DESARROLLO DE APLICACIONES WEB AVANZADO

LABORATORIO N° 02

Librerias en Node. Implementar. Reescribir

Alumno(s):		***Nota***	
Grupo:		***Ciclo:V***	

Criterio de Evaluación	*Excelente (4pts)*	*Bueno (3pts)*	*Requiere mejora (2pts)*	*No acept. (0pts)*	*Puntaje Logrado*
Identifica la utilidad de usar un gestor de paquetes					
Crear un paquete propio					
Creación y mantenimiento de paquetes publicados					
Realiza con éxito lo propuesto en los módulos JavaScript					
Es puntual y redacta el informe adecuadamente					

Laboratorio 2: Instalación y administración de las librerías para las aplicaciones web

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Identificar las principales características de NPM
- Importación y uso de librerías principales de NPM
- Creación de módulos propios.
- Exportar sus propios módulos.

Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

Equipos y Materiales:

- Una computadora con:
  - Windows 7 o superior
  - VMware Workstation 10+ o VMware Player 7+
  - Conexión a la red del laboratorio
- Máquinas virtuales:
  - Windows 7 Pro 64bits Español - Plantilla
- Instalador de node.js

Procedimiento:

Lab Setup

1. **Creación del equipo virtual**
   1.1. Encender el equipo
   1.2. Acceder empleando la cuenta de **usuario**: Tecsup, **contraseña**: _____
   1.3. Iniciar el Software VMWare.
   1.4. Abrir la plantilla ubicada en:
   E:\Equipos virtuales\Windows 7 Pro 64bits Español - Plantilla
   1.5. Crear un clon de la maquina anterior con los siguientes datos:
   1.6. Nombre del clon: **C15-DAWA**
   1.7. Ubicación del clon: **D:\C15-DAWA**
   1.8. Cerrar la plantilla
   1.9. Iniciar el equipo virtual **C15-DAWA**
   1.10. Identifíquese con la cuenta de usuario: **Redes**. Contraseña: **RCDTecsup2**
   1.11. Solicite el instalador de node.js al instructor e instálelo.

Identificación de npm

2. Node.js viene con NPM instalado por defecto en todas sus versiones actuales. No se deberá hacer ninguna configuración adicional para hacerlo funcionar.
   2.1. Ingrese el comando **npm** en una terminal. Adjunte la captura con los argumento principales de la función npm y además mencione la versión con la que está trabajando.

Creación de package.json

3. Procedemos a iniciar nuestro proyecto creando una carpeta donde residirá nuestro código.
   3.1. Creamos una carpeta en C:\ con el nombre de proyecto

```
C:\>mkdir proyecto
```

3.2. Nos situamos dentro de la misma para proceder a escribir el comando **npm init**

```
C:\>cd proyecto

C:\proyecto>npm init
```

3.3. Con el comando anterior, habremos iniciado el asistente de creación de nuestro manifiesto de proyecto, o también conocido como **package.json**. Proceda a seguir las instrucciones del asistente, añadiendo aquí las capturas correspondientes a cada paso con su explicación. El nombre de proyecto será **tecsup-2019-XXX** donde **XXX** será sus dos apellidos seguidos, en minúsculas y sin acentos. Ingrese su nombre como autor del proyecto, versión 0.0.1

3.4. Verifique la creación del archivo mencionado y adjunte una captura de su contenido.

Instalación de paquetes mediante NPM

4. Procederemos a instalar un paquete mediante NPM para conocer su utilización en la vida real.

4.1. Ingresamos el siguiente comando y esperamos se complete con éxito.

```
C:\proyecto>npm install underscore
```

4.2. Verifiquemos si hay algún cambio en la carpeta **proyecto**. De ser así, adjunte una captura de pantalla de lo que ha sucedido.

