



UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL  
FACULTAD REGIONAL BUENOS AIRES



# Preguntas?



# Lo que vamos a ver hoy!

- ❑ Qué es NodeJS?
- ❑ Funcionamiento
- ❑ Uso de NPM
- ❑ Introducción a programación asincrónica



# NodeJS



# Qué es NodeJS?

# NodeJS

## Qué es NodeJS?

Es una de las principales tecnologías de desarrollo del lado servidor.

Es utilizado por empresas como Netflix, PayPal, LinkedIn, Uber, Walmart, Ebay...

Se basa en JavaScript.

# NodeJS

## Qué es NodeJS?

### Ventajas:

- Uso del mismo lenguaje en el cliente (JS) y en el servidor
- Reutilización de código (cliente y servidor)
- Usa el motor V8 de Chrome
- Entrada/Salida sin bloqueo
- Orientado a eventos
- Liviano
- Gran cantidad de paquetes

# INSTALACIÓN





# Instalación

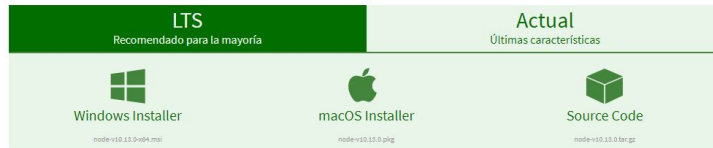
<https://nodejs.org/es/download/>



## Descargas

Versión actual: 10.13.0 (includes npm 6.4.1)

Descargue el código fuente de Node.js o un instalador pre-compilado para su plataforma, y comience a desarrollar hoy.



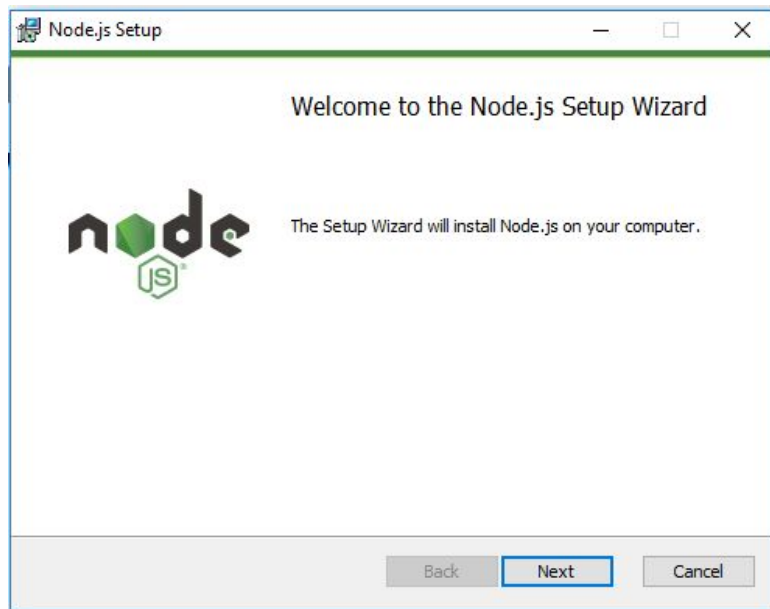
descargar la versión msi

Windows Installer (.msi)  
Windows Binary (.zip)  
macOS Installer (.pkg)  
macOS Binary (.tar.gz)  
Linux Binaries (x64)  
Linux Binaries (ARM)  
Source Code

32-bit	64-bit	
32-bit	64-bit	
	64-bit	
	64-bit	
	64-bit	
ARMv6	ARMv7	ARMv8
node-v10.13.0.tar.gz		

