**Módulo 1: Introducción a Node.js, Módulos, Paquetes npm, Eventos y Ficheros - Entrega P2P: Eventos**

**Objetivos**

* Aprender cómo funciona la programación asíncrona basada en eventos implementada por el modulo events y la clase EventEmitter
* Emplar callbacks, clases y módulos de Node.js.

**Descripción de la práctica**

Esta práctica proporciona un programa de ejemplo está formado por 4 ficheros: **habitacion.js**, **climatizador.js**, **termostato.js** y **main.js**. Este programa simula la habitación de una vivienda que tiene instalados un climatizador y un termostato que controla su temperatura. El programa de ejemplo usa del módulo events de Node.js para gestionar los eventos.

**¡Cuidado!** El código dado funciona en UNIX/Linux. Para ejecutar en Windows hay que aplicar los cambios que se indican en las instrucciones sobre compatibilidad UNIX-Windows para node.js.

En esta práctica el alumno deberá realizar dos tareas:

* **Tarea 1:** Ampliar el programa de ejemplo añadiendo un programador. El programador permitirá configurar que temperatura ideal se desea tener en la habitación en todo momento.
* **Tarea 2:** Sustituir el módulo events de Node.js por uno equivalente implementado por el alumno, de forma que el funcionamiento del programa de ejemplo no se vea afectado.

Para ejecutar el programa de ejemplo se debe invocar el comando:

$ node main.js

A continuación se describe cada uno de los ficheros que incluye la práctica:

**El fichero *habitación.js***

El fichero **habitacion.js** es un módulo que exporta la clase **Habitacion**. Cada instancia de la clase **Habitacion** simula una habitación de una vivienda, y mantiene el valor de su temperatura ambiente. Además, cada instancia de **Habitacion** tiene un temporizador que modifica la temperatura aleatoriamente en +/- 1ºC cada 10 segundos. Esta variación aleatoria de la temperatura simula los cambios de estación, o cualquier actividad externa que afecte a la temperatura de la habitación.

**El fichero *climatizador.js***

Este fichero es un módulo exporta la clase **Climatizador**. Cada instancia de la clase **Climatizador** simula un aparato climatizador instalado en una habitación, y que permite aumentar y disminuir la temperatura de la habitación. Toma aire de la habitación y lo expulsa un poco más caliente o un poco más frio.

El constructor de la clase **Climatizador** toma como parámetro un objeto **Habitacion** que representa la habitación donde está instalado el climatizador.

Los objetos climatizador tienen dos métodos: **enfriar** y **calentar**. Se usan para disminuir o aumentar ligeramente la temperatura de la habitación cada vez que se invocan.

**El fichero *termostato.js***

Este módulo exporta la clase **Termostato**.

Cada instancia de la clase **Termostato** simula un termostato instalado en una habitación, que puede configurarse con una temperatura ideal, y que emite eventos informando cuando la temperatura de la habitación es demasiado alta o baja respecto de la ideal. También emite un evento periódico informando de la temperatura actual de la habitación. La clase **Termostato** extiende a la clase **EventEmitter** para gestionar la emisión de eventos y sus escuchadores. El constructor de la clase Termostato toma como parámetro un objeto **Habitacion** que representa la habitación donde está instalado. Cada instancia de **Termostato** tiene tres propiedades:

* La propiedad **habitacion** mantiene una referencia a la habitación donde está instalado el termostato.
* La propiedad **temperaturaIdeal** almacena el valor de la temperatura ideal que se ha configurado.
* La propiedad **intervalId** identifica un temporizador interno que periódicamente sondea la temperatura de la habitación y emite los eventos **"tic"**, **"muchocalor"** y **"muchofrio"**. La propiedad intervalId se necesita para cancelar el temporizador interno cuando se apaga o se vuelve a encender el termostato.

Cada instancia de **Termostato** tiene tres métodos:

* El método **indicarTemperaturaIdeal** se usa para configurar la temperatura ideal deseada en la habitación.
* El método **encender** se usa para encender el termostato, inicializando el temporizador interno que lanza los eventos.
* El método **apagar** se usa para apagar el temporizador, detenido el temporizador interno encargado de lanzar los eventos. Las instancias de Termostato pueden emitir tres eventos: • El evento **"muchocalor"** se emite cuando la temperatura actual de la habitación supera a la temperatura ideal en más grados que los indicados por la constante **MARGEN\_ERROR**. • El evento **"muchofrio"** se emite cuando la temperatura actual de la habitación es inferior a la temperatura ideal en más grados que los indicados por la constante **MARGEN\_ERROR**. • El evento **"tic"** se emite cada medio segundo, y le pasa a los escuchadores como parámetro el valor de la temperatura actual de la habitación.