4.3. Abra el archivo package.json para verificar si ha habido algún cambio. De no ser así Ingrese el siguiente comando y ábralo nuevamente.

```
C:\proyecto>npm install underscore --save
```

4.4. Adjunte una captura del archivo **package.json** y explique lo que ha cambiado.

4.5. NPM permite instalar librerías para utilizarlas, independientemente de si estamos usando node.js o si queremos usarlas para nuestro proyecto web en el desarrollo del frontend. La diferencia de usar el argumento -**–save** es que se buscará un manifiesto de proyecto (**package.json**) donde agregar esta dependencia de nuestro proyecto.

4.6. Cree el archivo **index.js** con el siguiente contenido.

```
var _ = require("underscore");

var lista =[1,2,3,4,5,6]; _.each(lista,function(item) {
    console.log(item);
});
```

4.7. Ejecutemos el resultado e interpretemos lo obtenido.

4.8. Agreguemos la siguiente variable con un array de empleados de una compañía.

```
var employeesCollection = [
    {
        "id":1,
        "name":"Soni",
        "designation":"SE",
        "salary":25000
    },
    {
        "id":2,
        "name":"Rohit",
        "designation":"SSE",
        "salary":35000
    },
    {
        "id":3,
        "name":"Akanksha",
        "designation":"Manager",
        "salary":45000
    },
    {
        "id":4,
        "name":"Mohan",
        "designation":"Accountant",
        "salary":30000
    },
    {
        "id":5,
        "name":"Gita",
        "designation":"SSE",
        "salary":35000
    }
];
```

4.9. Le solicitan crear un reporte para obtener que personal está en que cargo de la compañía. Digite la siguiente función y ejecute el script.

```
var cargos = _.map(employeesCollection, function (employee) {
    return {nombre: employee.name, cargo: employee.designation};
});
console.log(cargos);
```

4.10. A veces, solamente nos pedirán los nombres de los empleados.

```
console.log(_.pluck(employeesCollection, "name"));
```

4.11. Underscore tiene funcione que realmente nos facilitarán la vida. Por ejemplo, la función **chain** permite “encadenar” distintas funciones de la misma librería para luego devolver un solo valor (de una forma parecida al Aggregation Framework de MongoDB)

```
var empleados_sse = _.chain(employeesCollection)
  .filter(function (employee) {
     return employee.designation === 'SSE';
  })
  .map(function (employee) {
     return {name: employee.name, id: employee.id};
  })
  .value();

  console.log(empleados_sse);
```

4.12. Explique el resultado obtenido de la última función.

Creación de un paquete en node.js

5. Uno de los ejemplos clásicos al momento de aprender a desarrollar tus propias librerías es desarrollar una que permita el reemplazo de cadenas de texto.
   5.1. Creamos una carpeta con el nombre **lib**.
   5.2. En dicha carpeta, crearemos el archivo **replace.js** con el siguiente código:

```
exports.replace = function(objetivo, reemplazos) {
    var param_encontrados = objetivo.match(/%(.*?)%/g);
    if (param_encontrados) {
        var nombre_param = null,
            valor_reemplazo = null;

        for (var i=0; i<param_encontrados.length; i++) {
            nombre_param = param_encontrados[i].replace(/%/g, '');
            valor_reemplazo = reemplazos[nombre_param];

            objetivo = objetivo.replace(param_encontrados[i], valor_reemplazo);
        }
    }
    return objetivo;
};
```

   5.3. La función exports le indica a node que partes son reconocidas como integrantes del módulo creado.
   5.4. Para probar nuestro código que reemplaza cadenas de texto, modificaremos el contenido de **index.js** para que luzca de la siguiente manera.

```
var param_replacer = require('./lib/replace');

var objetivo = "%hello% %world%! -- %world% %hello%!";
var idioma = "es";
var reemplazos = {
  "en": {
    "hello": "Hello",
    "world": "World"
  },
  "es": {
    "hello": "Hola",
    "world": "Mundo"
  }
};

var resultado = param_replacer.replace(objetivo, reemplazos[idioma]);

console.log(resultado);
```

