



POLYMER 1.

Value
THROUGH
TECHNOLOGY

Restricciones

Nombre del documento:

Clasificación de la Información: Interno

Restricciones

- › Los contenidos de este documento son propiedad de Softtek y son internos. Queda estrictamente prohibido cualquier reproducción total o parcial sin la autorización escrita por parte de Softtek.
- › Este documento está sujeto a cambios. Los comentarios, correcciones o dudas deberán ser enviados al autor.

Audiencia	Propósito
Desarrolladores	Capacitación

Tabla de Revisión

- › La siguiente tabla enlista las revisiones realizadas a este documento. Debe utilizarse para describir los cambios y adiciones cada vez que este documento vuelva a ser publicado. La descripción debe ser detallada e incluir el nombre de quien solicita los cambios.

Núm. de versión	Fecha de versión	Tipo de cambios	Dueño / Autor	Fecha de revisión / Expiración
1.1	08/8/2018	Creación del documento	Carlos Montero	08/08/2018

POLYMER.

Polymer es una librería para realizar Web Components de una manera más rápida y productiva, que además contiene un amplio catálogo de elementos disponibles para basar el desarrollo de nuevos componentes.

Polymer, cuando se describe a sí mismo, lo hace con las siguientes palabras: "Polymer is a new type of library". La clave en esa descripción es que Polymer está construido sobre la base de los estándares abiertos de los "Web Components". Por decirlo de otro modo, toda la "magia" que permite a Polymer funcionar es Javascript estándar, basada en especificaciones de la W3C.

Los propios creadores de Polymer se esfuerzan en insistir en el hecho de estar en frente de una librería y no un framework. Polymer no es MVC, como lo podría ser AngularJS o EmberJS. Tampoco hace uso de una arquitectura definida, que debas seguir para el desarrollo de una aplicación completa, pues se ciñe únicamente a la arquitectura para **crear componentes aislados que se puedan distribuir, reutilizar y combinar entre sí (es la definición general de los web components)**.

Por poner un ejemplo, AngularJS (en su versión 1.x) incluye las directivas, que nos permiten extender el HTML para crear tags enriquecidos que son capaces de hacer cosas diversas. Para quien no conoce las directivas de Angular, éstas permiten especializar etiquetas, aportando comportamientos diversos. Esas directivas están construidas en base a código propio de AngularJS, mientras que Polymer, que hace algo similar, está construido en base a las herramientas existentes dentro del Javascript de los navegadores modernos.

WEB COMPONENTS

Los Web Components nos ofrecen un estándar que va enfocado a la creación de todo tipo de componentes utilizables en una página web, para realizar interfaces de usuario y elementos que nos permitan presentar información (o sea, son tecnologías que se desarrollan en el lado del cliente). Los propios desarrolladores serán los que puedan, en base a las herramientas que incluye Web Components crear esos nuevos elementos y publicarlos para que otras personas también los puedan usar.

En resumen, este nuevo estándar viene a facilitar la creación de nuevos elementos que enriquezcan la web. Pero además, está pensado para que se puedan **reutilizar de una manera sencilla y también extender**, de modo que seamos capaces de crear unos componentes en base a otros.



WEB COMPONENTS

El ejemplo más típico que veremos por ahí es un mapa de Google. Hoy, si no usamos web components, cuando queremos mostrar un mapa en una página web, tenemos que crear código en tres bloques.

1. Un HTML con el elemento donde se va a renderizar el mapa
2. Un CSS para definir algún estilo sobre el mapa, por ejemplo sus dimensiones
3. Lo más importante, un Javascript para que puedas generar el mapa, indicando las coordenadas que deseas visualizar (para centrar la vista inicial) y muchos otros detalles de configuración que tu mapa necesite.

Son tres lenguajes diferentes, que se especifican en bloques de código separados y usualmente en archivos separados. Sin Web Components, para tener todos los bloques agrupados y tener un código único para embeber un elemento se usaba generalmente la etiqueta IFRAME, que permite cargar un HTML, CSS y Javascript y reducir su ámbito a un pequeño espacio de la página. Esta técnica se sigue utilizando, pero en el futuro se va a sustituir gracias a las bondades de los Web Components.

A partir de ahora podremos expresar un mapa de Google con una etiqueta propietaria, que no pertenece al estándar del HTML, que simplifica la tarea y la acota a un pequeño bloque independiente.

```
<google-map latitude="12.678" longitude="-67.211"></google-map>
```

Es un ejemplo tomados directamente de un Web Components real, creado por el equipo de Google.

WEB COMPONENTS

El ejemplo de la lamina anterior tiene como intención reflejar:

1. Es como si estuviéramos inventando etiquetas nuevas. Esa es una de las capacidades de los Web Components, pero no la única.
2. La etiqueta propietaria que estamos inventando es "google-map"
3. No tenemos el HTML por un lado, el CSS y el Javascript por otro. Es simplemente la etiqueta nueva y ésta ya es capaz de lanzar el comportamiento.
4. Obviamente, en algún lugar habrá un Javascript que se encargará de procesar esa etiqueta, pero será genérico para cualquier tipo de mapa y reutilizable. Lo que además debe verse es que en el HTML estás colocando información que antes estaría en el Javascript. Por ejemplo en el caso del mapa de google los atributos `latitude="12.678"` `longitude="-67.211"` antes eran datos que se escribían en el Javascript. Ahora se declaran en el HTML. El Javascript por tanto es genérico y no tendremos que programarlo nosotros, sino que nos vendrá dado por Google o por el creador del web component de turno.

ESPECIFICACIONES EN WEB COMPONENTS.

Ahora que ya hemos entendido alguna cosa de lo que son los componentes web, el concepto en sí, vamos a ser un poco más formales y describir las distintas especificaciones que podemos encontrar en los Web Components.

Custom Elements:

Esta especificación describe el método que nos permitirá crear nuevas etiquetas personalizadas, propietarias. Estas etiquetas las podremos ingeniar para dar respuesta a cualquier necesidad que podamos tener.

HTML Templates:

Incorpora un sistema de templating en el navegador. Los templates pueden contener tanto HTML como CSS que inicialmente no se mostrará en la página. El objetivo es que con Javascript se acceda al código que hay dentro del template, se manipule si es necesario y posteriormente se incluya, las veces que haga falta, en otro lugar de la página.

ESPECIFICACIONES EN WEB COMPONENTS.

HTML Imports:

Permite importar un pedazo de código que podrás usar en un lugar de tu página. Ese código podrá tener HTML, CSS y Javascript. El HTML no se visualizará directamente en la página, pero lo podrías acceder con Javascript e inyectar en algún lugar. Pero aunque se llame específicamente "HTML Imports", realmente sirve para cargar de una manera única tanto HTML como CSS como Javascript. Además podrás tener dentro un "HTML Template", con las ventajas que ellos aportan. Mediante código Javascript seremos capaces también de registrar componentes personalizados "Custom Elements" o realizar otro tipo de acciones sobre el contenido de la página que sean necesarias.

Shadow DOM:

Este sistema permite tener una parte del DOM oculta a otros bloques de la página. Se dice comúnmente que estamos encapsulando parte del DOM para que no interfiera con otros elementos de la página. Básicamente te sirve para solucionar un caso común que ocurre al incluir un plugin de terceros. A veces usan clases o identificadores para aplicar estilos que afectan a otros elementos de la página, descolocando cosas que no debería o alterando su aspecto. Pues con el Shadow DOM podemos hacer que los componentes tengan partes que no estarían visibles desde fuera, pudiendo colocar estilos que solo afectan al Shadow DOM de un web component y evitando que estilos de la página sean capaces de afectar al Shadow DOM.

Librerías para Web Components

En cuanto a librerías Javascript para producir Web Components hay que aclarar primero que realmente no hacen falta. Como has visto, los Web Components forman parte de un estándar, que está siendo discutido todavía en mayor medida, pero es un estándar. Eso quiere decir que, más tarde o temprano, todos los navegadores lo tendrán en su "core" y podrás usarlo con tan solo usar Javascript estándar, sin necesidad de ninguna librería adicional.

No obstante, lo cierto es que diversos actores se han apresurado a presentar algunas librerías que nos permiten desarrollar hoy mismo con la tecnología de los Web Components. Te las resumimos a continuación:

Polymer:

Es una librería impulsada por Google que actualmente es el mayor referente en cuanto a Web Components. Desarrollada para aprovechar la tecnología del estándar, facilitando la creación de interfaces de usuario reutilizables.

X-Tag:

Es la apuesta de Mozilla para la creación de Web Components, específicamente los custom elements, al alcance de todos los navegadores modernos.

Bosonic:

Herramientas y utilidades para crear web components incluso en navegadores no tan nuevos, como Internet Explorer 9.

Polymer o AngularJS o similares.

Volviendo a la posible competencia con otros frameworks hay que admitir tanto una cosa como la otra. Por una parte es perfectamente posible usar Polymer en conjunto con otras librerías y por otra parte sería muy factible construir una aplicación basada en Polymer sin usar ningún otro framework del lado del cliente. Esto es así porque en el core de la librería y adicionalmente en el catálogo de elementos de Polymer, de los que hablaremos después, están implementadas muchas de las funcionalidades que nos ofrecen los frameworks Javascript MVC. Ejemplos concretos son el binding o el acceso asíncrono a todo tipo de recursos mediante Ajax. Paralelamente, se están presentando constantemente complementos (nuevos componentes) que permiten a Polymer hacer más cosas típicas que vienes haciendo con frameworks como Angular como por ejemplo gestionar rutas de la aplicación por medio de un sistema de routing.

Lo que debe quedar claro es que Polymer se diferencia, con respecto a AngularJS o ReactJS, por estar construido encima de las especificaciones de los Web Components. Por ello no son librerías comparables y, gracias a basarse en estándares de la W3C, se podría suponer una vida más larga a Polymer que a otras alternativas para el desarrollo de interfaces de usuario y aplicaciones web.

Qué Contiene Polymer.

Polymer tiene diversos elementos enfocados en la creación de Web Components:

- Un completo sistema de Polyfills, que permiten dar soporte al estándar de Web Components a navegadores que no lo han implementado todavía de manera nativa.
- Un kit de herramientas destinadas a que cualquier desarrollador pueda crear sus propios componentes.
- Una enorme librería de elementos clasificados en varias áreas, en los cuales podremos basarnos para hacer nuevos componentes que den vida a páginas web y aplicaciones para móviles.

Desarrollo declarativo.

Lo interesante de los web components y Polymer es que te permite hacer lo que se conoce como "desarrollo declarativo". En vez de crear tus comportamientos escribiendo código Javascript que realice las cosas que necesitas, vamos a comenzar a desarrollar en base a la declaración de elementos con etiquetas HTML nuevas que realicen las tareas que necesitas.

O sea, para agregar un icono no necesitas tener una imagen, asociada a una URL en el atributo src.

Puedes implementarlo por medio de un Custom element, que viene a ser como una nueva etiqueta del HTML. Por ejemplo:

```
<iron-icon icon="icons:alarm"></iron-icon>
```

Pero un icono es un componente muy sencillo, quizás la diferencia entre usar una etiqueta IMG y este custom element es bien poca. Pero piensa por ejemplo en un panel lateral, que se colapsa automáticamente en pantallas de dimensiones pequeñas y que podemos desplegar para ver su contenido.

```
<paper-drawer-panel> <div drawer> Panel lateral </div> <div main> Panel del cuerpo </div> </paper-drawer-panel>
```

Si para hacer una conexión Ajax antes tenías que escribir un bloque entero de Javascript, con Polymer podrás escribirlo con un Custom Element:

```
<iron-ajax url$="https://restcountries.eu/rest/v1/name/spain"></iron-ajax>
```

Catalogo de elementos en Polymer.

El catálogo de elementos está dividido en varias secciones:

Fe: Iron Elements

Estos elementos forman parte del "core" de Polymer y generalmente están pensados para no usarlos de manera única, sino para usarlos en conjunto con otros. Por ejemplo encontraremos iconos, que podrías usar sueltos en una página, pero generalmente los combinarás con otros elementos para hacer barras de navegación, listas decoradas, etc.

Md: Paper Elements

Es una lista de interfaces de usuario útiles para muchos tipos de proyectos. Están basados en la línea estética y funcional del "material design" de Google, por lo que ya nos ofrecen una buena base para poder hacer aplicaciones bastante atractivas visualmente.

