Argentina programa 4.0

Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

primero la gente

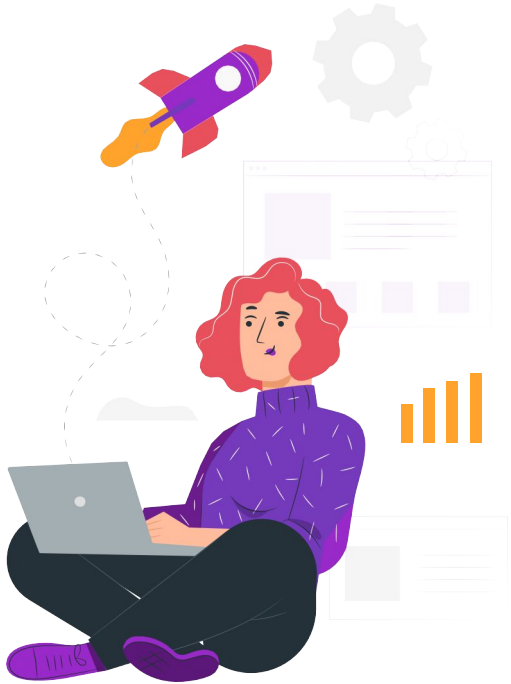
UNTREF
UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Clase 0: Introducción a la programación

Ministerio de
Desarrollo Productivo
Argentina

Agenda de hoy

A. Desarrollo de aplicaciones para servidores
B. Qué es Node JS
    a. JavaScript del lado del servidor
    b. Limitaciones del lenguaje JS
C. Instalación de Node JS
    a. el archivo package.json
    b. el archivo .gitignore
    c. la carpeta node_modules
D. Uso de la Ventana Terminal - Línea de comandos
E. Nivelación de JS (repaso de los conceptos más importantes)
    a. variables, constantes, condicionales, ciclos, funciones (parámetros, retorno),
F. console (log, warn, table, error)
    a. try - catch - finally



Desarrollo de aplicaciones para servidores

Desarrollo de aplicaciones para servidores

Cuando hablamos de la web, uno piensa inmediatamente en un navegador web y una dirección, o URL. Pero la web también es un correo electrónico, una red social, una red de contactos corporativos, una red que nos provee acceso a la diversión de plataformas de videojuegos, etcétera.

Y también, gran parte del funcionamiento de nuestros dispositivos móviles, se genera utilizando a la web como mecanismo de intercambio de datos (*Facebook, Twitter, WhatsApp, TikTok, entre otros sistemas de software conocidos*).

Desarrollo de aplicaciones para servidores

Por ello, podemos decir que, la web, pasó de ser un simple complemento en nuestras vidas a ser el centro de nuestro día a día.

En un principio, proveyéndonos acceso al correo electrónico y una navegación limitada, hasta nuestros días, donde nos brinda total información sobre nuestros intereses personales o profesionales, para nuestra vida laboral, como también nuestra vida académica, de ocio, y familiar, entre otros nichos.

