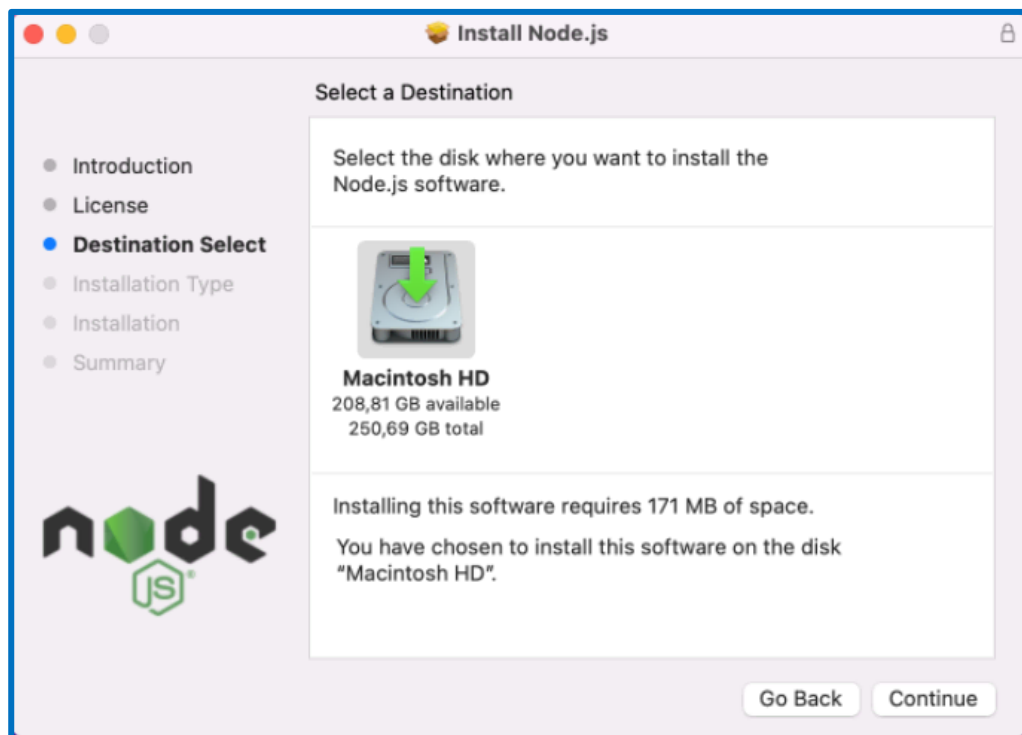
- Descarga la última versión del instalador Node.js desde la web oficial.
- Haz doble clic en el archivo descargado y se abrirá la ventana de instalación de Node.js. Haz clic en Continue.



[Instalación Node.js en MACOS](#)

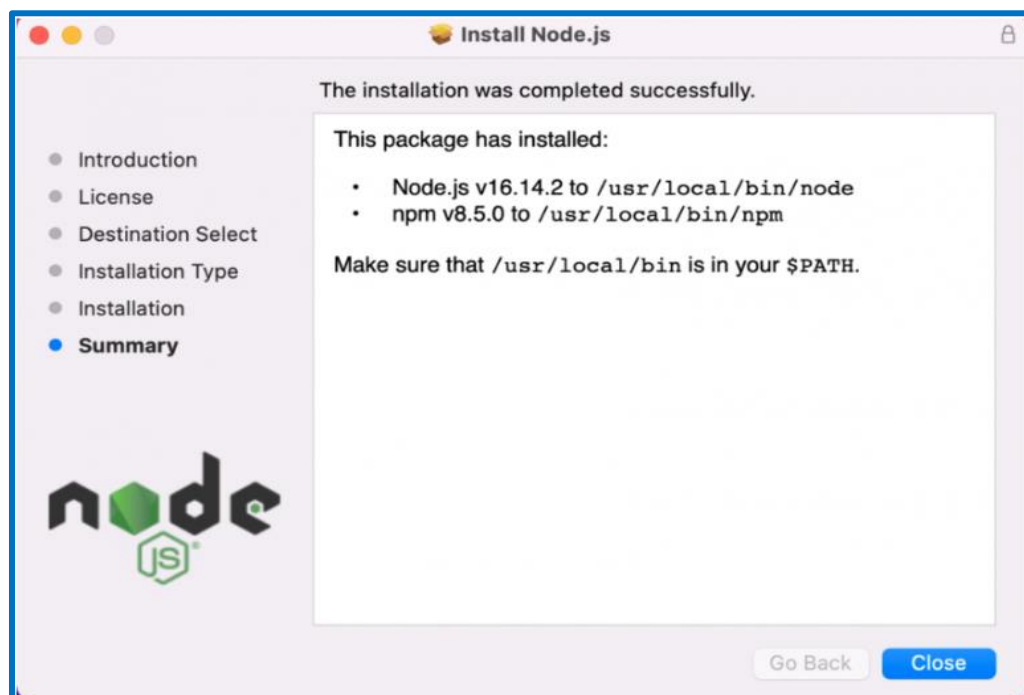
- Revisa el Acuerdo de Licencia de Software y selecciona Continue.

- Selecciona la carpeta de destino y pulsa Continue.



[MAC](#)

- Revisa el tipo de instalación y pulsa Install.
- Comenzará el proceso de instalación.
- Tanto Node.js como NPM estarán disponibles después de la instalación. Pulsa Close para finalizar la instalación.



[Cierre de instalación](#)

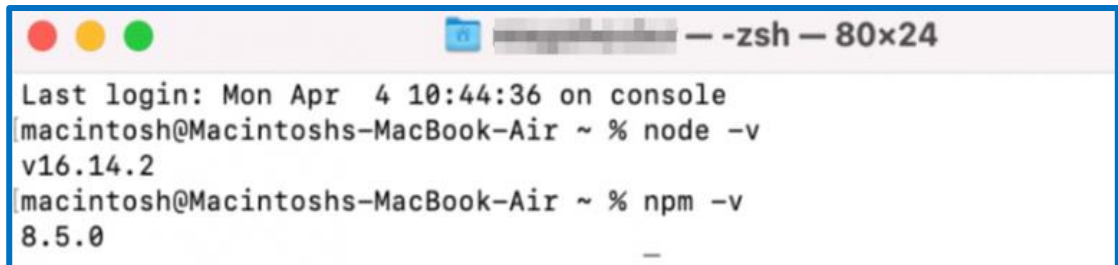
- A continuación, pulsa el icono de Launchpad en el dock y abre

Terminal.

- Verifica la instalación de Node.js y NPM escribiendo los siguientes comandos:

```
node -v
```

```
npm -v
```



```

Last login: Mon Apr  4 10:44:36 on console
macintosh@Macintoshs-MacBook-Air ~ % node -v
v16.14.2
macintosh@Macintoshs-MacBook-Air ~ % npm -v
8.5.0
  
```

[Verificación de instalación](#)

○ Linux

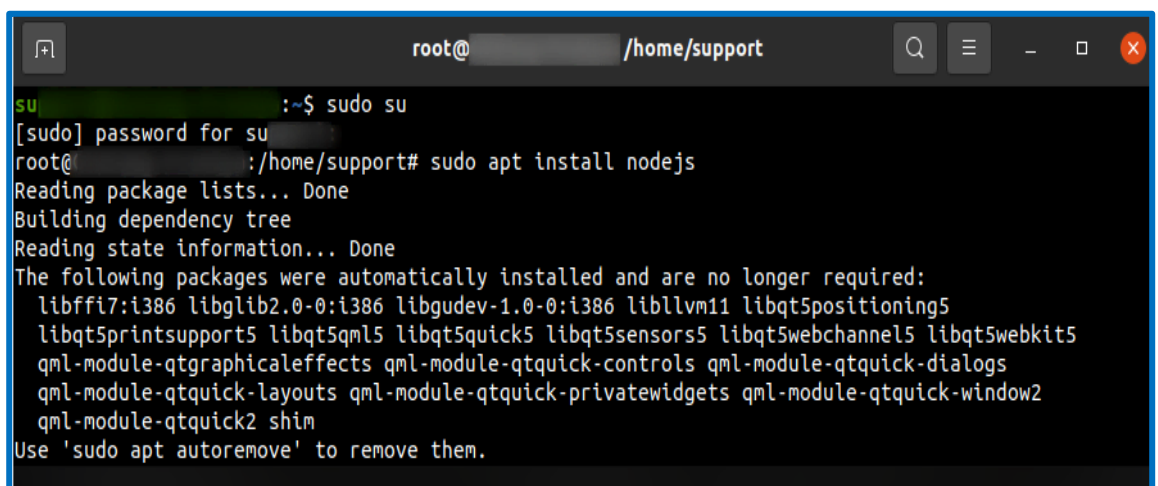
Sigue los siguientes pasos para instalar Node.js en Linux. En este tutorial usaremos Ubuntu.

- Haz clic en el botón Mostrar aplicaciones y abre Terminal.
- Ejecuta el siguiente comando para instalar Node.js:

```
sudo apt install nodejs
```

- Una vez que tu dispositivo Linux haya completado la instalación, tendrás que instalar NPM.

```
sudo apt install npm
```



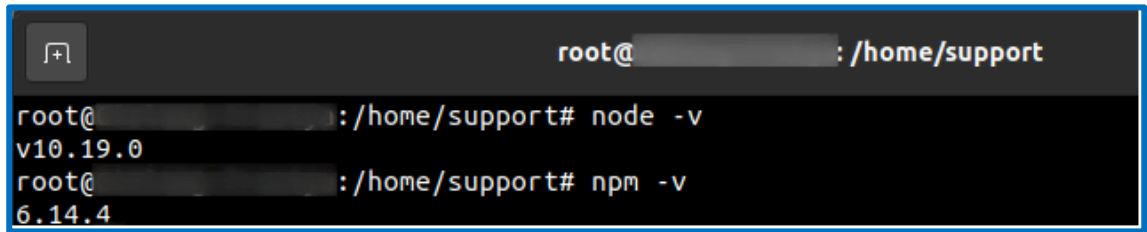
```

root@ /home/support
su ~$ sudo su
[sudo] password for su
root@ /home/support# sudo apt install nodejs
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
 libffi7:i386 libglib2.0-0:i386 libgudev-1.0-0:i386 libllvm11 libqt5positioning5
 libqt5printsupport5 libqt5qml5 libqt5quick5 libqt5sensors5 libqt5webchannel5 libqt5webkit5
 qml-module-qtgraphicaleffects qml-module-qtquick-controls qml-module-qtquick-dialogs
 qml-module-qtquick-layouts qml-module-qtquick-privatewidgets qml-module-qtquick-window2
 qml-module-qtquick2 shim
Use 'sudo apt autoremove' to remove them.
  
