

MP0488.

Desarrollo de interfaces de usuario
UF1. Confección de interfaces de usuario

**1.2. Librerías de
componentes disponibles**

Índice

☰	Objetivos	3
☰	¿Qué es una librería?	4
☰	Un poco de historia	5
☰	Tipos de librerías: simultaneidad	7
☰	Interfaces del sistema operativo	9
☰	Librerías y nombres de archivo de los sistemas operativos	11
☰	Lenguaje, frameworks y librerías	18
☰	Usando componentes: ¿Qué nos proporciona cada lenguaje?	20
☰	¿Qué es Javascript?	24
☰	¿Son lo mismo JAVA y JavaScript?	26
☰	Bibliotecas y Frameworks JavaScript	28
☰	Resumen	34

Objetivos

En esta lección perseguimos los siguientes objetivos:

1

Comprender el principio de la programación por bibliotecas y componentes, así como sus ventajas.

2

Conocer distintas tecnologías de bibliotecas y componentes.

3

Adquirir sentido crítico en la selección y uso de bibliotecas y componentes.



¡Ánimo y adelante!

¿Qué es una librería?

Librería hace referencia al conjunto de subprogramas con una UI definida para ser invocados y utilizados en el desarrollo de software o para comunicarnos con el dispositivo.

¡Ojo! Hay algunos términos que no debemos confundir.

Según vayamos programando, nos daremos cuenta de que la mayor parte del código y de llamadas a datos se utilizan en muchos sitios del programa que estamos desarrollando. Así sucede para los CMS como WordPress, Joomla, Prestashop, etc., donde se repiten una y otra vez los mismos códigos en diferentes secciones y páginas . De hecho, muchas de esas funciones repetidas están en un directorio (carpeta), separadas de las partes principales del programa que estamos implementando, y están preparadas para la acción (o lo que es lo mismo, **compiladas para poder usarlas cuando lo necesitemos**).

El término **librería** es una traducción incorrecta y hace referencia al término en castellano de **biblioteca**, o conjunto de implementaciones o subprogramas con una interfaz bien definida para ser invocados y que son utilizados para desarrollar software o comunicarnos con el propio dispositivo. O, lo que es lo mismo, **las librerías son una o más funciones que tenemos ya compiladas y preparadas para ser utilizadas en cualquier programa que desarrollemos**.

En los entornos de desarrollo más modernos, el concepto de biblioteca ha evolucionado hasta convertirse en **componente** (o elemento que ofrece un servicio predefinido y es capaz de comunicarse con otros componentes), completando el puzzle de software preparado para poder resolver un determinado problema, y que crea un API (**Applicattion Program Interface**) para que el programador la pueda explotar.

Un poco de historia

Los primeros conceptos de programación acerca de las librerías tenían la intención de separar las *definiciones de datos* de la *implementación del programa*.

Se utilizan desde el año 1.959 con lenguajes de programación como COBOL y FORTRAN. Simula adquiere el nombre de dos lenguajes de programación de simulación (Simula I y Simula 67), desarrollados en 1.960 en el Centro de Computación de Noruega en Oslo por Ole-Johan Dahl y Kristen Nygaard.

```
! todo programa empieza con un begin y termina con un end ;
Begin

    Class Saludo;
    Begin
        OutText("¡Hola Mundo!");
        OutImage;
    End of class saludo;

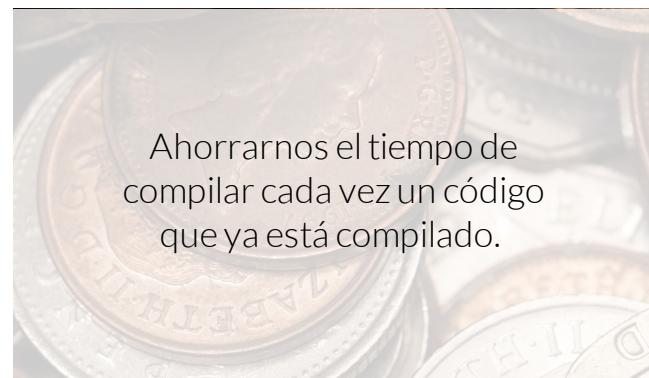
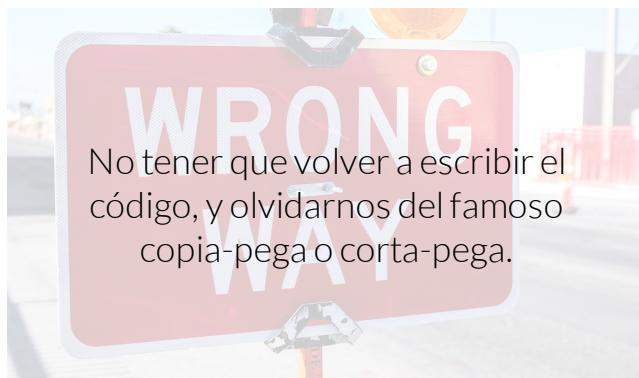
    REF(Saludo) objeto;
    objeto :- New Saludo;

End of module program;
```

Ejemplo de un programa codificado con el lenguaje de programación Simula67.

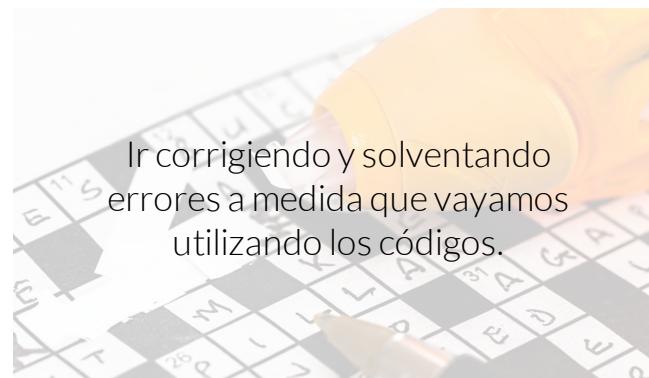
Con este lenguaje se introdujeron conceptos como objetos, clases, herencias y subclases, así como procedimientos virtuales y simulación de eventos. Sin embargo, no fue hasta el año 1.965 cuando se desarrolló por completo el lenguaje de programación Simula67, momento en que se comenzó a utilizar la inclusión de las clases de Simula en los archivos de las librerías, agregándolas, incluso, en tiempo de ejecución.

Ventajas del uso de bibliotecas y componentes



Ahorrarnos el tiempo de compilar cada vez un código que ya está compilado.

Disponer del código ya compilado que estará probado y será fiable.



