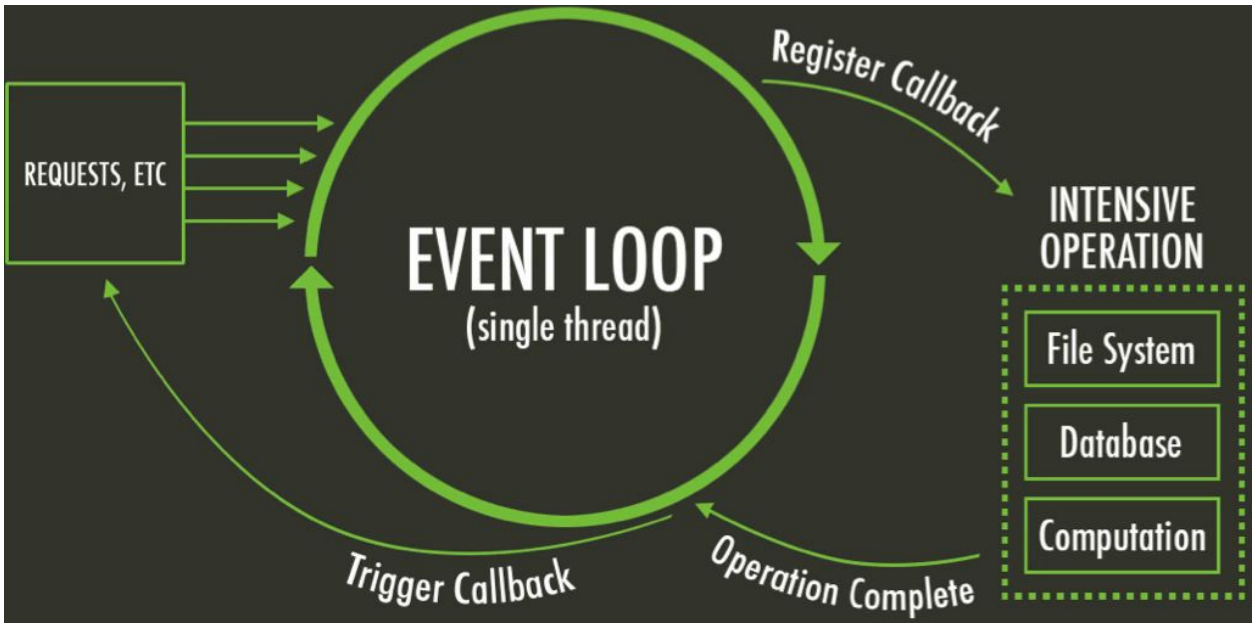
Evento loops.

Node.js es una tecnología asincrónica (async), lo que significa que los flujos de ejecución no son procedimentales en todas las operaciones.
Maneja un solo "thread" (proceso), el cual está en un loop constante verificando si se han disparados eventos. Cada evento tiene asociados N cantidad de callbacks. Es un modelo tradicional de emitters y listeners.



Como node.js es una tecnología muy usada para construir aplicaciones de red, nos basaremos en "Request" (visitas de usuario) a las mismas.
En el gráfico podemos ver el evento loop, el cual está constantemente monitoreando si un evento ha sido disparado.
Cada vez que un evento es lanzado los callbacks asociados a los mismos son ejecutados.
Los procesos asincrónicos son operaciones que se ejecutan "en paralelo" al flujo de ejecución de al apliaciones. Cuando un proceso asincrónico termina el mismo dispara un evento que es asociado al callback.
En Node.js las operaciones de acceso al sistema de archivo, bases de datos o trabajo en red son normalmente procesos asincrónicos, es por ello que las funciones (métodos) para operar en forma asincrónica llevan como argumento una funcion (callback).

Procesos de ejecución.

Es importante conocer el proceso de ejecución dentro de node, entender que sucede en su interior.
Primero que nada debemos entender como esta desarollado, observemos la siguiente imagen.