```

[Instalación en Linux](#)

- A continuación, verifica la versión de Node.js ejecutando estos comandos:

```
nodejs -v
npm -v
```



```

root@ [redacted] : /home/support
root@ [redacted] : /home/support# node -v
v10.19.0
root@ [redacted] : /home/support# npm -v
6.14.4

```

Verificación de instalación

• Creación de server http

Node.js facilita la creación de servidores HTTP mediante el módulo http, lo cual es útil para desarrollar aplicaciones web desde cero.

A continuación, mostraremos los pasos para crear un servidor HTTP básico:

○ **Importar el módulo http:**

```
const http = require('http');
```

- **Crear el servidor:** Utilizamos el método createServer del módulo http, el cual acepta una función callback que maneja las solicitudes y respuestas.

```
const PORT = 3000; // Puerto donde el servidor escucha
server.listen(PORT, () => {
  console.log(`Servidor escuchando en http://localhost:${PORT}`);
});
```

○ **Definir el puerto y hacer que el servidor escuche las solicitudes:**

```
const PORT = 3000; // Puerto donde el servidor escucha
server.listen(PORT, () => {
  console.log(`Servidor escuchando en http://localhost:${PORT}`);
});
```

Este servidor escucha en el puerto 3000 y, al recibir una solicitud, responde con el mensaje "¡Hola, este es mi primer servidor HTTP en Node.js!". Puedes verlo en el navegador accediendo a `http://localhost:3000`.

2. JAVASCRIPT

JavaScript es un lenguaje de programación ampliamente usado en desarrollo web, tanto para frontend como para backend, gracias a su versatilidad y compatibilidad.

- **Variables.**

Las variables son contenedores para almacenar datos. En JavaScript, existen tres palabras clave principales para declarar variables:

- **var:** Usada en versiones antiguas de JavaScript, tiene alcance de función y puede ser reasignada.
- **let:** Introducida en ES6, tiene alcance de bloque y permite reasignación.
- **const:** También introducida en ES6, tiene alcance de bloque y no permite reasignación (constantes).

Ejemplos:

```
var nombre = 'Juan';  
let edad = 25;  
const ciudad = 'Lima';
```

- **Condicionales.**

Los condicionales permiten ejecutar diferentes bloques de código según una condición.

Ejemplo:

- Con if, else if, y else:

```
let temperatura = 30;  
  
if (temperatura > 30) {  
    console.log("Hace mucho calor");  
} else if (temperatura > 20) {  
    console.log("El clima es agradable");  
} else {  
    console.log("Hace frío");  
}
```

- **switch:** Otra forma de realizar condicionales cuando tenemos varias opciones posibles.

```

let dia = 'lunes';

switch (dia) {
  case 'lunes':
    console.log("Inicio de semana");
    break;
  case 'viernes':
    console.log("Fin de semana cercano");
    break;
  default:
    console.log("Día normal");
}

```

- **Bucles.**

Los bucles nos permiten ejecutar un bloque de código repetidamente.

- **Bucle for:** Se usa cuando conocemos el número de iteraciones.

```

for (let i = 0; i < 5; i++) {
  console.log("Iteración número", i);
}

```

- **Bucle while:** Ejecuta el código mientras la condición sea verdadera.

```

let contador = 0;
while (contador < 5) {
  console.log("Contador:", contador);
  contador++;
}

```



```

let count = 0;
while (count < 5) {
  count++;
  console.log("Looping");
}

count = 0;

while (count < 10) {
  count++;
  console.log(count);
}

count = 0;

while (count < 10) {
  count++;
  if (count == 2) {
    break;
  }
  console.log(count);
}

count = 0;

while (count < 10) {
  count++;
  if (count == 2) {
    continue;
  }
  console.log(count);
}

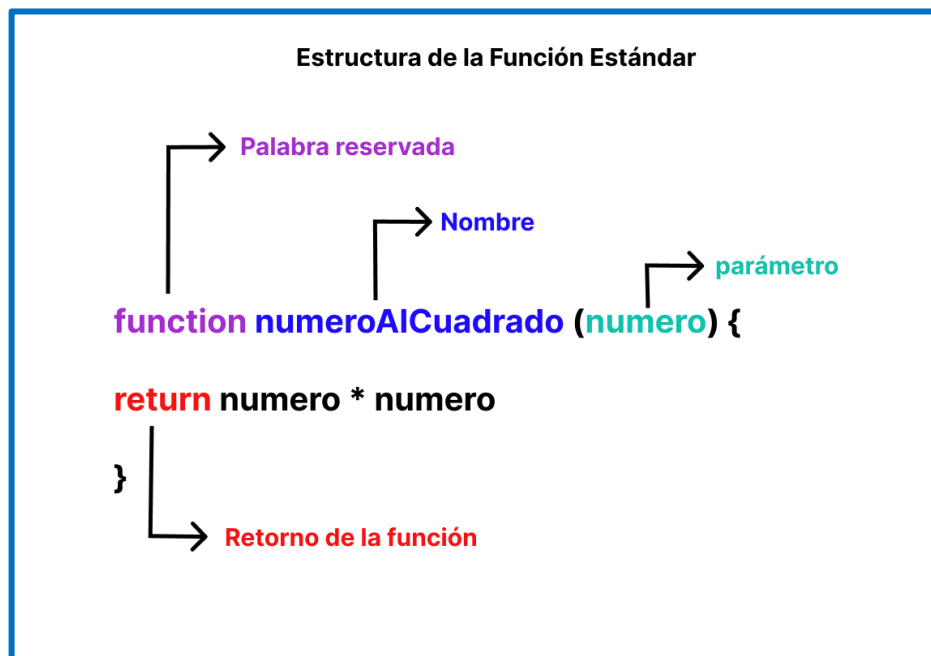
```

[Ejemplo de bucle while en Javascript.](#)

- **Bucle do...while:** Ejecuta el código al menos una vez, luego verifica la condición.

```
let numero = 0;
do {
  console.log("Número:", numero);
  numero++;
} while (numero < 5);
```

- **Funciones.**



Estructura de función estándar.

Las funciones son bloques de código reutilizables que realizan una tarea específica. En JavaScript existen varias maneras de definir funciones:

- **Funciones tradicionales:**

```
function saludo(nombre) {
  return "Hola, " + nombre;
}
console.log(saludo("Ana"));
```

- **Funciones anónimas:** Usadas principalmente para funciones de callback.

```
const suma = function(a, b) {
  return a + b;
};
console.log(suma(3, 4));
```

- **Funciones flecha (arrow functions):** Introducidas en ES6, son más concisas.

```
const resta = (a, b) => a - b;
console.log(resta(10, 4));
```

- **Eventos.**

Los eventos en JavaScript permiten a los desarrolladores responder a acciones del usuario, como hacer clic en un botón o mover el ratón.

- **Ejemplo básico de un evento click en JavaScript:**

Primero, asegurémonos de tener un elemento en HTML para interactuar:

```
<button id="miBoton">Haz clic aquí</button>
```

Luego, en JavaScript, podemos capturar el evento click:

```
document.getElementById("miBoton").addEventListener("click", () => {
    alert("¡Has hecho clic en el botón!");
});
```

3. MANIPULACION DEL DOM EN JAVASCRIPT

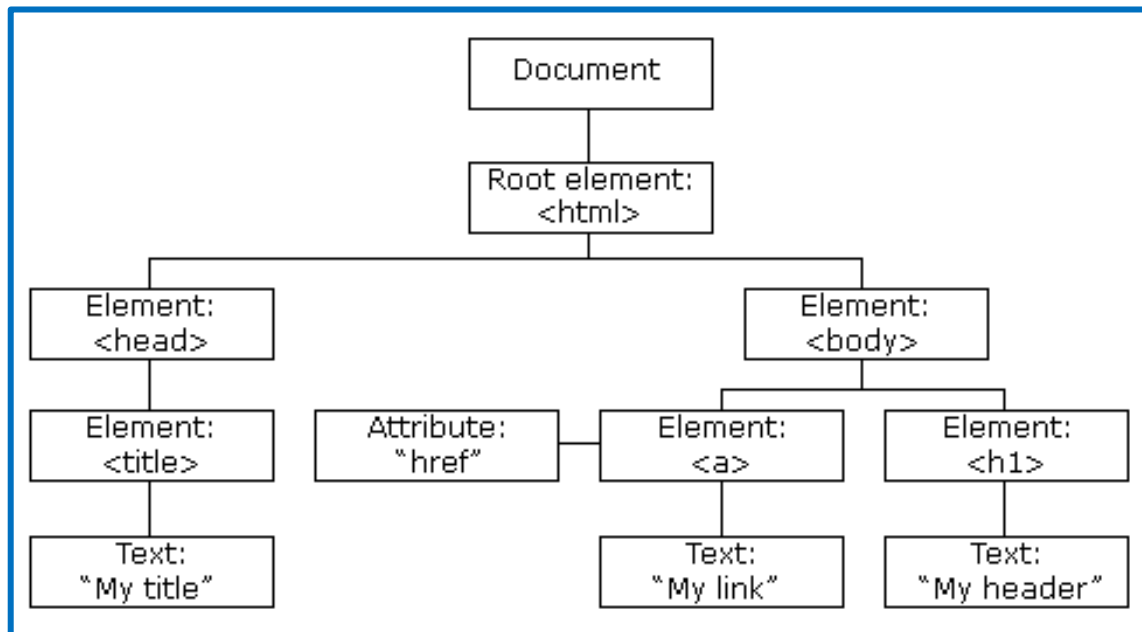
Todos los navegadores lo usan para trabajar y renderizar una página web. Todo comienza con un proceso conocido como Critical Rendering Path, el cual es un proceso que hacen todos los navegadores para que todo nuestro código HTML, CSS y Javascript se conviertan en píxeles en la pantalla de los usuarios. En este paso el navegador crea dos árboles, el primero de ellos DOM (Document Object Model) y el segundo CSSOM, se les dice árbol porque esa es la estructura de datos que representa, donde cada elemento es un nodo.