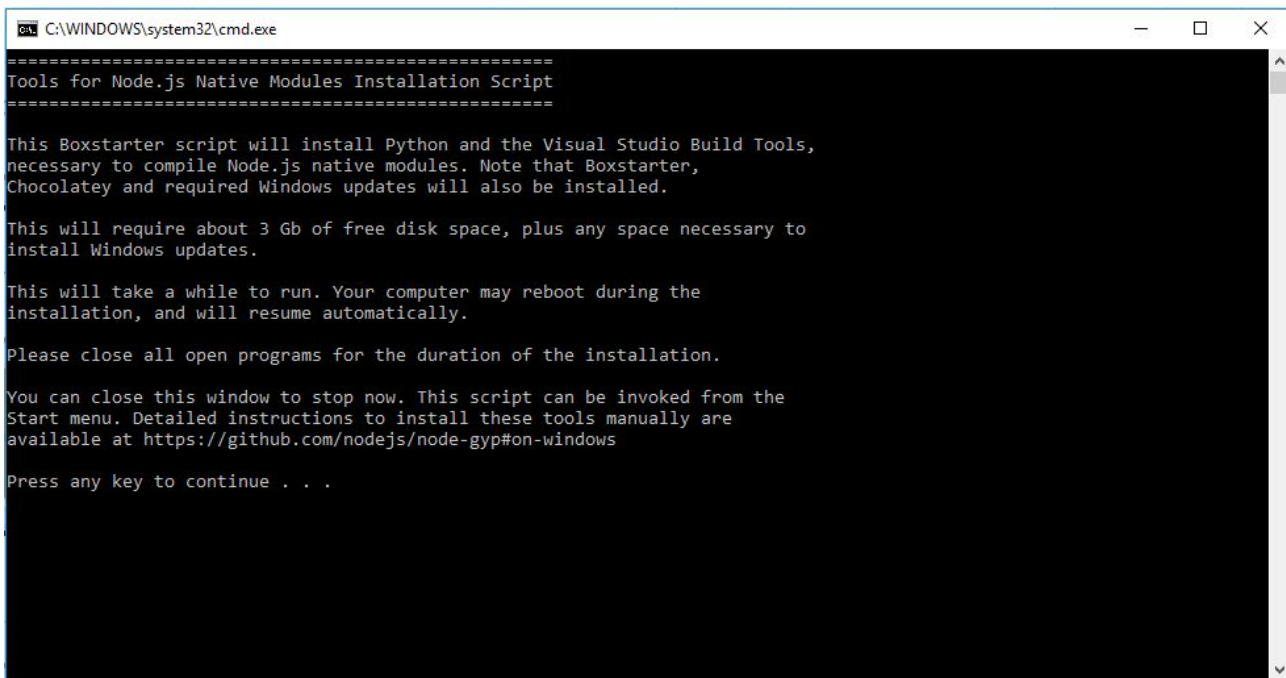
Plataformas adicionales

# Instalación



# Instalación

Al finalizar la  
instalación  
aparece esta  
pantalla



```
C:\WINDOWS\system32\cmd.exe

Tools for Node.js Native Modules Installation Script

This Boxstarter script will install Python and the Visual Studio Build Tools,
necessary to compile Node.js native modules. Note that Boxstarter,
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. Your computer may reboot during the
installation, and will resume automatically.

Please close all open programs for the duration of the installation.

You can close this window to stop now. This script can be invoked from the
Start menu. Detailed instructions to install these tools manually are
available at https://github.com/nodejs/node-gyp#on-windows

Press any key to continue . . .
```



# NPM



# Instalación

## **NPM**

Gestor de módulos para NodeJS.

Se instala automáticamente con el NodeJS.

# Instalación

## NPM

Para conocer la versión de node y npm que tenemos instalado (y así también saber si están instalados) usar desde la terminal, los siguientes comandos:

```
node -v
```

```
npm -v
```

# Instalación de módulos para NodeJS

# Instalación

## NPM - Instalación de módulos para NodeJS

Hay 2 formas de instalar módulos: local o globalmente

### Localmente (recomendado)

El módulo deseado se instalará localmente en el proyecto que estemos trabajando, en una carpeta llamada node\_modules.

```
$npm install [nombre_modulo] .
```

La carpeta node\_modules se crea automáticamente al instalar un módulo.



# Instalación

## NPM - Instalación de módulos para NodeJS

### Globalmente

Algunos módulos/aplicaciones, se pueden instalar para usarse desde cualquiera de nuestros proyectos.

```
$npm install -g [nombre_modulo] .
```

No es muy recomendable porque si actualizamos la versión del módulo/aplicación por un proyecto, estaremos afectando a todos los demás.

# Instalación

## NPM - Instalación de módulos para NodeJS

Ver la documentación de un módulo

(siempre que tenga el archivo .md creado):

```
$npm docs [nombre_modulo] .
```

Se abre el navegador y va a la página de la documentación del módulo.

# Instalación

## NPM - Instalación de módulos para NodeJS

### Utilizar los módulos

Desde el proyecto en Node.js

```
var modulo = require('modulo'); .
```

# Package.json

# Instalación

## NPM - package.json

Es un archivo fundamental para trabajar mejor y más fácil con npm.

