

# Índice

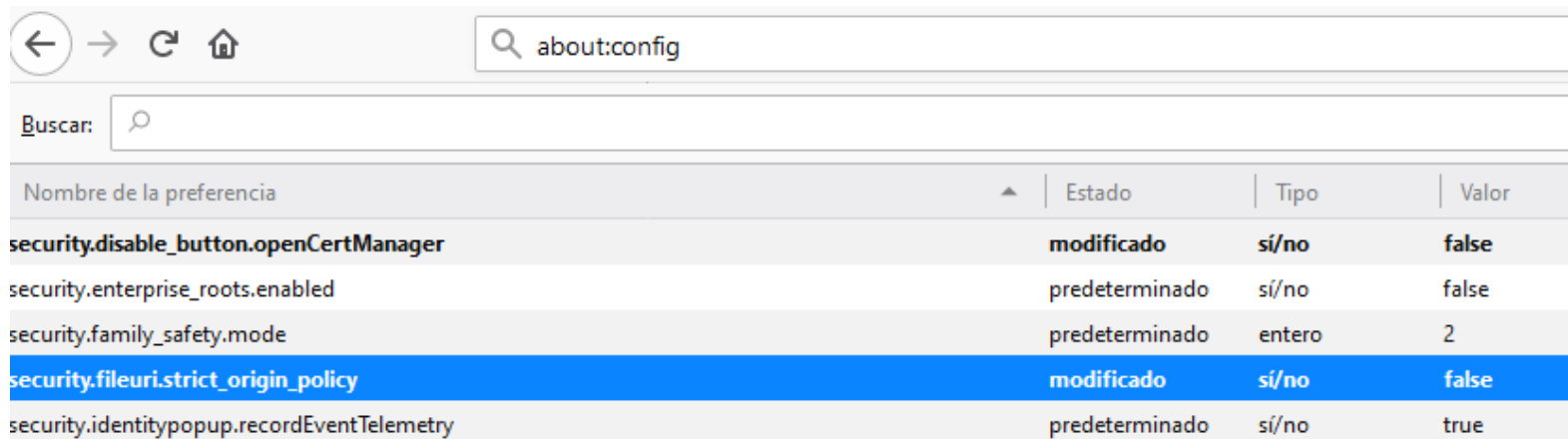
1. Herramientas
2. Cargando Datos
3. Selecciones y Operadores
4. Creando Elementos Nuevos
5. Insertando Datos
6. Aplicando Propiedades Dinámicas
7. Añadiendo Elementos SVG
8. `join()`: Actualización Dinámica de Elementos

# 1. Herramientas (I)

- Javascript se ejecuta en local, por lo tanto:
  - No es necesario usar un servidor para las pruebas
- Problema:
  - Vamos a cargar datos desde fichero, lo que genera problemas en determinados navegadores que no dejan hacerlo en local
    - Ej. Chrome
- Solución:
  - Instalar un servidor local en nuestra máquina
    - Ej. EasyPHP
  - Probar con navegadores que permitan cargar datos desde ficheros locales.
    - Ej. Mozilla Firefox
      - A partir de la versión 68 ya no lo permite por defecto. En la siguiente transparencia veremos cómo lograrlo
  - Chrome tiene extensiones que permiten instalar servidores HTML ligeros
    - Es otra opción
- Las pruebas de ejemplos y ejercicios se ha hecho con Firefox.

# 1. Herramientas (II)

- Solución a la nueva directiva de seguridad en Mozilla Firefox para poder probar javascript que cargan datos desde ficheros locales:
  1. Escribir como URL “about:config”
  2. Buscar la preferencia “security.fileuri.strict\_origin\_policy”
  3. Ponerlo a “false”, haciendo clic sobre esa preferencia



The screenshot shows the Firefox 'about:config' page. The address bar contains 'about:config'. Below the search bar, a table lists various preferences. The preference 'security.fileuri.strict\_origin\_policy' is highlighted in blue, indicating it is selected. Its state is 'modificado' (modified) and its value is 'false'.