El DOM está compuesto por nodo, donde cada nodo son etiquetas HTML, y el CSSOM trabaja con el árbol de estilos.

Por tanto, tenemos los siguientes puntos:

- **DOM**

- Es una representación del HTML.
- Estructura en forma de árbol que contiene nodos.
- Es dinámico, es decir, puede cambiar.



Estructura de árbol del DOM de HTML

- **CSSOM**

El Modelo de objetos CSS (CSS Object Model) es un conjunto de APIs que permite manipular CSS desde JavaScript. Así como el DOM (Document Object Model) es para HTML, el CSSOM (CSS Object Model) es para CSS. Permite leer y modificar el estilo de CSS de forma dinámica.

- **Web API**

Es una mezcla de DOM + JavaScript. Una Web API nos permite conectar el DOM con JavaScript para que podamos manipularlo (Agregar, edita y quitar elementos). Existe más de 70, el DOM es una de ellas, existen otras para hacer animaciones, manejo de archivos, Drag and drop.

- **Leer Nodos**

- **document.getElementById**

Nos permite obtener un elemento del DOM a través de su ID. Retorna un único nodo del HTML, porque solo en una página cada id debe ser único, es decir, no puede haber más de un elemento con el mismo ID.

- **document.getElementsByTagName**

Nos permite obtener una lista (NodeList) con todos los nodos que tienen definida la etiqueta proporcionada. Es un método que puede traer 0, 1 o n Nodos.

- **document.getElementsByClassName**

Funciona similar al anterior, pues nos retorna una lista, pero en este caso nos devuelve los nodos que entre sus clases contenga la especificada en el argumento del método, es decir, vamos a identificar una clase y traemos todos los elementos que contengan dicha clase.

- **document.querySelector**

Nos permite obtener cualquier elemento del DOM de acuerdo al argumento que le indiquemos, podemos pasarle una cadena de caracteres que contiene uno o mas selectores CSS, estos deben ir separado por comas. Este método retorna el primer elemento que coincida con el filtro, es decir, si existen varios elementos que coincidan con la búsqueda este nos retornara el primero que encuentre.

- **document.querySelectorAll**

Este método es casi igual al anterior a diferencia de que este devuelve una lista (NodeList) donde se encuentran todos los elementos que coincidan con el o los selectores indicados.

A diferencia del método anterior que nos devuelve la primera coincidencia, aquí vamos a obtener todos esos elementos que coincidan con el selector indicado.

- **Diferencias entre NodeList y Array**

Un NodeList puede parecer mucho a un Array, pero la realidad es que son dos estructuras completamente distintas. Por un lado, NodeList es una colección de nodos del DOM, extraídos del HTML y un Array es un tipo de dato especial en JavaScript, donde podemos almacenar cualquier tipo de dato. Ambos tienen similitudes, como acceder a la longitud, a través de `length`, acceder a los elementos a través de su índice usando `[índice]`. Ahora, hay que aclarar que en un NodeList no tenemos disponibles los principales métodos de Array que nos facilitan la vida, como `map()`, `filter()`, `reduce()`, `some()`. Un dato curioso y super interesante del NodeList es que es una especie de colección en vivo, ¿y qué quiere decir esto? Que si se agrega o se elimina algún elemento del DOM los cambios son aplicados inmediatamente y de forma automática al NodeList.

Es recomendable transformar los NodeList a Array, pues la mayoría de los motores de JavaScript están optimizados para trabajar con Arrays, en especial el motor V8 de Google. Ahora veamos dos formas de transformar un NodeList en un array.

//Forma 1: Spread Operator

```
const inputs = document.querySelectorAll("input");
const inputsArray = [... inputs];
```

```
// Usano la clase Array y su método from
const inputs = document.querySelectorAll("inputs");
const inputsArray = Array.from(inputs);
```

- **Crear nodos**

- **document.createElement**

Crea un elemento HTML especificado por su tagName. Este elemento es solo creado, es decir, se crea en memoria, normalmente es asignado en una variable, pero no pertenece al DOM, no forma parte de los nodos hasta que lo indiquemos.

- **document.createTextNode**

Este método nos permite crear un nodo de texto, generalmente es usado para escapar caracteres HTML. Recibe una cadena de texto que contiene los datos a poner en el nodo.

- **document.createAttribute**

Este método nos permite crear un nodo de tipo atributo. El DOM no impone que tipo de atributos pueden ser agregados a un particular elemento. Funciona de igual forma a los demás, solo crea un nodo, que puede o no en el DOM.

- **Agregar Nodos**

- **parentElement.appendChild**

Este método nos permite agregar un elemento (Nodo) al final de la lista de hijos de un elemento padre previamente especificado, es decir, agrega un elemento dentro de un padre, y este es colocado al final. Es importante resaltar que si el Nodo que se está insertando existe en el documento actual, es una referencia, este será removido del padre actual y será puesto dentro del padre que se está especificando. por otro lado este método retorna o devuelve el elemento Nodo insertado.

- **parentElement.append**

Este método es una evolución de appendChild, donde podemos agregar más de un nodo, cosa que no podemos hacer en el método anterior, podemos agregar texto. Este también agrega los nodos al final de los hijos del elemento padre especificado, y no retorna ningún elemento. No tiene soporte adecuado para IE 11.

- **parentElement.insertBefore**

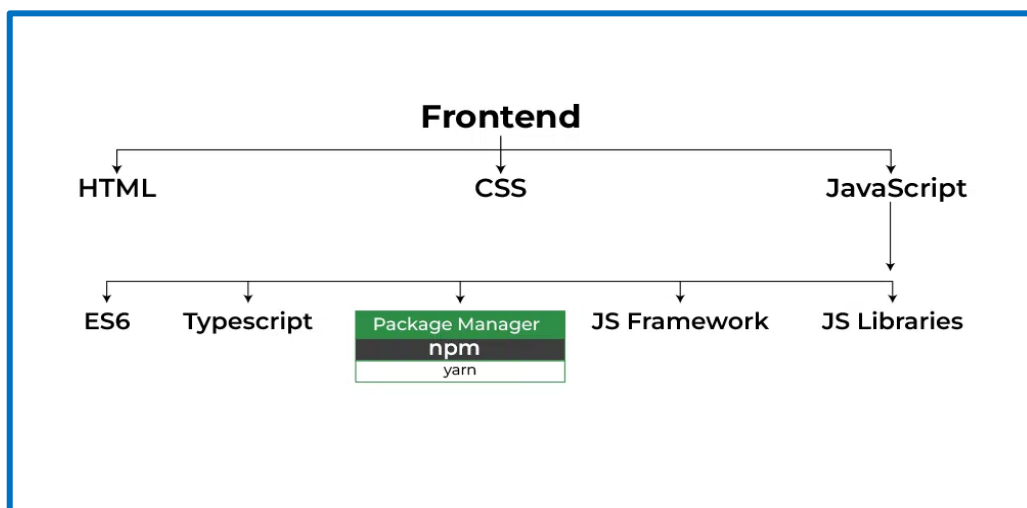
Este método permite insertar o agregar un nodo, antes del nodo que pasemos de referencia, y que pertenece al padre del que estamos accediendo a dicho método. Si insertamos un Nodo que ya esta en nuestro DOM el método lo cambia de lugar, es decir, lo quita de donde se encuentra actualmente y lo inserta en el sitio al que estamos haciendo referencia. Este método recibe dos argumentos, el primero de ellos es el Nodo que queremos insertar, y el segundo es el Nodo de referencia. Si el segundo argumento es null entonces el nodo se inserta al final de la lista de hijos del padre, es decir, actúa como el appendChild.

- **parentElement.insertAdjacentElement**

Este método permite insertar un nodo de acuerdo a una posición especificada, y lo ubica con respecto al elemento sobre el cual es llamado. Es uno de los más complicados de usar, pero el que más opciones de inserción nos proporciona. Este método recibe 2 argumentos, el primero de ellos donde se insertará el nuevo nodo, para esto contamos con cuatro opciones, las cuales son:

- **beforebegin:** Inserta el elemento antes de la referencia. Es decir, antes del nodo que llama al método en cuestión.
- **afterbegin:** Insertar el nodo dentro del elemento referencia, pero además de esto lo inserta antes de su primer hijo.
- **beforeend:** Al igual que la opción anterior lo agrega dentro del elemento referencia, con la particularidad de que ahora lo inserta después de su último hijo.
- **afterend:** Inserta el elemento después del nodo referencia, es decir, después del nodo que llama al método.

4. FUNDAMENTOS DE NODE PACKAGE MANAGER (NPM)



Node Package Manager en el proceso de programación.

- **¿Qué es Node.js y por qué es importante para los desarrolladores?**

Node.js es un entorno de ejecución basado en JavaScript que permite a los desarrolladores crear aplicaciones escalables y eficientes, tanto en el lado del servidor como en el cliente. A diferencia de otros entornos, Node.js se destaca por su modelo de programación asíncrono y su sistema de eventos, lo que lo hace ideal para aplicaciones que manejan una gran cantidad de conexiones simultáneas, como chats en tiempo real o plataformas de streaming.

- **Ventajas de usar Node.js en el desarrollo de aplicaciones**

Node.js ofrece una serie de ventajas que lo hacen destacar frente a otros entornos de desarrollo. A continuación, te explicamos algunas de las razones por las cuales es tan popular entre desarrolladores y empresas.