Node.js se basa en el motor V8 como hemos mencionado en capítulos anteriores, el cual está desarrollado en C/C++. Al igual que el motor V8, las librerías de acceso a red, el sistema de event loop y otras tantas librerías estan desarrolladas con las mismas tecnologías.
Fuera del ambiente C/C++ tenemos los módulos, los cuales estan desarrollados en JavaScript.
Cada vez que lanzamos una aplicacion en Node, el intérprete de node realiza una compilación del código (desarrollado por nosotros y las librerías requeridas). En éste proceso si existiera un error de sintaxis, uso indebido de variables (al usar el "strict mode"), nuestra aplicación directamente no se ejecutaría. Cuando el proceso de compilación es exitoso la aplicacion es ejecutada.

Módulos en Node.js.

Los módulos en node.js son librerías escritas normalmente en javascript que nos permiten realizar operaciones de distintos tipos (conectarnos a una base de datos, crear un archivo comprimido, etc).
Para poder incluir un módulo usamos la función "require()" pasando como argumento el nombre del módulo a leer.
Los módulos en su gran mayoría retornan una funcion (clase).
Los callback (cb) normalmente reciben argumentos, el cual el primero siempre es el error (si no hay error se recibe un null).

Events.

Node.js nos ofrece la posibilidad de crear nuestros propios eventos, establecer listeners y dispararlos.
Para lograr esto necesitamos instanciar EventEmitter del módulo nativo Events.
Un evento en sí no es más que un nombre (string). Cada evento tiene N cantidad de listener.
Cuando disparamos un evento podemos adjuntar los argumentos a pasarla a los listener (los callbacks de los mismos).

Ejemplo

```
var EventEmitter = require("events").EventEmitter;
var ee = new EventEmitter();



ee.on("cursando", function(a,b,c){
   console.log("alguien esta cursando: ",a,b,c);
});

ee.emit("cursando", "luis", "carlos", "mariela");
```

Métodos.

Método	Descripción.
addListener(event, listener)	Agrega un listener. Recibe un string como evento y un cb (function).
on(event, listener)	Es un alias de addListener.
once(event, listener)	Es igual que on(), con la diferencia que solo se ejecutará una única vez.
removeAllListeners(event)	Elimina todos los listener de un evento.

Ejemplo

```
var EventEmitter = require("events").EventEmitter;
```

```
var ee = new EventEmitter();



ee.once("cursando", function(a,b,c){
    console.log("yo me ejecuto una sola vez: ",a,b,c);
});

ee.on("cursando", function(a,b,c){
   console.log("(on) alguien esta cursando: ",a,b,c);
});

ee.addListener("cursando", function(a,b,c){
    console.log("(addListener) alguien esta cursando: ",a,b,c);
});

ee.emit("cursando", "luis", "carlos", "mariela");
ee.emit("cursando", "luis", "carlos", "mariela");
ee.emit("cursando", "luis", "carlos", "mariela");
```

```
yo me ejecuto una sola vez:  luis carlos mariela
(on) alguien esta cursando:  luis carlos mariela
(addListener) alguien esta cursando:  luis carlos mariela
(on) alguien esta cursando:  luis carlos mariela
(addListener) alguien esta cursando:  luis carlos mariela
(on) alguien esta cursando:  luis carlos mariela
(addListener) alguien esta cursando:  luis carlos mariela
```

Módulo fs.

El primero módulo que veremos será "fs", el cual es usado para las operaciones de I/O a nivel FileSystem.
Cuenta con métodos para varias operaciones, los cuales veremos en la siguiente tabla.

Métodos.

Método	Descripción.
exists(file, cb)	Chequea si existe un archivo. En el cb envía true o false.

existsSync(file)	Cheque si existe un archivo en forma sincrónica. Retorna true/false.
writeFile(file, data,[opts], cb)	Escribe data en un archivo en forma asincrónica. En el cb se pasan como argumento el err (si no hay error retorna null).
writeFileSync(file, data, [opts])	Igual que el anterior pero escribe en forma sincrónica.
readFile(file, [opts], cb)	Lee el contenido de un archivo y lo pasa como segundo argumento del cb.
readFileSync(filename, [opts])	Lee un archivo en forma sincrónica. Retorna el contenido del archivo.
mkdir(path, [mode], cb)	Crear un directorio. Retorna err como argumento del cb.
mkdirSync(path, [mode])	Crear un directorio en forma sincrónica.
rmdir(path, cb)	Elimina un archivo o directorio en forma asincrónica.
rmdirSync(path)	Elimina un archivo o directorio en forma sincrónica.