**El fichero *main.js***

Este fichero es el programa principal. Importa las clases **Habitacion**, **Climatizador** y **Termostato**, y crea una instancia de cada una de ellas. La constante **dormitorio** apunta a una habitación que inicialmente está a 22ºC.

Las constantes **climatizador** y **termostato** apuntan al climatizador y al termostato instalados en el dormitorio anterior. En el termostato se registran tres escuchadores. El primero escucha los eventos **"muchofrio"** para provocar que el climatizador caliente. El segundo escucha los eventos **"muchocalor"** para que provocar que el climatizador enfríe. El último escucha los eventos **"tic"** para mostrar por la consola la temperatura actual del dormitorio. Finalmente, se configura una temperatura ideal a 20ºC y se enciende el termostato.

**Tareas a realizar**

**Tarea 1: Programador**

La primera tarea que debe realizar el alumno es ampliar el programa de ejemplo añadiendo un programador que permita configurar la temperatura que se desea tener en la habitación en todo momento.

Debe crearse un fichero llamado programador.js que implemente un módulo que exporte una clase llamada **Programador**

El constructor de la clase **Programador** debe tomar como parámetro un objeto con la configuración de horas y temperaturas que se desea programar. El objeto con la configuración debe ser un array como el ilustrado a continuación.

[

{ hora: "07:00",

temperatura: 22

},

{ hora: "08:30",

temperatura: 18

},

{ hora: "18:00",

temperatura: 22

},

{ hora: "23:00",

temperatura: 20

}

]

Cada elemento del array es un objeto con dos claves: \* **hora**: El valor asociado a esta clave es un string con la hora a la que debe aplicarse la programación de la nueva temperatura ideal. El formato de la hora debe ser "hh:mm". \* **temperatura**: un número con la nueva temperatura ideal a programar.

Con la configuración del ejemplo se programaría una temperatura de 22ºC todos los días a las 7 de la mañana, 18ºC todos los días a las 8:30, 22ºC todas las tardes a las 6, y 20ºC todos los días a las 11 de la noche. Se recomienda usar el módulo Later.js (<https://bunkat.github.io/later>) para implementar el módulo programador.js. Recuerde instalarlo con `npm install. El módulo Later.js permite planificar instantes de tiempo en los que ejecutar tareas. Por ejemplo, para escribir por consola la palabra "hola" todos los días a las 18:00, se podría usar el siguiente código:

// Importar modulo Later.js:

const later = require('later');

// Usar zona horaria local:

later.date.localTime();

// Crear planificación para todos los dias a las 18:00

const sched = later.parse.text('at 18:00');

// Crear un temporizador que escriba indefinidamente "hola"

// en los instantes planificados anteriormente:

later.setInterval(() => console.log('hola'), sched);

Las instancias de la clase **Programador** deben emitir un evento llamado **"ideal"** cada vez que sea necesario reprogramar la temperatura ideal, siguiendo las instrucciones de la configuración pasada en el constructor. El evento **"ideal"** se emitirá pasando como parámetro el valor de la temperatura ideal a programar en el termostato.

Para poder emitir eventos, la clase **Programador** debe extender a la clase **EventEmitter**.

Finalmente, en el programa principal main.js deben añadirse los siguientes cambios: \* Importar el módulo programador.js para crear un objeto **Programador**. \* Añadir las sentencias necesarias para que cuando el programador emita un evento **"ideal"**, se ajuste la nueva temperatura ideal en el termostato.

**Tarea 2: Events**

En esta tarea se pide al alumno que cree su propia implementación del módulo **events**, y que la integre en el programa de ejemplo, sustituyendo al módulo events proporcionado por Node.js.

La implementación pedida al alumno es una versión reducida de la proporcionada por Node.js. El alumno solo debe implementar los métodos **emit** y **on**.