- **Rendimiento y escalabilidad:** Node.js es capaz de manejar miles de conexiones simultáneas sin bloquear el hilo de ejecución, gracias a su modelo de eventos no bloqueantes. Esto lo hace ideal para aplicaciones que requieren alto rendimiento, como servicios de streaming o plataformas de redes sociales. Empresas como LinkedIn migraron a Node.js, reduciendo la cantidad de servidores utilizados en un 90% mientras mejoraban el rendimiento.
- **Eficiencia en el desarrollo:** Utilizar JavaScript en ambos lados del desarrollo (cliente y servidor) simplifica enormemente el flujo de trabajo. Los desarrolladores pueden reutilizar código y habilidades, lo que reduce el tiempo de desarrollo y mejora la colaboración dentro de los equipos. Además, el ciclo de retroalimentación es más rápido, lo que permite iteraciones ágiles.
- **Ecosistema robusto a través de npm** Node.js cuenta con npm (**Node Package Manager**): el cual es uno de los gestores de paquetes más grandes del mundo. Con más de un millón de paquetes disponibles, los desarrolladores pueden aprovechar librerías y herramientas de código abierto para acelerar el desarrollo. Este vasto ecosistema permite integrar funcionalidades de manera rápida, como autenticación, procesamiento de imágenes o manejo de bases de datos.
- **Facilidad de integración:** Node.js se integra fácilmente con otras tecnologías, como bases de datos relacionales (MySQL, PostgreSQL) y no relacionales (MongoDB, Redis). También puede interactuar con servicios externos mediante APIs REST o WebSockets, lo que lo convierte en una excelente opción para aplicaciones modernas y microservicios.

Las API pueden funcionar de cuatro maneras diferentes, según el momento y el motivo de su creación.

- **API de SOAP**

Estas API utilizan el protocolo simple de acceso a objetos. El cliente y el servidor intercambian mensajes mediante XML. Se trata de una API menos flexible que era más popular en el pasado.

- **API de RPC**

Estas API se denominan llamadas a procedimientos remotos. El cliente completa una función (o procedimiento) en el servidor, y el servidor devuelve el resultado al cliente.

- **API de WebSocket**

La API de WebSocket es otro desarrollo moderno de la API web que utiliza objetos JSON para transmitir datos. La API de WebSocket admite la comunicación bidireccional entre las aplicaciones cliente y el servidor. El servidor puede enviar mensajes de devolución de llamada a los clientes conectados, por lo que es más eficiente que la API de REST.

- **API de REST**

Estas son las API más populares y flexibles que se encuentran en la web actualmente. El cliente envía las solicitudes al servidor como datos. El servidor utiliza esta entrada del cliente para iniciar funciones internas y devuelve los datos de salida al cliente. Veamos las API de REST con más detalle a continuación.

- **¿Qué son las API de REST?**

REST significa transferencia de estado representacional. REST define un conjunto de funciones como GET, PUT, DELETE, etc. que los clientes pueden utilizar para acceder a los datos del servidor. Los clientes y los servidores intercambian datos mediante HTTP.

La principal característica de la API de REST es que no tiene estado. La ausencia de estado significa que los servidores no guardan los datos del cliente entre las solicitudes. Las solicitudes de los clientes al servidor son similares a las URL que se escriben en el navegador para visitar un sitio web. La respuesta del servidor son datos simples, sin la típica representación gráfica de una página web.

- **¿Qué es una API web?**

Una API web o API de servicios web es una interfaz de procesamiento de aplicaciones entre un servidor web y un navegador web. Todos los servicios web son API, pero no todas las API son servicios web. La API de REST es un tipo especial de API web que utiliza el estilo arquitectónico estándar explicado anteriormente.

Los diferentes términos relacionados con las API, como API de Java o API de servicios, existen porque históricamente las API se crearon antes que la World Wide Web. Las API web modernas son API de REST y los términos pueden utilizarse indistintamente.

➤ ¿Qué son las integraciones de las API?

Las integraciones de las API son componentes de software que actualizan automáticamente los datos entre los clientes y los servidores. Algunos ejemplos de integraciones de las API son la sincronización automática de datos en la nube desde la galería de imágenes de su teléfono o la sincronización automática de la hora y la fecha en su laptop cuando viaja a otra zona horaria. Las empresas también pueden utilizarlas para automatizar de manera eficiente muchas funciones del sistema.



Integraciones API

- **Beneficios ofrecen las API de REST**

Las API de REST ofrecen cuatro beneficios principales:

- **Integración:** Las API se utilizan para integrar nuevas aplicaciones con los sistemas de software existentes. Esto aumenta la velocidad de desarrollo, ya que no hay que escribir cada funcionalidad desde cero. Puede utilizar las API para aprovechar el código existente.

- **Innovación:** Sectores enteros pueden cambiar con la llegada de una nueva aplicación. Las empresas deben responder con rapidez y respaldar la rápida implementación de servicios innovadores. Para ello, pueden hacer cambios en la API sin tener que reescribir todo el código.
 - **Ampliación:** Las API presentan una oportunidad única para que las empresas satisfagan las necesidades de sus clientes en diferentes plataformas. Por ejemplo, la API de mapas permite la integración de información de los mapas en sitios web, Android, iOS, etc. Cualquier empresa puede dar un acceso similar a sus bases de datos internas mediante el uso de API gratuitas o de pago.
 - **Facilidad de mantenimiento:** La API actúa como una puerta de enlace entre dos sistemas. Cada sistema está obligado a hacer cambios internos para que la API no se vea afectada. De este modo, cualquier cambio futuro que haga una de las partes en el código no afectará a la otra.
- **¿Cuáles son los diferentes tipos de API?**

Las API se clasifican tanto en función de su arquitectura como de su ámbito de uso. Ya exploramos los principales tipos de arquitecturas de API, ahora veamos el ámbito de uso.

- **API privadas:** Estas son internas de una empresa y solo se utilizan para conectar sistemas y datos dentro de la empresa.
 - **API públicas:** Están abiertas al público y pueden cualquier persona puede utilizarlas. Puede haber o no alguna autorización y coste asociado a este tipo de API.
 - **API de socios:** Solo pueden acceder a ellas los desarrolladores externos autorizados para ayudar a las asociaciones entre empresas.
 - **API compuestas:** Estas combinan dos o más API diferentes para abordar requisitos o comportamientos complejos del sistema.
- **Punto de conexión de la API y su importancia**

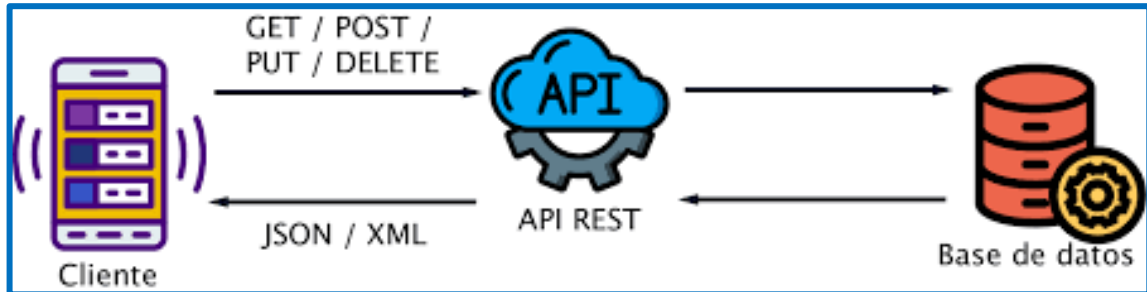
Los puntos de conexión de las API son los últimos puntos de contacto del sistema de comunicación de las API. Se trata de las URL de servidores, servicios y otras ubicaciones digitales específicas desde las que se envía y recibe información entre sistemas. Los puntos de conexión de las API son fundamentales para las empresas por dos motivos principales:

- **Seguridad:** Los puntos de conexión de las API hacen que el sistema sea vulnerable a los ataques. La supervisión de las API es crucial para evitar

su uso indebido.

- **Rendimiento:** Los puntos de conexión de las API, especialmente los de alto tráfico, pueden provocar cuellos de botella y afectar al rendimiento del sistema.

- **¿Cómo proteger una API de REST?**



API REST.

Todas las API deben protegerse mediante una autenticación y una supervisión adecuadas. Las dos maneras principales de proteger las API de REST son las siguientes:

- **Tokens de autenticación**

Se utilizan para autorizar a los usuarios a hacer la llamada a la API. Los tokens de autenticación comprueban que los usuarios son quienes dicen ser y que tienen los derechos de acceso para esa llamada concreta a la API. Por ejemplo, cuando inicia sesión en el servidor de correo electrónico, el cliente de correo electrónico utiliza tokens de autenticación para un acceso seguro.

- **Claves de API:** Las claves de API verifican el programa o la aplicación que hace la llamada a la API. Identifican la aplicación y se aseguran de que tiene los derechos de acceso necesarios para hacer la llamada a la API en cuestión. Las claves de API no son tan seguras como los tokens, pero permiten supervisar la API para recopilar datos sobre su uso. Es posible que haya notado una larga cadena de caracteres y números en la URL de su navegador cuando visita diferentes sitios web. Esta cadena es una clave de la API que el sitio web utiliza para hacer llamadas internas a la API.

- **Crear una API**

Se requiere la debida diligencia y esfuerzo para crear una API con la que otros desarrolladores quieran trabajar y en la que confíen. Los siguientes son los cinco pasos necesarios para un diseño de API de alta calidad:

- **Planificación de la API:** Las especificaciones de la API, como OpenAPI, proporcionan el esquema para el diseño de su API. Es mejor pensar en

los diferentes casos de uso por adelantado y asegurarse de que la API cumple con los estándares de desarrollo actuales.

- **Creación de la API:** Los diseñadores de API crean prototipos de API mediante código reutilizable. Una vez probado el prototipo, los desarrolladores pueden personalizarlo a las especificaciones internas.
- **Prueba de la API:** Las pruebas de la API son las mismas que las del software y deben hacerse para evitar errores y defectos. Las herramientas de pruebas de la API pueden utilizarse para reforzar la prueba de la API contra los ciberataques.
- **Documentación de la API:** Aunque las API son explicativas, la documentación de las mismas sirve de guía para mejorar su uso. Las API bien documentadas que ofrecen una gama de funciones y casos de uso tienden a ser más populares en una arquitectura orientada a servicios.
- **Comercialización de la API:** Así como Amazon es un mercado en línea para la venta minorista, los mercados de API existen para que los desarrolladores compren y vendan otras API. Publicar su API puede permitirle monetizarla.