Prácticas y ejemplo.

```
var fs = require("fs");


fs.writeFile('prueba.txt', 'Estoy en el curso de Node.js', function (err) {
    //si hay error, lanzamos una exception
    if (err) throw err;
    //mostramos mensaje en pantalla
    console.log('Archivo guardado');
});


var cbs = function(e,d){

    if(e) throw e;
    console.log('Terminé la operacion.');
```

```
};

fs.mkdir("pruebas", cbs);
fs.mkdir("pruebas/a", cbs);
fs.mkdir("pruebas/b", cbs);
fs.mkdir("pruebas/c", cbs);
```

En base al código anterior podemos observar:

- El error de ejecutar operaciones asincrónicas y no esperar que las mismas terminen.
- Comprobar que no existan archivos / directorios en forma previa.
- El uso de callbacks.

Ejercicios.

Modificar el código dado como ejemplo para que cumpla con las siguientes reglas:

- Comprobar que no existan previamente los archivos o directorios a crear. En caso de que existan, eliminarlos.
- Informar al usuario de que archivos/directorios existían previamente, cuales fueron eliminados y cuales creados.
- Crear un archivo de textos con todos los logs de las operaciones realizadas por el script.
- Generar una clase (object) llamada myFiles.
- Generar un método en la clase myFiles llamado doFile(fileName, fileType[dir,file], [content]) que almacene en un array un objeto con los argumentos.
- Generar un método en la clase myFiles llamado doFilesNow() que al ejecutarse recorra el array que almacena los objetos con los archivos/directorios a crear.

Ejemplo solución 1

```
var fs = require("fs");

var logs = "";
var log = function(str){
 console.log(str);
 logs=logs+"\n\r"+str;
}

var rmdir = function(d, cb){
   fs.rmdir(d, function(){
    log("Eliminado el directorio> "+d);
```

```
    cb();
  });
}

var mkdir = function(dir, cb){

  var _mkdir = function(name, cb2){
        fs.mkdir(name, function(e){
         log("Creado directorio> "+ name);
         cb2();
        });
   }//end _mkdir

  fs.exists(dir, function(b){
    if(b){
     log("el directorio> "+dir+" existe");
      rmdir(dir, function(){
          _mkdir(dir, cb);
        });
    }//existe
    else{
       _mkdir(dir, cb);
    }//no existe
  });
}

mkdir("pruebas", function(){
        mkdir("pruebas/a", function(){
                mkdir("pruebas/b", function(){
                        mkdir("pruebas/c", function(){
                          fs.writeFile("logs.log",logs,function(e){
                                console.log("log guardados");
                          });//end writeFile
                        })
                })
        })
});
```

Ejemplo solución 2 (partial)

```
var events = require("events");
var eventEmitter = events.EventEmitter;
var ee = new eventEmitter();
var fs = require("fs");
```

```
//mkdir -> created
//rmdir -> deleted

var cursor = 0;
var directories = ["prueba","prueba/a","prueba/b"];

ee.on("created", function(name){
        console.log("Un directorio fue creado: ", name);
        var n = next();
        if(n!==false) mkdir(n);
});

ee.on("deleted", function(){
        console.log("Un directorio fue elimiado");
});


var next = function(){
  return cursor == directories.length ? false : directories[cursor++];
};

var mkdir = function(name){
    fs.mkdir(name, function(e){
        if(e) console.log(e);
        ee.emit("created", name);
    });
};

mkdir("pepe");

/*
var item = next();

while(item){
    console.log(item);
        item = next();
}*/
```