Ventajas de usar package.json en nuestros proyectos

- No tenemos que instalar módulos uno a uno ya que se van a descargar de forma automática.
- Facilita la instalación de nuestra aplicación a otros desarrolladores.
- Todos los archivos y documentación de una determinada aplicación se almacena en un solo lugar.

# Instalación

## NPM - package.json

Se debe crear en el raíz de nuestro proyecto. La estructura quedará

```
app.js  
package.json  
node_modules
```

### Estructura básica de package.json

```
{  
  "name": "miapp",  
  "version": "0.0.1",  
  "dependencies": {  
    "nombre_modulo": "version",  
    "nombre_modulo2": "version"  
  }  
}
```



# Instalación

## NPM - package.json

No es necesario instalar de a uno los diferentes módulos. Sólo se debe ejecutar por línea de comandos:

```
$npm install
```

Entonces:

1. NPM lee las dependencias incluídas en el archivo package.json
2. Instala automáticamente los módulos necesarios

A su vez, al compartir nuestra aplicación, no será necesario copiar la carpeta node\_modules ya que se podrá generar automáticamente.

# Correr una aplicación



# Instalación

## NPM - Correr una aplicación - Nivel básico

1. Crear la carpeta donde va a estar nuestro proyecto
2. Escribir en un archivo, el programa a ejecutar. Nombrarlo con extensión .js
3. En la consola ejecutar `node nombre_archivo.js` y oprimir la tecla Enter



# Instalación

## NPM - Correr una aplicación - Utilizando package.json

1. Crear la carpeta donde va a estar nuestro proyecto
2. Ejecutar el comando `$npm init -f`

Esto creará en la carpeta, el archivo package.json que tendrá la siguiente forma:

```
{
  "name": "primerProyecto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

# Instalación

## NPM - Correr una aplicación - Utilizando package.json

3. Escribir en un archivo, el programa a ejecutar. Nombrarlo con extensión .js
4. Modificar package.json de la siguiente manera:

```
{
  "name": "primerProyecto",
  "version": "1.0.0",
  "description": "",
  "main": "primerPrograma.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node primerPrograma"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```



# Instalación

## NPM - Correr una aplicación - Utilizando package.json

5. En la consola, en la carpeta del proyecto, ejecutar el comando `npm start`

# Introducción a programación asincrónica

## Introducción a programación asincrónica

### *Ejemplo de proceso sincrónico*

1. Son las 11PM y aún estoy programando para llegar con una entrega
2. Tengo hambre
3. Guardo los cambios del código (commit?)
4. Camino hasta la pizzería cerca de mi casa
5. Pido la pizza y espero que me la entreguen
6. Vuelvo a casa, me como la pizza

## Introducción a programación asincrónica

### *Ejemplo de proceso **asincrónico***

1. Son las 11PM y aún estoy programando para llegar con una entrega
2. Tengo hambre
3. Llamo a la pizzería y hago mi pedido
4. Continúo con mi desarrollo
5. Luego de un rato suena el timbre <- Evento
6. Guardo los cambios
7. Salgo, pago la pizza, me como la pizza

## Introducción a programación asincrónica

método asincrónico

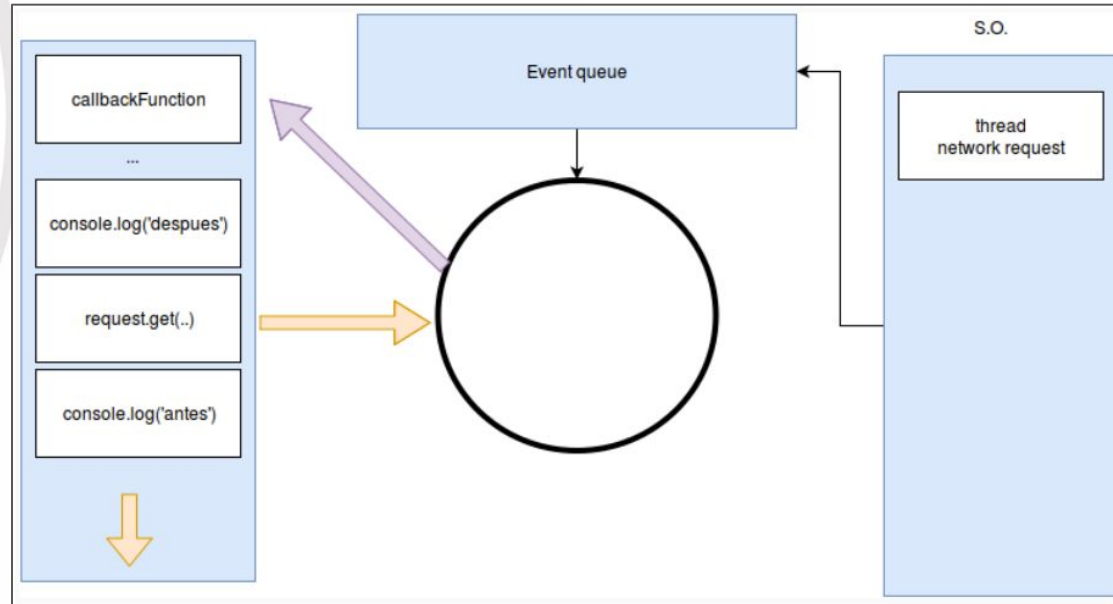
```
console.log('antes');  
request.get(  
  {url: 'https://www.google.com?q=UTN'},  
  | callbackFunction  
);  
console.log('después');
```