Desarrollo de aplicaciones para servidores

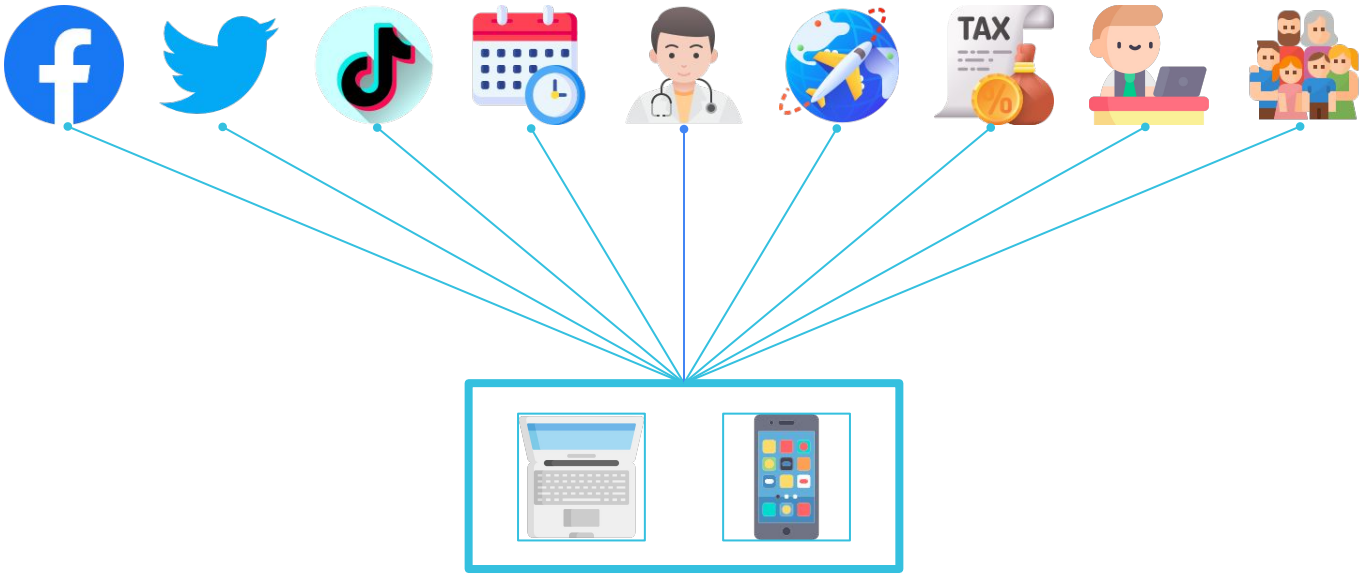
Y, para que todo esto ocurra, hoy dependemos del acceso a Internet, para realizar prácticamente cualquier actividad que requiera una computadora de por medio.

Desde escribirnos con familiares y amigos que se encuentran en un punto distante, pasando por realizar trámites en el Estado o en un Municipio, abonar impuestos, planificar un viaje de vacaciones, y organizar una reunión con amigos.

💡 **Todo esto, y más, se realiza hoy a través de Internet**

Desarrollo de aplicaciones para servidores

El uso de productos y servicios de internet, que consumimos de forma periódica.



Dispositivos personales

Desarrollo de aplicaciones para servidores

La web se ha convertido en el centro de gestión de gran parte de las tareas de nuestra vida.

Incluso, aquellas actividades que por el momento no realizamos con la web como medio de gestión, en algún momento también se terminarán digitalizando.

Desarrollo de aplicaciones para servidores

Como usuarios, estamos acostumbrados a interactuar con aplicaciones web, aplicaciones instaladas en nuestra computadora, o aplicaciones instaladas en nuestro dispositivo móvil.

Cualquier sea el caso, **todas estas aplicaciones de software dependen hoy de Internet** como medio de transporte. **Y dependen también de aplicaciones backend** (*aplicaciones remotas, o de servidor*), siendo estas últimas las que resuelven gran parte de las tareas mencionadas.

¿Y qué es una aplicación de backend?

Una aplicación de backend, backend app, aplicación de servidor, etcétera, es la parte de software que **se encarga de gestionar los datos y la lógica de negocio de una aplicación nativa, web, o aplicación móvil**.

Es una parte "*invisible*" de la aplicación, que no dispone de una interfaz gráfica para el usuario final, pero es más que esencial para que las otras aplicaciones que utilizamos nosotros funcionen correctamente.