2. MANIPULACIÓN DE DATOS DEL TIPO JSON EN JAVASCRIPT

JSON son las siglas de JavaScript Object Notation, y no es más que un formato ligero de datos, con una estructura (notación) específica, que es totalmente compatible de forma nativa con JavaScript. Como su propio nombre indica, JSON se basa en la sintaxis que tiene JavaScript para crear objetos.



[JSON](#)

Sin embargo, su contenido puede ser simplemente un array, un number, un string, un boolean o incluso un null. Ahora vamos a crear un objeto vacío para que contenga más datos para ejemplificarlo:

```

{
  "name": "Manz",
  "life": 3,
  "totalLife": 6,
  "power": 10,
  "dead": false,
  "props": ["invisibility", "coding", "happymood"],
  "senses": {
    "vision": 50,
    "audition": 75,
    "taste": 40,
    "touch": 80
  }
}

```

Si comparamos un JSON con un objeto JavaScript, aparecen algunas ligeras diferencias y matices:

- Las propiedades del objeto deben estar entrecomilladas con «comillas dobles»
- Los textos string deben estar entrecomillados con «comillas dobles»
- Sólo se puede almacenar tipos como string, number, object, array, boolean o null.
- Tipos de datos como function, date, regexp u otros, no es posible almacenarlos en un JSON.
- Tampoco es posible añadir comentarios en un JSON.

Hay formatos derivados de JSON que sí permiten comentarios, como es el caso de JSON5.

Mucho cuidado con las comillas mal cerradas o las comas sobrantes (antes de un cierre de llaves, por ejemplo). Suelen ser motivos de error de sintaxis frecuentemente. Existe una cómoda extensión para Visual Code llamada Fix JSON que te corrige los errores de formato de un JSON.

• Utilización de JSON

Como hemos mencionado, si analizamos bien la sintaxis de un JSON, nos habremos dado cuenta de que es muy similar a un objeto declarado JavaScript, pero con ciertas diferencias:

```
const user = {
  name: "Manz",
  life: 99,
}
```

En JavaScript tenemos una serie de métodos que nos facilitan la tarea de pasar de object de JavaScript a JSON y viceversa, pudiendo trabajar con contenido de tipo (que contenga un JSON) y objetos JavaScript según interese.

El primero de ellos, el método `.parse()` nos va a permitir pasar el contenido de texto string de un JSON a object. En contrapartida, el método `.stringify()` nos va a permitir pasar de object de JavaScript a contenido de texto con el JSON en cuestión.

○ Convertir JSON a objeto

La acción de convertir JSON a objeto JavaScript se le suele denominar parsear. Es una acción que analiza un JSON válido y devuelve un objeto JavaScript con dicha información correctamente estructurada. Para ello, utilizaremos el mencionado método `JSON.parse()`:

```
const json = `{
  "name": "Manz",
  "life": 99
}`;

const user = JSON.parse(json);

user.name; // "Manz"
user.life; // 99
```

Como se puede ver, `user` es un objeto generado a partir del JSON almacenado en la variable `JSON` y podemos consultar sus propiedades y trabajar con ellas sin problemas.

○ Convertir objeto a JSON

La operación inversa, convertir un objeto JavaScript a JSON también se puede realizar fácilmente haciendo uso del método `JSON.stringify()`. Este impronunciable método se puede utilizar para transformar un objeto de JavaScript a JSON rápidamente:

```
const user = {
  name: "Manz",
  life: 99,
  talk: function () {
```

```
    return "Hola!";
  },
};
```

```
JSON.stringify(user);
// '{"name":"Manz","life":99}'
```

Observa que, como habíamos dicho, las funciones no están soportadas por JSON, por lo que si intentamos convertir un objeto que contiene métodos o funciones, `JSON.stringify()` no fallará, pero simplemente devolverá un string omitiendo las propiedades que contengan funciones (u otros tipos de datos no soportados).

Además, se le puede pasar un segundo parámetro al método `.stringify()`, que será un array que actuará de filtro a la hora de generar el objeto. Observa el siguiente ejemplo:

```
const user = {
  name: "Manz",
  life: 99,
  power: 10,
};
```

```
JSON.stringify(user, ["life"])
// '{"life":99}'
```

```
JSON.stringify(user, ["name", "power"])
// '{"name":"Manz","power":10}'
```

```
JSON.stringify(user, [])
// '{}'
```

```
JSON.stringify(user, null)
// '{"name":"Manz","life":99,"power":10}'
```

Observa que el penúltimo caso, no se conserva ninguna propiedad, mientras que el último, se conserva todo.

Por último, también podemos añadir un tercer parámetro en el método `.stringify()` que indicará el número de espacios que quieres usar al crear el string del JSON resultante. Observa que hasta ahora, el string está minificado y aparece todo junto en la misma línea.

Observa lo que ocurre en los siguientes casos:

```
const user = {
  name: "Manz",
```



```

    life: 99
  };

JSON.stringify(user, null, 2);
// {
//   "name": "Manz",
//   "life": 99
// }

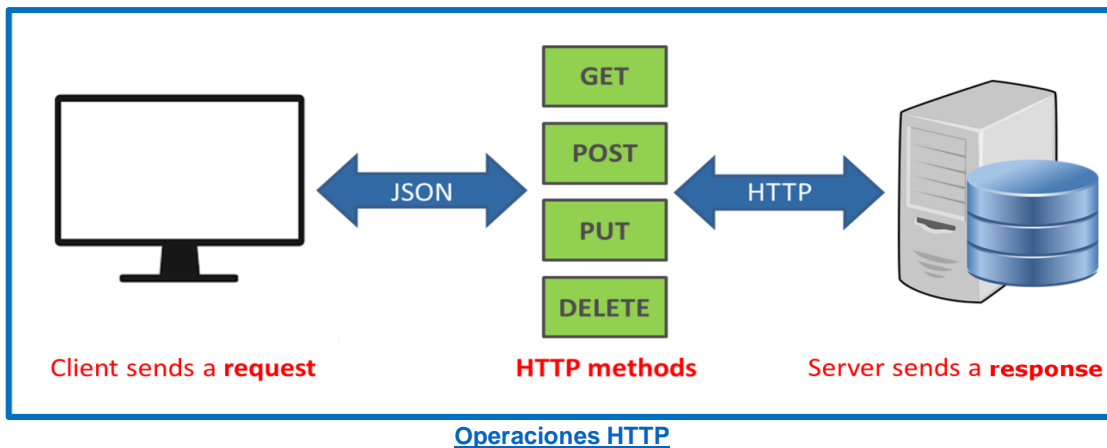
JSON.stringify(user, null, 4);
// {
//   "name": "Manz",
//   "life": 99
// }

JSON.stringify(user, ["name"], 1);
// {
//   "name": "Manz"
// }

```

En el primer caso, json2, el resultado se genera indentado a 2 espacios. En el segundo caso, json4, el resultado se genera indentado a 4 espacios. En el tercer y último caso, json1, se filtran las propiedades, dejando sólo "name" y se genera indentando a 1 espacio.

3. OPERACIONES HTTP ESTÁNDAR API REST



En una API RESTful, las operaciones HTTP básicas permiten crear, leer, actualizar y eliminar recursos. Estas operaciones se conocen como CRUD:

- **GET, POST.**

- **GET**

- **Función:** Recuperar datos de un servidor sin modificar nada en el sistema.

- **Uso común:** Consultar recursos como listas de usuarios, productos, o detalles específicos.

- **Ejemplo de uso:**

GET /api/usuarios

Este ejemplo recupera la lista completa de usuarios.

○ POST

- **Función:** Enviar datos al servidor para crear un nuevo recurso.

- **Uso común:** Agregar un nuevo usuario, crear un nuevo pedido, etc.

- **Ejemplo de uso:**

POST /api/usuarios

Cuerpo de la solicitud (Body):

```
{
  "nombre": "Ana",
  "email": "ana@example.com"
}
```

Este ejemplo crea un nuevo usuario con el nombre y email especificados.

● PUT.

- **Función:** Modificar o actualizar un recurso existente en el servidor.

- **Uso común:** Actualizar datos completos de un usuario o un registro específico.

- **Ejemplo de uso:**

PUT /api/usuarios/1

Cuerpo de la solicitud (Body):

```
{
  "nombre": "Ana María",
  "email": "anamaria@example.com"
}
```

Este ejemplo actualiza el usuario con ID 1, cambiando su nombre y

email.

- **DELETE.**

- **Función:** Eliminar un recurso del servidor.
- **Uso común:** Borrar un registro específico como un usuario, un pedido, etc.
- **Ejemplo de uso:**

DELETE /api/usuarios/1

Este ejemplo elimina al usuario con ID 1.

4. LENGUAJE XML.

El lenguaje de marcado extensible (XML) permite definir y almacenar datos de forma compartible. XML admite el intercambio de información entre sistemas de computación, como sitios web, bases de datos y aplicaciones de terceros. Las reglas predefinidas facilitan la transmisión de datos como archivos XML a través de cualquier red, ya que el destinatario puede usar esas reglas para leer los datos de forma precisa y eficiente.

```
<?xml version="1.0"?>
<documento>
  <NAME>Jhon</NAME>
  <BREED>Siamese</BREED>
  <AGE>6</AGE>
  <ALTERED>yes</ALTERED>
  <DECLAWED>no</DECLAWED>
  <LICENSE id="Objeto">Cn5454</LICENSE>
  <OWNER>Boris Direx</OWNER>
</documento>
```

Ejemplo de XML.

- **Importancia de XML**

El lenguaje de marcado extensible (XML) es un lenguaje de marcado que proporciona reglas para definir cualquier dato. A diferencia de otros lenguajes de programación, XML no puede realizar operaciones de computación por sí mismo.

En cambio, se puede implementar cualquier software o lenguaje de programación para la administración estructurada de datos.