Para que el módulo implementado por el alumno se integre fácilmente en el programa de ejemplo, deben cumplirse los siguientes requisitos: \* El módulo debe llamarse events.js y situarse junto a los demás ficheros javascripts. \* Para requerirlo desde termostato.js y desde programador.js deberá sustituir las llamadas require("events") por require("./events"). \* El módulo debe exportar una clase que se llame **EventEmitter**. \* Las instancias de **EventEmitter** deben tener una propiedad donde se deben guardar todos los escuchadores que se hayan registrado usando el método **on**. Se debe guardar el nombre de los eventos junto con la lista de escuchadores interesados en cada evento. \* Las instancias de **EventEmitter** deben tener el método **on**, para que los escuchadores se registren. Este método toma como parámetro el nombre de un evento, y el método a ejecutar cuando se emita ese evento. \* Las instancias de **EventEmitter** deben tener el método **emit**, que se emplea para emitir un evento. Este método toma como parámetros el nombre del evento a emitir, y los argumentos que hay que pasar a las funciones escuchadoras interesadas en ese evento.

Una vez sustituido el módulo events de Node.js por el desarrollado por el alumno, el comportamiento del programa de prueba no debe variar.

**Descargar el código del proyecto**

El proyecto debe clonarse en el ordenador desde el que se está trabajando:

$ git clone https://github.com/ging-moocs/MOOC\_node\_mod1-eventos\_entrega

A continuación se debe acceder al directorio de trabajo y abrir el fichero index.html con el editor de la elección del alumno.

$ cd MOOC\_node\_mod1-eventos\_entrega

Para ejecutar el programa de ejemplo se debe invocar el comando:

$ node main.js

**Prueba de la práctica**

Para ayudar al desarrollo, se provee una herramienta de autocorrección que prueba las distintas funcionalidades que se piden en el enunciado. Para utilizar esta herramienta debes tener node.js (y npm) (<https://nodejs.org/es/>) y Git instalados.

Para instalar y hacer uso de la [herramienta de autocorrección](https://www.npmjs.com/package/moocauto) en el ordenador local, ejecuta los siguientes comandos en el directorio del proyecto:

$ npm install -g moocauto ## Instala el programa de test

$ moocauto ## Pasa los tests al fichero a entregar

............................ ## en el directorio de trabajo

... (resultado de los tests)

También se puede instalar como paquete local, en el caso de que no se dispongas de permisos en el ordenador desde el que estás trabajando:

$ npm install moocauto ## Instala el programa de test

$ npx moocauto ## Pasa los tests al fichero a entregar

............................ ## en el directorio de trabajo

... (resultado de los tests)

Se puede pasar la herramienta de autocorrección tantas veces como se desee.

**Entrega de la práctica**

El alumno debe subir un fichero comprimido ZIP incluyendo todos los ficheros de la práctica excepto el directorio node\_modules (si existe).

**Evaluación de la práctica**

La evaluación de la práctica se realizará mediante revisión por pares (P2P). Cada alumno tendrá que revisar la práctica de 3 de sus compañeros y otros 3 revisarán la suya. Se puede utilizar la herramienta de autocorrección (moocauto) como ayuda para revisar la práctica de los compañeros.

El objetivo de este curso es sacar el máximo provecho al trabajo que están dedicando, por lo que les recomendamos que utilicen la evaluación para ayudar a sus compañeros enviando comentarios sobre la corrección del código, su claridad, legibilidad, estructuración y documentación.

Dado que es un curso para principiantes, ante la duda les pedimos que sean benevolentes con sus compañeros, porque muchos participantes están empezando y los primeros pasos siempre son difíciles.

**OJO! Una vez enviada la evaluación, está no se puede cambiar.** Piensen bien su evaluación antes de enviarla.

**RÚBRICA**: Se puntuará el ejercicio a corregir sumando el % indicado a la nota total si la parte indicada es correcta:

* **40%:** Tarea 1
* **60%:** Tarea 2

Módulo 1. Introducción A Node.Js, Módulos, Paquetes Npm, Eventos Y Ficheros

Ejercicio P2P Obligatorio

Las instrucciones del ejercicio están disponibles en el siguiente enlace:  
<https://github.com/ging-moocs/MOOC_node_mod1-eventos_entrega/blob/master/README.md>

**Entrega de la práctica**

El alumno debe subir un fichero comprimido ZIP incluyendo todos los ficheros de la práctica excepto el directorio node\_modules (si existe).

**Instrucciones para la evaluación por pares.**

La evaluación de la práctica se realizará mediante revisión por pares (P2P). Cada alumno tendrá que revisar la práctica de 3 de sus compañeros y otros 3 revisarán la suya. Se puede utilizar la herramienta de autocorrección (moocauto) como ayuda para revisar la práctica de los compañeros.