antes  
después  
...



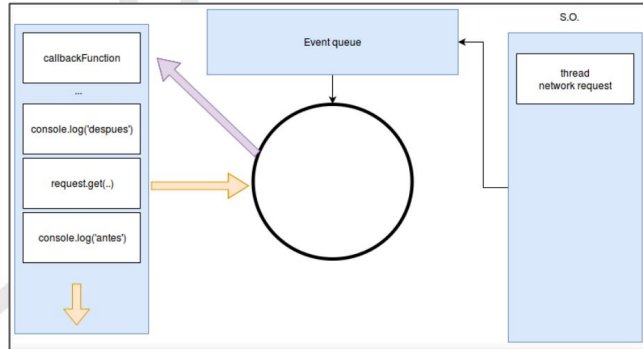
## Introducción a programación asíncrona

### Esquema de ejecución del ejemplo



## Introducción a programación asíncrona

### Esquema de ejecución del ejemplo



- Se ejecuta el primer `console.log`
- Se realiza una petición asíncrona a un servidor de Internet (se realiza la petición y sigue la ejecución del código en la línea siguiente de inmediato - `console.log`-, sin esperar la respuesta)
- Se ejecuta el segundo `console.log`
- ...
- Cuando se recibe la respuesta del servidor de Internet, se llama a la función `callbackFunction` (que fue definida por el usuario en otra sección del código)

## Introducción a programación asincrónica

### Ventajas de la programación Asincrónica

- Código más limpio
- Permite mejor performance de la aplicación
- Es “casi” obligatorio utilizarlo en NodeJS

## Introducción a programación asincrónica

### Formas de programación asíncrona en NodeJS

- Callbacks
- Promises
- Generators
- Async/Await

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### Callbacks

Es una función que es llamada al finalizar un proceso asincrónico.

En el ejemplo del pedido de pizza, sería una función que es llamada cuando suena el timbre (se terminó el proceso de pedido de pizza).


## Introducción a programación asincrónica

### Formas de programación asíncrona

#### Callbacks - Ejemplo en NodeJS

```
function hacerALos3seg() {  
  console.log('Me llamo luego de 3 segundos');  
}
```

```
console.log("Paso 1");  
setTimeout(hacerALos3seg, 3000);  
console.log("Paso 2");
```



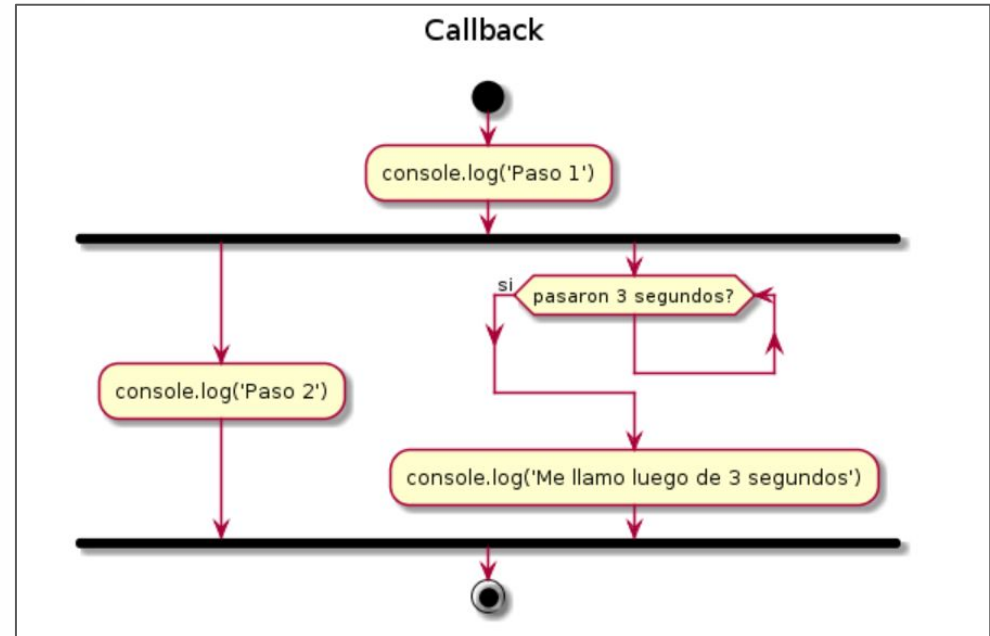
```
Paso 1  
Paso 2  
Me llamo luego de 3 segundos
```