Por ejemplo, imagine un documento de texto con comentarios. Los comentarios pueden ofrecer sugerencias como las siguientes:

- Ponga el título en negrita.
- Esta oración es un encabezado.
- Esta palabra es el autor.

Estos comentarios mejoran la usabilidad del documento sin repercutir en su contenido. Del mismo modo, XML utiliza símbolos de marcado para proporcionar más información sobre los datos. Otros programas, como los navegadores y las aplicaciones de procesamiento de datos, utilizan esta información para procesar datos estructurados de manera más eficiente.

• Etiquetas XML

Los símbolos de marcado, denominados etiquetas en XML, se utilizan para definir los datos. Por ejemplo, para representar los datos de una librería, puede crear etiquetas como <libro>, <título> y <autor>. El documento XML de un solo libro tendría el siguiente contenido:

```
<libro>
<título>Introducción a Amazon Web Services</título>
<autor>Mark Wilkins</autor>
</libro>
```

Las etiquetas ofrecen una sofisticada codificación de datos para integrar los flujos de información en diferentes sistemas.

• Beneficios de usar XML

○ Respaldo para las transacciones interempresariales

Cuando una empresa vende un bien o servicio a otra empresa, las dos empresas necesitan intercambiar información como el costo, las especificaciones y los plazos de entrega. Con el lenguaje de marcado extensible (XML), pueden compartir toda la información necesaria electrónicamente y cerrar negocios complejos de forma automática, sin intervención humana.

○ Conservación de la integridad de los datos

XML le permite transferir datos junto con la descripción de los datos, lo que evita la pérdida de la integridad de los datos. Puede usar esta información descriptiva para hacer lo siguiente:

- Verificar la precisión de los datos.
- Personalizar automáticamente la presentación de datos para diferentes usuarios.
- Almacenar datos de forma coherente en múltiples plataformas.

- **Mejora de la eficiencia de búsqueda**

Los programas de computación, como los motores de búsqueda, pueden ordenar y categorizar archivos XML de forma más eficiente y precisa que otros tipos de documentos. Por ejemplo, la palabra marca puede ser un sustantivo o un verbo. Basándose en las etiquetas XML, los motores de búsqueda pueden categorizar con precisión marca para resultados de búsqueda relevantes. Por lo tanto, XML ayuda a las computadoras a interpretar el lenguaje natural de manera más eficiente.

- **Diseño de aplicaciones flexibles**

Con XML, puede actualizar o modificar cómodamente el diseño de su aplicación. Muchas tecnologías, especialmente las más nuevas, vienen con compatibilidad con XML incorporada. Pueden leer y procesar automáticamente los archivos de datos XML para que pueda realizar cambios sin tener que volver a formatear toda la base de datos.

- **Aplicaciones de XML**

El lenguaje de marcado extensible (XML) es la tecnología subyacente en miles de aplicaciones, que van desde herramientas de productividad comunes, como el procesamiento de textos hasta el software de publicación de libros e incluso sistemas de configuración de aplicaciones complejos.

- **Transferencia de datos**

Puede usar XML para transferir datos entre dos sistemas que almacenan los mismos datos en diferentes formatos. Por ejemplo, su sitio web almacena las fechas en formato MM/DD/AAAA, pero su sistema de contabilidad almacena las fechas en formato DD/MM/AAAA. Puede transferir los datos del sitio web al sistema de contabilidad mediante XML. Los desarrolladores pueden escribir código que convierta automáticamente lo siguiente:

- Datos del sitio web a formato XML.
- Datos XML a datos del sistema contable.

- Los datos del sistema de contabilidad de vuelta a formato XML.
- Datos XML de vuelta a datos del sitio web.

- **Aplicaciones web**

XML da estructura a los datos que se ven en las páginas web. Otras tecnologías de sitios web, como HTML, funcionan con XML para presentar datos coherentes y relevantes a los visitantes del sitio web. Por ejemplo, consideremos un sitio web de comercio electrónico que vende ropa. En lugar de mostrar toda la ropa a todos los visitantes, el sitio web utiliza XML para crear páginas web personalizadas basadas en las preferencias del usuario. Muestra productos de marcas específicas filtrando la etiqueta <marca>.



Aplicaciones web.

- **Documentación**

Puede usar XML para especificar la información estructural de cualquier documento técnico. Luego, otros programas procesan la estructura del documento para presentarla de manera flexible. Por ejemplo, hay etiquetas XML para un párrafo, un elemento de una lista numerada y un encabezado. Con estas etiquetas, otros tipos de software preparan automáticamente el documento para usos como impresión y publicación de páginas web.

- **Tipo de datos**

Muchos lenguajes de programación admiten XML como tipo de datos. Con esta compatibilidad, puede escribir fácilmente programas en otros lenguajes que funcionen directamente con archivos XML.

- **Componentes de un archivo XML**

Un archivo de lenguaje de marcado extensible (XML) es un documento basado en texto que se puede guardar con la extensión .xml. Puede escribir XML de forma similar a otros archivos de texto. Para crear o editar un archivo XML, puede usar cualquiera de las siguientes opciones: editores de texto como Notepad o Notepad++, editores XML en línea, navegadores web y cualquier archivo XML incluye los siguientes componentes.

- **Documento XML**

Las etiquetas `<xml></xml>` se utilizan para marcar el principio y el final de un archivo XML. El contenido de estas etiquetas también se denomina documento XML. Es la primera etiqueta que cualquier software buscará para procesar código XML.

- **Declaración XML**

Un documento XML comienza con alguna información sobre el propio XML. Por ejemplo, podría mencionar la versión XML que sigue. Esta apertura se denomina declaración XML. A continuación, se muestra un ejemplo.

```
<?xml version="1.0" encoding="UTF-8"?>
```

- **Elementos XML**

Todas las demás etiquetas que cree en un documento XML se denominan elementos XML. Los elementos XML pueden contener las siguientes características: texto, atributos y otros elementos.

Todos los documentos XML comienzan con una etiqueta principal, que se denomina elemento raíz.

Por ejemplo, eche un vistazo al archivo XML que aparece a continuación:

```
<ListaInvitación>
<familia>
  <tía>
    <nombre>Cristina</nombre>
    <nombre>Estefanía</nombre>
  </tía>
</familia>
</ListaInvitación>
```

`<ListaInvitación>` es el elemento raíz; familia y tía son otros nombres de elementos.

○ Atributos XML

Los elementos XML pueden tener otros descriptores denominados atributos. Puede definir sus propios nombres de atributos y escribir los valores de los atributos entre comillas, como se muestra a continuación.

```
<edad de la persona="22">
```

○ Contenido XML

Los datos de los archivos XML también se denominan contenido XML. Por ejemplo, en el archivo XML, es posible que veas datos como este.

```
<amigo>
  <nombre>Carlos</nombre>
  <nombre>Esteban</nombre>
</amigo>
```

Los valores de los datos Carlos y Esteban son el contenido.

5. CONSULTAS SQL Y BASE DE DATOS



[Comando básico en SQL.](#)

- **Operaciones Básicas de SQL (CRUD)**

- **SELECT (Leer):** Recupera datos de una tabla.

```
SELECT * FROM usuarios WHERE id = 1;
```

Este ejemplo obtiene los datos del usuario con ID 1.

- **INSERT (Crear):** Agrega un nuevo registro en una tabla.

```
INSERT INTO usuarios (nombre, email) VALUES ('Ana', 'ana@example.com');
```

Este ejemplo añade un nuevo usuario.

- **UPDATE (Actualizar):** Modifica un registro existente en una tabla.

```
UPDATE usuarios SET nombre = 'Ana María' WHERE id = 1;
```

Este ejemplo actualiza el nombre del usuario con ID 1.

- **DELETE (Eliminar):** Borra un registro específico de una tabla.

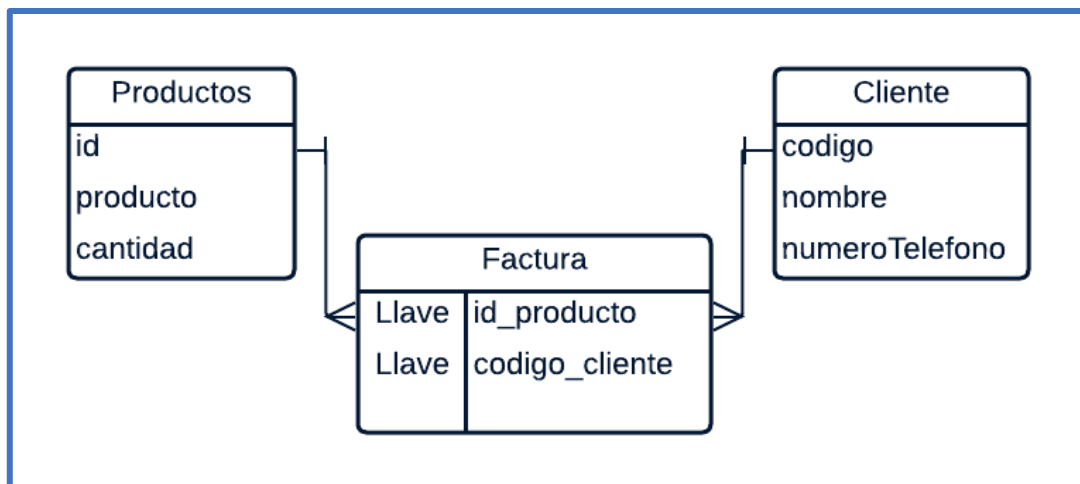
```
DELETE FROM usuarios WHERE id = 1;
```

Este ejemplo elimina el usuario con ID 1.

- **Tipos de Bases de Datos**

En desarrollo de APIs, es esencial conocer los tipos de bases de datos que se pueden usar.

- **Bases de datos relacionales:** Utilizan SQL y almacenan datos en tablas relacionadas (ej., MySQL, PostgreSQL).



Ejemplo de gráfico de base de datos relaciones.

- **Bases de datos no relacionales (NoSQL):** Utilizan estructuras como documentos, claves-valor o grafos (ej., MongoDB, Redis).