Ir corrigiendo y solventando errores a medida que vayamos utilizando los códigos.

```

<html>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css">
        <style></style>
      <nav class="navbar navbar-expand-md navbar-dark bg-dark"></nav>
      <todo-form>
        <style></style>
        <div class="card todo-form"></div>
      </todo-form>
      <hr>
      <todo-list ref="list">
        <style></style>
        <h2>Tasks</h2>
        <ul ref="todos" class="list-group">
          <todo-task ref="task-1517176192142" id="task-1517176192142">
            <todo-task ref="task-1517176320397" id="task-1517176320397">
              <todo-task ref="task-1517176329096" id="task-1517176329096">
                <todo-task ref="task-1517176334849" id="task-1517176334849">
                  </ul>
                </todo-list>
              </todo-task>
            </todo-task>
          </ul>
        </todo-list>
      </main>
    </todo-application>
  </div>
  <script src="src/main.js">

```

Utilización de bibliotecas y componentes de JavaScript en un website.

Tipos de librerías: simultaneidad

Como hemos mencionado, las librerías contienen código que proporciona servicios a otros programas completamente independientes.

Es decir, pasan a formar parte de éstos, de modo que permiten que el nuevo código y los datos que necesitamos se compartan, se carguen y puedan modificarse de forma modular.

Los archivos ejecutables y las librerías se hacen referencia entre sí en un proceso que denominamos enlace o link, que por lo general es realizado automáticamente por un programa determinado, al que llamamos **enlazador**. Este programa busca un conjunto de librerías y otros módulos en un orden determinado.

Lo más importante es que una librería, a diferencia de (casi) el resto de los archivos, no espera ser utilizada de forma autónoma, o lo que es lo mismo, será utilizada (enlazada en la etapa de **linking**) por uno o varios programas o procesos de forma simultánea o independiente.

LIBRERÍAS ESTÁTICAS

Piezas de código que se incorporan a la aplicación cuando es enlazada, formando un elemento único en un tiempo denominado **tiempo de compilación**. Este tipo de librerías poseen referencias (direcciones para saltos) y otras llamadas de la rutina de programa, y pueden estar en la parte principal o en un módulo dependiendo de otro.

Se resuelven en **direcciones fijas o reubicables** (desde una base común) asignando memoria. Ciertos lenguajes de programación más avanzados utilizan el **enlace inteligente**, mediante el cual el **vinculador** conoce o se integra con el compilador para usar las referencias externas y el código en una librería, provocando que los programas sean más pesados y menos flexibles a la hora de modificar su código.

La mayor parte de los lenguajes de programación almacenan en una forma relativa o simbólica módulos o librerías que no se resuelven hasta la asignación de direcciones finales estáticas a todos los códigos y módulos.

LIBRERÍAS DINÁMICAS

Las librerías dinámicas o bibliotecas compartidas son piezas de código que permanecen separadas de la aplicación y se enlazan automáticamente cada vez que se ejecuta la aplicación en tiempo de ejecución, por lo que el programa es menos pesado y evita la duplicación de código cuando se requiere usar la misma librería.

El inconveniente es que estas librerías suelen encontrarse en directorios específicos del sistema, lo que ocasiona algunos problemas de dependencias y versionados de una misma librería. Si una librería compartida de la que depende un ejecutable se elimina, se mueve, se renombra o se copia una versión incompatible de la misma a otro lugar, el archivo ejecutable no se cargará, es decir, se pondrá en marcha la **dependencia**: cada programa hace referencia a las librerías sólo por sus identificadores únicos completos.

Este tipo de librerías tiene tan sólo las limitaciones establecidas por las licencias de software.

Interfaces del sistema operativo

A continuación, te mostraremos una clasificación de Librerías de Componentes para los diferentes sistemas operativos del mercado.

En un sistema operativo todo está estructurado en capas, de modo que una capa o ofrece una interfaz a la capa superior, y una capa $o+1$ es representada por un conjunto de funciones que determinan la forma en que desde ésta se accede a la capa o .

La implementación de la capa o es independiente de la interfaz (comúnmente se dice que es transparente a la capa $o+1$). De este modo, no hay que preocuparse de cómo es la capa o , ya que está implementada.

Una interfaz debe especificar con precisión las funciones que ofrece y cómo se usan (argumentos, valores de retorno, etc.).

Interfaz de usuario (UI)

Cuando no existían los dispositivos que utilizamos hoy en día, el usuario tenía que comunicarse con el sistema tecleando órdenes que le permitían realizar multitud de acciones.

El sistema ofrecía una utilidad específica, denominada **intérprete de comandos (shell)** en la terminología de Unix/Linux), que presentaba como interfaz un conjunto de comandos bien especificados en un manual.

Hoy en día, las GUI facilitan enormemente la forma de interacción del usuario mediante objetos y conceptos intuitivos (iconos, clicks del ratón, arrastrar y soltar...), aunque no se ha perdido un pequeño resquicio de los intérpretes de comando: **el CMD, o símbolo de sistema**.

Interfaz de administración

El administrador del sistema es un profesional que conoce las herramientas y funciones específicas que el sistema operativo ofrece para la gestión de todo el sistema, tanto usuarios como recursos (incluidos los de red).

Sólo él puede usarlas, pues requieren privilegios especiales, basándose en una extensión del intérprete de comandos, aunque el uso de estas herramientas no excluye la utilización de la interfaz gráfica.

Interfaz de programación

A la hora de desarrollar aplicaciones sobre un sistema operativo, el programador utiliza un conjunto de funciones para acceder a los servicios del sistema operativo: la **interfaz de llamadas al sistema**.

Sus funciones no difieren de otras de las librerías que ofrece el lenguaje de programación, aunque las llamadas a un sistema operativo son específicas de ese sistema y, por lo tanto, incompatibles con las de otro sistema operativo ya que se refieren a objetos y conceptos específicos de ese sistema.

Librerías y nombres de archivo de los sistemas operativos

Las **librerías de los sistemas operativos** son, como ya hemos visto, una forma sencilla y potente de *modularizar* y *reutilizar* código. Además, permiten acelerar la carga de ciertas partes necesarias del sistema.

La desventaja de las librerías en los sistemas operativos proviene del uso de ejecutables completamente separados y de su llamada mediante un procedimiento remoto (RPC) a través de una red a otra computadora. Esto se debe a que, aunque maximiza la reutilización del sistema operativo porque el código necesario para admitir la librería es el mismo código que se utiliza para proporcionar soporte de aplicaciones y seguridad para cualquier otro programa, no requiere que la librería exista en la misma máquina, así que se pueden reenviar las solicitudes a través de la red pese a ser un peligro en los tiempos que corren. Así, un uso intensivo de estas llamadas remotas puede producir sobrecarga en las peticiones a red y de carga del propio sistema operativo.