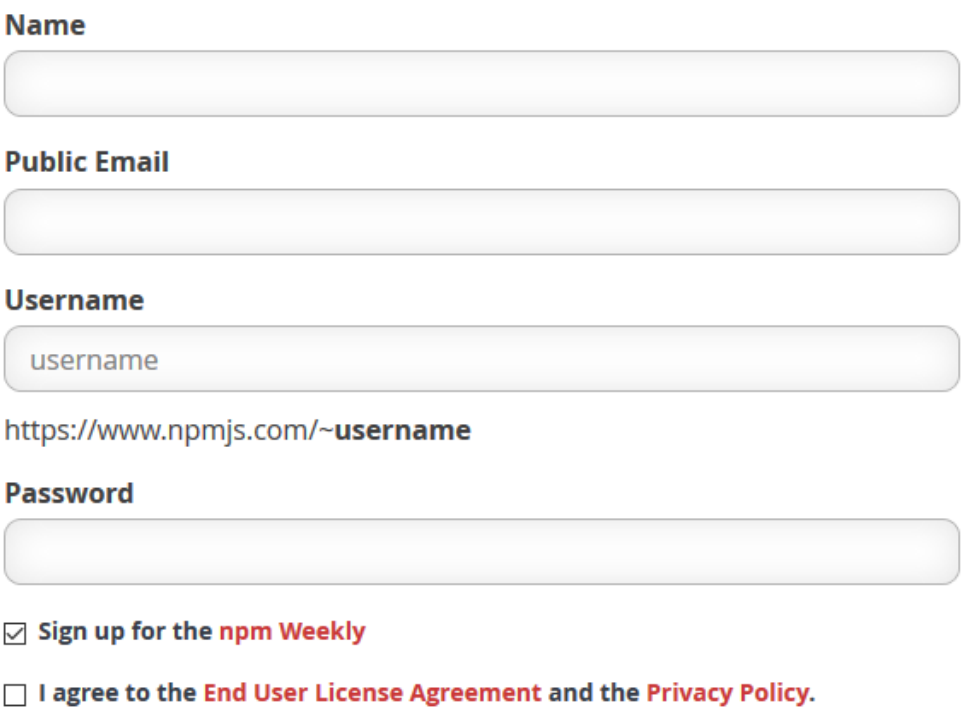
5.5. Adjunte una captura del resultado y agregue como podría mejorar el código o hacerlo más funcional.

Publicación en NPM

6. Lo importante de haber desarrollado un módulo es poder reutilizarlo en algún nuevo proyecto o compartirlo con la comunidad. Procederemos entonces a ver cómo se publica un módulo en NPM
   6.1. Primero deberemos modificar nuestro index.js para que luzca de la siguiente manera.

```
var param_replacer = require('./lib/replace');
if (typeof exports !== 'undefined') {
    if (typeof module !== 'undefined' && module.exports) {
        exports = module.exports = param_replacer;
    }
    exports.param_replacer = param_replacer;
} else {
    root.param_replacer = param_replacer;
}
```

   6.2. Ingrese a la página web de npm: https://www.npmjs.com/
   6.3. Proceda a crear una cuenta en npm. Ingrese los datos solicitados en el formulario.



   6.4. Recuerde su usuario y contraseña, ya que estos serán solicitados al momento de publicar nuestro módulo.
   6.5. Usamos el comando **npm adduser** para poder validar nuestras credenciales en el proyecto que tenemos creado. Procedemos a ingresar la data que nos solicite.

```
C:\proyecto>npm adduser
```

   6.6. Para publicar un paquete en npm, basta con solicitarlo.

```
C:\proyecto>npm publish
```

6.7. Procederemos a probar dicha publicación y reutilizar el código ya creado. Ingresemos la siguiente cadena de comandos para instalar el paquete replace. Coloque el nombre que le corresponde.

```
C:\proyecto>cd..

C:\>mkdir otro_proyecto

C:\>cd otro_proyecto

C:\otro_proyecto>npm install tecsup-2017-XXXXX
```

6.8. Deberíamos obtener un error, explique el por qué y cómo lo solucionará.

6.9. Después de solucionarlo, utilice el código llamándolo directamente desde su nuevo proyecto. Agregue el código que utilizaría para hacer pruebas con su módulo replace. Puede basarse en la forma en la que usamos underscore.js

Entendimiento de ejecución de funciones en JavaScript

7. Es importante tomar en cuenta que nuestro desarrollo debe ser siempre lo más óptimo posible, debido a que al desarrollar Aplicaciones Web estamos sometidos a un alto tráfico y muchas peticiones.