El objetivo de este curso es sacar el máximo provecho al trabajo que están dedicando, por lo que les recomendamos que utilicen la evaluación para ayudar a sus compañeros enviando comentarios sobre la corrección del código, su claridad, legibilidad, estructuración y documentación.

Dado que es un curso para principiantes, ante la duda les pedimos que sean benevolentes con sus compañeros, porque muchos participantes están empezando y los primeros pasos siempre son difíciles.

**OJO! Una vez enviada la evaluación, está no se puede cambiar.** Piensen bien su evaluación antes de enviarla.

**RÚBRICA**: Se puntuará el ejercicio a corregir sumando el % indicado a la nota total si la parte indicada es correcta:

* **40%:** Tarea 1
* **60%:** Tarea 2

# Módulo 1: Introducción a Node.js, Módulos, Paquetes npm, Eventos y Ficheros - Entrega P2P: Eventos

## Objetivos

* Aprender cómo funciona la programación asíncrona basada en eventos implementada por el modulo events y la clase EventEmitter
* Emplar callbacks, clases y módulos de Node.js.

## Descripción de la práctica

Esta práctica proporciona un programa de ejemplo está formado por 4 ficheros: **habitacion.js**, **climatizador.js**, **termostato.js** y **main.js**. Este programa simula la habitación de una vivienda que tiene instalados un climatizador y un termostato que controla su temperatura. El programa de ejemplo usa del módulo events de Node.js para gestionar los eventos.

**¡Cuidado!** El código dado funciona en UNIX/Linux. Para ejecutar en Windows hay que aplicar los cambios que se indican en las instrucciones sobre compatibilidad UNIX-Windows para node.js.

En esta práctica el alumno deberá realizar dos tareas:

* **Tarea 1:** Ampliar el programa de ejemplo añadiendo un programador. El programador permitirá configurar que temperatura ideal se desea tener en la habitación en todo momento.
* **Tarea 2:** Sustituir el módulo events de Node.js por uno equivalente implementado por el alumno, de forma que el funcionamiento del programa de ejemplo no se vea afectado.

Para ejecutar el programa de ejemplo se debe invocar el comando:

$ node main.js

A continuación se describe cada uno de los ficheros que incluye la práctica:

### El fichero *****habitación.js*****

El fichero **habitacion.js** es un módulo que exporta la clase **Habitacion**. Cada instancia de la clase **Habitacion** simula una habitación de una vivienda, y mantiene el valor de su temperatura ambiente. Además, cada instancia de **Habitacion** tiene un temporizador que modifica la temperatura aleatoriamente en +/- 1ºC cada 10 segundos. Esta variación aleatoria de la temperatura simula los cambios de estación, o cualquier actividad externa que afecte a la temperatura de la habitación.

### El fichero *****climatizador.js*****

Este fichero es un módulo exporta la clase **Climatizador**. Cada instancia de la clase **Climatizador** simula un aparato climatizador instalado en una habitación, y que permite aumentar y disminuir la temperatura de la habitación. Toma aire de la habitación y lo expulsa un poco más caliente o un poco más frio.

El constructor de la clase **Climatizador** toma como parámetro un objeto **Habitacion** que representa la habitación donde está instalado el climatizador.

Los objetos climatizador tienen dos métodos: **enfriar** y **calentar**. Se usan para disminuir o aumentar ligeramente la temperatura de la habitación cada vez que se invocan.

### El fichero *****termostato.js*****

Este módulo exporta la clase **Termostato**.

Cada instancia de la clase **Termostato** simula un termostato instalado en una habitación, que puede configurarse con una temperatura ideal, y que emite eventos informando cuando la temperatura de la habitación es demasiado alta o baja respecto de la ideal. También emite un evento periódico informando de la temperatura actual de la habitación. La clase **Termostato** extiende a la clase **EventEmitter** para gestionar la emisión de eventos y sus escuchadores. El constructor de la clase Termostato toma como parámetro un objeto **Habitacion** que representa la habitación donde está instalado. Cada instancia de **Termostato** tiene tres propiedades:

* La propiedad **habitacion** mantiene una referencia a la habitación donde está instalado el termostato.
* La propiedad **temperaturaIdeal** almacena el valor de la temperatura ideal que se ha configurado.
* La propiedad **intervalId** identifica un temporizador interno que periódicamente sondea la temperatura de la habitación y emite los eventos **"tic"**, **"muchocalor"** y **"muchofrio"**. La propiedad intervalId se necesita para cancelar el temporizador interno cuando se apaga o se vuelve a encender el termostato.