- **Conexión entre API RESTful y Bases de Datos**

Al diseñar un API RESTful, las consultas SQL se ejecutan en la base de datos en respuesta a las solicitudes HTTP (GET, POST, PUT, DELETE). La API actúa como un "puente" entre el cliente y la base de datos.

6. CONCEPTOS DE VIRTUALIZACIÓN

La virtualización es una tecnología que se puede usar para crear representaciones virtuales de servidores, almacenamiento, redes y otras máquinas físicas. El software virtual imita las funciones del hardware físico para ejecutar varias máquinas virtuales a la vez en una única máquina física. Las empresas recurren a la virtualización para utilizar sus recursos de hardware de manera eficiente y obtener retornos mayores de sus inversiones. También potencia los servicios de computación en la nube que ayudan a las organizaciones a administrar la infraestructura de manera más eficaz.

- **Importancia de la virtualización**

Al utilizar la virtualización, es posible interactuar con cualquier recurso de hardware con mayor flexibilidad. Los servidores físicos consumen electricidad, ocupan espacio de almacenamiento y necesitan mantenimiento. Con frecuencia el acceso a estos está limitado por la proximidad física y el diseño de la red. La virtualización resuelve todas estas limitaciones al abstraer la funcionalidad del hardware físico en el software. Es posible administrar, mantener y utilizar la infraestructura de hardware como una aplicación en la web.

- **Ejemplo de virtualización**

Imagine una empresa que necesita servidores para tres funciones:

- Almacenar los correos electrónicos de la empresa de forma segura.
- Ejecutar una aplicación orientada a los clientes.
- Ejecutar aplicaciones empresariales internas.



Importancia de virtualidad.

Cada una de estas funciones tiene diferentes requisitos de configuración:

- La aplicación de correo electrónico requiere más capacidad de almacenamiento y un sistema operativo de Windows.
- La aplicación orientada a los clientes requiere un sistema operativo Linux y una gran capacidad de procesamiento para gestionar grandes volúmenes de tráfico del sitio web.
- La aplicación empresarial interna requiere iOS y más memoria interna (RAM).

Para cumplir estos requisitos, la empresa configura tres servidores físicos dedicados diferentes para cada aplicación. La empresa debe realizar una elevada inversión inicial y llevar a cabo el mantenimiento y las actualizaciones continuas de una máquina a la vez. La empresa tampoco puede optimizar su capacidad de computación. Paga el 100 % de los costos de mantenimiento de los servidores, pero únicamente utiliza una fracción de sus capacidades de almacenamiento y procesamiento.

○ **Uso eficiente del hardware**

En el caso de la virtualización, la empresa crea tres servidores digitales, o máquinas virtuales, en un único servidor físico. Especifica los requisitos del sistema operativo para las máquinas virtuales y puede utilizarlas como los servidores físicos. Sin embargo, la empresa ahora tiene menos hardware y gastos relacionados.

○ **Infraestructura como servicio**

La empresa puede ir un paso más allá y utilizar una instancia en la nube o una máquina virtual de un proveedor de computación en la nube, como AWS. AWS administra todo el hardware subyacente. Además, la empresa puede solicitar recursos de servidor con distintas configuraciones. Todas las aplicaciones se ejecutan en estos servidores virtuales sin que los usuarios noten ninguna diferencia. También se hace más fácil la administración de los servidores para el equipo de TI de la empresa.

- **La virtualización**

Para comprender de forma correcta la máquina virtual basada en kernel (KVM), primero debe comprender algunos conceptos básicos de la virtualización. La virtualización es un proceso que permite a una computadora compartir sus recursos de hardware con varios entornos separados de forma digital. Cada entorno virtualizado se ejecuta dentro de los recursos asignados, como la memoria, la potencia de procesamiento y el almacenamiento. Con la virtualización, las organizaciones pueden cambiar entre diferentes sistemas operativos en el mismo servidor sin tener que reiniciar.

Las máquinas virtuales y los hipervisores son dos conceptos importantes en la virtualización.

- **Máquina virtual**

Una máquina virtual es un equipo definido por software que se ejecuta en un equipo físico con un sistema operativo y recursos informáticos independientes. La computadora física se denomina máquina host y las máquinas virtuales son máquinas invitadas. Se pueden ejecutar varias máquinas virtuales en una sola máquina física. Un hipervisor extrae las máquinas virtuales del hardware de la computadora.

- **Hipervisor**

El hipervisor es un componente de software que administra varias máquinas virtuales en una computadora. Garantiza que cada máquina virtual reciba los recursos asignados y no interfiera con el funcionamiento de otras máquinas virtuales. Existen dos tipos de hipervisores.

- **Hipervisor de tipo 1**

Un hipervisor de tipo 1, o hipervisor bare metal, es un programa de hipervisor que, en vez de instalarse en el sistema operativo, se instala de forma directa en el hardware de la computadora. Por lo tanto, los hipervisores de tipo 1 tienen un mejor rendimiento y se utilizan con

frecuencia para las aplicaciones empresariales. KVM utiliza el hipervisor de tipo 1 para alojar varias máquinas virtuales en el sistema operativo Linux.

➤ Hipervisor de tipo 2

También conocido como hipervisor alojado, el hipervisor de tipo 2 está instalado en un sistema operativo. Los hipervisores de tipo 2 son adecuados para la informática del usuario final.



• Beneficios de la virtualización

La virtualización proporciona varios beneficios a cualquier organización:

○ Utilización eficiente de los recursos

La virtualización mejora los recursos de hardware que se utilizan en el centro de datos. Por ejemplo, en lugar de ejecutar un servidor en un sistema informático, se puede crear un grupo de servidores virtuales en el mismo sistema informático, al utilizar y devolver servidores al grupo según sea necesario. Tener menos servidores físicos subyacentes libera espacio en el centro de datos y supone un ahorro de dinero en electricidad, generadores y aparatos de refrigeración.

○ Administración automatizada de las TI

Ahora que las computadoras físicas son virtuales, se pueden administrar mediante el uso de herramientas de software. Los administradores crean programas de implementación y configuración para definir plantillas de máquinas virtuales. Es posible duplicar la infraestructura de forma repetida y coherente y evitar las configuraciones manuales propensas a errores.

- **Recuperación de desastres más rápida**

Cuando eventos como los desastres naturales o los ataques cibernéticos afectan negativamente a las operaciones empresariales, recuperar el acceso a la infraestructura de TI y sustituir o arreglar un servidor físico puede llevar horas o incluso días. Por el contrario, al utilizar entornos virtualizados, el proceso tarda minutos. Esta rápida respuesta mejora significativamente la capacidad de recuperación y facilita la continuidad del negocio para que las operaciones puedan continuar según lo previsto.

- **Funcionamiento de la virtualización**

La virtualización utiliza un software especializado, llamado hipervisor, para crear varias instancias en la nube o máquinas virtuales en un solo equipo físico.

- **Instancias en la nube o máquinas virtuales**

Después de instalar el software de virtualización en la computadora, podrá crear una o más máquinas virtuales. Se puede acceder a las máquinas virtuales de la misma manera que se accede a otras aplicaciones en la computadora. La computadora se llama host y la máquina virtual se llama huésped. Varios huéspedes se pueden ejecutar en el host. Cada huésped tiene su propio sistema operativo, que puede ser el mismo o diferente del sistema operativo del host.

Desde la perspectiva del usuario, la máquina virtual funciona como un servidor típico. Tiene ajustes, configuraciones y aplicaciones instaladas. Los recursos de computación, como las unidades centrales de procesamiento (CPU), la memoria de acceso aleatorio (RAM) y el almacenamiento aparecen de la misma manera que en un servidor físico. También es posible configurar y actualizar los sistemas operativos huéspedes y sus aplicaciones según sea necesario sin afectar al sistema operativo host.

- **Tipos de virtualización**

La tecnología de virtualización permite obtener las funciones de distintos tipos de infraestructura física y todas las ventajas de un entorno virtualizado. Se puede ir más allá de las máquinas virtuales para crear un conjunto de recursos virtuales en el entorno virtual.

- **Virtualización de servidores**

La virtualización de servidores es un proceso que particiona un servidor físico en múltiples servidores virtuales. Es una forma eficaz y rentable de

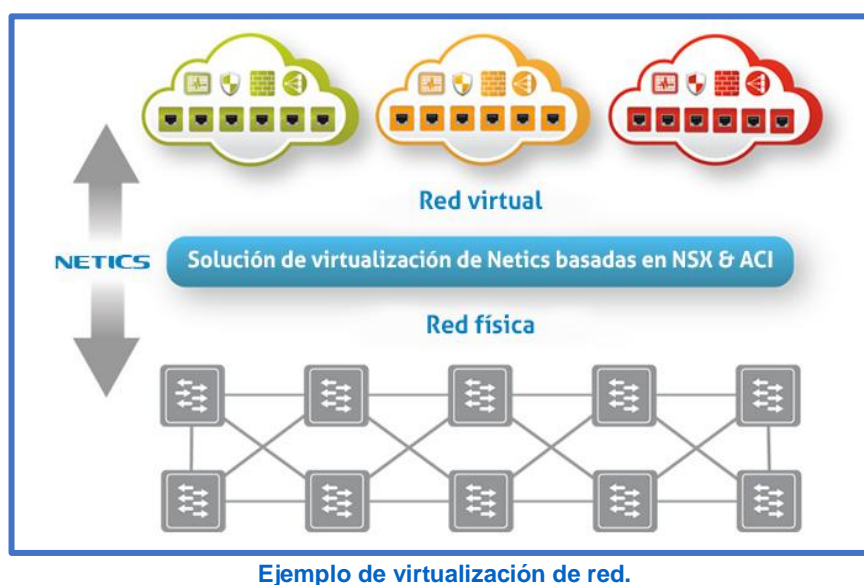
utilizar los recursos del servidor y de implementar los servicios de TI en una organización. Sin la virtualización de servidores, los servidores físicos únicamente aprovechan una pequeña cantidad de sus capacidades de procesamiento, lo que provoca que los dispositivos queden inactivos.