7.1. Deduzca el resultado de la siguiente porción de código. Explique su razonamiento.

```
var nodes = document.getElementsByTagName('button');
for (var i = 0; i < nodes.length; i++) {
   nodes[i].addEventListener('click', function() {
      console.log('You clicked element #' + i);
   });
}
```

_____
_____
_____
_____

7.2. Implemente el código antes mencionado y soluciónelo. Proponga una manera de que imprima lo deseado.

7.3. Deduzca el resultado de la siguiente porción de código. Explique el porqué de su respuesta

```
function printing() {
   console.log(1);
   setTimeout(function() { console.log(2); }, 1000);
   setTimeout(function() { console.log(3); }, 0);
   console.log(4);
}

printing();
```

7.4. Los operadores corto circuíto sirven para hacer evaluaciones booleanas, es usado con diversos lenguajes de programación, veamos ahora un ejemplo simple de cómo funciona en Javascript. Los operadores son: && y || y lo que hacen es asignar el valor del segundo operando basándose en la evaluación del primero, el operador && es muy útil cuando se desea encontrar objetos nulos antes de acceder a sus atributos.

```
if (persona) {
    var nombre = persona.obtenerNombre();
}
```

7.5. Esta porción de código puede ser optimizada de la siguiente manera. Anote su utilidad y en qué casos se utilizaría.

```
var nombre = persona && persona.obtenerNombre();
```

7.6. Inclusive, se puede llegar a reducir sentencias if…else

```
if (nombre) {
    var encargado = nombre;
} else {
    var encargado = "Algún nombre";
}
```

```
var encargado = nombre || "Algun nombre";
```

7.7. En Javascript no existen palabras reservadas como public o private, no hay class, solo objetos que heredan de otros y todos los campos de cualquier objeto son públicos. Pero, entonces ¿Cómo es posible usar Javascript en diversos aspectos y con una orientación a objetos? Pues para ello existe algo llamado **closure** (clausura). Si una función externa devuelve una función interna, la función interna tiene acceso a las variables de la función externa, incluso si la función externa es devuelta posteriormente. Para que esto quede claro podemos ver lo siguiente:

```
function funcionExterna(variable) {
    var  variableDeLaFuncionExterna = variable;
    return function funcionInterna() {
        return variableDeLaFuncionExterna;
    }
}

var foo = functionExterna(20);
foo(); // Esto devolvera 20
```

7.8. Si nos percatamos la funcion interna está siendo llamada antes de que la funcion externa sea retornada y funciona. Esto es un closure. Con un ejemplo voy a tratar de explicarlo mejor, si tuviéramos el siguiente código:

```
var futbolista = {
    "goles": 0,
    "anotaGol": function(gol){
        if (gol > 0) this.goles += gol;
    }
};

futbolista.anotaGol(5);
console.log(futbolista.goles); //Devuelve 5
futbolista.anotaGol(3);
console.log(futbolista.goles); //Devuelve 8
```

7.9. Se podría modificar a futbolista, incluso sin hacer uso de su anotaGol, eso podría ser peligroso, sobre todo en proyectos grandes donde hay muchos archivos y variables, que fácilmente podrían confundir a todos. En ese caso sería mejor algo como esto:

```
var jugador = function(){
    var acumuladorPrivado = 0;
    return {
        "obtenerGoles":function(){
        return acumuladorPrivado;
        },
        "anotaGol":function(gol){
            if (gol > 0) acumuladorPrivado += gol;
        }
    };
};

var sergio = new jugador();
console.log(sergio.acumuladorPrivado);
console.log(sergio.obtenerGoles());
sergio.anotaGol(4);
console.log(sergio.obtenerGoles());
```