Nombre de la preferencia	Estado	Tipo	Valor
security.disable_button.openCertManager	modificado	sí/no	false
security.enterprise_roots.enabled	predeterminado	sí/no	false
security.family_safety.mode	predeterminado	entero	2
<b>security.fileuri.strict_origin_policy</b>	<b>modificado</b>	<b>sí/no</b>	<b>false</b>
security.identitypopup.recordEventTelemetry	predeterminado	sí/no	true

## 2. Cargando Datos - Introducción

- Los datos a usar para las representaciones pueden ser definidos en el código
  - Hemos visto cómo hacerlo en el tema anterior
  - Ejemplo código:

```
var datos = [10,20,30,40];  
for (i=0; i<datos.length; i++) {  
    console.log(datos[i]);  
}
```
- Aquí vamos a ver cómo podemos cargar datos desde ficheros externos en D3
  - Recordar que desde la versión 5 se usa el API Fetch (<https://fetch.spec.whatwg.org/>) para traer o cargar datos
  - Aquí vamos a usar la opción simple, sin control de errores
    - Más adelante veremos de manera práctica como gestionar los errores que se puedan producir en la carga.
- Para los ejemplos vamos a usar la versión 6 de d3.

## 2. Cargando Datos - csv

- csv: formato de fichero con campos separado por comas. Es una forma estándar de pasar tablas a ASCII
- Ejemplo fichero (nombre, por ejemplo, "datosCSV.csv"):

```
Fruta,Precio
Pera,1.45
Naranja,0.76
Manzana Golden,2.77
```

- Lectura mediante el método `d3.csv()` (se añade el código para verlo en consola):
  - Ejemplo para ver el objeto en el que se almacena el fichero leído

```
d3.csv("datosCSV.csv").then(function(data) {
  console.log(data);
});
```

Función anónima usada como "callback function": función llamada sólo tras ser leído correctamente el fichero; se le pasa como argumento el objeto creado con los datos leídos (para ser más precisos: el array de objetos)

- Ejemplo para ver cada dato leído:

```
d3.csv("datosCSV.csv").then(function(data) {
  for(i=0;i<data.length;i++){
    console.log(data[i].Fruta + " " + data[i].Precio);
  }
});
```

O con bucle `foreach()`:

```
d3.csv("datosCSV.csv", function(data) {
  data.forEach(function(d) {
    console.log(d.Fruta + " " + d.Precio);
  });
});
```

Nuevamente usamos una función anónima: para cada elemento del array "data" se ejecutará la función, pasando como argumento a ella el elemento correspondiente del array (recordemos, un dato de tipo objeto)

## 2. Cargando Datos – Otros Formatos

- Campos separado por tabuladores (tsv):
  - Método D3: `d3.tsv()`
    - Ej.: `d3.tsv("datosTSV.tsv").then(function(data) { ... });`
- Campos separados por cualquier carácter:
  - Método D3: `d3.dsv()`
    - Ej.: `var ssv = d3.dsv(" ", "text/plain"); // fichero usando el espacio como separador`  
`ssv("datosSSV.txt").then(function(data) { ... });`
- Fichero JSON
  - Método D3: `d3.json()`. Ejemplo:

Fichero	Código Javascript 1	Código Javascript 2
<pre>{ "frutas": [   {     "Fruta": "manzana",     "Precio": 1.47   },   {     "Fruta": "naranja",     "Precio": 2.00   } ]}</pre>	<pre>d3.json("datos.json").then(function(data) {   console.log(data); });</pre>	<pre>d3.json("datos.json").then(function(data) {   for (i=0; i&lt;data.frutas.length; i++) {     console.log(data.frutas[i].Fruta + " " +       data.frutas[i].Precio);   } });</pre>

## 3. a 7.

- Usar Tema 2 del libro:
  - Fabio Nelli, Create Web Charts with D3, Ed. Apress, 2014
  - Páginas 20 a 38
- Documento en Web:
  - D3\_Tema2\_D3Basico\_Aptdos3-7.pdf
- Recordando:
  - **Atributos** HTML: valores adicionales que configuran los elementos HTML o ajustan su comportamiento de diversas formas
    - Particulares de cada etiqueta
      - Básicos (aplicables a casi todas las etiquetas): id, class, style, title
      - De svg: ancho y alto del elemento
    - Enlace: <https://developer.mozilla.org/es/docs/Web/HTML/Atributos>
  - **Propiedades** HTML: definidos mediante el atributo “style” directamente en las etiquetas HTML o mediante css, modifican las propiedades (formato o apariencia) de los elementos afectados
    - Ej. color/tamaño fuente, tipo/color línea de la caja, etc.
    - Enlace: [http://www.w3schools.com/html/html\\_css.asp](http://www.w3schools.com/html/html_css.asp)
    - Lista propiedades css3: <http://www.w3schools.com/cssref/>
  - Ej. `<circle cx="25" cy="25" r="20" style="fill:purple; stroke:green; stroke-width:10; opacity:0.9" />`  