Otras alternativas pasan por las **librerías de generación de código**, aplicaciones que pueden generar o transformar código de bytes a ciertos sistemas utilizados por algunos lenguajes de programación (JAVA entre ellos), a algunos marcos de acceso a datos y para pruebas de generación de objetos proxy dinámicos.

Para realizar los enlaces a estas librerías estáticas y dinámicas en los principales sistemas operativos se utiliza el **nombramiento de archivos**. Es decir, se nombra cada uno de los archivos que formarán parte de estas librerías para el uso de un sistema operativo y la documentación del sistema operativo que estemos utilizando. De este modo se reduce el esfuerzo necesario para la lectura del código fuente y mejora (dando más rapidez de carga y más limpieza si se estructura adecuadamente) la apariencia, los tiempos de carga y las solicitudes que pueden ocupar la memoria que utiliza el sistema operativo.

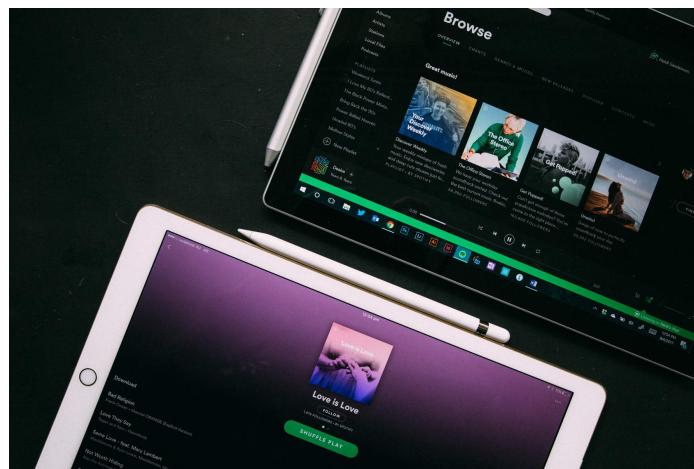
Esta elección depende, o bien de la empresa, o bien de convenciones como las que utiliza [W3C](#), es decir, depende de los intereses que haya que satisfacer en cada momento y para un elevado número de usuarios.

Ahora vamos a estudiar los nombres de archivo pertenecientes a **librerías (estáticas y dinámicas)** en los sistemas operativos informáticos más conocidos del mercado... Veamos

Microsoft Windows

El sistema busca la librería en el directorio actual, en el directorio del sistema (con la función `kernel32 SetDllPath()`) y en las rutas de la variable de entorno **PATH**.

Para los enlaces dinámicos, los archivos suelen tener el sufijo **DLL (Dynamic Linking Library)**, aunque otras extensiones (OCX para **bibliotecas OLE**, etc.) pueden conllevar problemas de compatibilidad (**DLL HELL**), a resolver mediante la aplicación *Dependency Walker*. Según se compilen los archivos, tienen el sufijo **OBJ** o **LIB (bibliotecas de importación o carga dinámica selectiva)**.



Dispositivos con Microsoft Windows.

El **Modelo de Objetos Componentes (COM)** define un estándar y proporciona mecanismos para localizar y versionar archivos, describir la interfaz e implementar poderosos *back-ends* para GUI (Visual Basic y ASP), que se puedan programar desde lenguajes de scripting (HTML).

MacOS e iOS



Dispositivos con MacOS y iOS.

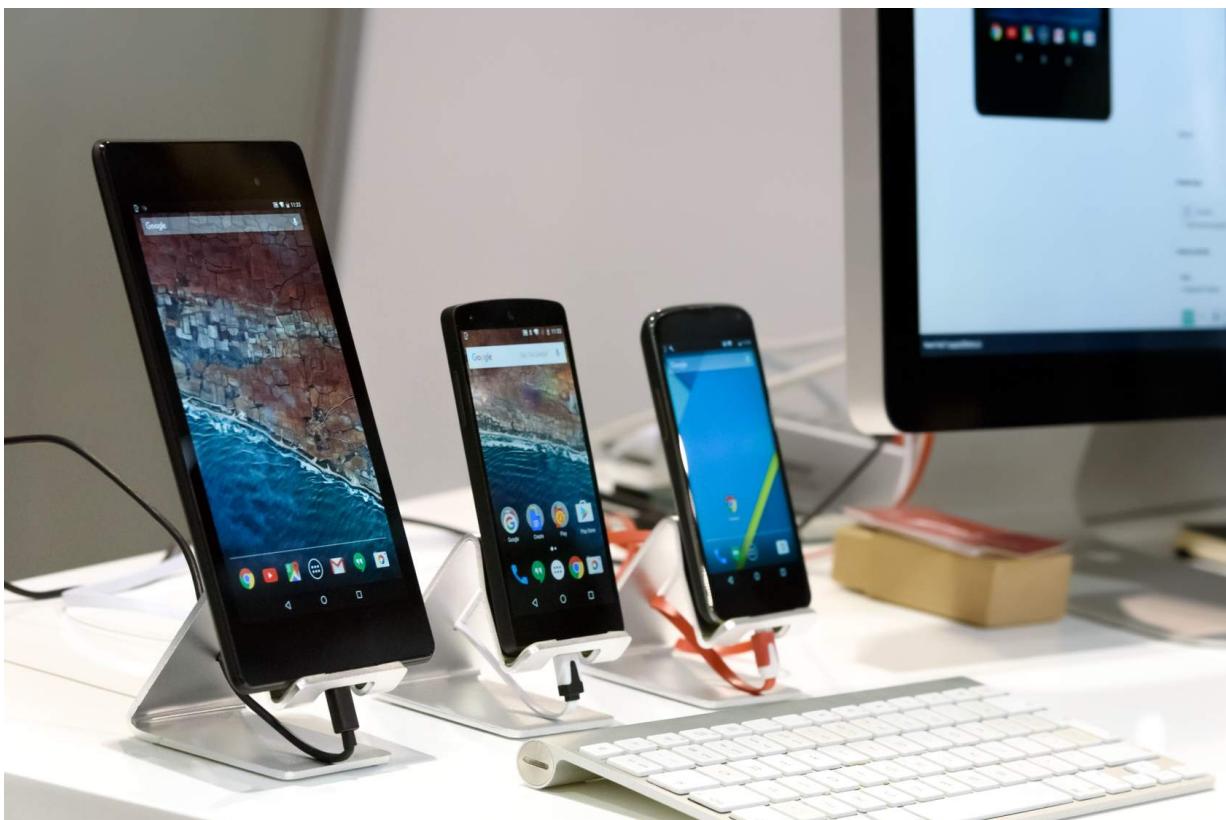
Hereda el nombramiento de archivos de librerías estáticas de **BSD**, con la biblioteca almacenada en un archivo con el sufijo **A**, pudiendo usar **SO** o **DYLIB** para librerías vinculadas dinámicamente.