Qué podemos realizar con una aplicación de backend

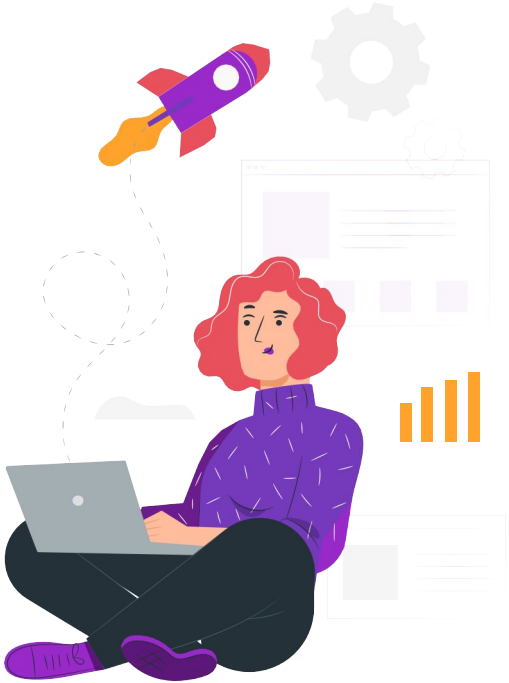
A. autenticar y gestionar usuarios

B. gestionar sesiones de trabajos

C. procesar pagos de compras

D. enviar correos electrónicos predefinidos

E. gestionar archivos (audio, video, imágenes, documentos)



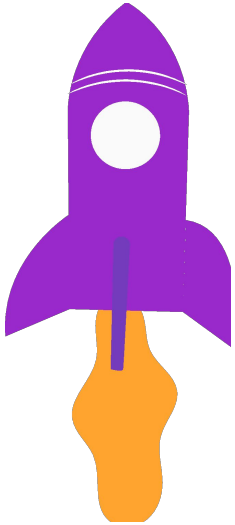
Entre otras tantas funciones más...

¿Y qué es una aplicación de backend?

Resumen

En la programación backend, se incluyen tareas como la gestión de bases de datos, el procesamiento de estos, y la comunicación entre diferentes componentes de una aplicación, además de mecanismos de seguridad necesarios.

Por lo tanto, **la programación backend es la parte invisible de una aplicación o sitio web que hace posible que todo funcione** correctamente en el "*detrás de escena*".



Node Js

Node JS

Entre los diferentes lenguajes de programación mencionados para construir aplicaciones backend, JavaScript era uno de ellos.

Pero, en realidad, **JavaScript nace como un lenguaje de programación integrado a los navegadores web, para poder procesar la lógica del lado del usuario**, o sea, en aplicaciones frontend.

Node JS

Aún así desde el año 2009 y, gracias a **Node.js**, JavaScript fue portado como un lenguaje de programación funcional del lado del servidor y, por ello, hoy tenemos la posibilidad de construir este tipo de aplicaciones utilizando un lenguaje flexible, simple, pero a su vez poderoso, como lo es JS.

Veamos entonces, qué es Node.js, y cómo podemos sacar provecho de este para la construcción de aplicaciones de backend.

Node JS

Node.js es un entorno de tiempo de ejecución de JavaScript de código abierto y multiplataforma que permite a los desarrolladores construir aplicaciones de servidor escalables y de alta velocidad.

A diferencia de los navegadores web, que ejecutan JavaScript en el lado del cliente, Node.js permite a los desarrolladores ejecutar JavaScript en el lado del servidor.

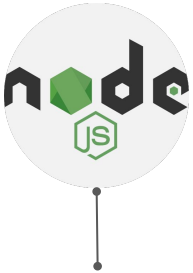
Node JS

Está construido sobre el motor JavaScript V8, el cual le proporciona un rendimiento rápido y eficiente.

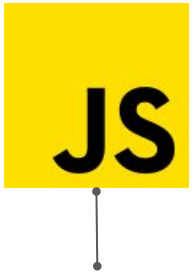
Además, Node.js **utiliza un modelo de E/S sin bloqueo** y **orientado a eventos**. Esto lo hace un lenguaje adecuado para aplicaciones en tiempo real que requieren una gran cantidad de conexiones de red y entrada/salida de datos.