GO: Google Web Components

Es un catálogo de componentes que forman un wrapper a diversas API de Google. Por ejemplo podemos usar esos "envoltorios" para crear mapas de Google, Acceder a documentos de Drive, trabajar con calendarios, etc. abstrayéndonos de cómo funcionan por dentro esas APIs.



Catalogo de elementos en Polymer.

Au: Gold Elements

Estos sirven específicamente para el comercio electrónico, formularios, sistemas de validación de tarjetas de crédito, etc.

Ne: Neon Elements

Sirven para crear animación y efectos especiales en componentes.

Pt: Platinum Elements

En esta clasificación encontramos elementos que sirven para asuntos relacionados con la operativa de aplicaciones para móviles. Como por ejemplo, poder trabajar offline y poder operar con la página aunque no se tenga conexión, trabajo con notificaciones, etc.

Mo: Molecules

Son envoltorios de librerías de terceros, para usar mediante la arquitectura de los componentes web.

Puedes explorar el catálogo de elementos de Polymer en: <https://elements.polymer-project.org/>

Que necesitamos para iniciar con Polymer.



- ATOM
- NODE
- BOWER

ATOM ya se instalo en el modulo de CSS, para esta parte del curso ya debes estar familiarizado con su uso.

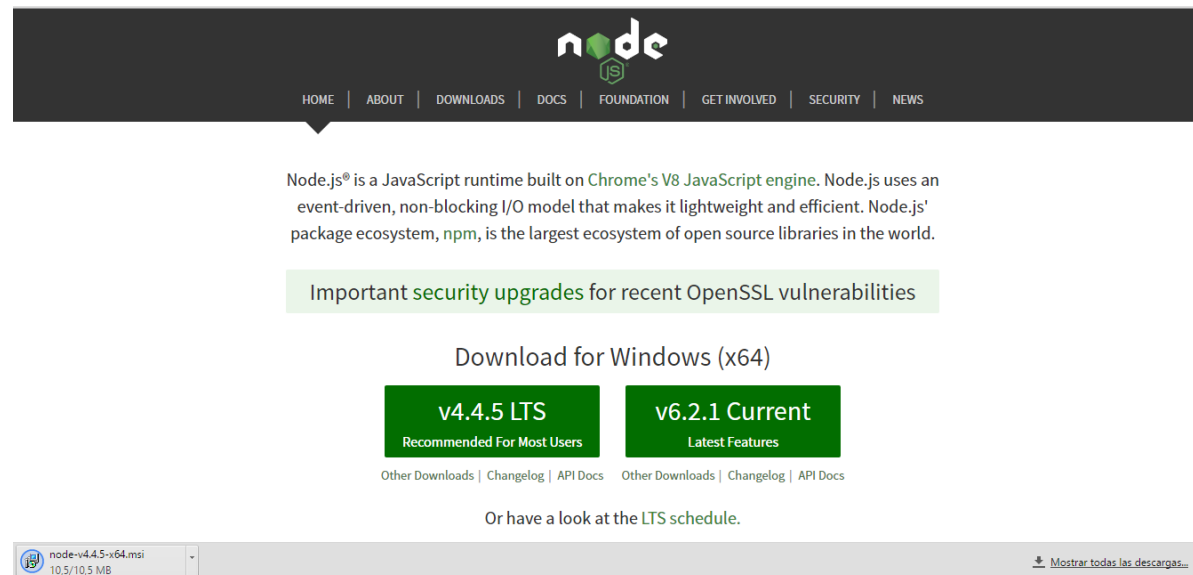


Que necesitamos para iniciar con Polymer.

Node.js es un entorno en tiempo de ejecución multiplataforma, open source, basado en el lenguaje de programación **ECMAScript** y en el motor V8 de Chrome. **Node.js** cuenta con un ecosistema de paquetes NPM que es el mayor ecosistema de librerías de código abierto en el mundo.

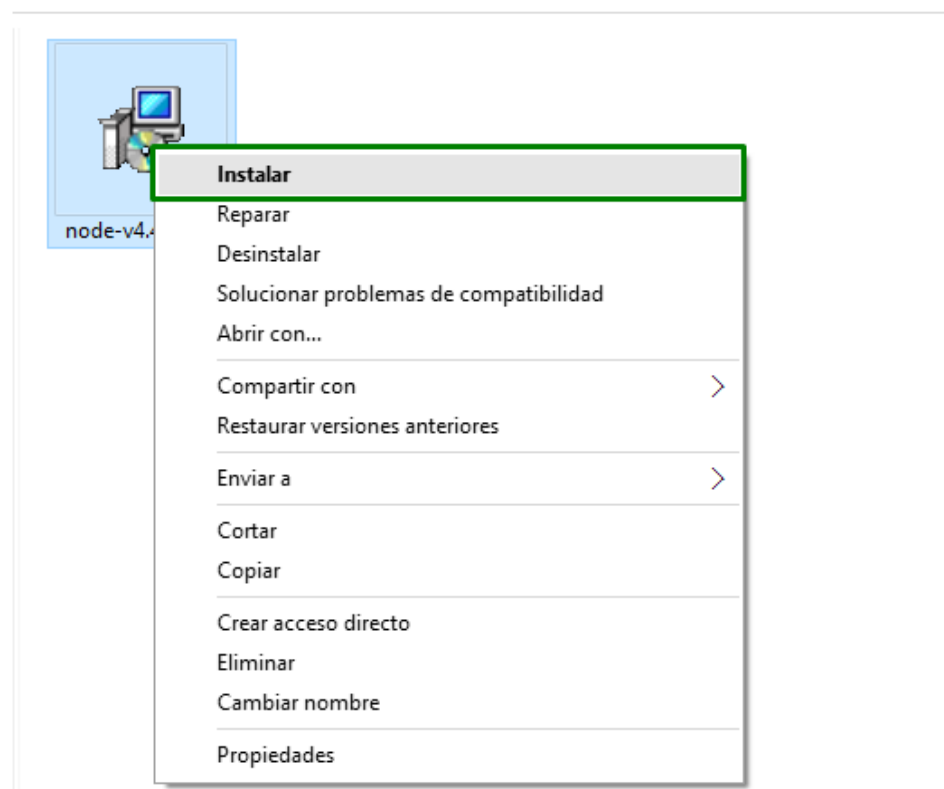
Fue creado por **Ryan Dahl** en 2009.

Para instalarlo en Windows nos dirigimos a la página oficial de node [“https://nodejs.org/en/”](https://nodejs.org/en/), y descargamos la versión más reciente.



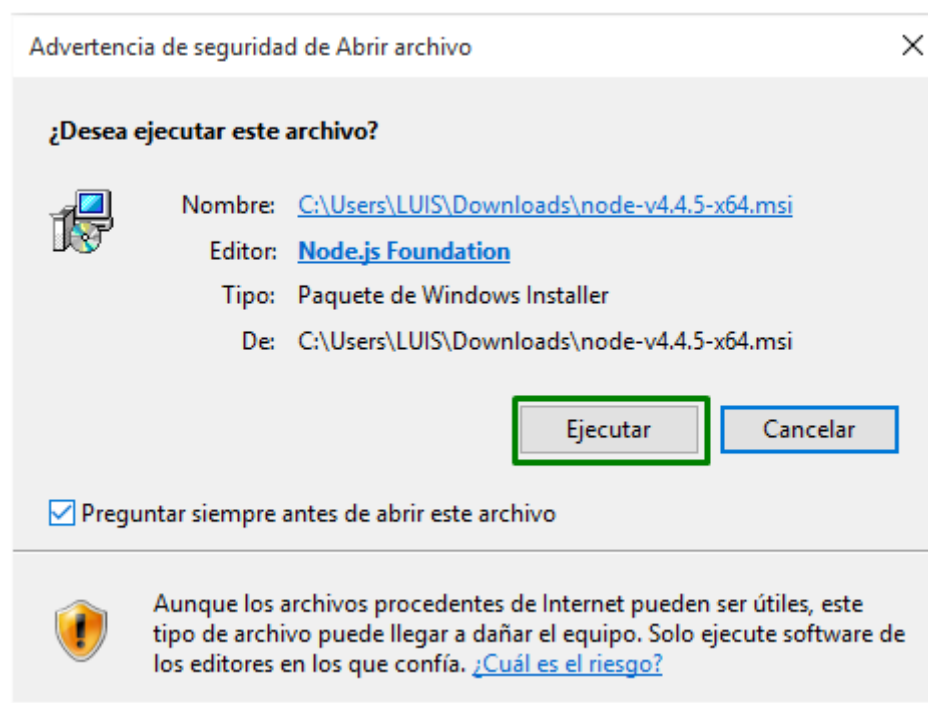
Que necesitamos para iniciar con Polymer.

Una vez descargado el archivo has clic derecho y escoge la opción **Instalar**



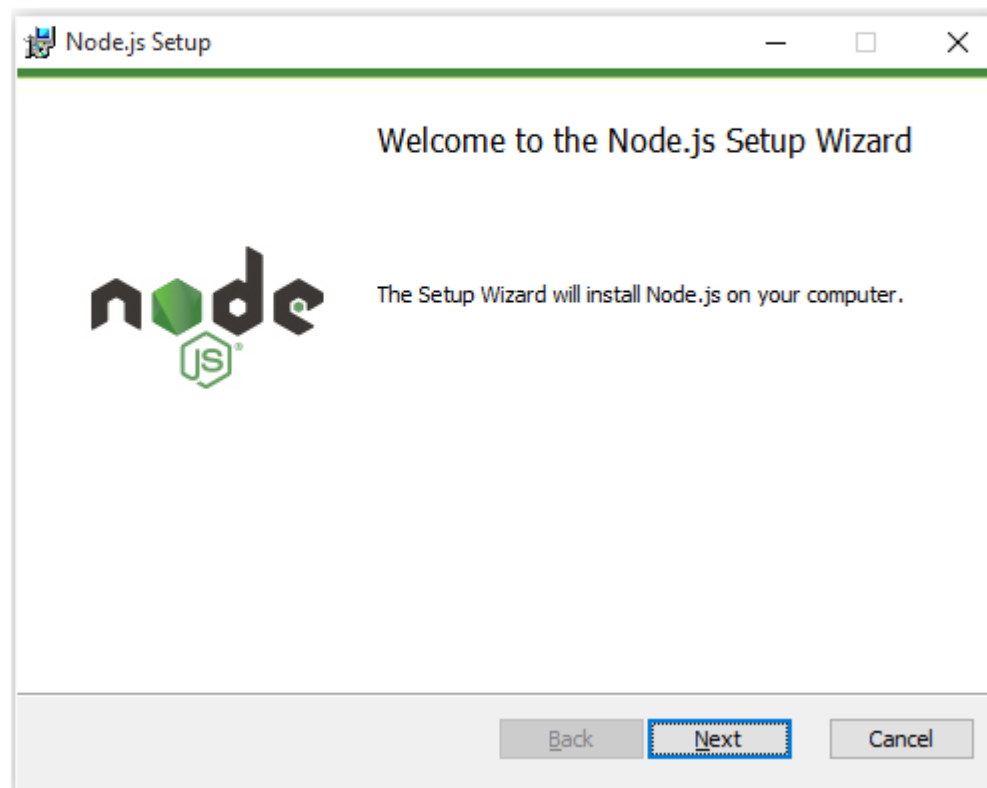
Que necesitamos para iniciar con Polymer.

En esta ventana has clic en el botón con la leyenda “Ejecutar”.