La mayoría de las librerías son marcos ubicados dentro de directorios especiales, llamados **paquetes**, que envuelven los archivos y metadatos necesarios. El nombre de fichero de una librería dinámica tiene siempre la forma **lib+nombrepaqete+.SO+vención**.

FreeBSD, Android, Linux y sistemas Unix

El sistema almacena los archivos **libfoo.a** y **libfoo.so** en directorios como **/lib** , **/usr/lib** o **/usr/local/lib**, de modo que los nombres de archivo siempre comienzan con **L**I**B** y terminan con un sufijo de **.A** (librería estática) o **.SO** (objeto compartido, librería enlazada dinámicamente), pudiendo incluir la versión principal o el número de versión completo.

La mayoría de los sistemas de tipo Unix tienen una ruta de búsqueda que especifica los directorios del sistema de archivos. Se recomienda a los desarrolladores que coloquen en sus librerías dinámicas rutas de búsqueda predeterminadas.



Dispositivos con Android.

GTK Widgets (Unix, Windows & MacOS X)

```
2196: function(scope, element, attr, ngSwitchController) {
2197:   var selectedScope, element, attr, ngSwitchController;
2198:   var selectedTranscludes = [];
2199:   var previousElements = [];
2200:   var selectedElements = [];
2201:   var previousElements = [];
2202:   var selectedScopes = [];
2203:
2204:   scope.$watch(expr, function ngSwitchWatchAction(value) {
2205:     var i, ii;
2206:     for (i = 0, ii = previousElements.length; i < ii; ++i) {
2207:       previousElements[i].remove();
2208:     }
2209:     previousElements.length = 0;
2210:
2211:     for (i = 0, ii = selectedScopes.length; i < ii; ++i) {
2212:       var selected = selectedElements[i];
2213:       selectedScopes[i].$destroy();
2214:       previousElements[i] = selected;
2215:       $animate.leave(selected, function() {
2216:         previousElements.splice(i, 1);
2217:       });
2218:     }
2219:
2220:     selectedElements.length = 0;
2221:     selectedScopes.length = 0;
2222:
2223:     if ((selectedTranscludes = ngSwitchController.cases['!' + value] || ngSwitchController.cases[value])) {
2224:       scope.$eval(attr.change);
2225:       forEach(selectedTranscludes, function(selectedTransclude) {
2226:         var selectedScope = scope.$new();
2227:         selectedScopes.push(selectedScope);
2228:         selectedScope.$on('$destroy', function() {
2229:           selectedScopes.pop();
2230:         });
2231:       });
2232:     }
2233:   });
2234:
```

Programación de librerías.

GTK significa **GIMP Toolkit**, aplicación de dibujo o librería de componentes base del entorno Gnome y, por tanto, la base de miles de aplicaciones libres y comerciales.

Aunque su plataforma original era el sistema **X-Window** asociado a los sistemas Unix, se ha exportado a otros sistemas operativos como Microsoft Windows y Mac OS X.

wxWidgets - "wxWindows" (multiplataforma)

Hace muchos años, debido a los problemas que suponían los sistemas basados en X-Windows y Microsoft Windows, se buscó un *apaño* que permitiese desarrollar aplicaciones multiplataforma capaces de usar lo común entre plataformas y emular el resto. El resultado se llamó **wxWindows**, que tuvo que cambiar a **wxWidgets** por problemas legales con el nombre.



Dispositivos variados (multiplataforma).

ActiveX (exclusivo de Microsoft Windows)

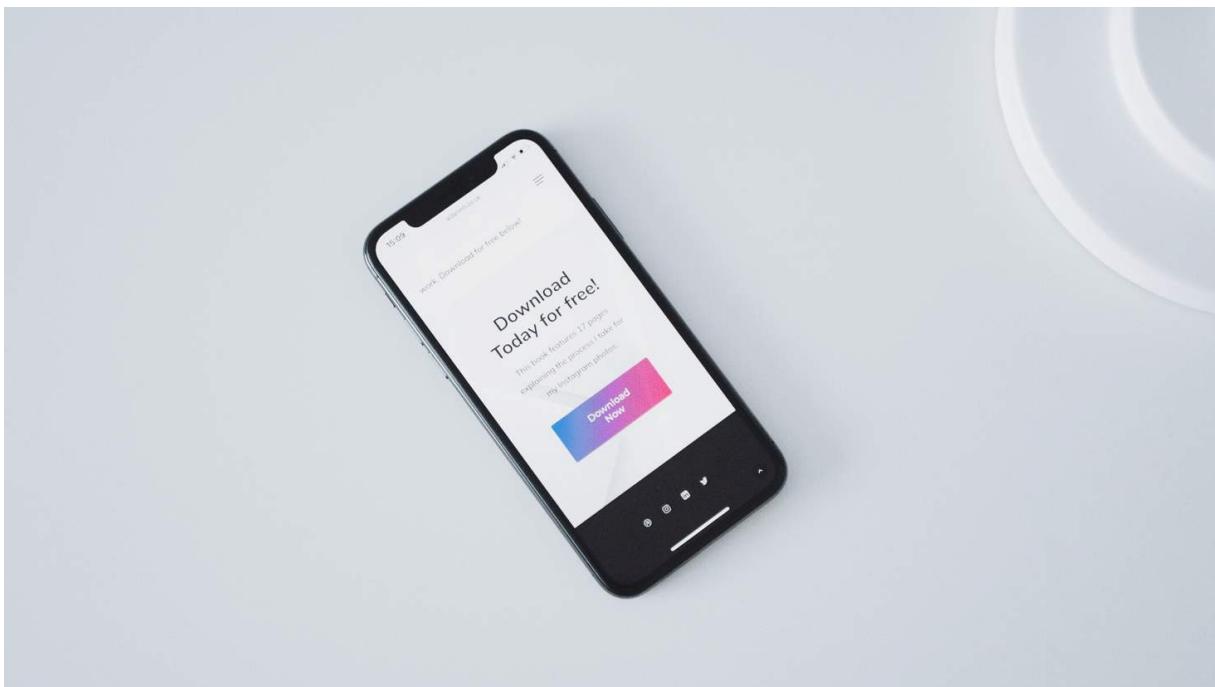
Este modelo de componentes de **Microsoft COM (Component Object Model)**, en el que se basa el sistema operativo Windows, permite almacenar los componentes en una biblioteca de enlace dinámico o parte de un ejecutable (varios componentes dentro de un archivo).