Node JS

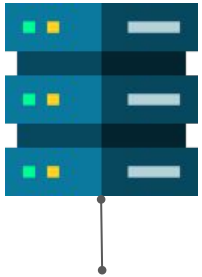
Características de Node JS en el desarrollo de aplicaciones backend



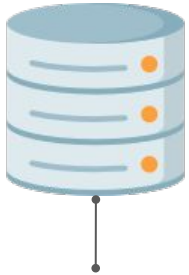
Entorno de ejecución de aplicaciones, comúnmente de backend



Utiliza a JavaScript como lenguaje de programación



Las aplicaciones se alojan en servidores



Interactúa con todo tipo de bases de datos (*Relacionales*, *NoSQL*, *planas*)

Utiliza mecanismos de seguridad para validar usuarios y permisos

Node JS

Ventajas y usos

Es ampliamente **utilizado para construir aplicaciones web, APIs, servidores de juegos, aplicaciones de chat** en tiempo real y otras aplicaciones de servidor.

Cuenta con una gran comunidad de desarrolladores y una amplia gama de **bibliotecas y módulos** disponibles para su uso en proyectos de Node.js.

Limitaciones del lenguaje JS

Consideraciones

El motor V8 utilizado para interpretar a JavaScript del lado del servidor le quita algunos poderes a este lenguaje de programación como, por ejemplo, trabajar con el DOM HTML.

Por ello, funciones JS como **alert(), prompt()** o **confirm(),** y objetos como document, no estarán disponibles en Node.js, para integrar aplicaciones de servidor.

Node Js

Descarga e instalación

En el sitio web oficial: www.nodejs.org, encontraremos el link de descarga de este entorno de ejecución.

Existen dos versiones de Node JS:

- ➔ **LTS**
- ➔ Actual

Descarga e instalación

Diferencias entre las versiones para descargar.

LTS

Esta versión cuenta con las características probadas y aprobadas de Node JS. La sigla en cuestión proviene de (*Long Time Support*), con soporte extendido en el tiempo, más estable y, por ello, **será la versión que debemos elegir para su descarga e instalación**.

ACTUAL

La otra alternativa cuenta con características modernas, muy nuevas, en su mayoría experimentales. Es un entorno pensado para probar novedades de Node JS. **No está no esta pensada para ser implementado en ambientes profesionales** o corporativos.

Ejercicio práctico

Abrimos un espacio de trabajo, para poder instalar y configurar las herramientas necesarias.

Si no lo tienes, descarga e instala VS Code

Luego, descarga e instala Node JS

La profe te compartirá los links de acceso para la descarga de estas aplicaciones

Break

En 10 minutos continuamos

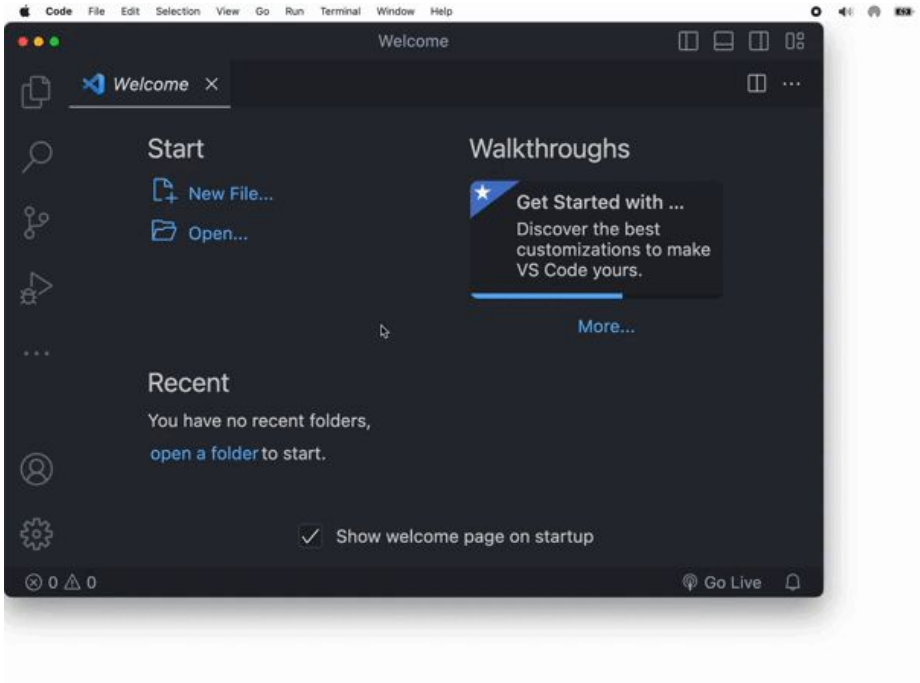
Verificar la instalación

Abre Visual Studio Code, despliega el **menú Terminal**, y elige el primer punto del menú: **New Terminal**.