Cada instancia de **Termostato** tiene tres métodos:

* El método **indicarTemperaturaIdeal** se usa para configurar la temperatura ideal deseada en la habitación.
* El método **encender** se usa para encender el termostato, inicializando el temporizador interno que lanza los eventos.
* El método **apagar** se usa para apagar el temporizador, detenido el temporizador interno encargado de lanzar los eventos. Las instancias de Termostato pueden emitir tres eventos: • El evento **"muchocalor"** se emite cuando la temperatura actual de la habitación supera a la temperatura ideal en más grados que los indicados por la constante **MARGEN\_ERROR**. • El evento **"muchofrio"** se emite cuando la temperatura actual de la habitación es inferior a la temperatura ideal en más grados que los indicados por la constante **MARGEN\_ERROR**. • El evento **"tic"** se emite cada medio segundo, y le pasa a los escuchadores como parámetro el valor de la temperatura actual de la habitación.

### El fichero *****main.js*****

Este fichero es el programa principal. Importa las clases **Habitacion**, **Climatizador** y **Termostato**, y crea una instancia de cada una de ellas. La constante **dormitorio** apunta a una habitación que inicialmente está a 22ºC.

Las constantes **climatizador** y **termostato** apuntan al climatizador y al termostato instalados en el dormitorio anterior. En el termostato se registran tres escuchadores. El primero escucha los eventos **"muchofrio"** para provocar que el climatizador caliente. El segundo escucha los eventos **"muchocalor"** para que provocar que el climatizador enfríe. El último escucha los eventos **"tic"** para mostrar por la consola la temperatura actual del dormitorio. Finalmente, se configura una temperatura ideal a 20ºC y se enciende el termostato.

### Tareas a realizar

### Tarea 1: Programador

La primera tarea que debe realizar el alumno es ampliar el programa de ejemplo añadiendo un programador que permita configurar la temperatura que se desea tener en la habitación en todo momento.

Debe crearse un fichero llamado programador.js que implemente un módulo que exporte una clase llamada **Programador**

El constructor de la clase **Programador** debe tomar como parámetro un objeto con la configuración de horas y temperaturas que se desea programar. El objeto con la configuración debe ser un array como el ilustrado a continuación.

[

{ hora: "07:00",

temperatura: 22

},

{ hora: "08:30",

temperatura: 18

},

{ hora: "18:00",

temperatura: 22

},

{ hora: "23:00",

temperatura: 20

}

]

Cada elemento del array es un objeto con dos claves: \* **hora**: El valor asociado a esta clave es un string con la hora a la que debe aplicarse la programación de la nueva temperatura ideal. El formato de la hora debe ser "hh:mm". \* **temperatura**: un número con la nueva temperatura ideal a programar.

Con la configuración del ejemplo se programaría una temperatura de 22ºC todos los días a las 7 de la mañana, 18ºC todos los días a las 8:30, 22ºC todas las tardes a las 6, y 20ºC todos los días a las 11 de la noche. Se recomienda usar el módulo Later.js (<https://bunkat.github.io/later>) para implementar el módulo programador.js. Recuerde instalarlo con `npm install. El módulo Later.js permite planificar instantes de tiempo en los que ejecutar tareas. Por ejemplo, para escribir por consola la palabra "hola" todos los días a las 18:00, se podría usar el siguiente código:

// Importar modulo Later.js:

const later = require('later');

// Usar zona horaria local:

later.date.localTime();

// Crear planificación para todos los dias a las 18:00

const sched = later.parse.text('at 18:00');

// Crear un temporizador que escriba indefinidamente "hola"

// en los instantes planificados anteriormente:

later.setInterval(() => console.log('hola'), sched);

Las instancias de la clase **Programador** deben emitir un evento llamado **"ideal"** cada vez que sea necesario reprogramar la temperatura ideal, siguiendo las instrucciones de la configuración pasada en el constructor. El evento **"ideal"** se emitirá pasando como parámetro el valor de la temperatura ideal a programar en el termostato.

Para poder emitir eventos, la clase **Programador** debe extender a la clase **EventEmitter**.

Finalmente, en el programa principal main.js deben añadirse los siguientes cambios: \* Importar el módulo programador.js para crear un objeto **Programador**. \* Añadir las sentencias necesarias para que cuando el programador emita un evento **"ideal"**, se ajuste la nueva temperatura ideal en el termostato.