ActiveX incorpora interfaces específicas que añaden mayor funcionalidad y se pueden programar por Powerbuilder, Borland Delphi, Borland C++ Builder, Microsoft Visual Basic, Microsoft Visual C++, Microsoft Office (mediante Visual Basic for Applications), Microsoft Internet Explorer (mediante VBScript or JavaScript), Microsoft Visual J++, etc.

Qt (Windows)

En general, las bibliotecas deben cumplir normas como, por ejemplo, que exista un archivo de cabecera con los prototipos. Si la biblioteca es estática, sólo hace falta ella misma; si la biblioteca es dinámica, hace falta ella misma y un archivo con las definiciones, o que la propia aplicación se encargue de cargar la biblioteca cuando lo desee.

En Qt el lugar adecuado es el archivo de proyecto (**.pro**), donde, por ejemplo, se pueden introducir rutas de búsqueda de archivos de cabecera.



Dispositivos con GUI en fase de prototipado

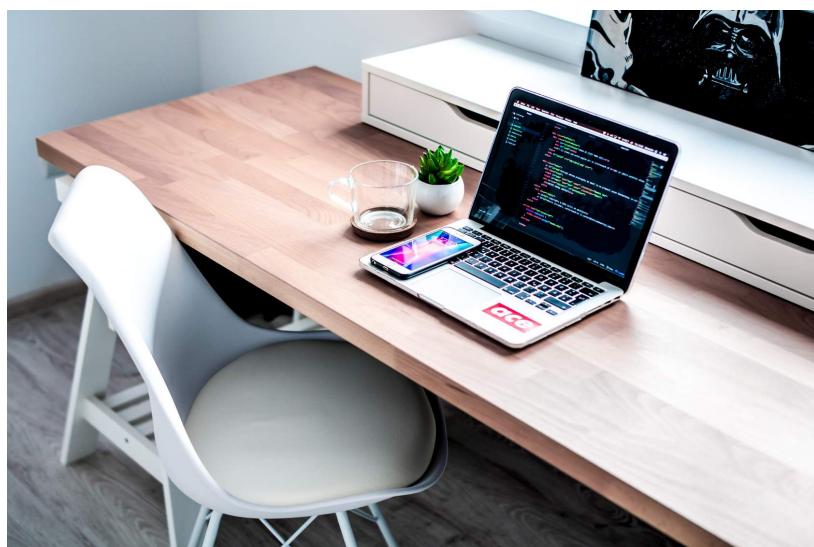
Debemos tener en cuenta que las librerías deben colocarse de forma que, en primer lugar, esté la de más alto nivel y al final, la de nivel más bajo. El motivo es que al compilar se van leyendo consecutivamente y cargando sólo lo necesario de cada una de ellas.

Lenguaje, frameworks y librerías

El uso de las herramientas adecuadas en un proyecto afecta de forma notable a su desarrollo.

Por eso, el lenguaje de programación, los *framework* y las librerías son importantes.

Aunque el concepto de *framework* no es sencillo de definir, se trata de un esquema para el desarrollo y/o la implementación de software, es decir, es el marco que separa la gestión de los datos, las operaciones y la presentación.



Ejemplo típico de Framework: [jQuery](#) (librería Javascript).

Algunos de estos *frameworks* pueden llegar a definir los nombres de ficheros, su propia estructura y las convenciones de programación, o lo que es lo mismo, la normalización a la hora de implementar software.

¿Por qué permitir un completo desorden de código, ficheros de datos y demás desarrollos, si lo mejor es estructurar y normalizar la información? Esto es lo que pretende un *framework*.

Para que las librerías y componentes (externos) puedan ser enlazados a nuestros desarrollos es necesaria la etapa de **enlazado** o *linking*. El comportamiento es prácticamente el mismo que el de los demás programas o entornos, como el caso de algunas librerías que pueden requerir de otras para poder funcionar (por ejemplo, en los CMS o Gestores de Contenidos que vemos en el mercado - WordPress, Magento, Prestashop, etc.), que redefinen o alteran la librería original o la hacen disponible para otras tecnologías.

Para programar hay en el mercado una abundancia de lenguajes, librerías, *frameworks*, o tecnologías disponibles para desarrollar un proyecto. **Podemos elegir cualquiera de ellas o la combinación que más se ajuste a las necesidades del proyecto, las que mejor conozcamos o las preferidas en base a sus características**, incluyendo su comunidad, documentación, si tiene un desarrollo activo, fecha de la última versión, etc.

Por el contrario, esta abundancia requiere un análisis de las más usadas, con mejor documentación o con un mantenimiento activo, lo que nos requerirá tiempo y esfuerzos extra. Por ejemplo, el uso de lenguajes de programación que utilizan *frameworks* nos evitará tener que desarrollar mucho código, ganaremos mayor flexibilidad y habrá menos errores con la dependencia de su uso.



Lo mejor es elegir el lenguaje de programación que nos permita reemplazarlo en un futuro.

Usando componentes: ¿Qué nos proporciona cada lenguaje?

Desde el punto de vista de un programador, un componente está formado por varias secciones: **Configuración, Implementación y Modulación.**

Todos los componentes que se ven en las aplicaciones que utilizamos están escritos con un código que los dota de la apariencia y funcionalidad que el usuario espera, ya sea un botón, un cuadro de texto, un desplegable, etc.

Los primeros desarrolladores tuvieron que trabajar mucho para dar ese aspecto y funcionalidad a cada parte de su aplicación, hasta que se comenzaron a publicar las API de los sistemas operativos, momento en el que surgieron multitud de librerías que facilitan ese acceso para diferentes lenguajes de programación.

Librerías: Lenguajes de programación más conocidos

JAVA (LWJGL, AWT y Javax.swing)

Son el resultado de compilar el código fuente desarrollado por quien implementa la JRE (Java Runtime Environment), y ofrecen apoyo para el desarrollo en Java. Estas librerías facilitan al programador un conjunto definido de funciones para realizar tareas comunes, proporcionando una interfaz abstracta para tareas que son altamente dependientes del hardware de la plataforma destino y de su sistema operativo.