Dentro de este se abrirá una ventana de Terminal (Comandos).

Allí **puedes verificar que Node JS esté correctamente instalado** en tu computadora. Para ello, escribe lo siguiente:

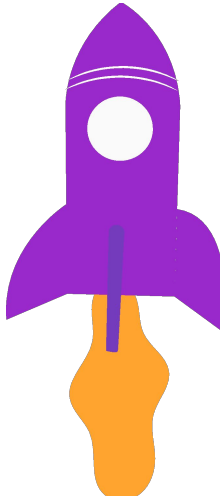
node --version



Verificar la instalación

En la **ventana Terminal, o de Línea de Comandos, de tu sistema operativo, también puedes verificar la correcta instalación de Node JS**.

El uso de una u otra opción, es indistinto. Dependerá de lo que más cómodo te resulte.



Node JS

Los elementos claves que encontraremos en un proyecto Node.js

El archivo **package.json** será un archivo que se crea dentro de cada proyecto Node.js.

Es clave, y no debe eliminarse de la carpeta de cada proyecto que desarrollemos.

El archivo **.gitignore**. Por defecto no existe. De necesitarlo lo crearemos nosotras de forma manual. Es útil para no subir al repositorio de archivos información sensible.

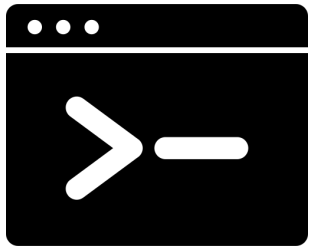
La carpeta **Node_Modules**, se creará cuando comencemos a instalar librerías, frameworks y otras dependencias para utilizar en nuestros proyectos Node.js.

Uso de la ventana terminal

Uso de la ventana Terminal

En **la ventana Terminal** se suelen combinar comandos propios del sistema operativo, como crear Carpetas, Archivos, etc... con los comandos propios de Node JS.

Todos los sistemas operativos de escritorio cuentan con una ventana de Terminal, o Línea de comandos. Mac y Linux comparten muchas cosas en común, por lo tanto, su estructura de línea de comandos es similar.

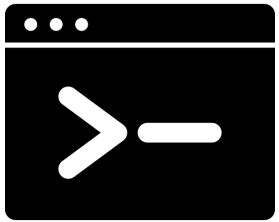


Uso de la ventana Terminal

Windows utiliza la clásica ventana de línea de comandos, pero desde hace algunos años permite instalar una herramienta adicional la cual brinda una ventana Terminal similar a Mac y Linux.

Si utilizas Windows, te sugerimos que instales esta aplicación oficial de Microsoft, para que puedas acostumbrarte al entorno Terminal desde tus primeros pasos con él.

Esto te hará más fácil adaptarte, en un futuro cercano, a tener que trabajar desde una computadora con Mac o Linux.



Uso de la ventana Terminal

Comandos más utilizados en la ventana Terminal

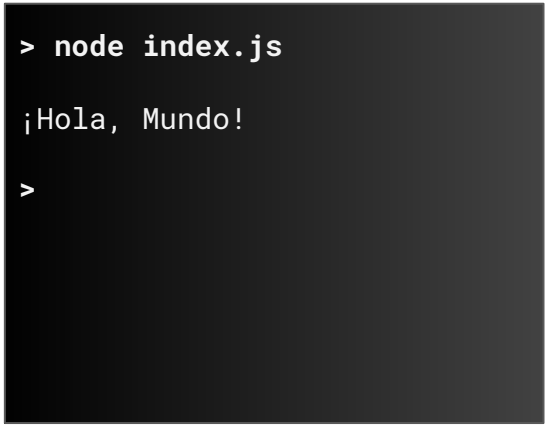
COMANDO	DESCRIPCIÓN	COMANDO	DESCRIPCIÓN
md	crear carpeta/directorio	node --version	verificar la versión instalada de Node JS
cd folder	ingresar a una carpeta/directorio	node -v	mismo comando anterior, acotado
ls	listar archivos y subcarpetas	clear	limpiar el contenido de la pantalla
dir		cls	