Ejemplo 3 (partial)

```
var events = require("events");
var eventEmitter = events.EventEmitter;
var ee = new eventEmitter();
var fs = require("fs");
```

```
//mkdir -> created
//rmdir -> deleted

var cursor = 0;
var directories = ["prueba","prueba/a","prueba/b"];

ee.on("created", function(name){
        console.log("Un directorio fue creado: ", name);
        var n = next();
        if(n!==false) mkdir(n);
});

ee.on("deleted", function(name){
        console.log("Un directorio fue elimiado: ",name);
});


var next = function(){
  return cursor == directories.length ? false : directories[cursor++];
};

var mkdir = function(name){
    var _mkdir = function(){
        fs.mkdir(name, function(e){
           if(e) console.log(e);
           ee.emit("created", name);
        });//end fs.mkdir
    };//end _mkdir


       fs.exists(name, function(e){
         if(e){
            fs.rmdir(name,function(){
              ee.emit("deleted",name);
              _mkdir();
             });//end fs.rmdir
          }else{
             _mkdir();
          }//end else
       });//end exits
};

mkdir("pepe");
```

```
var item = next();
if(item!==false) mkdir(item);
```

Ejemplo (partial 4).

```
module.exports = function(){

var events = require("events");
var eventEmitter = events.EventEmitter;
var ee = new eventEmitter();
var fs = require("fs");


//mkdir -> created
//rmdir -> deleted

this.mkdirs = function(directories){
var cursor = 0;

ee.on("created", function(name){
        console.log("Un directorio fue creado: ", name);
        var n = next();
        if(n!==false) mkdir(n);
});

ee.on("deleted", function(name){
        console.log("Un directorio fue elimiado: ",name);
});


var next = function(){
  return cursor == directories.length ? false : directories[cursor++];
};

var mkdir = function(name){
    var _mkdir = function(){
        fs.mkdir(name, function(e){
           if(e) console.log(e);
           ee.emit("created", name);
        });//end fs.mkdir
    };//end _mkdir


       fs.exists(name, function(e){
         if(e){
```

```
            fs.rmdir(name,function(){
              ee.emit("deleted",name);
              _mkdir();
             });//end fs.rmdir
          }else{
             _mkdir();
          }//end else
       });//end exits
};

var item = next();
if(item!==false) mkdir(item);
}//end mkdirs
};//module.exports
```

Test del capítulo 6.

El event loop es:

- Un proceso asincrónico
- Un módulo de node.js
- Un loop que chequea la integridad de las variables.
- Un loop que chequea si se ha disparado un evento.

Cual se los siguientes método del módulo "fs" crea un directorio?

- fs.watch
- fs.createDir
- fs.makeDirectory
- fs.mkdir
- fs.newDir

Cual de los siguiente métodos de fs reciben como argumento un callback?

- fs.writeFileSync
- fs.mkdirSync
- fs.readFile
- fs.rmdirSync

Que retorna el siguiente código en node?

```
var fs = require("fs").writeFile("testing.tpm", "node es genial", function(e){
    console.log(e);
});
```

- Error
- TypeError: Bad argument
- if(e) throw e;
- null
- return binding.fstat(fd);

Modulos de usuario.

Los módulos de usuarios son simples archivos .js que en su interior exportan una funcion u objeto.
La forma de requerirlos es la misma que los módulos nativos o los instalados por npm, con la diferencia que deberemos indicar su ubicación (teniendo en cuenta que el directorio de trabajo es el mismo en donde se encuentra nuestro script).

Ejemplo 1.

mymod.js
```
module.exports = function(){
  this.hi = function(name){
    console.log("hola ", name)
  };
};
```

app.js
```
var mymod = require("./mymod");
var mymodinstance = new mymod();
mymodinstance.hi("nodejs");
```

Test del capítulo 7.

Cual de las siguientes líneas es incorrecta para declarar un módulo de usuario?

- module.exports = function(){};
- module.exports = return {};

- module.exports = {};

Cual de las siguientes líneas es válida?

- require("./mymodule")[1];
- require({file:"./mymodule"});
- require(["./mymodule"]);