- **LWJGL (Lightweight Java Game Library):** es una solución destinada a la creación de juegos de calidad comercial escritos en el lenguaje Java que proporciona a los desarrolladores acceso a diversas librerías multiplataforma como OpenGL (Open Graphics Library) y OpenAL (Open Audio Library), permitiendo la creación de juegos de alta calidad con gráficos y sonido 3D, además del acceso a controladores de juegos como GamePads, volantes y Joysticks.
- **AWT (Abstract Window Toolkit):** está contenida en la rama java de la API del lenguaje, lo que permite desarrollar GUI. Está compuesta por componentes como los Buttons, Labels, etcétera.
- Por último, **Javax.swing** proporciona una serie de clases e interfaces que amplían la funcionalidad del anterior, ya que es la versión más moderna del API de Java. Están escritos en Java y son independientes de la plataforma.

Python (Pygame)

Posee un archivo que denomina **módulo** (contiene las declaraciones ejecutables y definiciones de funciones y su propio espacio de nombres, usado de forma global). Estos módulos pueden ser importados a otros módulos, o al módulo principal, para inicializarlo solamente la primera vez que se importa. El nombre del archivo es el nombre del módulo con el sufijo .py agregado.

Python viene con una librería de módulos estándar, de modo que algunos se integran en el intérprete (proveen acceso a operaciones que no son parte del núcleo del lenguaje), aunque el conjunto de módulos es una opción de configuración que depende de la plataforma.

Pygame es un conjunto de módulos del lenguaje Python que permite la creación de videojuegos en dos dimensiones de una manera sencilla, orientado al manejo de sprites. Pygame esta basado en la librería SDL 1.2.

C y C++ (Allegro y SDL)

Las librerías estáticas, denominadas también **librerías-objeto**, son colecciones de ficheros objeto (**compilados**) agrupados en un solo fichero de extensión .lib, .a (principalmente), junto con uno o varios ficheros de cabecera .h. En el caso de las librerías dinámicas, durante la construcción de la aplicación se incluyen en los archivos fuente los ficheros de cabecera. Posteriormente, el linker incluye en el ejecutable los módulos correspondientes a las funciones y clases de librería que hayan sido utilizadas en el programa, de forma que el conjunto entra a formar parte del ejecutable. Cualquiera que sea el caso (tanto la utilización como la construcción) son diferentes según se trate de librerías estáticas o dinámicas.

Allegro: es una librería libre y de código abierto para la programación de videojuegos desarrollada en lenguaje C. Cuenta con funciones para gráficos, manipulación de imágenes, texto, sonidos, dispositivos de entrada (teclado, ratón y joystick), etc. Aunque ofrece una API en lenguaje C, actualmente existen envolventes y bibliotecas adicionales que permiten utilizarlo en otros lenguajes, como C++, Java, C#, Visual Basic.NET, Python, D, Lua, Pascal, Perl, Ruby, Go, Scheme, Common Lisp, Ocaml, Forth y Mercury.

SDL (Simple DirectMedia Layer): es un conjunto de librerías desarrolladas en el lenguaje de programación C que proporciona funciones básicas para realizar operaciones de dibujo en dos dimensiones, gestión de efectos de sonido y música, además de carga y gestión de imágenes. Es utilizado por otros lenguajes de programación como C++, Ada, C#, BASIC, Erlang, Lua, Java, Python, etc. También proporciona herramientas para el desarrollo de videojuegos y aplicaciones multimedia.

VCL (Windows)

Es el modelo nativo de componentes de C++ Builder, basado en la **Visual Component Library (VCL)**, desarrollada por Borland para Delphi, cuyos componentes son clases C++, pudiendo usarse únicamente con Borland C++ Builder y Borland Delphi.

JavaScript (jQuery, Underscore, D3.js y Modernizr)

Javascript es un lenguaje de programación que permite a los desarrolladores crear acciones en sus páginas web. Se trata de un lenguaje basado en acciones, es decir, que posee menos restricciones, además de poder ser utilizado Windows y sistemas X-Windows. Existen dos tipos de JavaScript: **Navigator JavaScript**, que se ejecuta en el cliente, y **LiveWire Javascript**, que se ejecuta en el server.

Jquery es, sin lugar a dudas, la librería más popular que existe en JavaScript. Principalmente es usada para manipular el **DOM (Document Object Model)**, cuya estructura de tipo árbol representa, por ejemplo, todos los elementos de una página web. Con esta librería se pueden hacer llamadas a **AJAX**.

Por otro lado, la librería **Underscore** posee más de cien funciones que nos ayudan a manipular arreglos, objetos y demás, aunque la funcionalidad más llamativa es la de **ayudantes de programación funcional (functional programming helpers)**.

En el caso de la librería **D3.js**, se utiliza para la visualización de información, ya que ofrece la opción de crear varios gráficos tipo Excel e incluso 3D, ofreciendo una API bastante completa en la cual podemos usar selectores y manipular elementos del DOM de una forma similar a como lo hacemos con JQuery.

No podía faltar **Modernizr**, la librería que se encarga de validar si nuestro navegador soporta una u otra característica de HTML5, lo que facilita a los desarrolladores actuar en consecuencia.

AJAX (Ajax.NET Professional, AjaxRequest Library, Bajax y Rico)

El lenguaje de programación **Asynchronous Javascript and XML** se utiliza principalmente para solicitar información a un servidor, pero también se encarga de las incompatibilidades entre los diferentes navegadores.

Ajax.NET Professional es una de las primeras librerías AJAX disponibles para Microsoft ASP.NET y trabaja con .NET 1.1 y 2.0.

En el caso de **AjaxRequest Library**, se simplifica y extiende las capacidades del objeto **XMLHttpRequest**, y permite desarrollar los proyectos sin tener que preocuparnos por los procesos a bajo nivel.

Por otro lado, está **Bajax**, una pequeña librería JavaScript para usar AJAX en nuestras páginas web, totalmente independiente del lenguaje de programación, con la que se puede mostrar contenido dinámico usando comandos simples.

Rico es una librería de efectos Ajax disponible en OpenRico que simplifica el desarrollo de aplicaciones. Dispone de algunos efectos gráficos, tablas actualizables y secciones de drag & drop.

XML (tinyXML, JAXP y xerces)

XML (Extensible Markup Language) es un subconjunto de **SGML** (Estándar **Generalised Mark-up Language**), simplificado y adaptado a Internet, no una versión mejorada de HTML, y por supuesto, no es un lenguaje para hacer páginas WEB. Se utiliza principalmente para representar información estructurada en la web, de modo que esta información pueda ser almacenada, transmitida, procesada, visualizada e impresa por diversos tipos de aplicaciones y dispositivos.

TinyXML es una pequeña librería autocontenido, y suele incluir los archivos fuentes en el código del sistema que la utiliza.