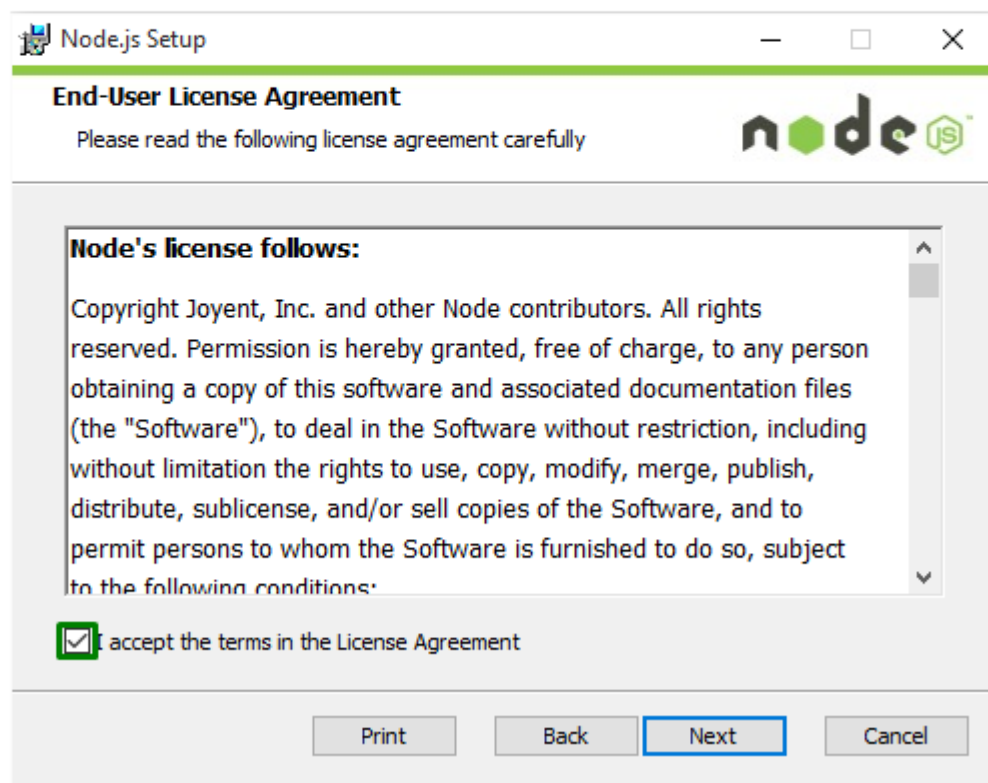
Que necesitamos para iniciar con Polymer.

Aquí presiona en el botón **Next**



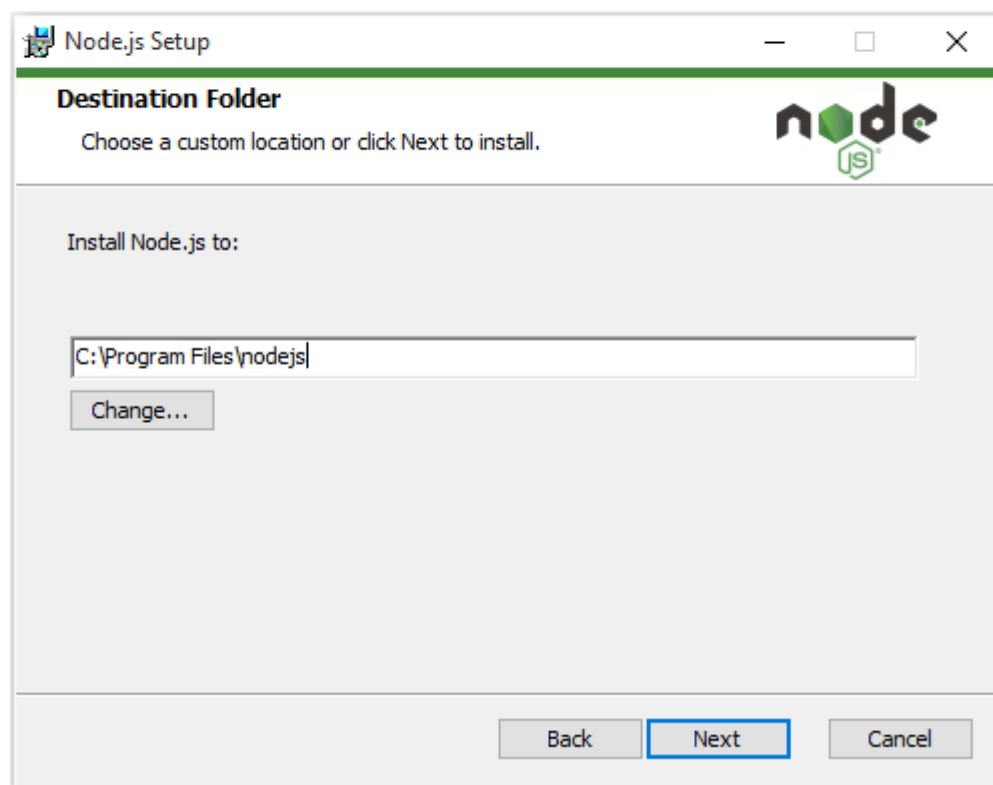
Que necesitamos para iniciar con Polymer.

Ahora selecciona la casilla donde dice (**I accept the terms in the License Agreement**) como en la imagen y luego clic en el botón **Next**



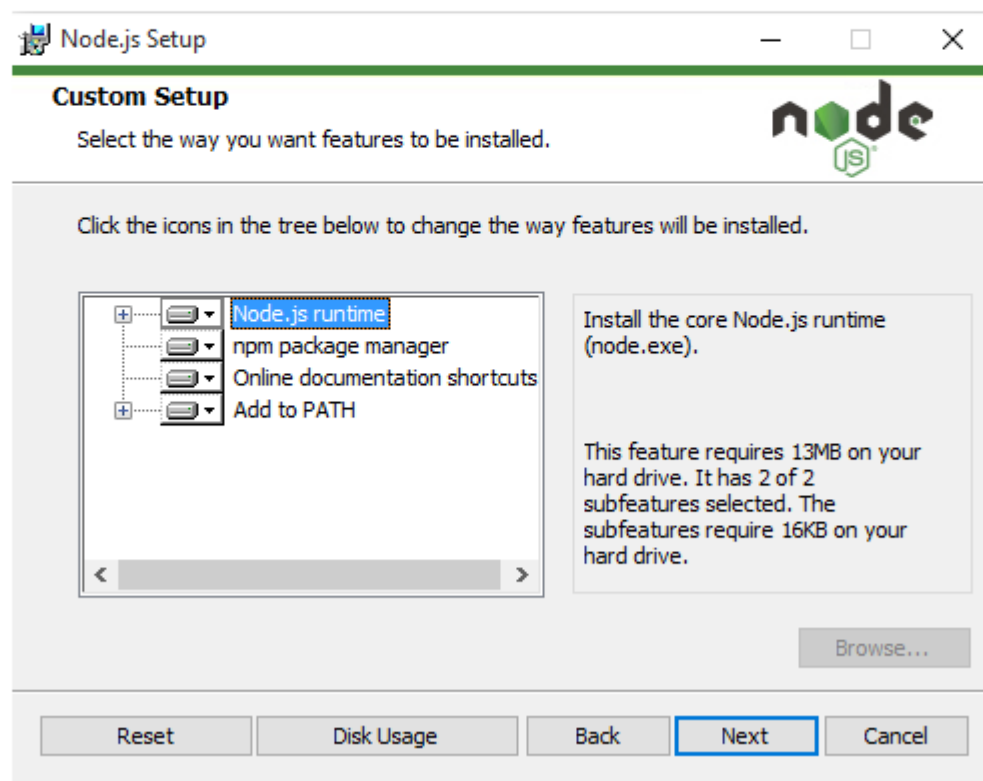
Que necesitamos para iniciar con Polymer.

Aquí puedes cambiar la ruta de instalación si lo deseas y si no lo dejas igual. Para continuar has clic en el botón **Next**



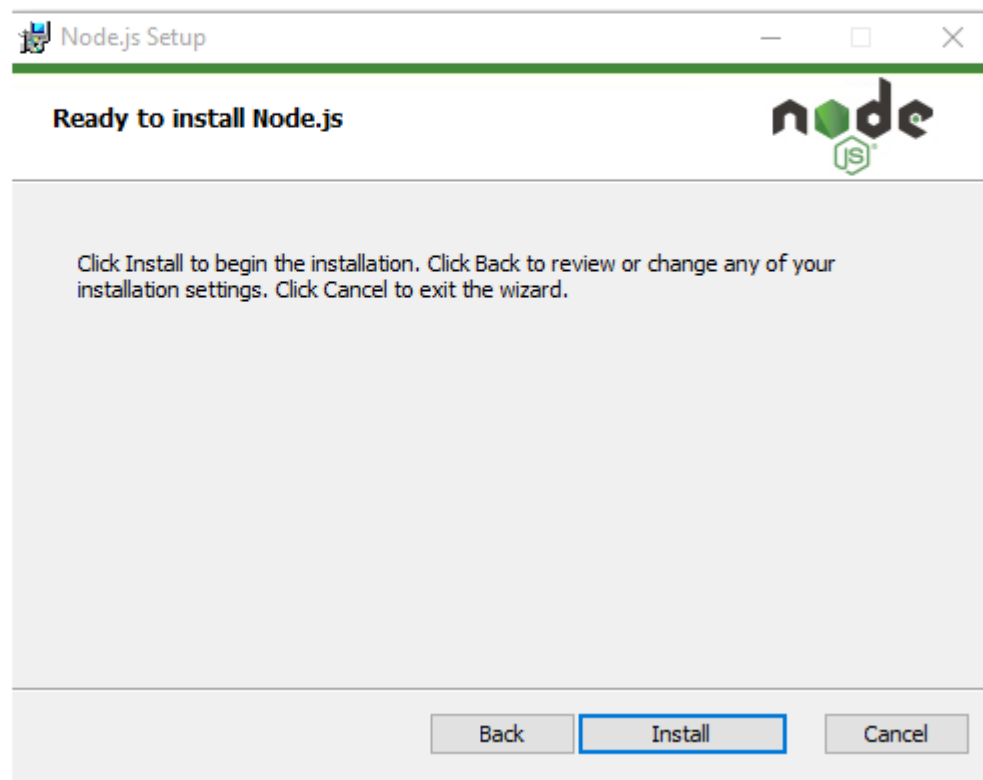
Que necesitamos para iniciar con Polymer.

Aquí tienes la opción de personalizar la instalación, una vez culminado has clic en el botón **Next**



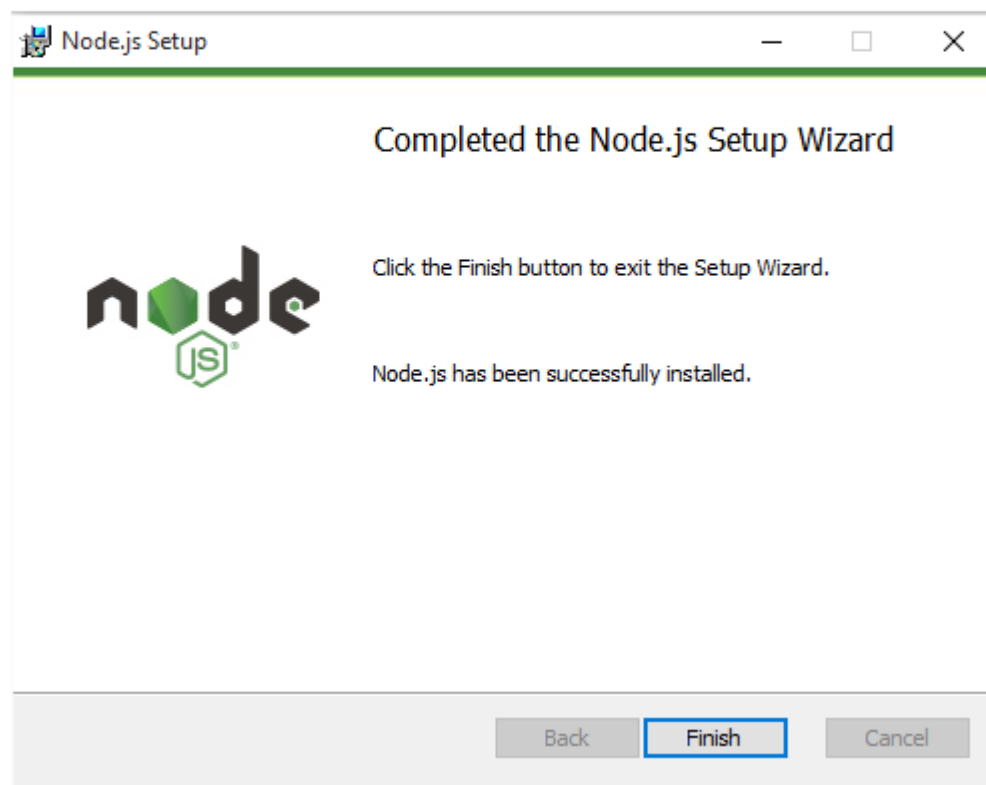
Que necesitamos para iniciar con Polymer.