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### Callbacks - Ejemplo en NodeJS

**hacerALos3seg()** es una función que es usada como callback, es decir, que va a ser llamada al finalizar un proceso (en este caso, cuando pasen 3 segundos)





PENSEMOS...





## Introducción a programación asincrónica

### Callbacks

Código del ejemplo de la pizzería



Cuál sería la salida por pantalla?

```
// Ejemplo de proceso de pedido de pizza async con callbacks

// Pizzeria
function pedirPizza(tipoDePizza, entregarPizzaFn) {
  console.log("Estoy cocinando");
  // ....
  // Cuando termino de hacer la pizza
  entregarPizzaFn();
}

// Mi casa
console.log("Por llamar a la pizzeria");
pedirPizza("Pizza de palmitos", function() {
  console.log("llego la pizza");
});
console.log("Sigo trabajando normalmente");
```



# SIGAMOS...



## Introducción a programación asincrónica

### Formas de programación asíncrona

#### Callback Hell

El método asincrónico por medio de Callbacks tiene una gran desventaja: el código puede quedar rápidamente ilegible y muy difícil de seguir.

A esta desventaja, se la llama Callback Hell (Infierno de Callbacks).

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### Callback Hell - Ejemplo

```
// Ejemplo de callback hell

function doAsync1(fn) {
  console.log("doAsync1");
  fn();
}

function doAsync2(fn) {
  console.log("doAsync2");
  fn();
}

function doAsync3(fn) {
  console.log("doAsync3");
  fn();
}

function doAsync4(fn) {
  console.log("doAsync4");
  fn();
}

console.log("Iniciando");
doAsync1(function () {
  doAsync2(function () {
    doAsync3(function () {
      doAsync4(function () {
        // ...
      })
    })
  })
});

console.log("Finalizando");
```

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### *Conclusión de Callbacks*

##### **Ventajas**

- Muy conocidas (posiblemente las hayas usado)
- Se usa en muchas funciones
- Es sencillo para operaciones básicas

##### **Desventajas**

- El código puede quedar ilegible fácilmente

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### Promises

Es una abstracción de la programación asincrónica. Es un objeto que hace de “interlocutor” entre la función que hace una operación asincrónica, y el llamador. Este “interlocutor” maneja la respuesta de la función asincrónica (ya sea un valor o una excepción).

**Definición simplificada:** es el resultado de una operación asincrónica, cuyo estado puede ser pendiente, completada, o rechazada.

## Introducción a programación asíncronica

### Formas de programación asíncrona

### Promises - Ejemplo en NodeJS

Función	Descripción
<u>funcionQueProcesaLaRespuestaOk</u>	Es la función que es llamada cuando la respuesta del proceso asíncronico es correcta (sin errores). Solo muestra el mensaje ok
<u>funcionQueProcesaElError</u>	Es la función que es llamada cuando la respuesta del proceso asíncronico retorna un error. Solo muestra el mensaje Con error
<u>procesoAsincronicoConPromise</u>	Es la función asíncronica, la cual puede retornar una respuesta correcta (usando el método resolve()) o un error por medio de la función (reject())

```
function funcionQueProcesaLaRespuestaOk() {
  console.log('ok');
}

function funcionQueProcesaElError() {
  console.log('Con error');
}

function procesoAsincronicoConPromise() {
  return new Promise(function(resolve, reject) {
    let todoOk = true;
    if (todoOk) {
      resolve();
    } else {
      reject();
    }
  });
}

// Forma 1 de llamar una Promise
console.log('Paso 1');
procesoAsincronicoConPromise()
  .then(funcionQueProcesaLaRespuestaOk)
  .catch(funcionQueProcesaElError);
console.log('Paso 2');

// Forma 2 de llamar una Promise
procesoAsincronicoConPromise()
  .then(funcionQueProcesaLaRespuestaOk, funcionQueProcesaElError);
```