JAXP (Java API for XML Processing) es la librería que permite procesar, tanto el estándar DOM, como el estándar SAX, y ha sido diseñada para ser flexible y uniformar el desarrollo de aplicaciones Java con Xml.

Xerces es muy utilizada en sus formatos C++ y Java. Posee una gran cantidad de funcionalidades extra, no siempre disponibles en otras librerías. JDOM permite leer, escribir, crear y manipular ficheros XML de forma sencilla e intuitiva. Está programada en Java, lo que le permite utilizar las capacidades particulares del lenguaje, simplificando significativamente su manejo. Se basa en el procesamiento de un documento XML y la construcción de un árbol.

Existen librerías cuya especificación estándar define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D (**wXWINDOWS**, **QT**, **GTK**, **DirectX**-
Direct3D u **OpenGL**).

¿Qué es Javascript?

JavaScript (JS) es un lenguaje muy utilizado para crear pequeños programas, que luego serán insertados en páginas web y en programas más grandes, orientados a objetos mucho más complejos que nos permiten diferentes efectos e interactuar con nuestros usuarios.

JavaScript fue creado por **Brendan Eich** y vio la luz en el año 1.995 con el nombre de **LiveScript**, posteriormente nombrado **JavaScript**. Nació como un lenguaje sencillo destinado a añadir algunas características interactivas a las páginas web, ya que HTML sólo permitía crear páginas estáticas, donde se podía mostrar textos con estilos, pero se necesitaba interactuar con los usuarios.

Estas son algunas de sus características:

- Es uno de los lenguajes basado en acciones que posee menos restricciones.
- Utiliza Windows y sistemas X-Windows.
- Gran parte de la programación en este lenguaje está centrada en describir objetos.

Lo primero que tenemos que tener en cuenta es que hay dos tipos de **JavaScript**:

- **Cliente:** realiza peticiones a una aplicación o a otro dispositivo. Técnicamente es denominado Navigator JavaScript.
- **Server (servidor):** es más reciente. Técnicamente se denomina LiveWire Javascript. Responde a las peticiones de los dispositivos o aplicaciones cliente.

Actualmente disponemos de JavaScript en Internet:

- Correo y chat.
- Buscadores de información.
- Reloj, contadores de visitas y fechas.
- Calculadoras, validadores (acceso a intranets, formularios y documentación).
- Detectores de navegadores e idiomas.

Arquitectura Cliente-Servidor

Esta arquitectura **se refiere a un modelo de comunicación que vincula a varios dispositivos informáticos a través de una red de modo que las tareas se distribuyen entre los servidores** (que proveen los servicios) **y los clientes** (que demandan dichos servicios). Es decir, el cliente le pide un recurso al servidor, que le brinda una respuesta.

Este tipo de modelo permite repartir la capacidad de procesamiento.



Granja de servidores [imgIX](#).

¿Son lo mismo JAVA y JavaScript?

Esta pregunta puede parecer obvia a muchos programadores que aún no conocen ambas tecnologías. **La respuesta es que no, Java y JavaScript no son lo mismo.**

Cierto es que comparten una palabra al principio de su nombre, por lo que muchas personas piensan que **JAVA** y **JavaScript** están relacionados, o que JavaScript es un subconjunto de **JAVA** y cosas parecidas... Nada más lejos de la realidad.

JAVA nació en 1.991 en la empresa **Sun Microsystems**, de la mano de **James Gosling**. Su objetivo inicial era crear aplicaciones para receptores de televisión y dispositivos embebidos, aunque al final haya terminado funcionando en casi todas partes.

JavaScript es un lenguaje de programación posterior, cuyo nombre en código durante el desarrollo del lenguaje era **Mocha**, aunque se pensó lanzarlo finalmente como **LiveScript** para denominarlo definitivamente **JavaScript** por la gran influencia en su sintaxis por parte del lenguaje **Java**, aunque las similitudes son únicamente estéticas.

Aparte de esa raíz común en el nombre, **JAVA** y **JavaScript** son lenguajes totalmente diferentes:

- **JavaScript es un lenguaje interpretado, y JAVA es compilado:** los programas JavaScript son archivos de texto que pueden leer tanto los ordenadores como las personas, mientras que los de JAVA se compilan a un archivo especial optimizado para que lo lea y lo ejecute un ordenador.
- **JAVA depura errores en dos fases y JavaScript en una.**
- **JAVA es un lenguaje orientado a objetos puro, y JavaScript está basado en prototipos** (simulaciones de orientación a objetos).
- **JAVA es fuertemente tipado** (las variables tienen un tipo determinado) y **JavaScript es débilmente tipado**, ya que en JAVA todas las variables tienen un tipo determinado y una vez definidas no se pueden cambiar.
- **JAVA tiene ámbito por bloque y JavaScript lo tiene por función:** el acceso a las variables depende de dónde las hayamos definido.
- **Las funciones en JavaScript son multi-argumento siempre y en JAVA es necesario indicarlo.**
- **JavaScript es estándar, JAVA no.**

En resumen, y rompiendo falsos mitos y leyendas urbanas:

1. JavaScript NO es lo mismo que JAVA.
2. JavaScript NO es un subconjunto de JAVA.
3. JavaScript NO es una versión de JAVA para programar páginas web.
4. JavaScript NO comparte prácticamente nada con JAVA.
5. JavaScript NO es más fácil que JAVA.
6. Si sabes programar en JAVA NO sabes programar en JavaScript, y viceversa.
7. JavaScript NO es sólo un lenguaje para páginas web, al igual que JAVA no es un lenguaje sólo para servidor.

Bibliotecas y Frameworks JavaScript

Hay millones de páginas web. Y muchas de ellas son interactivas de la misma manera, ya que probablemente, reutilizaron en su construcción el mismo código para no perder el tiempo escribiendo.