AtributosPropiedades

## 8. join(): Actualización Dinámica de Elementos (I)

- A partir de la versión 5 se introduce la función `join()` que permite de manera sencilla realizar las siguientes operaciones:
  - enter: introducir nuevos elementos en la visualización
  - update: actualizar los elementos de la visualización ya existentes
  - exit: actuar sobre los elementos de la visualización que desaparecen
    - `join()` fusiona (merge) y ordena automáticamente los elementos resultantes de las operaciones anteriores
- Sintaxis: `selection.join(enter[, update][, exit])`
  - Documentación: [https://github.com/d3/d3-selection#selection\\_join](https://github.com/d3/d3-selection#selection_join)
  - Si solo se pone el primer argumento funciona igual que la función `enter()` que hemos visto.
  - Ejemplo código página 34 del documento D3\_Tema2\_D3Basico\_Aptdos3-7.pdf, pero con `join()` en vez de `enter()` (ponemos solo la parte javascript):

```
<script type="text/javascript">
  var fruits = ['Apples', 'Pears', 'Bananas', 'Oranges', 'Strawberries'];
  var list = d3.select('ul');
  var fruits = list.selectAll('li').data(fruits);
  fruits.join('li').append('li').text(function (d) {
    return d;
  });
  fruits.text(function (d) {
    return d;
  });
</script>
```



## 8. join(): Actualización Dinámica de Elementos (II)

- Sintaxis con los tres argumentos:

Sintaxis función anónima con “=>”	Sintaxis función anónima con “function”
<pre>selection.data(data)   .join(     nuevos =&gt; nuevos.append(...),     actualizar =&gt; actualizar,     eliminados =&gt; eliminados.remove()   ) ...</pre>	<pre>selection.data(data)   .join(     function(nuevos) {nuevos.append(...)},     function(actualizar) {actualizar},     function(eliminados) {eliminados.remove()}   ) ...</pre>

- Secciones:
  - nuevos: parte que se ejecuta para todos los elementos del array de datos (*data*) que no tengan asociados un elemento html de la selección (*selection*)
  - actualizar: parte que se ejecuta para todos los elementos del array de datos (*data*) que ya tienen asociado un elemento html de la selección (*selection*)
  - eliminados: parte que se ejecuta para los elementos html de la selección (*selection*) que no tienen asociado un elemento del array de datos (*data*).

## 8. join(): Actualización Dinámica de Elementos (III)

- A definir: la forma de asociar (binding) elemento del array de datos - elemento html en la selección
  - Por defecto data() lo hace de manera secuencial: primer elemento de ambos, segundo elemento, etc.
  - Esta forma no suele ser adecuada con join, siendo normalmente necesario especificar lo forma de realizar esa asociación. Forma de hacerlo: con el segundo argumento de la función data().
    - `selection.data([data[, key]])` ([https://github.com/d3/d3-selection#selection\\_data](https://github.com/d3/d3-selection#selection_data))
    - “key” es la clave (valor único) que se usará para asociar dato-elemento en la selección
- La mejor manera de entenderlo es con un ejemplo.
  - Esta en [Tema 2. Material extra - ejemplo con join\(\)](#).
  - Seguimos con el ejemplo de estos apuntes para probar carga de datos
  - Simulamos mediante una función generadora ([https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/function\\*](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/function*)) una carga dinámica de datos que varían (variación en el catálogo y en el precio de las futas)
  - Con la función join() actualizamos la página, añadiendo nuevas frutas (sección enter), modificando los precios de las existente (sección update) o eliminando las que ya no están (sección exit)
  - Se han añadido operaciones adicionales para ir ampliando el conocimiento de d3
    - Concretamente: si el precio sube se colora en rojo, si baja en verde y si se mantiene en negro, pero solo el precio. Esto implica, entre otras cosas, guardar los valores anteriores.
  - Hay dos versiones, con la función flecha (`=>`) y con la sintaxis normal (`function(){..}`)
  - Como clave (key) se usa el valor del campo Fruta (único para cada elemento).
  - Abrir los ficheros, entender su contenido y ejecutarlos.