Uso de la ventana Terminal

También, la ventana Terminal, será la herramienta desde donde ejecutaremos las aplicaciones backend, o cualquier archivo JS con lógica.

Para ello, usaremos el comando **node** seguido del nombre del archivo JS.

El resultado de las operaciones/tareas resultantes de este código, se verán en la Consola o ventana Terminal.



Nivelación / Repaso

Nivelación / Repaso

Previo a sumergirnos de lleno en el mundo de Node.js, realizaremos un repaso de los conceptos más importantes de JS.

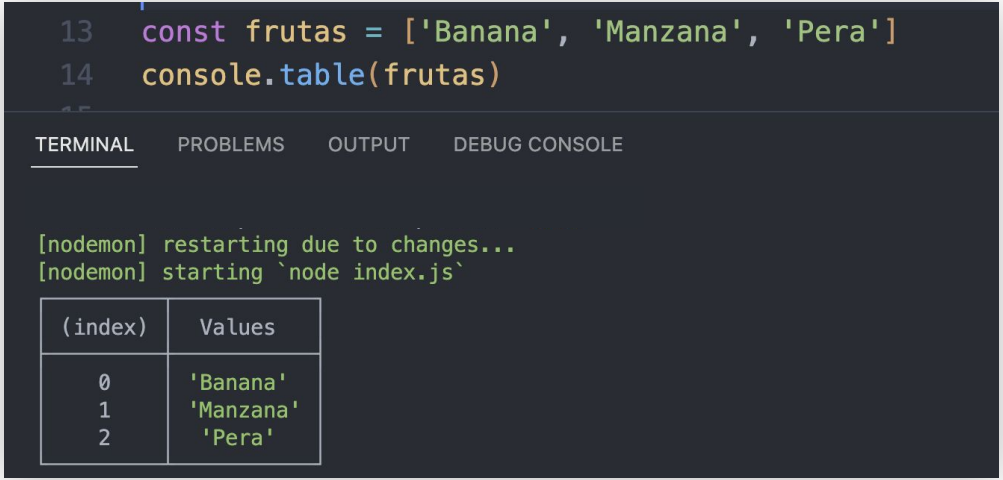
Seguramente conozcas la mayoría, pero como **JS es un lenguaje que evoluciona cada 6 semanas** (*literalmente*), seguro encontrarás temas que aún no conoces o no llegaste a experimentar con ellos.

El objeto Console

El objeto **console** y los métodos **.log()**, **.warn()** y **.error()**, son mensajes en la consola que denotan diferentes tonos de información.

Se usan para depurar de forma correcta en JS. **En JS del lado del servidor, se suelen ver todos con la impronta del método .log().**

El objeto Console



El objeto console: el método **.table()** es un método del objeto console para **estructurar la información de array de elementos y array de objetos**.

Este sí está disponible en Node JS aunque, cuando el contenido del array es demasiado, suele romperse la estructura de tabla.

Variables

Hello World

```
let nombreCompleto = "Joe McMillian";
let ocupacion = "Founder";
let empresa = "McMillian Security";
```

La declaración de variables se realiza a través de la **palabra reservada let**.

También tenemos disponible el uso de la **palabra reservada var** pero, desde la versión 2015 de EcmaScript, la misma se desestima para la implementación de código JS moderno.

constantes

constantes JS

```
const empresa = 'CARDIFF COMPUTERS';

console.log(empresa);
```

Las constantes también son un tipo de variable, pero tal como su nombre lo indica, su valor no puede cambiar "*es constante a lo largo del ciclo de vida de la aplicación*".