### Tarea 2: Events

En esta tarea se pide al alumno que cree su propia implementación del módulo **events**, y que la integre en el programa de ejemplo, sustituyendo al módulo events proporcionado por Node.js.

La implementación pedida al alumno es una versión reducida de la proporcionada por Node.js. El alumno solo debe implementar los métodos **emit** y **on**.

Para que el módulo implementado por el alumno se integre fácilmente en el programa de ejemplo, deben cumplirse los siguientes requisitos: \* El módulo debe llamarse events.js y situarse junto a los demás ficheros javascripts. \* Para requerirlo desde termostato.js y desde programador.js deberá sustituir las llamadas require("events") por require("./events"). \* El módulo debe exportar una clase que se llame **EventEmitter**. \* Las instancias de **EventEmitter** deben tener una propiedad donde se deben guardar todos los escuchadores que se hayan registrado usando el método **on**. Se debe guardar el nombre de los eventos junto con la lista de escuchadores interesados en cada evento. \* Las instancias de **EventEmitter** deben tener el método **on**, para que los escuchadores se registren. Este método toma como parámetro el nombre de un evento, y el método a ejecutar cuando se emita ese evento. \* Las instancias de **EventEmitter** deben tener el método **emit**, que se emplea para emitir un evento. Este método toma como parámetros el nombre del evento a emitir, y los argumentos que hay que pasar a las funciones escuchadoras interesadas en ese evento.

Una vez sustituido el módulo events de Node.js por el desarrollado por el alumno, el comportamiento del programa de prueba no debe variar.

## Descargar el código del proyecto

El proyecto debe clonarse en el ordenador desde el que se está trabajando:

$ git clone https://github.com/ging-moocs/MOOC\_node\_mod1-eventos\_entrega

A continuación se debe acceder al directorio de trabajo y abrir el fichero index.html con el editor de la elección del alumno.

$ cd MOOC\_node\_mod1-eventos\_entrega

Para ejecutar el programa de ejemplo se debe invocar el comando:

$ node main.js

## Prueba de la práctica

Para ayudar al desarrollo, se provee una herramienta de autocorrección que prueba las distintas funcionalidades que se piden en el enunciado. Para utilizar esta herramienta debes tener node.js (y npm) (<https://nodejs.org/es/>) y Git instalados.

Para instalar y hacer uso de la [herramienta de autocorrección](https://www.npmjs.com/package/moocauto) en el ordenador local, ejecuta los siguientes comandos en el directorio del proyecto:

$ npm install -g moocauto ## Instala el programa de test

$ moocauto ## Pasa los tests al fichero a entregar

............................ ## en el directorio de trabajo

... (resultado de los tests)

También se puede instalar como paquete local, en el caso de que no se dispongas de permisos en el ordenador desde el que estás trabajando:

$ npm install moocauto ## Instala el programa de test

$ npx moocauto ## Pasa los tests al fichero a entregar

............................ ## en el directorio de trabajo

... (resultado de los tests)

Se puede pasar la herramienta de autocorrección tantas veces como se desee.

## Entrega de la práctica

El alumno debe subir un fichero comprimido ZIP incluyendo todos los ficheros de la práctica excepto el directorio node\_modules (si existe).

## Evaluación de la práctica

La evaluación de la práctica se realizará mediante revisión por pares (P2P). Cada alumno tendrá que revisar la práctica de 3 de sus compañeros y otros 3 revisarán la suya. Se puede utilizar la herramienta de autocorrección (moocauto) como ayuda para revisar la práctica de los compañeros.

El objetivo de este curso es sacar el máximo provecho al trabajo que están dedicando, por lo que les recomendamos que utilicen la evaluación para ayudar a sus compañeros enviando comentarios sobre la corrección del código, su claridad, legibilidad, estructuración y documentación.

Dado que es un curso para principiantes, ante la duda les pedimos que sean benevolentes con sus compañeros, porque muchos participantes están empezando y los primeros pasos siempre son difíciles.

**OJO! Una vez enviada la evaluación, está no se puede cambiar.** Piensen bien su evaluación antes de enviarla.

**RÚBRICA**: Se puntuará el ejercicio a corregir sumando el % indicado a la nota total si la parte indicada es correcta:

* **40%:** Tarea 1
* **60%:** Tarea 2