Ahora hacemos clic en el botón **Install**



Que necesitamos para iniciar con Polymer.

Esta ventana muestra un mensaje **Node.js ha sido instalado exitosamente**





Que necesitamos para iniciar con Polymer. **Softtek®**

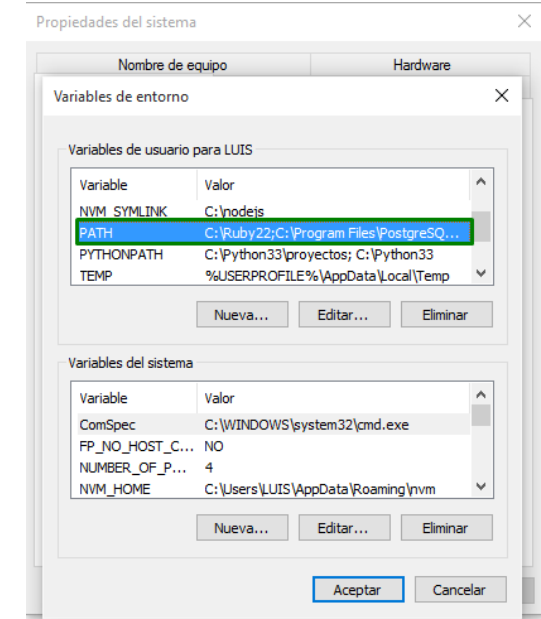
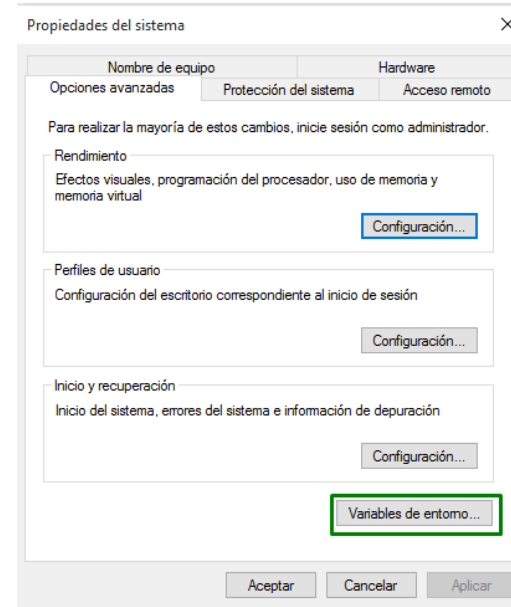
Para culminar vamos a verificar si **Node.js** esta instalado correctamente.

Abrimos el terminal **CMD** de windows y escribimos `node -v` el cual mostrara la versión del **Node.js** instalado.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.10240]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\LUIS>node -v
v4.4.5

C:\Users\LUIS>
```



Si no muestra la versión, ve a variables de entorno de S.O Windows y colócate en la parte final y agrega un punto y coma (;).

Continuando pega la ruta donde se instalo **Node.js** C:\Program Files\nodejs

Que necesitamos para iniciar con Polymer (BOWER).



Por qué Bower

Los desarrolladores frontend hemos sobrevivido toda una vida sin Bower. Con esto quiero decir que no es necesario usar Bower para realizar un proyecto o para instalar paquetes del lado del cliente. Solo es una utilidad que nos facilita el día a día, pero acostumbrarse a usarla puede ayudar bastante en determinados procesos.

Sin Bower, cuando queremos instalar una librería Javascript como jQuery o un framework de diseño como Bootstrap, simplemente íbamos al sitio de esos paquetes que requeríamos para un trabajo, los descargábamos y los colocábamos a mano en una carpeta de nuestro proyecto, o bien configurábamos una ruta absoluta al CDN en las etiquetas SCRIPT o LINK, necesarias para incluir esos paquetes. Con Bower ese trabajo, que no es pesado pero sí requiere de varios pasos (ir a un sitio web, descargar una librería, colocarla en una carpeta), se puede hacer con un sencillo comando de consola:

```
bower install jquery
```

Quizás todavía no le veas mucha utilidad todavía pero piensa en el momento en el que tienes que mantener toda una serie de librerías externas y actualizarlas con sus nuevas versiones. Requiere volver a descargar la librería a actualizar, borrar la antigua, subir la nueva, etc. Puede que hacer esto para una sola librería no represente mucho trabajo, pero a veces tenemos docenas de paquetes de terceros en un proyecto y entonces mantenerlos actualizados sí te puede llevar un rato. ¿No sería mejor hacerlo con un simple comando de consola? Obviamente, sí.

Que necesitamos para iniciar con Polymer (BOWER).



Una vez instalado NodeJS en nuestro equipo, al instalar NodeJS se instala también un gestor de paquetes Javascript llamado “npm”. Este gestor sirve para descargar e instalar en nuestro sistema cualquier programa basado en NodeJS y se opera por línea de comandos.

La manera de instalar Bower en nuestro equipo es ejecutando el correspondiente comando de instalación, vía npm, como se muestra en la siguiente imagen en pocas palabras abre el git bash en tu equipo entra a la carpeta donde vas a realizar tus practicas y ejecuta las líneas de la imagen.

```
npm install -g bower
```

Que necesitamos para iniciar con Polymer (BOWER).



Archivo bower.json

Lo ideal es que te crees un archivo bower.json en un carpeta raíz del proyecto. Este archivo te sirve para especificar de una manera formal todas las dependencias que tiene tu proyecto. De esa manera serás capaz de pedirle a Bower que te las instale todas de una vez, que las actualice todas de una vez, si es que se encuentran versiones nuevas que te interese instalar y también, cuando subas a producción un proyecto, que te permita aprovisionarlo con todas las librerías externas necesarias. Este archivo es muy fácil de construir, con sintaxis JSON, indicando una serie de campos que necesita para definir tu proyecto y sus dependencias.

Para crear por primera vez tu archivo bower.json lo más cómodo es lanzar el comando “bower init” desde la raíz de tu proyecto.

Esta tema en el curso es solo de conocimiento adicional, no se pregunta en examen de certificación ni En las practicas en clase, pero ya en ambientes laborales te puede servir.

```
bower init
```

Que necesitamos para iniciar con Polymer (BOWER).



La instrucción de la lamina anterior, pondra en marcha un script en el terminal, que recabará todas las informaciones necesarias para identificar tu proyecto. Aunque realmente, si no sabes algún dato de los que te pide, no necesitas preocuparte porque simplemente te ofrecerá un valor predeterminado que podrás aceptar pulsando “enter”. Es por ello, que si tienes prisa puedes hacer “enter, enter, enter...” dejando todas las opciones de manera predeterminada.

Cuando el proceso acabe verás el archivo bower.json en la raíz de tu proyecto, en la misma carpeta donde hiciste el “bower init”. Tendrá una forma parecida a esta:

```
{
  name: 'Prueba Bower',
  version: '0.0.0',
  authors: [
    'Carlos Montero orozco'
  ],
  description: 'Esto es una simple prueba',
  main: '',
  moduleType: [],
  license: 'MIT',
  homepage: 'http://www.softtek.com',
  ignore: [
    '**/*.*',
    'node_modules',
    'bower_components',
    'test',
    'tests'
  ]
}
```

Que necesitamos para iniciar con Polymer (BOWER).



Ahora vamos a editarlo para indicar nuestras dependencias, con tu editor de código de preferencia. Realmente es tan sencillo como colocar un nuevo campo "dependencies" cuyo valor será un objeto que define los nombres de los paquetes que tenemos como dependencias junto con sus versiones.

```
{
  "name": "Prueba Bower",
  "version": "0.0.0",
  "authors": [
    "Miguel Angel Alvarez "
  ],
  "dependencies": {
    "jquery": "~2.1.4",
    "bootstrap": "~3.3.5",
    "angular": "1.4.7",
    "angular-route": "1.4.7",
  }
}
```

- › Como podrás apreciar, algunas veces indicamos una versión exacta de una dependencia, pero a veces se le agrega el carácter "~" delante. Esto quiere decir que aceptas que se actualice la versión. Para que te sirva de referencia:
- › Si indicas "~1.2" estás diciendo que puede actualizar la versión de 1.2 a 1.3 cuando aparezca, 1.4, etc. Pero nunca subiría a la 2.0. tienen otros gestores de paquetes.
- › Una vez tenemos nuestro archivo bower.json podemos instalar las dependencias que hemos declarado gracias al comando "bower install". En seguida lo comentamos con mayor detalle.

Nota: No coloqué todos los campos del bower.json para no reproducir de nuevo el mismo texto de antes. Lo importante aquí es ver el mencionado campo "dependencies".

Que necesitamos para iniciar con Polymer (BOWER).



Comandos típicos de Bower

bower init

Es un atajo para generar un archivo bower.json, en el que definir las propiedades de un proyecto.

bower install

Si hacemos "bower install", sin ningún parámetro adicional, bower leerá lo que hayamos configurado en el archivo bower.json, instalando todas las dependencias que hayamos definido. Todos los componentes se colocarán en una carpeta específica llamada generalmente "bower_components".

bower install NOMBRE_PAQUETE

Este comando sirve para instalar un paquete, sin necesidad de nombrarlo entre las dependencias definidas en el archivo bower.json.

Que necesitamos para iniciar con Polymer (BOWER).



bower update

Este comando ejecuta la actualización de los paquetes, conforme a lo indicado en el archivo bower.json, ya que somos nosotros como desarrolladores los que debemos informar el rango de versiones que permitimos se actualicen.

bower uninstall NOMBRE_PAQUETE

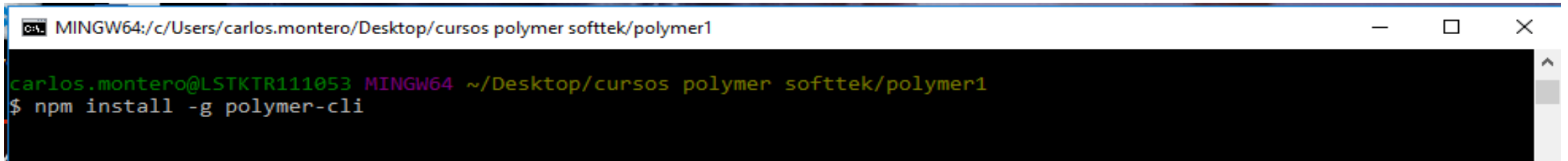
Sirve para desinstalar un paquete completamente del directorio de componentes de Bower. Este paquete debemos nombrarlo de manera obligatoria y si deseamos que se elimine también su referencia en la sección "dependencies" de bower.json deberemos indicarlo con la opción "--save".

Existen varios otros comandos que podrás leer en la documentación de Bower.

<https://bower.io/docs/api/>

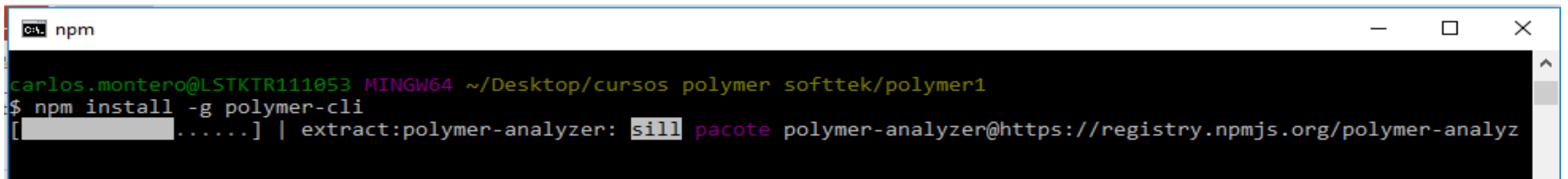
INSTALANDO POLYMER

Una vez instalado NODEjs y BOWER, accedemos al gitbash y nos posicionamos en la carpeta donde vamos a trabajar y es opcional aplicar el bower.json con el comando “bower init”.
Para correr cualquier comando es necesario tener conexión a internet y permiso de acceso a los sitios. De preferencia realiza estas acciones en una ubicación que no sea Softtek o BBVA. Ingresa las siguientes Lineas como se muestra en la imagen.