En JavaScript, la manera de reutilizar código es usando una biblioteca, un archivo de JS que contiene muchas funciones *precocinadas* creadas por un programador que la hace disponible para todo el mundo. Al incluir el código de JS en cualquiera de nuestros sites, cargamos en el <HEAD> el archivo o archivos que necesitemos para posteriormente programar o utilizar las bibliotecas que necesitemos, de la misma forma que se podría cargar un CSS.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="utf-8" />
        <title>Incluyendo JS en HTML5</title>
        <script type="text/javascript">
            function alerta() {
                alert('hola mundo!');
            }
        </script>
    </head>
    <body>
        <section>
            <span onclick="alerta();">Clic aqu&iacute;</span>
        </section>
    </body>
</html>
```

Ejemplo típico JS: *Hola Mundo* con una función creada desde 0.

Hay miles de bibliotecas de JS que se pueden importar para una web y muchos son los aspectos en que éstas nos pueden ayudar si pensamos que una web se compone de la UI (HTML y CSS), interactividad (JS + DOM) y datos (que a menudo traemos por medio de JS):

- Manipulación del DOM (**Modelo de Objetos del Documento**) y eventos del DOM.
- AJAX / recuperación de datos.
- Efectos y animación.
- Plantillas de HTML, diseño de página, widgets de interfaz de usuario.
- Gráficas, tablas, modelado de datos, rutas y navegación.
- Accesibilidad, soporte para múltiples navegadores y soporte móvil.

Una de las partes más difíciles del desarrollo web es decidir qué bibliotecas utilizar y su versión. **No hay una respuesta correcta**. Debes conocer las opciones y después tomar una decisión informada, esta es la solución más efectiva.

Veamos algunos **ejemplos de bibliotecas JS** muy conocidas en el mercado actual:

- [jQuery](#)
- [jQuery UI](#).
- [jQuery Mobile](#).
- [Sizzle](#).
- [Qunit](#).
- [modernizr](#).

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Hola mundo con jQuery</title>
        <script type="text/javascript" src="http://ajax.microsoft.com/ajax/-jquery/jquery-1.4.4.min.js"></script>
        <script type="text/javascript">
            $(document).ready(function() {
                $('#hola').text("Hola, Mundo!! Soy jQuery!!"); });
        </script>
    </head>
    <body>
        <div id="hola">Hola mundo</div>
    </body>
</html>
```

```
</script>
</head>
<body>
  <div id="hola"></div>
</body>
</html>
```

Ejemplo de carga de la biblioteca jQuery: *Hola Mundo* cargado con una biblioteca muy utilizada en JS.

Frameworks

Hay algunas bibliotecas que hacen todo, desde la recuperación de datos hasta la manipulación del DOM y widgets elegantes en la interfaz de usuario.

Si se utiliza uno de estos frameworks, típicamente estarás trayendo mucho JS a la página. Para algunos desarrolladores, estos frameworks hacen demasiado, por lo que prefieren usar bibliotecas más pequeñas con tareas más enfocadas. Veamos algún ejemplo:

- [AngularJS](#).
- [ReactJS](#).
- [Vue.js](#).
- [Ember.js](#).
- [Aurelia](#).
- [nodeJS](#).

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="Ejemplo uso frameworks JS">
    <meta name="author" content="JS">
```

```
<link rel="icon" href="..../favicon.ico">

<title>Ejemplo Framework Bootstrap</title>

<!-- Bootstrap core CSS -->
<link href="..../dist/css/bootstrap.min.css" rel="stylesheet">

<!-- IE10 viewport hack for Surface/desktop Windows 8 bug -->
<link href="..../assets/css/ie10-viewport-bug-workaround.css" rel="stylesheet">

<!-- Custom styles for this template -->
<link href="jumbotron.css" rel="stylesheet">

<!-- Just for debugging purposes. Don't actually copy these 2 lines! -->
<!--[if lt IE 9]><script src="..../assets/js/ie8-responsive-file-warning.js"></script><![endif]-->
<script src="..../assets/js/ie-emulation-modes-warning.js"></script>

<!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
<!--[if lt IE 9]>
<script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>
<script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
<![endif]-->
</head>

<body>

<nav class="navbar navbar-inverse navbar-fixed-top">
<div class="container">
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="#">Project name</a>
</div>
<div id="navbar" class="navbar-collapse collapse">
<form class="navbar-form navbar-right">
<div class="form-group">
<input type="text" placeholder="Email" class="form-control">
</div>
<div class="form-group">
```

```
<input type="password" placeholder="Password" class="form-control">
</div>
<button type="submit" class="btn btn-success">Sign in</button>
</form>
</div><!--/.navbar-collapse -->
</div>
</nav>

<!-- Main jumbotron for a primary marketing message or call to action -->
<div class="jumbotron">
<div class="container">
<h1>Hello, world!</h1>
<p>This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and three supporting pieces of content. Use it as a starting point to create something more unique.</p>
<p><a class="btn btn-primary btn-lg" href="#" role="button">Learn more &raquo;</a></p>
</div>
</div>

<div class="container">
<!-- Example row of columns -->
<div class="row">
<div class="col-md-4">
<h2>Heading</h2>
<p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui. </p>
<p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
</div>
<div class="col-md-4">
<h2>Heading</h2>
<p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui. </p>
<p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
</div>
<div class="col-md-4">
<h2>Heading</h2>
<p>Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.</p>
<p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
</div>
```

```
</div>

<hr>

<footer>
    <p>&copy; 2016 Company, Inc.</p>
</footer>
</div> <!-- /container -->

<!-- Placed at the end of the document so the pages load faster --&gt;
&lt;script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/j-
query.min.js"&gt;&lt;/script&gt;
&lt;script&gt;window.jQuery || document.write('&lt;script src=".../assets/js/ven-
dor/jquery.min.js"&gt;&lt;/script&gt;')&lt;/script&gt;
&lt;script src=".../dist/js/bootstrap.min.js"&gt;&lt;/script&gt;
<!-- IE10 viewport hack for Surface/desktop Windows 8 bug --&gt;
&lt;script src=".../assets/js/ie10-viewport-bug-workaround.js"&gt;&lt;/script&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

Ejemplo de carga del *framework* Bootstrap.

Ejemplo

Puedes ver el ejemplo anterior en funcionamiento pulsando en este enlace:

[IR AL EJEMPLO](#)

Aunque programar nosotros solos provoca una gran satisfacción, es conveniente superar el síndrome del "No inventado aquí" y sacar partido al código que tan generosamente han donado al bien común otros desarrolladores.

Resumen

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

- Una librería es un conjunto de subprogramas con una UI bien definida para ser invocados y utilizados en el desarrollo de software o para comunicarnos con el dispositivo en cuestión.
- Una librería se puede integrar de forma estática o dinámica. La utilización de las librerías optimiza el tiempo de desarrollo y pruebas.
- Un *framework* es un esquema para el desarrollo y/o la implementación de software, es decir, el marco que separa la gestión de los datos, las operaciones y la presentación.
- El lenguaje de programación, los *framework* y las librerías importan. Elegir el lenguaje de programación adecuado es fundamental.
- JAVA y JavaScript no son lo mismo. JavaScript es un lenguaje muy utilizado para crear pequeños programas que luego son insertados en una página web y en programas más grandes, orientados a objetos mucho más complejos, que nos permiten diferentes efectos e interactuar con nuestros usuarios.



PROEDUCA