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### Promises - Ejemplo en NodeJS

Existen 2 formas de llamar a una función que retorna una Promise

```
funcionQueRetornaPromise().then(funcionQueProcesaLaRespuestaCorrecta).catch(funcionQueProcesaError);
```

```
funcionQueRetornaUnaPromise().then(funcionQueProcesaLaRespuestaCorrecta, funcionQueProcesaError);
```

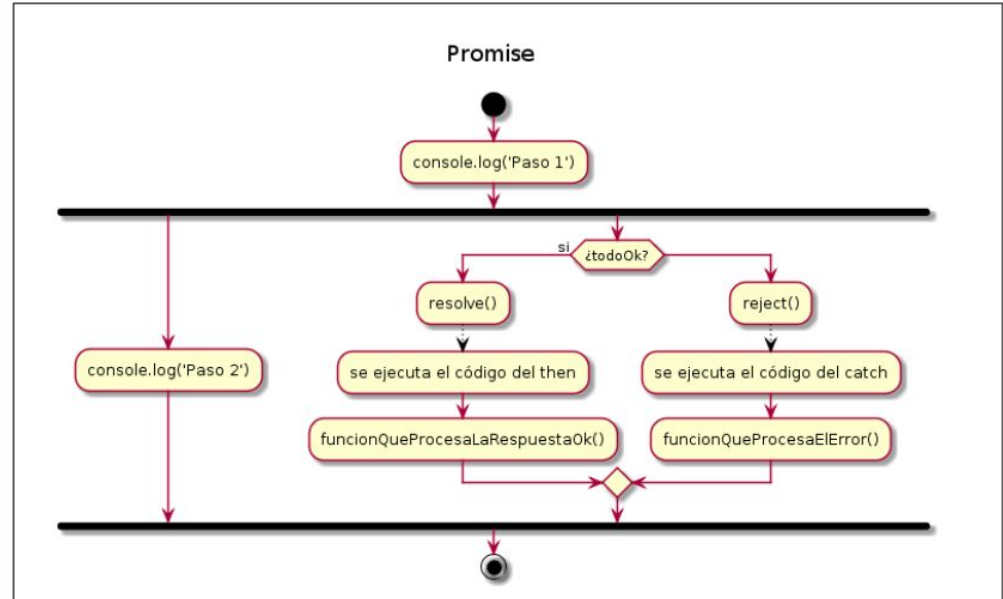


## Introducción a programación asíncrona

### Formas de programación asíncrona

#### Promises - Ejemplo en NodeJS

Paso 1  
Paso 2  
ok





PENSEMOS...



## Introducción a programación asincrónica

### Formas de programación asíncrona

#### Promises



Ejercicio:

Cómo es la salida por pantalla si se cambia

`let todoOk = true;` por `let todoOk = false;`?



# SIGAMOS...



## Introducción a programación asíncrona

### Formas de programación asíncrona

#### Promises - Ejemplo de la pizzería

Tanto a `resolve()` como a `reject()` se le pueden pasar parámetros.

Los parámetros serán pasados a las funciones que procesan la respuesta.

```
// Ejemplo de proceso de pedido de pizza async con promises

// Pizzeria (Generalmente el código de la pizzería va a estar implementado por la
operación que deseamos usar)
var pedirPizza = new Promise(function(resolve, reject) {
  console.log("Estoy cocinando");
  // Aca se hacen todos los procesos

  var todoSalioBien = false;
  if ( todoSalioBien ) {
    resolve(); // Todo ok => retorno por curso normal
  }
  else {
    reject(Error('Hubo un error en tu pedido')); // Hubo un error => lo retorno por
error
  }
});

// Mi casa
pedirPizza.then(function() {
  console.log( "Llego la pizza ok" );
}).catch(function( errorMsg ) {
  console.log("No me pudieron entregar la pizza y me llamaron para decirme "+
errorMsg );
});
```

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### Promises - Ejemplo de la pizzería