```
MINGW64:/c:/Users/carlos.montero/Desktop/cursos polymer softtek/polymer1  
carlos.montero@LSTKTR111053 MINGW64 ~/Desktop/cursos polymer softtek/polymer1  
$ npm install -g polymer-cli
```

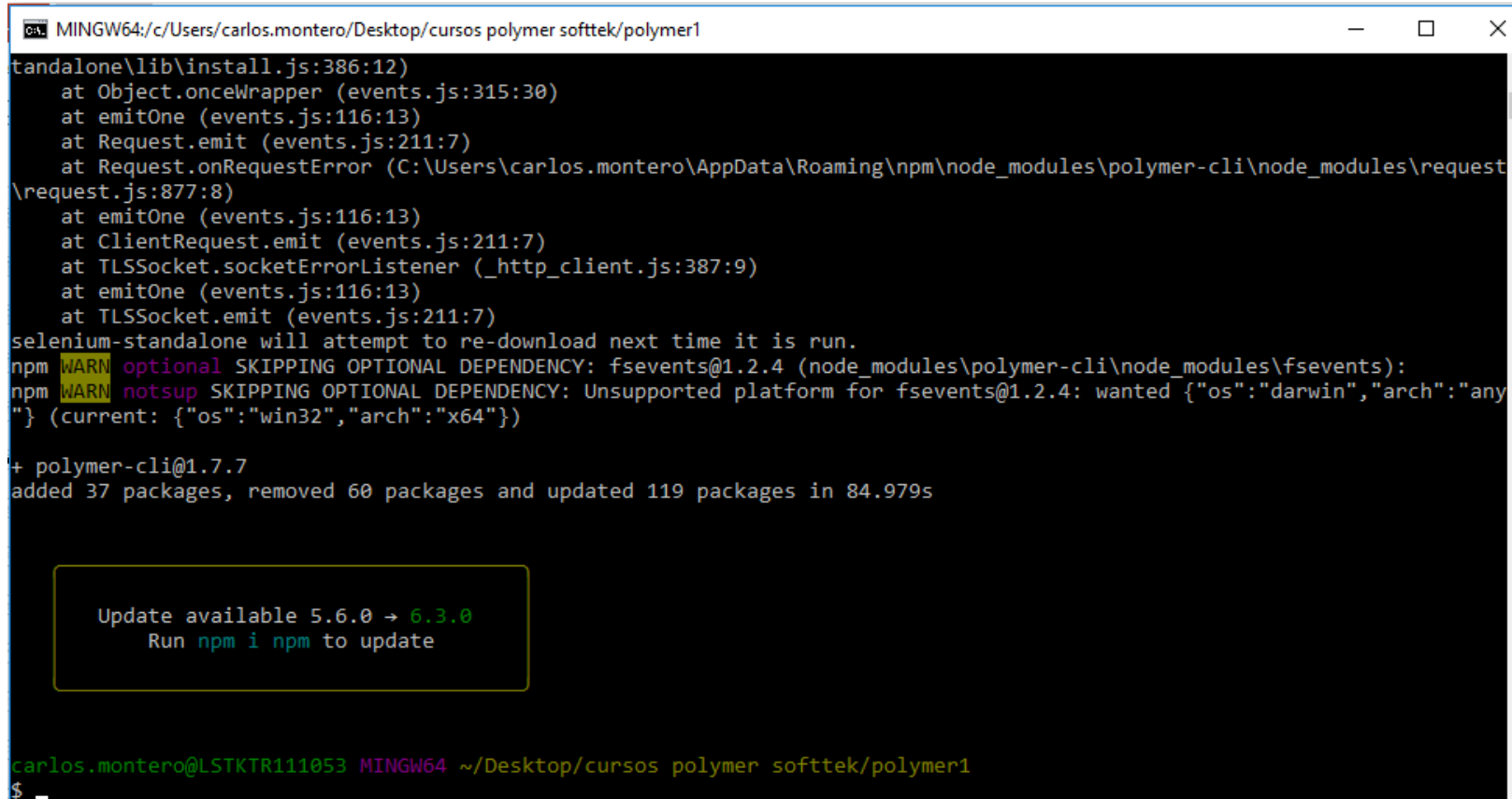
Se visualizara el avance durante el proceso de instalación.



```
npm  
carlos.montero@LSTKTR111053 MINGW64 ~/Desktop/cursos polymer softtek/polymer1  
$ npm install -g polymer-cli  
[.....] | extract:polymer-analyzer: sill pacote polymer-analyzer@https://registry.npmjs.org/polymer-analyz
```

INSTALANDO POLYMER

Al terminar la ejecución del comando “npm install –g polymer-cli”, muestra algo similar a la siguiente imagen.



```
MINGW64:/c:/Users/carlos.montero/Desktop/cursos polymer softtek/polymer1
tandalone\lib\install.js:386:12)
    at Object.onceWrapper (events.js:315:30)
    at emitOne (events.js:116:13)
    at Request.emit (events.js:211:7)
    at Request.onRequestError (C:\Users\carlos.montero\AppData\Roaming\npm\node_modules\polymer-cli\node_modules\request
\request.js:877:8)
    at emitOne (events.js:116:13)
    at ClientRequest.emit (events.js:211:7)
    at TLSSocket.socketErrorListener (_http_client.js:387:9)
    at emitOne (events.js:116:13)
    at TLSSocket.emit (events.js:211:7)
selenium-standalone will attempt to re-download next time it is run.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\polymer-cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ polymer-cli@1.7.7
added 37 packages, removed 60 packages and updated 119 packages in 84.979s

Update available 5.6.0 → 6.3.0
Run npm i npm to update

carlos.montero@LSTKTR111053 MINGW64 ~/Desktop/cursos polymer softtek/polymer1
$
```



INSTALANDO POLYMER

Ahora si vamos a iniciar desde cero creando nuestros propios componentes de polymer ejecutamos La siguiente instrucción “bower install --save Polymer/polymer”, como se muestra en la imagen.

```
carlos.montero@LSTKTR111053 MINGW64 ~/Desktop/cursos polymer softtek/polymer1
$ bower install --save Polymer/polymer
```

Durante la instalación muestra algo similar a la siguiente imagen

```
carlos.montero@LSTKTR111053 MINGW64 ~/Desktop/cursos polymer softtek/polymer1
$ bower install --save Polymer/polymer
bower polymer#*          cached https://github.com/Polymer/polymer.git#3.0.2
bower polymer#*          validate 3.0.2 against https://github.com/Polymer/polymer.git#*
bower polymer#*          new version for https://github.com/Polymer/polymer.git#*
bower polymer#*          resolve https://github.com/Polymer/polymer.git#*
bower polymer#*          download https://github.com/Polymer/polymer/archive/v3.0.5.tar.gz
bower polymer#*          extract archive.tar.gz
bower polymer#*          resolved https://github.com/Polymer/polymer.git#3.0.5
bower polymer#^3.0.5      install polymer#3.0.5
bower                    no-json No bower.json file to save to, use bower init to create one

polymer#3.0.5 bower_components\polymer
```



INSTALANDO POLYMER

Con el paso anterior podemos ya crear nuestro primer componente polymer, en laminas posteriores Se explicara la estructura de un componente polymer.

Ahora si necesitamos un ejemplo de los que están por default en polymer, ejecutamos la siguiente Instrucción “polymer init” como se muestra en la siguiente imagen.

```
MINGW64:/c:/Users/carlos.montero/Desktop/cursos polymer softtek/polymer1
carlos.montero@LSTKTR111053 MINGW64 ~/Desktop/cursos polymer softtek/polymer1
$ polymer init
```

Durante la instalación muestra algo similar a la siguiente imagen y debemos seleccionar una opción, en la siguiente lamina se indica a que se refiere cada opción.

```
polymer
carlos.montero@LSTKTR111053 MINGW64 ~/Desktop/cursos polymer softtek/polymer1
$ polymer init
? Which starter template would you like to use?
1) polymer-3-element - A simple Polymer 3.0 element template
2) polymer-3-application - A simple Polymer 3.0 application
3) polymer-3-starter-kit - A Polymer 3.x starter application template, with navigation and "PRPL pattern" loading
4) polymer-2-element - A simple Polymer 2.0 element template
5) polymer-2-application - A simple Polymer 2.0 application
6) polymer-2-starter-kit - A Polymer 2.x starter application template, with navigation and "PRPL pattern" loading
7) shop - The "Shop" Progressive Web App demo
Answer:
```

INSTALANDO POLYMER

A que se refieren cada una de las opciones.

- 1.- Elemento de polymer versión 3, solo trae la estructura de un elemento.
- 2.- Aplicación de polymer versión 3, trae rutas y una app shell cascaron de una aplicación.
- 3.- Polymer startkit una app web para modificar tiene 3 paginas, para versión 3 de polymer.
- 4.- Elemento de polymer versión 2, solo trae la estructura de un elemento.
(esta opción se va ocupar en las practicas en el modulo de polymer 2)
- 5.- Aplicación de polymer versión 2, trae rutas y una app Shell cascaron de una aplicación.
- 6.- Polymer startkit una app web para modificar tiene 3 paginas, para versión 2 de polymer.
- 7.- una aplicación **E-commerce** completa para poder visualizar la forma que esta desarrollada.



INSTALANDO POLYMER

Al seleccionar una opción por ejemplo la 4, nos va a solicitar el nombre del componente “Element name”, para crear el nombre del componente por regla general **debe ser con letras minúsculas y debe contener un guion medio** por ejemplo “practicapolymer-uno”, como se muestra en la imagen.

```
CA. polymer
carlos.montero@LSTKTR111053 MINGW64 ~/Desktop/cursos polymer softtek/polymer1
$ polymer init
? Which starter template would you like to use? polymer-init-polymer-2-element:app
info: [init] Running template polymer-2-element...
? Element name (polymer1-element) practicapolymer-uno_
```

Posteriormente solicita una descripción del componente la cual es opcional, con esto se termina la creación De un componente de los precargados por polymer.

```
Setup Complete!
Check out your new project README for information about what to do next.

carlos.montero@LSTKTR111053 MINGW64 ~/Desktop/cursos polymer softtek/polymer1
$
```

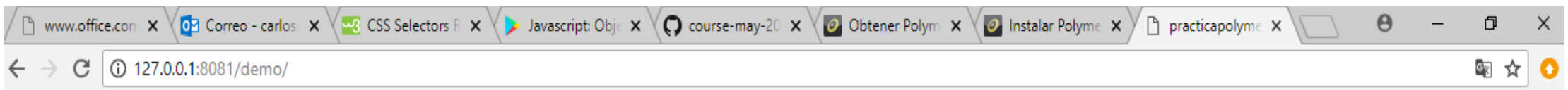


INSTALANDO POLYMER

Para las dos opciones que se vieron en las laminas anteriores es necesario poder ver su comportamiento. En el navegador, para esto ejecutamos el comando “polymer serve –open”, el cual abre el navegador Predeterminado en nuestro equipo y muestra el componente creado.

Si no se desea que se muestre en el navegador predeterminado se omite la propiedad “-open”, en el comando. Y con el resultado que genera se copia la url y se pega en el navegador de nuestra elección.