7.10. Otro caso común, presente siempre en Javascript, es la herencia parasitaria. Adjunte una captura de lo obtenido de este ejemplo y adjunte sus conclusiones del comportamiento de este lenguaje ante esta herencia.

```
var Persona = function(nombre, edad){
 return {
  nombre: nombre,
  ingresarEdad: function(e) { edad: e; },
  obtenerEdad: function() {return edad;}
 };
};

var Programador = function(nombre, edad, especialidad){
 var e = Persona(nombre, edad);
 e.especialidad = especialidad;
 return e;
};

var aurora = new Persona("Aurora", 27);
var sergio = new Programador("Sergio", 29, "Javascript");

sergio.obtenerEdad();
```

Resolución de problemas

8. Procederemos a resolver distintos problemas con JavaScript. Esto reforzará nuestro conocimiento del lenguaje, que siendo utilizado tanto en front-end como en back-end, es necesario no solamente saber su sintaxis principal, sino entender su funcionamiento (distinto a otros lenguajes al momento de declarar variables, clases o arreglos).
   8.1. Se presentarán a continuación distintos casos de código, por lo que se pide adjuntar su código con lo faltante a lo presentado, además de una captura del resultado deseado. Si es que desea optimizar el código se considerará como un bonus en este laboratorio.
   8.2. Obtener el mayor de dos números.

```
(function() {

    var mayor = function(o1, o2) {
        if ( o1.tamano > o2.tamano ) {
            console.log('o1 es mayor');
        } else {
            console.log('o2 es mayor');
        }
    };

    var x = // escribir codigo aqui
    var y = // escribir codigo aqui

    mayor(x, y);
}());
```

   8.3. Obtener el álbum favorito de una persona.

```
(function() {
    var album_favorito = function( coleccion ) {
        if ( coleccion.length === 0 ) {
            return;
        }
        var mas_reproducido = coleccion[0].tocado,
            mas_indice  = 0;
        for ( var i=0, len = coleccion.length; i < len; i++ ) {
            if ( coleccion[i].tocado > mas_reproducido ) {
                mas_reproducido = coleccion[i].tocado;
                mas_indice  = i;
            }
        }
        return { play: mas_reproducido, index: mas_indice };
    };

    var music = // escribe tu codigo aqui
    var fav = album_favorito( music );

    console.log( "Tu album favorito fue tocado " + fav.play + " veces" );
}());
```