Ejemplo	Descripción
resolve('mensaje ok')	<p>Pasa como parámetro 'mensaje ok' a la función que se pase como parámetro en el .then(). Ej: para <code>.then(funcion1)</code></p> <p>Pasa como parámetro 'mensaje ok' a funcion1 <code>funcion1('mensaje ok')</code></p>
reject('mensaje error')	<p>Pasa como parámetro 'mensaje error' a la función que se pasa como parámetro a .catch. Ej: para <code>.catch(funcionError)</code></p> <p>Pasa como parámetro 'mensaje error' a <code>funcionError</code>. <code>funcionError('mensaje error')</code></p>

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### *Promises - Concatenación del procesamiento de la respuesta*

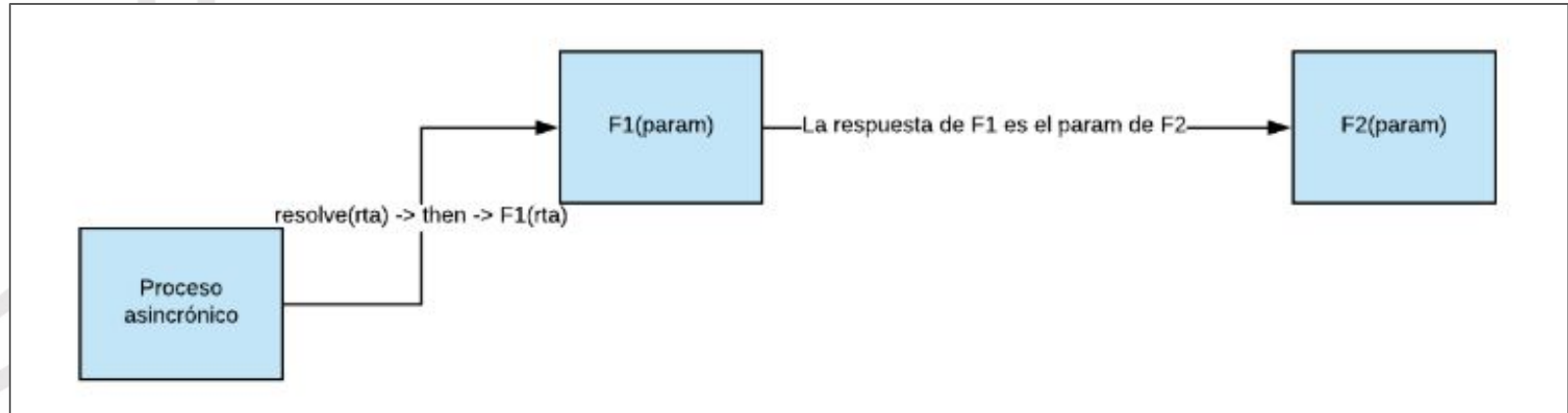
Se pueden concatenar funciones para dividir el procesamiento de la respuesta.

Esto permite que varias funciones procesen la respuesta, o que cada función vaya convirtiendo la respuesta a un formato que necesite la siguiente función.

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### *Promises - Concatenación del procesamiento de la respuesta*





## Introducción a programación asincrónica

### Formas de programación asíncrona

#### *Promises - Concatenación del procesamiento de la respuesta*

Ejemplo	Descripción
<u>fnAsinc()</u> .then(f1).then(f2);	<ul style="list-style-type: none"><li>• El <b>parámetro</b> pasado a <u>resolve()</u> dentro de <u>fnAsinc</u> es pasado como <b>parámetro</b> a f1</li><li>• El <b>valor</b> que retorna f1, es pasado como <b>parámetro</b> a f2</li></ul>

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### *Promises - Concatenación - Ejemplo pizzería*

La secuencia de ejecución del código es la siguiente:

1. Se muestra el mensaje “Paso 1”
2. Se llama al método `procesoAsincronicoConPromise()`  
-> Continúa en A
3. Se muestra mensaje “Paso 2”

```
function funcionQueProcesaLaRespuestaOk(respuesta) {
  console.log('ok', respuesta);
  return '-pase por respuesta ok 1-';
}
function funcionQueProcesaAlFinalRespuestaOk(respuesta) {
  console.log('ok2', respuesta);
}
function funcionQueProcesaElError(error) {
  console.log('Con error', error);
}

function procesoAsincronicoConPromise() {
  return new Promise(function(resolve, reject) {
    let todoOk = true;
    if (todoOk) {
      resolve('-pase por resolve-');
    } else {
      reject();
    }
  });
}

// Forma 1
console.log('Paso 1');
procesoAsincronicoConPromise()
  .then(funcionQueProcesaLaRespuestaOk)
  .then(funcionQueProcesaAlFinalRespuestaOk)
  .catch(funcionQueProcesaElError);

console.log('Paso 2');
```