○ Virtualización del almacenamiento

La virtualización del almacenamiento combina las funciones de los dispositivos de almacenamiento físico, como el almacenamiento conectado a la red (NAS) y la red de área de almacenamiento (SAN). Se puede agrupar el hardware de almacenamiento del centro de datos, aunque sea de diferentes proveedores o de diferentes tipos. La virtualización del almacenamiento utiliza todo el almacenamiento físico de datos y crea una gran unidad de almacenamiento virtual que se puede asignar y controlar mediante un software de administración. Los administradores de TI pueden optimizar las actividades de almacenamiento, como el archivado, las copias de seguridad y la recuperación, porque pueden combinar varios dispositivos de almacenamiento en red de forma virtual en un único dispositivo de almacenamiento.

○ Virtualización de red

Cualquier red de computadoras dispone de elementos de hardware como conmutadores, enrutadores y firewalls. Una organización con oficinas en múltiples ubicaciones geográficas puede tener varias tecnologías de red diferentes que trabajan juntas para crear su red empresarial. La virtualización de la red es un proceso que combina todos estos recursos de red para centralizar las tareas administrativas. Los administradores pueden ajustar y controlar estos elementos virtualmente sin tocar los componentes físicos, lo que simplifica enormemente la administración de la red.



○ Virtualización de aplicaciones

La virtualización de aplicaciones extrae las funciones de las aplicaciones de modo que se ejecuten en sistemas operativos distintos de aquellos para los que fueron diseñadas. Por ejemplo, los usuarios pueden ejecutar una aplicación de Microsoft Windows en una máquina Linux sin cambiar la configuración de la máquina. Para lograr la virtualización de las aplicaciones, siga estas prácticas:

- **Streaming de aplicaciones:** los usuarios transmiten la aplicación desde un servidor remoto, de modo que se ejecute únicamente en el dispositivo del usuario final cuando sea necesario.
- **Virtualización de aplicaciones basada en el servidor:** los usuarios pueden acceder a la aplicación remota desde un navegador o una interfaz de cliente sin necesidad de instalarla.
- **Virtualización local de la aplicación:** el código de la aplicación se envía con su propio entorno para que se ejecute en todos los sistemas operativos sin necesidad de cambios.

○ Virtualización de escritorios

En la mayoría de las organizaciones existe personal sin conocimientos técnicos que utiliza sistemas operativos de escritorio para ejecutar aplicaciones empresariales comunes. Por ejemplo, es posible que se presenten los siguientes casos de personal:

- Un equipo de atención al cliente que requiere una computadora de escritorio con Windows 10 y un software de gestión de las relaciones con los clientes.
- Un equipo de marketing que requiere Windows Vista para las aplicaciones de ventas

Puede utilizar la virtualización de escritorios para ejecutar estos diferentes sistemas operativos de escritorio en máquinas virtuales, a las que los equipos pueden acceder de forma remota. Este tipo de virtualización permite administrar los escritorios de forma eficiente y segura, con lo que se ahorra dinero en hardware de escritorio. Los siguientes son tipos de virtualización de escritorios.

○ Infraestructura de escritorio virtual

La infraestructura de escritorio virtual ejecuta escritorios virtuales en un servidor remoto. Los usuarios pueden obtener acceso a estos mediante el uso de dispositivos cliente.

- **Virtualización de escritorio local**

Al utilizar la virtualización de escritorio local, se ejecuta el hipervisor en una computadora local y se crea una computadora virtual con un sistema operativo diferente. Es posible alternar entre el entorno local y el virtual de la misma manera que se puede alternar entre las aplicaciones.



Virtualización de escritorio

- **Diferencia la virtualización de la computación en la nube**

La computación en la nube consiste en suministrar recursos informáticos bajo demanda a través de Internet con precio de pago por uso. En lugar de comprar, poseer y mantener un centro de datos físico, es posible acceder a los servicios tecnológicos, como la potencia de computación, el almacenamiento y las bases de datos, a medida que se necesitan a través de un proveedor de servicios en la nube.

La tecnología de virtualización hace posible la computación en la nube. Los proveedores de servicios en la nube crean y mantienen sus propios centros de datos. Crean diferentes entornos virtuales que utilizan los recursos de hardware subyacentes. Posteriormente, es posible programar el sistema de manera que acceda a estos recursos en la nube mediante el uso de API. Las necesidades de la infraestructura se pueden satisfacer como un servicio completamente administrado.

- **AWS como ayuda con la virtualización y la computación en la nube**

Al usar AWS, dispondrá de varias formas de crear, implementar y salir al mercado rápidamente con tecnología de vanguardia. Por ejemplo, se podría beneficiar de cualquiera de los siguientes servicios:

- Utilice Amazon Elastic Compute Cloud (Amazon EC2) para controlar de manera precisa la infraestructura. Elija los procesadores, el almacenamiento y la red que desee.

- Utilice AWS Lambda para la computación sin servidor y así podrá ejecutar código sin tener en cuenta los servidores.
- Utilice Amazon Lightsail para implementar servidores virtuales, almacenamiento, bases de datos y redes por un precio bajo y predecible.

- **La diferencia la virtualización de servidores del uso de contenedores**

El uso de contenedores es una forma de implementar el código de la aplicación de manera que se ejecute en cualquier entorno físico o virtual sin necesidad de hacer cambios. Los desarrolladores empaquetan el código de la aplicación con las bibliotecas relacionadas, los archivos de configuración y otras dependencias que el código necesita para ejecutarse. Este paquete único de software, llamado contenedor, se puede ejecutar de forma independiente en cualquier plataforma. El uso de contenedores es un tipo de virtualización de aplicaciones.

Puede concebir la virtualización de servidores como la construcción de una carretera para conectar dos lugares. Hay que recrear un entorno virtual completo y luego ejecutar la aplicación en este. A modo de comparación, el uso de contenedores es como construir un helicóptero que pueda volar a cualquiera de esos lugares. La aplicación se encuentra dentro de un contenedor y se puede ejecutar en todo tipo de entornos físicos o virtuales.

REFERENCIAS

- *Arquitectura de una API REST · Desarrollo de aplicaciones web.* (s/f). Gitbooks.io. <https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html>
- Boneu, M. S. (2020, diciembre 22). *Variables en JavaScript – Guía para principiantes sobre var, const y let.* freecodecamp.org. <https://www.freecodecamp.org/espanol/news/javascript-variables-para-principiantes/>
- buitrago/, P. O. (2024, mayo 7). *Manejo de formularios del lado del cliente con JavaScript: explicado con código de ejemplo.* freecodecamp.org. <https://www.freecodecamp.org/espanol/news/manejo-de-formularios-del-lado-del-cliente-con-javascript-explicado-con-codigo-de-ejemplo/>
- *Como manipular el DOM con Javascript.* (s/f). Platzi. <https://platzi.com/tutoriales/2193-dom/9548-como-manipular-el-dom-con-javascript/>
- *Condicionales.* (s/f). Gitbook.io. <https://makeitrealcamp.gitbook.io/javascript-book/condicionales>
- Coppola, M. (2022, septiembre 19). *Frontend y backend: qué son, en qué se diferencian y ejemplos.* Hubspot.es. <https://blog.hubspot.es/website/frontend-y-backend>
- David-Engel. (s/f). *Paso 3: Conexión con SQL mediante Node.js - Node.js driver for SQL Server.* Microsoft.com. <https://learn.microsoft.com/es-es/sql/connect/node-js/step-3-proof-of-concept-connecting-to-sql-using-node-js?view=sql-server-ver16>
- *Formato JSON.* (s/f). Lenguajejs.com. <https://lenguajejs.com/javascript/objetos/json/>
- *Git - git documentation.* (s/f). Git-scm.com. <https://git-scm.com/docs/git#Documentation/git.txt--v>
- *Git y GitHub.* (s/f). MDN Web Docs. https://developer.mozilla.org/es/docs/Learn/Tools_and_testing/GitHub
- *Guía definitiva de Node.js: Introducción para desarrolladores.* (2024, septiembre 18). Evolve Academy. <https://evolveacademy.es/guia-definitiva-de-node-js-introduccion-para-desarrolladores/>
- Infante, D. C. H. (2022, mayo 20). *Qué es Node.js: Casos de uso comunes y cómo instalarlo.* Tutoriales Hostinger. <https://www.hostinger.es/tutoriales/que-es-node-js>
- *Manipulación del DOM en JavaScript.* (2019, junio 24). *Por amor al código.* <https://juanmirod.github.io/2019/06/24/chuleta-dom.html>

- Master, S. Q. L. (2022, diciembre 7). SQL en JavaScript. *Programar en SQL*. <https://www.programarsql.com/sql-en-javascript/>
- Mota, A. E. (2023, octubre 13). *Algoritmos que todo desarrollador de JavaScript debe conocer*. Listopro Community. <https://community.listopro.com/algoritmos-que-todo-desarrollador-de-javascript-debe-conocer/>
- Palma, I. (2020, diciembre 14). *Bucles JavaScript Explicados: For Loop, While Loop, Do...while Loop, y más*. freecodecamp.org. <https://www.freecodecamp.org/espanol/news/javascript-bucles-explicados-for-while-loops/>
- ¿Qué es la virtualización? (s/f). Amazon.com. <https://aws.amazon.com/es/what-is/virtualization/>
- ¿Qué es NPM? (s/f). Lenguajejs.com. <https://lenguajejs.com/npm/introduccion/que-es/>
- ¿Qué es una interfaz de programación de aplicaciones (API)? (s/f). Amazon.com. <https://aws.amazon.com/es/what-is/api/>
- ¿Qué es XML? (s/f). Amazon.com. <https://aws.amazon.com/es/what-is/xml/>
- ¿Qué son las funciones? (s/f). Lenguajejs.com. <https://lenguajejs.com/javascript/funciones/que-son/>
- ¿Qué son los eventos? (s/f). Lenguajejs.com. <https://lenguajejs.com/javascript/eventos/que-son-eventos/>
- *Tipos de datos y estructuras en JavaScript*. (s/f). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/JavaScript/Data_structures



RDA
RECURSO DIDÁCTICO PARA EL APRENDIZAJE