```
$ polymer serve
info: [cli.command.serve]   Files in this directory are available under the following URLs
applications: http://127.0.0.1:8081
reusable components: http://127.0.0.1:8081/components/practicapolymer-uno/
```



Basic practicapolymer-uno demo

Para cerrar el servidor web que se genero en el git bash, teclear “ctrl + c”, y con esto termina el Servicio.

Estructura de componente Polymer

Para entender rápidamente las ventajas de polymer y como usarlo en nuestros desarrollos, supongamos que en una pagina web vas a mostrar una marquesina desplazándose, actualmente buscas el código para la solicitud y lo encuentras en CSS, este código debe tener aproximadamente 40 líneas de código, cada vez que quieras ocupar este efecto debes copiar y adaptar en código html.

Con polymer solo mandas a llamar la etiqueta y atreves de un formato json envías el texto a mostrar a si como la propiedad de tipo de letra, tamaño, color de fondo etcétera .

Con css:

```
/* marquesina que se despalaza */
.bounce {
  height: 50px;
  overflow: hidden;
  position: relative;
  background: white; /*Color del fondo*/
  color: red; /* Color de la letra*/
  /*border: 1px solid orange;*/ /*marco con borde de color opcional*/
}
.bounce p { /*para el texto */
  position: absolute;
  width: 100%;
  height: 100%;
  margin: 0;
  line-height: 50px;
  text-align: center;
  /* posición inicial */
  -moz-transform: translateX(50%);
  -webkit-transform: translateX(50%);
  transform: translateX(50%);
  /* aplica animacion al elemento */
  -moz-animation: bouncing-text 5s linear infinite alternate;
  -webkit-animation: bouncing-text 5s linear infinite alternate;
  animation: bouncing-text 5s linear infinite alternate;
}
/* define movimiento */
@-moz-keyframes bouncing-text {
  0% { -moz-transform: translateX(50%); }
  100% { -moz-transform: translateX(-50%); }
}
@-webkit-keyframes bouncing-text {
  0% { -webkit-transform: translateX(50%); }
  100% { -webkit-transform: translateX(-50%); }
}
@keyframes bouncing-text {
  0% {
    -moz-transform: translateX(50%);
    -webkit-transform: translateX(50%);
    transform: translateX(50%);
  }
}
//Texto en archivo index.html
<div class="bounce">
  <p>!!! texto marquesina !!!</p>
</div>
```



HTML File



css_marquezina.css

```
<!-- con POLYMER : -->
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>MARQUESINA</title>
<script src="../../webcomponentsjs/webcomponents-loader.js"></script>
<link rel="import" href="../../iron-demo-helpers/demo-pages-shared-styles.html">
<link rel="import" href="../../iron-demo-helpers/demo-snippet.html">
<link rel="import" href="../../mar-quesina.html">
<custom-style>
<style is="custom-style" include="demo-pages-shared-styles"> </style> </custom-style>
</head>
<body>
<template>
<mar-quesina mar-data='{ "info_1": "Texto de marquesina", "Tamaño_letra": "20" }'>
</mar-quesina>
</template>
</body>
</html>
```




Estructura de componente Polymer

En la lamina anterior, notarás que en el ejemplo de polymer hay una etiqueta (tag), de nombre: **<mar-quesina></mar-quesina>**

La etiqueta también le asigna a mar-data, datos en formato json esto porque a si esta diseñado, pueden existir desarrollos que solo se llame con la etiqueta.

Esta etiqueta esta referenciada en el archivo html donde se mande a llamar de la siguiente manera la

puedes ubicar en la lamina anterior:

<link rel="import" href="../mar-quesina.html">

Como notarás mandas a llamar un archivo html dentro de este archivo se encuentra tu desarrollo polymer

y tiene la siguiente estructura:

```
1 <link rel="import" href="../polymer/polymer-element.html">
2 <dom-module id="mar-quesina">
3   <template>
4     <style>
5
6   </style>
7
8   <template is="dom-repeat" items="[[marData]]" as="item">
9     <section class="stats">
10      <div class="content"> <picture class="cove2"></picture></div>
11    </section>
12  </template>
13 </template>
14
15 <script>
16   Polymer({
17     is: 'mar-quesina',
18     properties: {
19       tipoBooleano: {
20         type: Boolean,
21         value: true
22       },
23     }
24   });
25
26 </script>
27 </dom-module>
```

REFERENCIAS.

NOMBRE ETIQUETA POLYMER

ESTILOS.

HTML

CIERRA EL TEMPLATE QUE CONTIENE LOS ESTILOS Y EL HTML, DE LA ETIQUETA POLYMER

JAVA SCRIPT

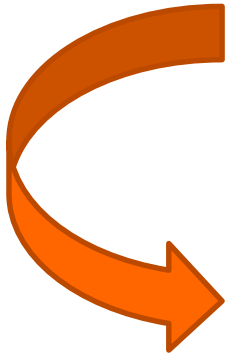
MI PRIMER ELEMENTO POLYMER

1. Accedamos al git bash y creamos dentro de nuestra carpeta de trabajo la carpeta polymer1
2. Creamos la subcarpeta practica 1
3. Ingresamos la sentencia “polymer init”
4. Seleccionamos la opción “4”.
5. Damos un nombre a nuestra etiqueta “clase-uno”.
6. Damos una descripción a nuestra etiqueta es opcional puede ir sin valor “practica en clase”.
7. Abrimos otra ventana con el git bash y nos ubicamos en la carpeta que creamos “practica 1”.
8. Ingresamos sentencia “polymer serve”.
9. Copiamos la URL de la segunda opción <http://127.0.0.1:8081>, y la pegamos en nuestro explorador.
10. Abrimos nuestro folder “practica 1”, con Atom.
11. Validamos que las referencias están correctas ingresando una etiqueta <H1> en el archivo “clase-uno.html”.
12. Actualizamos nuestro navegador y si vemos el texto ingresado en la etiqueta <H1>, las referencias están correctas.



MI PRIMER ELEMENTO POLYMER

Si no muestra el texto de la etiqueta entonces validamos nuestras referencias en el archivo index.html y clase-uno.html, el archivo index a verificar se encuentra dentro de la carpeta “demo”. Las referencias deben a apuntar a la carpeta “bower_components”, si la carpeta no se creo, se debe a cuestiones de seguridad o permisos del equipo; (solicitar al instructor la carpeta). En la siguiente imagen se muestra como se genero la referencia en el archivo index y como se modifico.



```
<script src="../../webcomponentsjs/webcomponents-loader.js"></script>

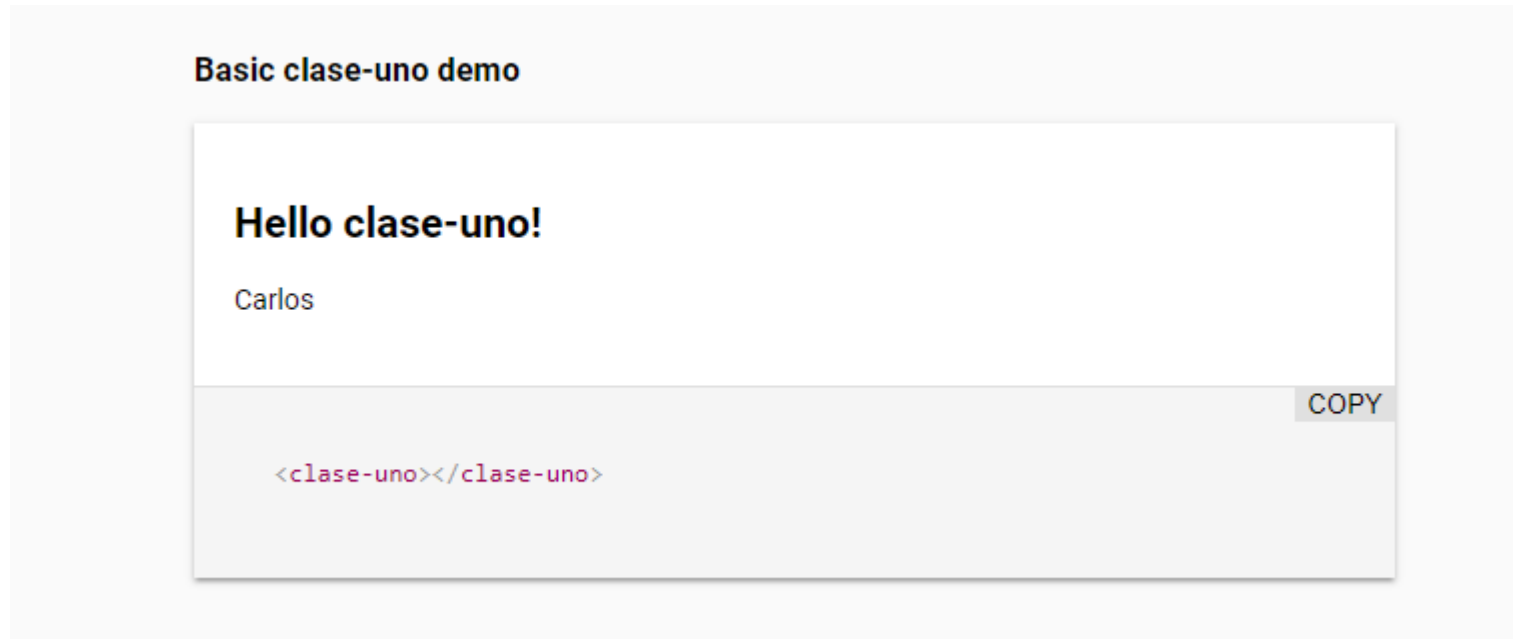
<link rel="import" href="../../iron-demo-helpers/demo-pages-shared-styles.html">
<link rel="import" href="../../iron-demo-helpers/demo-snippet.html">
<link rel="import" href="../../clase-uno.html">
```

```
<script src="../bower_components/webcomponentsjs/webcomponents-loader.js"></script>

<link rel="import" href="../bower_components/iron-demo-helpers/demo-pages-shared-styles.html">
<link rel="import" href="../bower_components/iron-demo-helpers/demo-snippet.html">
<link rel="import" href="../clase-uno.html">
```

MI PRIMER ELEMENTO POLYMER

Realizamos lo mismo de las referencias con el archivo “clase-uno.html”.
Salvamos y actualizamos nuestro navegador y debemos ver algo a sí:





MI PRIMER ELEMENTO POLYMER

Ahora vamos a eliminar del archivo “clase-uno.html”, las siguientes líneas:

```
:host {  
  display: block;  
}
```

```
<h2>Hello [[prop1]]!</h2>
```

```
/**  
`clase-uno`  
* practica en clase  
*  
* @customElement  
* @polymer  
* @demo demo/index.html  
*/
```

```
class ClaseUno extends Polymer.Element {  
  static get is() { return 'clase-uno'; }  
  static get properties() {  
    return {  
      prop1: {  
        type: String,  
        value: 'clase-uno'  
      }  
    };  
  }  
}  
window.customElements.define(ClaseUno.is, ClaseUno);
```



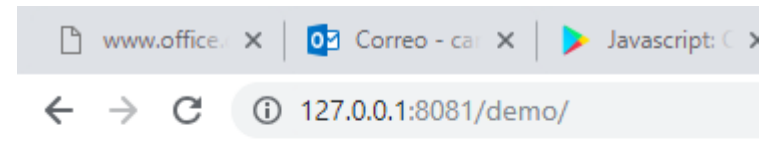
MI PRIMER ELEMENTO POLYMER

En el archivo “clase-uno.html”, agregamos las siguientes líneas de código, dentro de la etiqueta `<script></script>`.

```
Polymer({  
  is: 'clase-uno'  
});
```

En el archivo “Index.html”, se elimina la etiqueta `<custom-style></custom-style>`, y su contenido y dentro de la etiqueta `<body>`, solo se deja la etiqueta `<h3>` y el llamado a nuestro componente de polymer `<clase-uno>`.

```
<body>  
<h3>Basic clase-uno demo</h3>  
<clase-uno></clase-uno>  
</body>
```



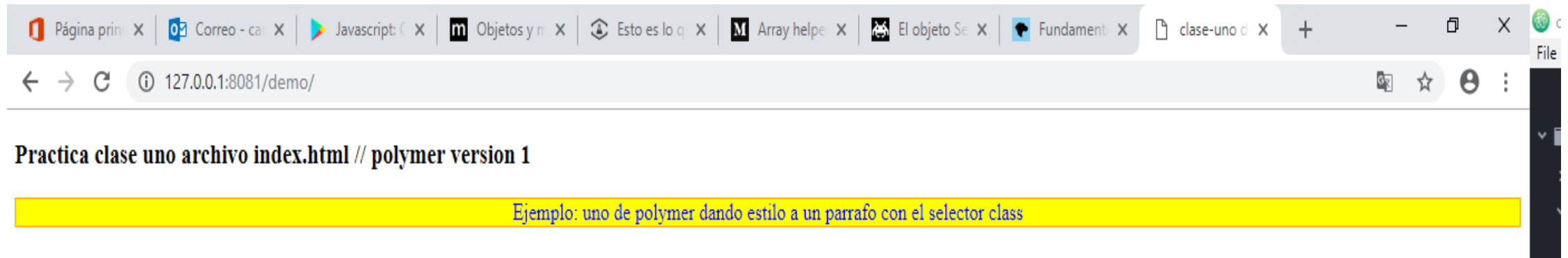
Basic clase-uno demo

Carlos etiqueta



MI PRIMER ELEMENTO POLYMER

Con esto ya tenemos nuestra base para iniciar a desarrollar en polymer, cambia el texto en la etiqueta `<h3>` del archivo “index.html”, por “Practica clase uno archivo index.html // polymer version 1”, Y dentro de nuestro archivo “clase-uno.html”, solo dejamos una etiqueta `<p>`, con el texto “Ejemplo: uno de polymer dando estilo a un parrafo con el selector class”, y a esta etiqueta `<p>`, se le asigna la clase “ejeuno”. Esta clase en el apartado `<style>` deberá centrar el texto en la pantalla, El texto será de color azul, tendrá un recuadro con borde color naranja a 1 pixel y un fondo amarillo. Se elimina todo código dentro de la etiqueta `<style>`, antes de anexar el código para nuestra clase.



PROPIEDADES Y TIPOS DE DATOS.

Para ver las propiedades y tipos de datos vamos a crear el archivo “ejemplo-dos.html”, y en el archivo “clase-uno.html”, agregamos una etiqueta <p>, con el texto “ejemplo 2 propiedades y tipos de datos”, como se muestra a continuación:

```
</style>

<p class= "ejeuno">Ejemplo: uno de polymer dando estilo a un parrafo con el selector class</p>
<br>
<p class= "ejeuno">Ejemplo 2 propiedades y tipos de datos</p>
<ejemplo-dos></ejemplo-dos>
```

En este mismo archivo “clase-uno.html”, vamos a generar la referencia con el archivo “ejemplo-dos.html”, como se muestra a continuación:

```
1  <link rel="import" href="../bower_components/polymer/polymer-element.html">
2  <link rel="import" href="../ejemplo-dos.html">
3
4  <dom-module id="clase-uno">|
5    <template>
6    <style>
```




PROPIEDADES Y TIPOS DE DATOS.

En el archivo “ejemplo-dos.html”, vamos a tener nuestra referencia a “polymer-element.html” y nuestro <dom-module> con el Id “ejemplo-dos”, la etiqueta <style>, vacía al igual que la sección de código html, por ultimo la etiqueta <script>, con los valores por default que genero nuestro archivo “clase-uno.html”.