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### *Promises - Concatenación - Ejemplo pizzería*

- A. `procesoAsincronicoConPromise()` llama a la función `resolve` pasando como parámetro “-pase por resolve-”
- B. La función `resolve()` finaliza la Promise y llama a la primer función que encuentra en el `.then`, la cual es `funcionQueProcesaLaRespuestaOk`, y le pasa como parámetro “-pase por resolve-”
- C. La función `funcionQueProcesaLaRespuestaOk()` muestra el mensaje:  
ok  
-pase por resolve-

```
function funcionQueProcesaLaRespuestaOk(respuesta) {
  console.log('ok', respuesta);
  return '-pase por respuesta ok 1-';
}
function funcionQueProcesaAlFinalRespuestaOk(respuesta) {
  console.log('ok2', respuesta);
}
function funcionQueProcesaElError(error) {
  console.log('Con error', error);
}

function procesoAsincronicoConPromise() {
  return new Promise(function(resolve, reject) {
    let todoOk = true;
    if (todoOk) {
      resolve('-pase por resolve-');
    } else {
      reject();
    }
  });
}

// Forma 1
console.log('Paso 1');
procesoAsincronicoConPromise()
  .then(funcionQueProcesaLaRespuestaOk)
  .then(funcionQueProcesaAlFinalRespuestaOk)
  .catch(funcionQueProcesaElError);

console.log('Paso 2');
```

## Introducción a programación asíncronica

### Formas de programación asíncrona

#### *Promises - Concatenación - Ejemplo pizzería*

- D. La función `funcionQueProcesaLaRespuestaOk()` retorna la cadena “-pase por respuesta ok 1-”
- E. Se ejecuta la siguiente función de la Promise (el próximo `then`), que es la función `funcionQueProcesoAlFinalRespuestaOk()` y se le pasa como parámetro lo retornado por `funcionQueProcesaLaRespuestaOk()`, que es “-pase por respuesta ok 1-”
- F. La función `funcionQueProcesoAlFinalRespuestaOk` muestra el mensaje:  
ok 2  
-pase por respuesta ok 1-

```
function funcionQueProcesaLaRespuestaOk(respuesta) {  
  console.log('ok', respuesta);  
  return '-pase por respuesta ok 1-';  
}  
function funcionQueProcesoAlFinalRespuestaOk(respuesta) {  
  console.log('ok2', respuesta);  
}  
function funcionQueProcesaElError(error) {  
  console.log('Con error', error);  
}  
  
function procesoAsincronicoConPromise() {  
  return new Promise(function(resolve, reject) {  
    let todoOk = true;  
    if (todoOk) {  
      resolve('-pase por resolve-');  
    } else {  
      reject();  
    }  
  });  
}  
  
// Forma 1  
console.log('Paso 1');  
procesoAsincronicoConPromise()  
  .then(funcionQueProcesaLaRespuestaOk)  
  .then(funcionQueProcesoAlFinalRespuestaOk)  
  .catch(funcionQueProcesaElError);  
  
console.log('Paso 2');
```

## Introducción a programación asincrónica

### Formas de programación asíncrona

#### *Conclusión de Promises*

#### Ventajas

- Código más legible.
- Es la forma más habitual de trabajar con procesos asincrónicos.
- Se utiliza en varios frameworks JavaScript (Angular, React, Node)
- Podemos concatenar funciones que procesan la respuesta.
- Podemos enfocarnos y trabajar por “partes”, dividiendo un problema complejo en varios más pequeños.

#### Desventajas

- Existen algunos métodos en NodeJS para los cuales el uso de promises no es la opción predeterminadas (se usan Callbacks)

# PRÁCTICA

# PRÁCTICA NodeJS

## Ejercicio 1

Verificar si está instalado el nodejs y npm. Si no está, instalarlos.

Crear una carpeta para el primer ejercicio en nodejs

Crear el package.json en esa carpeta.

# PRÁCTICA NodeJS

## Ejercicio 2

Descargar los archivos ejercicio01.js y ejercicio02.js

Completar ambos ejercicios con el código necesario para hacerlos funcionar.

Ejecutar cada uno de los ejercicios.



FIN!!!!