Variables vs. Constantes

¿Cuándo usar variables y cuándo usar constantes?

Todo aquel valor que puede o requiere cambiarse en algún momento debe ser declarado como una variable.

Aquellos valores que deben ser estáticos o inalterables, debemos declararlos como constante.

Funciones

funciones JS

```
let nombreCompleto = "Joe McMillian";

function mostrarMiNombre() {
    console.log("2- Me llamo: ", nombreCompleto);
}
mostrarMiNombre()
```

Las funciones se utilizan exactamente igual que cuando creamos aplicaciones web con JavaScript.

Se estructuran con la palabra reservada function, y luego son invocadas en la parte de nuestra aplicación que necesite utilizarlas.

Funciones

funciones JS

```
let nombreCompleto = "Joe McMillian";

function mostrarMiNombre(nombre) {
    console.log("2- Me llamo: ", nombre);
}
mostrarMiNombre(nombreCompleto)
```

Pueden recibir uno o más parámetros.

Si son más de un parámetro, se deben separar por una coma uno de otros.

El resultado siempre es el mismo.

Funciones

funciones JS

```
function obtenerMaximo(num1, num2, num3) {
    return Math.max(1, 4, 10);
}
obtenerMaximo(2, 5, 10);  //devuelve 10
```

También podemos utilizar las funciones con retorno y/o parámetros.

Para ello, **la palabra reservada return**, será nuestra aliada.

Condicionales

condicionales

```
const nombreCompleto = 'Joe McMilliam';

if (nombreCompleto !== '') {
    console.log(`Bienvenid@ `${nombreCompleto}`);
} else {
    console.error(`No se reconoce el usuario, o el usuario está vacío`);
}
```

Los condicionales son básicamente aquellas expresiones que **nos permiten ejecutar código en base al resultado del análisis de una expresión**.

if - else son los condicionales más utilizados en JS, en casi todas las situaciones y lógica de código.

Condicionales

condicionales

```
const nombreCompleto = 'Joe McMilliam';
const login = false;

if (nombreCompleto !== '' && login === false) {
    console.log(`Bienvenid@ `${nombreCompleto}`);
    login = true;
} else {
    console.error(`No se reconoce el usuario, o el usuario está vacío`);
}
```

Podemos utilizar operadores lógicos (**AND** y **OR**), dentro del análisis de una o más expresiones.

Si utilizamos **AND (&&)**, ambas expresiones deben devolver como resultado TRUE.

Si utilizamos **OR (||)**, una de las dos expresiones evaluadas debe devolver como resultado TRUE.

Condicionales

Para este último escenario, tengamos siempre presente la Tabla de la Verdad. La misma es originaria del mundo de las matemáticas, y se aplica con la misma lógica en el mundo de la programación.

Es muy efectiva cuando de evaluar múltiples expresiones se trata.

a	b	$c = f(a, b)$
V	V	V
V	F	F
F	V	V
F	F	V

Condicionales

condicionales

```
const nombreCompleto = 'Joe McMilliam';

if (nombreCompleto !== '' && grupo === 'admin')
    console.log(`Bienvenid@ `${nombreCompleto}`);
else
    console.error('No se reconoce el usuario o el usuario está vacío.');
```

JS moderno permite simplificar la estructura condicional if, if - else.

Si la misma resuelve la acción precedida por la condición en una sola línea de código, también se puede prescindir del uso de las llaves de bloque.

Ciclos de iteración

Como todo lenguaje de programación, JavaScript también cuenta con ciclos de iteración.

Estos ciclos permiten repetir una tarea N cantidad de veces.

Esta repetición se realiza de forma controlada (*de principio a fin*), o de forma indefinida (*hasta que alguien indique cuándo terminar*).

Ciclos de iteración

Las repeticiones controladas se dan de la mano del ciclo for. En este se estructura una serie de parámetros, que le indican cuándo comienza y cuándo termina.