```
<link rel="import" href="../../bower_components/polymer/polymer-element.html">

<dom-module id="ejemplo-dos">
  <template>
    <style>
    </style>
  </template>

  <script>
    Polymer({
      is: 'ejemplo-dos'
    });
  </script>
```

PROPIEDADES Y TIPOS DE DATOS.

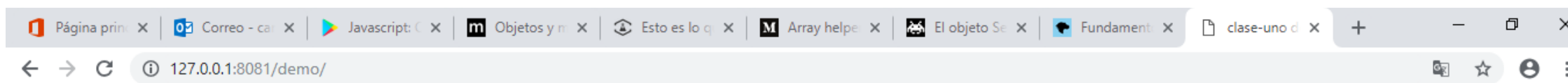


Dentro de la etiqueta `<script>`, vamos a ingresar una propiedad del tipo booleano con un valor `true`, y posterior a esto en donde va el código "html" con una etiqueta `<p>`, mostramos el valor de la propiedad, como se muestra en la siguiente imagen. Una vez realizado el valor booleano genera las propiedades para Entero y decimal con `Number`, `String`, `Array` y `Object`. Recuerda que al terminar cada propiedad con `}` si vas anexar otra debe incluir una coma `,`, si ya es la ultima propiedad va sin coma.

```
<dom-module id="ejemplo-dos">
  <template>
    <style>
    </style>
    <p> esta es una propiedad tipo boolean: "[[tipoBooleano]]"</p>
  </template>
<script>
Polymer({
  is: 'ejemplo-dos', /* <- No olvidemos la coma */
  /* estructura para generar una propiedad
  nombrePropiedadEnCamelCase:{
    type: Tipo de la propiedad (Obligatorio)
    value: Valor inicial de la propiedad (Opcional)
  } */
  properties: {
    tipoBooleano:{
      type: Boolean,
      value: true
    }
  }
});
</script>
</dom-module>
```

PROPIEDADES Y TIPOS DE DATOS.

Resultado Final en la pagina Web:



Practica clase uno archivo index.html // polymer version 1

Ejemplo: uno de polymer dando estilo a un parrafo con el selector class

Ejemplo 2 propiedades y tipos de datos

esta es una propiedad tipo boolean: "true"

esta es una propiedad tipo entero: 100

esta es una propiedad tipo decimal: 100.12345678901235

esta es una propiedad tipo array: "a,b,c"

ver array por indice: "b"

esta es una propiedad tipo object: "50"

Como ver una posición del array:

<p> ver array por índice: "[[tipoArray.1]]" </p>

Como ver un valor del objeto:

<p> esta es una propiedad tipo object: "[[tipoObject.age]]" </p>

Paso de Valor

Para el paso de valor, creamos la etiqueta “ejemplo-tres”, y generamos nuevamente en el archivo “clase-uno.html”, la referencia y el llamado de la siguiente forma:

```
<link rel="import" href="../ejemplo-tres.html">
```

```
<br>
<p class= "ejeuno">Ejemplo 3 paso de valor [[]]</p>
<ejemplo-tres recibe="valor 1"></ejemplo-tres>
<!-- para pasar mas de un valor ejemplo: <ejemplo-tres recibe_1="valor 1" recibe_2="valor 2"></ejemplo-tres>-->
```

Generamos el código JS en el archivo “ejemplo-tres.html”, de la siguiente forma:

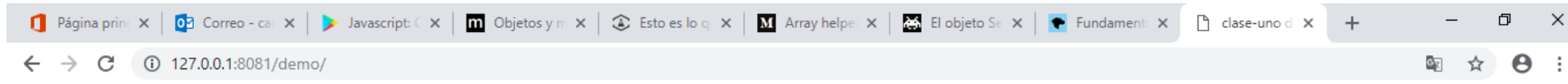
```
Polymer({
  is: 'ejemplo-tres',
  properties: {
    recibe:{
      type: String,
      value: 'valor 0'
    }
  }
});
```

Por ultimo en la sección del html en el archivo “ejemplo-tres.html”, generamos el código Que va a mostrar el valor recibido en pantalla:

<p> paso de valor de clase-uno.html a objeto polymer: [[recibe]]</p>

Paso de Valor

Resultado:



Practica clase uno archivo index.html // polymer version 1

Ejemplo: uno de polymer dando estilo a un parrafo con el selector class

Ejemplo 2 propiedades y tipos de datos

esta es una propiedad tipo boolean: "true"

esta es una propiedad tipo entero: 100

esta es una propiedad tipo decimal: 100.12345678901235

esta es una propiedad tipo array: "a,b,c"

ver array por indice: "b"

esta es una propiedad tipo object: "50"

Ejemplo 3 paso de valor [[]]

paso de valor de index.html a objeto polymer: valor 1

METODOS

Para los métodos creamos la etiqueta “ejemplo-cuatro”, y generamos nuevamente en el archivo “clase-uno.html”, la referencia y el llamado de la siguiente forma:

```
<link rel="import" href="../ejemplo-cuatro.html">
```

```
<br>  
<p class= "ejeuno">Ejemplo 4 metodos</p>  
<ejemplo-cuatro recibe="valor metodos"></ejemplo-cuatro>
```

METODOS

En el archivo “ejemplo-cuatro.html”, en la sección de HTML, con una etiqueta <p>, indicamos que Hay que presionar el botón y verificar en la consola el resultado.
en esta sección también se crea un botón que invoca al método “show_valor_metodo”.
En la sección del script se crea una propiedad del tipo string sin valor y el método que manda a la consola el valor recibido.

```
<p> para ver el valor que arroja el metodo hay que presionar el boton y luego ver el LOG </p>
<button on-tap="show_valor_metodo">Metodo</button>
</template>
<script>
Polymer({
  is: 'ejemplo-cuatro',
  properties: {
    recibe:{
      type: String,
    }
  },
  show_valor_metodo: function (){
    /*document.write(this.get('recibe'));*/
    console.log(this.get('recibe'));
  }
});
</script>
</dom-module>
```

CICLO DE VIDA DE UN COMPONENTE POLYMER



El ciclo de vida de los Custom Elements, de Javascript estándar, está compuesto por varias fases y los programadores disponemos de una serie de funciones "callback" que nos permiten realizar cosas cuando se hayan completado estas fases. La diferencia con Polymer es que los nombres de las funciones callback cambian, así como la manera de declararlas.

La declaración de funciones para ejecutar cosas en los diferentes estados de un componente se realiza mediante métodos que tienes que definir en el objeto que envías a la función Polymer() para registrar un componente.

```
Polymer({  
  is: 'componente-ejemplo',  
  nombreMetodo: function() {  
    // código del método  
  }  
});
```


CICLO DE VIDA DE UN COMPONENTE POLYMER



El detalle aquí es conocer las funciones que existen en Polymer para implementar acciones en el ciclo de vida:

- **created:** Se ha creado un ejemplar de un custom-element. Osea, el navegador ha creado un elemento que es un custom element de Web Components
- **attached:** El elemento se ha insertado dentro del documento, se ha insertado dentro del DOM
- **detached:** El elemento se ha retirado del DOM de la página
- **attributeChanged:** un atributo del componente se ha añadido, quitado o su valor ha cambiado

Nota: Estos métodos son los mismos que en los web components, solo que en el estándar tienen la palabra "Callback" atrás, como sufijo: createdCallback, attachedCallback, etc. Incluso dentro de un componente Polymer podrías usar la función nativa de Javascript si lo deseas o como alternativa fallback.

ready: este método es propio de Polymer y no existe en el estándar. Se ejecuta cuando se ha terminado de crear e inicializar todo el DOM local de un elemento, osea, todo el HTML que contiene en su TEMPLATE

CICLO DE VIDA DE UN COMPONENTE POLYMER

Para realizar el ejercicio de ciclo de vida genera en el archivo “index.html”, una etiqueta <a>, con un “href”, a la pagina “ciclovida.html”, y esta va a mandar a llamar al componente “ciclo-vida.html”.

Para una mejor presentación genera en el componente “clase-uno.html”, el estilo para el ejemplo cinco sin mandar a llamar ningún componente.



HTML File



HTML File

binding de una dirección y de dos direcciones

El binding es un enlace a un dato, de modo que todos los elementos que estén "bindeados" a ese dato puedan recibir cualquier cambio de manera automática.

Ya vimos que dentro de un componente puedo bindear datos, generalmente propiedades, hacia el template. De modo que si cambia esa propiedad también cambie la vista sin que tengamos que hacer nada. Ahora vamos a ver cómo entre componentes también puede producirse el binding, permitiendo que un dato que se usa en un componente también pueda viajar hacia otros componentes.

Envío de datos Literales:

```
<rompe-cadena str="Estas son las dos caras de la moneda" len="10"></rompe-cadena>
```

Envío de datos variable:

```
<rompe-cadena str="{{cadena}}" len="10"></rompe-cadena>
```

binding de una dirección y de dos direcciones



interoperabilidad entre componentes

En una aplicación basada en componentes (y esto aplica a todo el desarrollo basado en el estándar Web Components) tenemos una arquitectura basada en Custom Elements, por medio de un árbol de elementos que interactúan entre sí para cumplir el propósito de las aplicaciones. Esa actuación coordinada entre los componentes, necesaria para la resolución de los problemas se conoce como interoperabilidad. En Polymer se resuelve permitiendo que los componentes se pasen datos los unos a los otros por medio del binding y también gracias a la comunicación de eventos personalizados.

Pongamos que queremos hacer un componente que muestre un párrafo en un espacio reducido de la página. El párrafo tiene mucho texto así que a veces se puede recortar. Pero también queremos que el usuario pueda verlo entero si lo requiere, pulsando un botón para evitar ese recorte.

Nota: Esta tarea es bastante simple como para hacer un único componente, quizás no es el mejor ejemplo, pero piensa en una aplicación pequeña o mediana, la cantidad de subtareas que puede tener. Cada una de ellas se encapsula en un componente y eso permite que su complejidad sea pequeña, facilitando el desarrollo, pero sobre todo el mantenimiento, y ubicando perfectamente cada cosa en su sitio.

Para esta tarea podemos pensar en dos componentes distintos. Uno que se encargue únicamente del recorte de la cadena y otro que se encargue de la interacción con el usuario, permitiendo captar el evento de recorte o despliegue del párrafo cuando se pulsa el correspondiente botón. Quebrar esta tarea en dos componentes todavía es una solución más acertada cuando pensamos que ya existe el componente de romper una cadena, porque lo tenemos desarrollado.

binding de una dirección y de dos direcciones



En nuestro componente padre "mostrar-recortar" tendremos el siguiente template:

```
<template>
  <p>
    <rompe-cadena str="{{cadena}}" len="{{longitud}}"></rompe-cadena>
    <iron-icon icon="{{icono}}" on-tap="mostrarRecortar"></iron-icon>
  </p>
</template>
```

Como ves, unos componentes incluyen a otros y así se va generando ese árbol de componentes que son capaces de hacer funcionar una aplicación completa.

binding de una dirección y de dos direcciones

Para definir si queremos binding de una dirección y de dos direcciones, utilizamos dos tipos de notaciones distintas:

La notación de los dobles corchetes "[[]]" significa que se debe implementar binding de una sola dirección.

En este caso el valor bindeado viaja del padre al hijo pero no del hijo al padre.

La notación de las dobles llaves "{()}" significa binding de dos direcciones.

Es decir, el valor bindeado viaja del padre al hijo y también del hijo al padre.

Nota: Ojo con el binding de dos direcciones, puesto que es un poco más pesado para la aplicación, en términos de tiempo de procesamiento. Por eso en muchas ocasiones es preferible usar bindeo de una sola dirección y usar mecanismos como los eventos para producir el aviso a los padres cuando los hijos desean cambiar el valor de las propiedades.

binding de una dirección y de dos direcciones

Qué son propiedades

En los componentes de Polymer tenemos propiedades, declaradas mediante la declaración "properties". Eso son lo que llamamos propiedades del componente y son las que hemos usado repetidas veces a lo largo del manual. Polymer facilita la asignación de valores a esas propiedades del componente, por medio del template.