8.4. Función que permite hacer sumas de muchas variables.

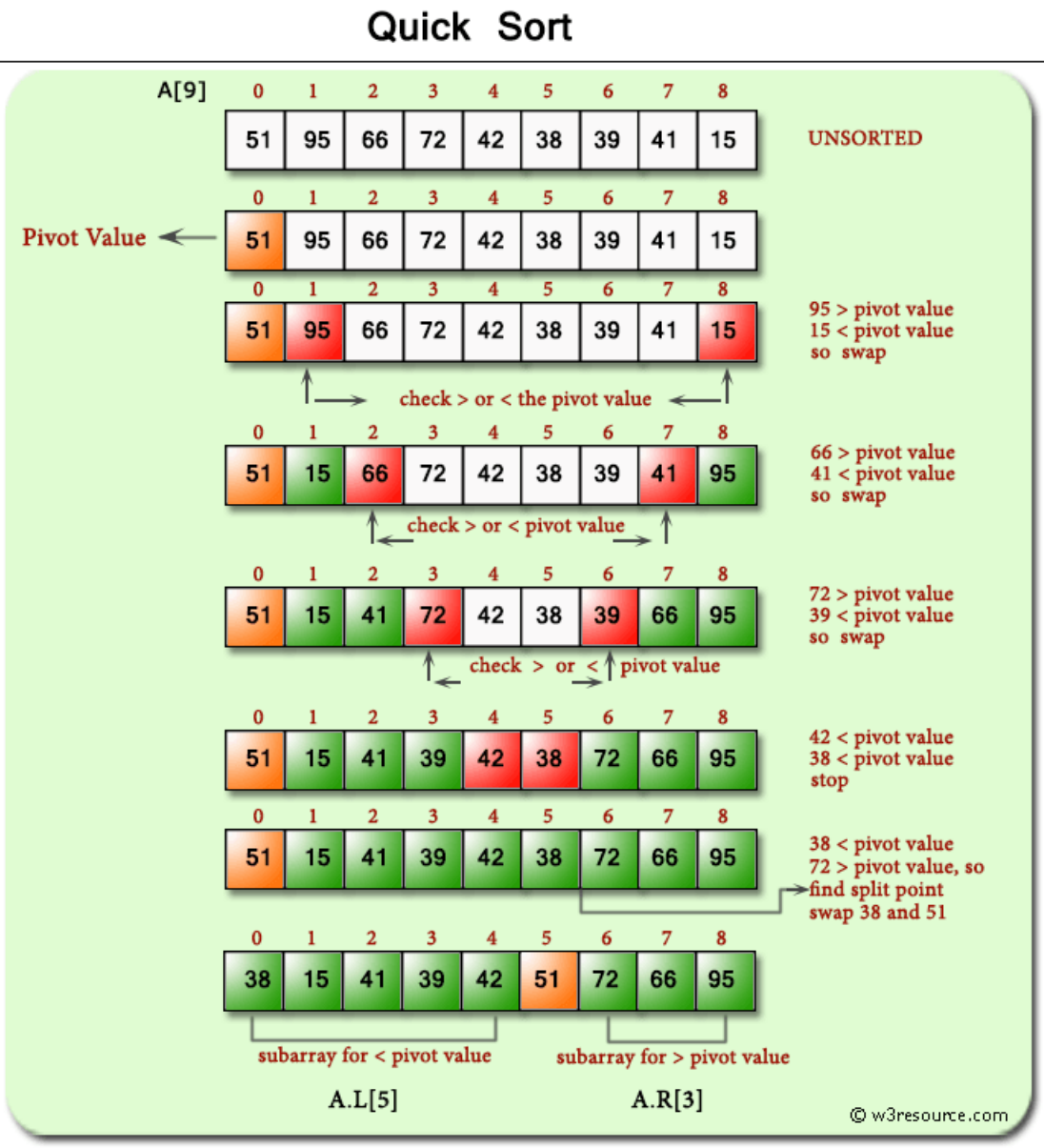
```
(function () {
  // escribe tu codigo aqui

  var s1 = Sumar();
  var s2 = Sumar();

  s1.agregar(10);
  s1.agregar(20);

  s2.agregar(30);
  s2.agregar(5);

  // imprime 30
  console.log(s1.obtenerSumaActual());

  // imprime 35
  console.log(s2.obtenerSumaActual());
}());
```