Este tipo de iteración se la denomina: "*Ciclo por conteo*".

Veamos un ejemplo:

Ciclos de iteración

Ciclos por conteo

```
const paisesLatinos = ['Argentina', 'Bolivia', 'Chile', 'Uruguay', 'Paraguay'];

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

En este ejemplo, sabemos que el array dispone de 5 elementos.

En **el ciclo for**, le indicamos que recorra el array desde la primera posición hasta la última, indicada por **la propiedad .length**.

Así es como implementamos un "*ciclo por conteo*".

Ciclos de iteración

La estructura de repetición (iteración) de forma indefinida, se da de la mano de while. En este otro caso, **el ciclo while ejecutará una tarea determinada, nunca, o repetidas veces**.

En este caso, while evalúa una expresión la cual debe dar como resultado un valor booleano del tipo true.

Allí, el ciclo while comienza a iterar, y seguirá haciéndolo hasta tanto ese valor booleano cambie a **false**.

Ciclos de iteración

El ciclo while inicia su ejecución porque se cumple la condición de la variable ejecutar la cual, al evaluarla, devuelve un valor true.

Debemos tener cuidado con este ejemplo, porque no estamos controlando dentro del ciclo while, el cambio de valor de la variable.

El **no cambiar su valor**, lleva este ciclo a lo que se conoce como **loop infinito**.

Ciclos por repetición

```
let ejecutar = true;

while(ejecutar) {
  console.log('Itero mediante el ciclo while');
}
```

Ciclos de iteración

En este otro caso, la variable ejecutar inicia su valor en false.

Dada esta condición, el ciclo while no iterará en ningún momento, porque, al ejecutar dicha línea de código, la condición esperada no se está cumpliendo.

Ciclos por repetición

```
let ejecutar = false;

while(ejecutar) {
  console.log('Itero mediante el ciclo while');
}
```

Ciclos de iteración

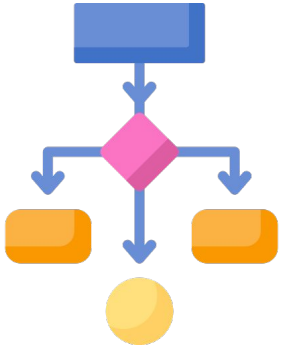
La estructura de repetición (iteración) de forma indefinida, se da de la mano de **do - while**.

A diferencia del ciclo while simple, **la expresión evaluada se hace al final del ciclo do lo que conlleva a que este ciclo se ejecute, al menos, una vez**.

try - catch - finally

Una pieza clave que existe desde los orígenes de javascript, es el uso de **la estructura try - catch - finally**.

Corresponde a un bloque de código que ejecuta una tarea específica en el código de JS y, ante cualquier error inesperado, atrapa el mismo y lo podemos controlar de forma efectiva.



try - catch - finally

Su estructura es muy simple. Dentro del apartado **try {...}**, definimos una operación o tarea de JS.

Si todo va bien, se ejecuta ese código y finaliza la operación de la instrucción en cuestión.

try catch finally

```
try {
  console.log("Intento hacer algo...");
}
catch(error) {
  ...
}
```

try - catch - finally

En el apartado **catch {...}**, definimos una operación que sólo se ejecutará si por algún motivo ocurre un error inesperado dentro del bloque de ejecución contenido en **try {...}**.

Catch nos permite capturar el error y/o manejarlo.

try catch finally

```
try {
  console.log("Intento hacer algo");
} catch(error) {
  console.error("Ocurrió un error inesperado.", error);
}
```

try - catch - finally

Finalmente, y de manera opcional, podemos incluir la sentencia **finally {...}**.

La misma ejecutará otro bloque de código, más allá de que el código controlado anteriormente, haya ido por buen camino, o se haya interceptado algún tipo de error.

try catch finally

```
try {
  console.log("Intento hacer algo");
} catch(error) {
  console.error("Ocurrió un error inesperado.", error);
} finally {
  console.warn("Este mensaje lo verás siempre");
}
```

Muchas gracias.

Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

primero
la gente