```
<paper-input label="Escribe algo" value="{{valorEscrito}}"></paper-input>
```

Ves dos casos interesantes en el anterior código:

A una propiedad como "label" le podemos asignar directamente un literal de cadena.
A la propiedad "value" le estamos asignando el valor de otra propiedad del componente padre ("valorEscrito").

Polymer se encarga de asignar esos valores a las propiedades. En el caso de "label" simplemente le asigna ese valor literal, pero en el caso del "value" tenemos un binding de propiedad. En este caso "value" es una propiedad del elemento "paper-input", propia de Polymer.

binding de una dirección y de dos direcciones

Qué son atributos

Sin embargo, hay veces que quieres asignar valores a atributos de un elemento y no a propiedades declaradas de Polymer. Es el caso de atributos como "class", "style", "hidden"... cualquier elemento del HTML tiene esos atributos, por ejemplo una etiqueta DIV, P, etc. También hay etiquetas que tienen atributos específicos de ellas mismas, por ejemplo la etiqueta A (un enlace) tiene el atributo "href". Incluso elementos de Polymer pueden tener atributos estándar del HTML como class.

En general, como ves, todo lo que son atributos de las etiquetas, que vienen del lenguaje HTML, son lo que llamamos atributos.

Cómo funciona el binding a atributos

Como decía, el binding a propiedades de elementos de Polymer no tiene ningún secreto. Lo que tenemos que aprender es el binding de atributos, que es prácticamente lo mismo, solo que tenemos que indicar un carácter "\$" en el template.

Se debe colocar un \$ justo después del nombre del atributo, por ejemplo:

<div class\$="[[claseActiva]]">Tendrá un class cuyo valor se bindea</div>

binding de una dirección y de dos direcciones



Sobre el binding a atributos debes saber algunas cosas:

- Sólo se pueden hacer de una dirección (one way binding). Para hacer de dos direcciones hay alternativas en algunos casos.
- Internamente Polymer los implementa con el método nativo `setAttribute()`
- En algunos navegadores puede que funcione el binding a atributos sin colocar el "\$", pero no te confunda, pues en otros browsers puede no funcionar.

binding de una dirección y de dos direcciones

Genera los ejemplos seis y siete, generado el llamado desde el archivo “clase-uno”, con el siguiente Código y no olvides generar la referencia.
Para el caso de los archivos “ejemplo-seis” y “ejemplo-siete”, te los anexo para que sean implementados.

<p class= "ejeuno">Ejemplo 6 binding one way</p>

<ejemplo-seis></ejemplo-seis>

<p class= "ejeuno">Ejemplo 7 binding two way</p> <ejemplo-siete></ejemplo-siete>



ejemplo-seis.html



ejemplo-seis-hijo.
html



ejemplo-siete



ejemplo-siete-hij
o

DATA BINDING HELPER ELEMENTS



El componente dom-repeat es un tipo de template de Polymer que permite hacer una iteración entre los elementos de un array.

Hasta este punto en el Manual de Polymer no hemos entrado todavía en detalle con los templates de repetición, una de las herramientas más usadas a la hora de construir la parte visual de un componente

Los templates de repetición, o templates de tipo "dom-repeat", nos permiten iterar entre los elementos de un array y producir una repetición de elementos en la vista de un componente Polymer.

El dom-repeat, junto con otros tipos de templates específicos de Polymer los tienes documentados en la clasificación Data binding helper elements y es que son elementos preparados para aprovechar el sistema de binding de Polymer. En ellos bindearemos un array y el propio componente permitirá que un grupo de elementos de un template se repita tantas veces como elementos en el array, bindeando a su vez el elemento actual a cada ítem repetido. Enseguida lo veremos mejor con un ejemplo.

DATA BINDING HELPER ELEMENTS

Implementar un dom-repeat

Para implementar un dom-repeat tenemos que partir de un array. Ese array será el que usaremos para hacer la repetición. Ese array de datos lo podrías tener escrito tal cual en el código, aunque lo corriente es que lo recibas de una consulta Ajax a un servicio web o API REST

```
properties: {  
  cervezas: {  
    type: Array,  
    value: function() {  
      return [  
        {  
          name: 'Brahma',  
          type: 'Pilsen'  
        },  
        {  
          name: 'Saint Beer',  
          type: 'Weiss'  
        },  
        {  
          name: 'Coruja',  
          type: 'Pale ale'  
        }  
      ]  
    }  
  }  
}
```

DATA BINDING HELPER ELEMENTS



Como puedes ver, es un array donde cada uno de los elementos es un objeto. Podríamos haber usado arrays donde elementos son valores primitivos, como cadenas o números, pero lo cierto es que casi siempre iteras por items que son objetos.

Una vez que tenemos nuestro array, de cervezas en este caso, lo podemos recorrer de manera declarativa con nuestro template de repetición, de esta manera:

```
<template is="dom-repeat" items="[[cervezas]]">
  <h2>[[item.name]]</h2>
  <p>Tipo de cerveza: [[item.type]]</p>
</template>
```

Este código lo colocarás tal cual en el template de tu componente. Como puedes observar, el dom-repeat es un tipo específico de template, por lo que estamos en la práctica anidando templates. Para indicar que este template es de repetición usamos is="dom-repeat", lo que significa que éste elemento de repetición es una especialización del elemento template de los web components.

A dom-repeat le tienes que indicar el array con el que vas a realizar la repetición en el atributo "items". En este caso le hemos pasado el array de cervezas usando el sistema de binding de Polymer.

DATA BINDING HELPER ELEMENTS



Alias del ítem

Algo sencillo que puede mejorar nuestro código y hacerlo más fácil de leer es asignar un alias al ítem dentro del template. Si cada uno de los ítems del array es una cerveza, podemos asignar el alias "cerveza" para ese ítem actual mediante el atributo `as="cerveza"`.

En el código nos quedaría así:

```
<template is="dom-repeat" items="[[cervezas]]" as="cerveza">
  <h2>[[cerveza.name]]</h2>
  <p>Tipo de cerveza: [[cerveza.type]]</p>
</template>
```

Nota: Colocar el alias es una decisión del programador a la hora de hacer su código. Este código y el anterior template hacen exactamente lo mismo.

Ahora que tenemos un alias, el nombre de la cerveza lo tendremos en `cerveza.name` y su tipo en `cerveza.type`.

DATA BINDING HELPER ELEMENTS

Genera el ejemplo-ocho.html, e ingresa la propiedad dom-repeat para cargar un Array que debes declarar como propiedad.

```
<p> ejemplo-ocho.html </p>
  <h1>DATA BINDING HELPER ELEMENTS </h1>
  <strong> dom-repeat --> "se utiliza para moverse en un array u objeto sin necesidad de usar un
ciclo for" ejemplo:: </strong>
  <p></p>

<div> Lista de empleados: </div>
<template is="dom-repeat" items="{{employees}}">
  <div># [[index]]</div> <!-- muestra elconsecutivo-->
  <div>First name: <span>[[item.first]]</span></div>
  <div>Last name: <span>[[item.last]]</span></div>
</template>
```

Array: { return [{first: 'Bob', last: 'Smith'}, {first: 'Sally', last: 'Johnson'}]; }

DATA BINDING HELPER ELEMENTS



Genera el webcomponent “ejemplo-ocho-hijouno.html”, y que se mande a llamar desde “ejemplo-ocho.html”, para generar nuevamente un Array y un método que se activa al presionar un botón que se encuentra dentro del templete del dom-repeat.

```
<p>hijo 1</p>
<template is="dom-repeat" id="menu" items="{{menuItems}}">
  <div>
    <span>{{item.name}}</span>
    <span>{{item.ordered}}</span>
    <button on-click="order">Order</button>
  </div>
</template>
```

```
Array: type: Array,
      value: function() { return [
        {name: 'Pizza', ordered: 0},
        {name: 'Pasta', ordered: 0},
        {name: 'Toast', ordered: 0}
      ]; }
```

```
Metodo: order: function (e) {
  e.model.set('item.ordered', e.model.item.ordered+1);
```


DATA BINDING HELPER ELEMENTS

Genera el webcomponent “ejemplo-ocho-hijos.html”, y que se mande a llamar desde “ejemplo-ocho.html”, para generar nuevamente un Array y un método que va a filtrar los datos del array.

```
/*Datos en el Dom: */
<p>ingresa parametro de busqueda:</p>
  <input value="{{searchString::input}}">
</p></p>
<template is="dom-repeat" items="{{employees}}" as="employee"
  filter="{{computeFilter(searchString)}}">
  <div>{{employee.lastname}}, {{employee.firstname}}</div>
</template>
```

```
type: Array,
value: function() { return [
  { firstname: "Jack", lastname: "Aubrey" },
  { firstname: "Anne", lastname: "Elliot" },
  { firstname: "Stephen", lastname: "Maturin" },
  { firstname: "Emma", lastname: "Woodhouse" }
]; }
```

```
/* Metodo */
computeFilter: function (string) {
  if (!string){
    return null;
  } else {
    string = string.toLowerCase();
    return function(employee){
      var first = employee.firstname.toLowerCase();
      var last = employee.lastname.toLowerCase();
      return (first.indexOf(string) != -1 ||
        last.indexOf(string) != -1);
    }
  }
}
```

DATA BINDING HELPER ELEMENTS

Genera el webcomponent “ejemplo-ocho-hijotres.html”, y que se mande a llamar desde “ejemplo-ocho.html”, para generar nuevamente un Array con de 4 X 2 { **firstname: "Jack", lastname: "Aubrey"** }, y en el dom se ocupara un alias y la propiedad index-as para enumerar los elementos del array.

```
<template is="dom-repeat" items="{{employees}}" as="employee" index-as="report_no">
```

la propiedad as --> renombra el nombre de la propiedad en el codigo pasa de employees a employee

la propiedad index-as genera dentro del dom-repeat el numero de renglones iniciado por cero ver en codigo "ejemplo-ocho-hijotres" --> index-as="report_no"

Lista de empleados:

0

First name: Jack

Last name: Aubrey

1

First name: Anne

Last name: Elliot

2

First name: Stephen

Last name: Maturin

3

First name: Emma

Last name: Woodhouse

DATA BINDING HELPER ELEMENTS



Genera el webcomponent “ejemplo-ocho-hijocuatro.html”, y que se mande a llamar desde “ejemplo-ocho.html”, y copia del componente que te anexo el código para que puedas Ver como se ocupan las propiedades: push, pop, unshift, shift, y splice.

Una vez que corrió el código correctamente en tu equipo incrementa una columna en el array que indique la ciudad de nacimiento y genera un estilo a tu gusto a la tabla donde se muestra el array. Por ultimo al llamar a tu webcomponent debes pasar el valor que indica el ancho de cada columna y modifica el texto del botón que ocupa la propiedad splice por splice.

<th style="width:200px">Number</th>



Polymer 1

Curso de capacitación para personal asignado a
BBVA

Value
THROUGH
TECHNOLOGY