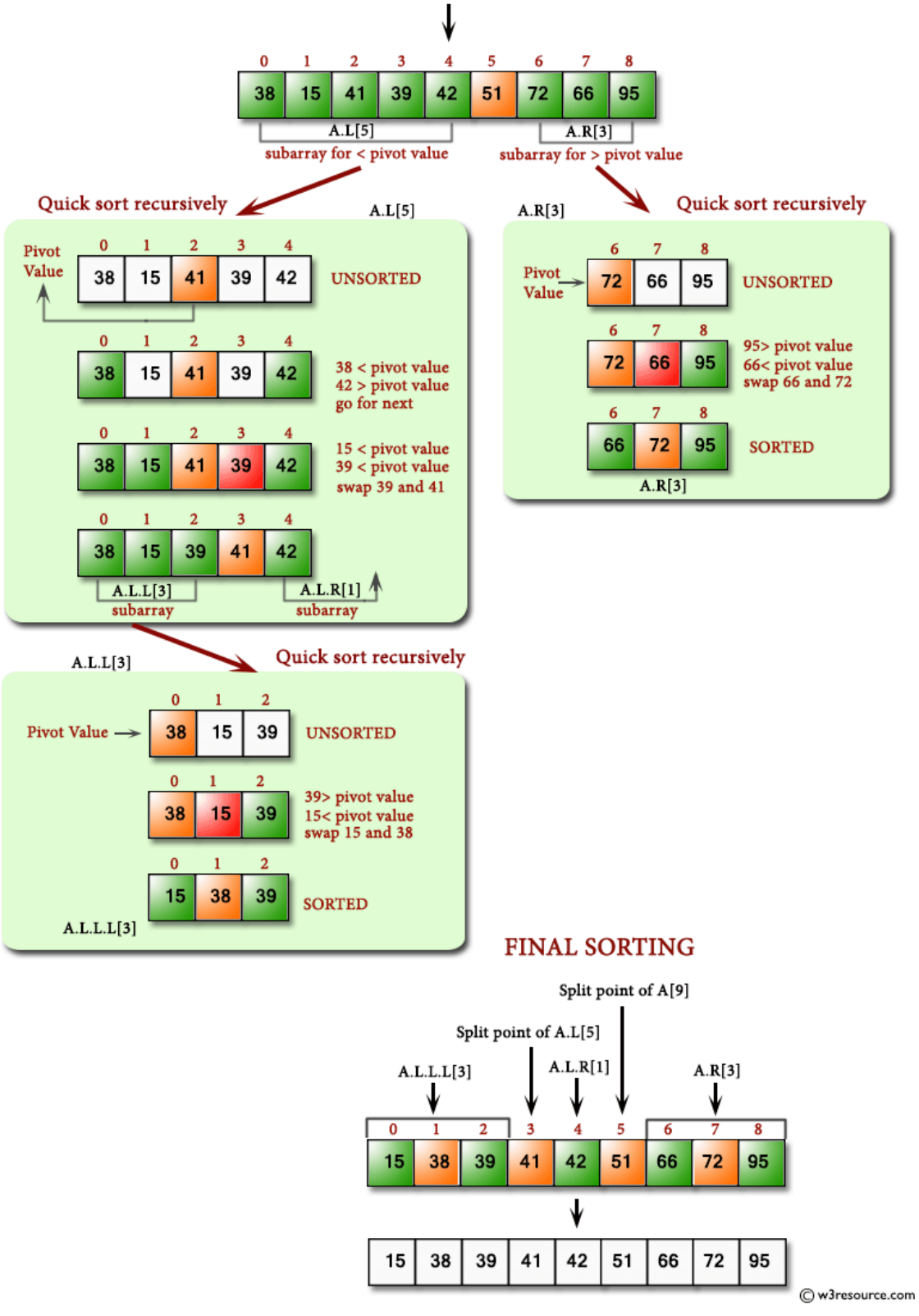
8.5. Función que devuelve el álbum más tocado de una serie de reproducciones.

```
(function() {
  // escribe tu codigo aqui

  var box = MusicBox(),
      a1 = Album("The Who", "Tommy"),
      a2 = Album("Tom Waits", "Closing Time"),
      a3 = Album("John Cale", "Paris 1919"),
      favorito;

  box.addAlbum(a1);
  box.addAlbum(a2);
  box.addAlbum(a3);

  a1.play() ; // imprime "Tocando The Who - Tommy"
  a2.play(); // imprime "Tocando Tom Waits - Closing Time"
  a1.play(); // imprime "Tocando The Who - Tommy"

  favorito = box.favoriteAlbum();

  // imprime "tu album favorito es The Who - Tommy"
  console.log("tu album favorito es " + favorito);
}());
```

8.6. Escriba una función para saber si una variable entregada es un número primo. Recuerde que usted puede recibir cualquier tipo de variable, tanto string como number como boolean. El desafío es tener una función que compruebe esto en al menos veinte líneas.

8.7. Desarrolle una función JavaScript que permita utilizar el algoritmo de ordenamiento QuickSort. Se adjunta una gráfica del funcionamiento del mismo.





9. **Finalizar la sesión**
   9.1. Apagar el equipo virtual
   9.2. Apagar el equipo

Conclusiones:

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.