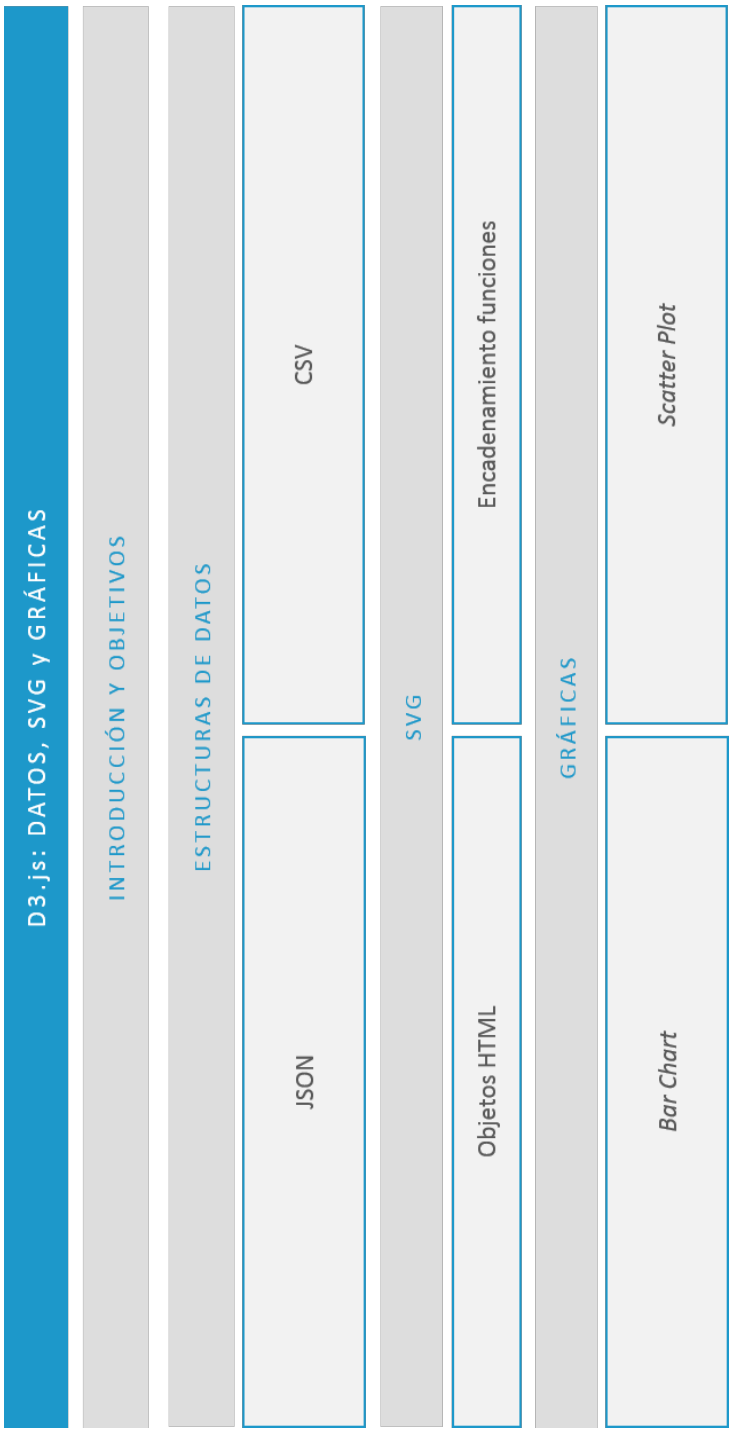
Herramientas de Visualización

Tema 4. D3.js. Datos, SVG y gráficas

Índice

Esquema



4.1. Introducción y objetivos

Ya hemos comenzado a experimentar con D3 en el tema anterior. Hemos visto la forma de trabajar con esta biblioteca, pero todavía no hemos creado ninguna gráfica.

En este tema introduciremos los elementos más importantes para empezar a dibujar y para ello, trabajaremos con ficheros CSV o JSON. También abordaremos los elementos SVG (*Scalable Vector Graphics*) en HTML y empezaremos a dibujar de una manera sencilla un *Bar Chart* y un *Scatter Plot,* todo con D3 y con elementos HTML.

Jugaremos con los estilos de las gráficas y anotaremos ciertos textos en los gráficos, pero algo simple, ya tendremos tiempo para hacer cosas más complejas.

Comenzamos este tema con una frase de Confucio:

«Life is really simple, but men insist on making it complicated» (Brainy Quote).

Siguiendo la línea del tema anterior, aconsejamos asentar primero los conceptos. Si saltamos rápidamente a intentar modificar visualizaciones, el aprendizaje puede complicarse, no hay nada malo en empezar lento pero seguro, aunque se tengan conocimientos de programación o de JavaScript, con D3 es fácil perderse.

Estudiar este tema no es diferente a los anteriores, recomendamos practicar con todas las instrucciones del código. Elige Brackets u otra herramienta para que puedas repetir los pasos. Además, el apartado sobre las estructuras de datos JSON y CSV es importante, así que insistimos en la relevancia de asimilar su contenido. Entender cómo funciona la gestión de datos permitirá modificar futuras visualizaciones de manera más ágil.

Por otro lado, en el resto de apartados del tema se asientan conceptos que luego podremos ir desarrollando con más facilidad, por lo que se aconseja probar todo el

código.

4.2. Trabajar con diferentes estructuras de datos JSON y CSV

CSV

Empezaremos creando un archivo CSV en Brackets. Lo llamaremos «comida.csv»:

- Comida,Preferencia.
- Manzana,9.
- Huevos fritos,10.
- Ensalada,5.
- Galleta,2.
- Leche,8.

¿Cómo cargamos el archivo con D3?:

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="styles.css"/>
<script language="javascript" type="text/javascript"
src="http://d3js.org/d3.v3.min.js"></script>
<script type="text/javascript">
function read(){
d3.csv("comida.csv", function(data) {
data.forEach(function(d){
d3.select("body").append("p").text(d.Comida + " " + d. Preferencia);
});
});
}
</script>
</head>
<body onload="read()">
</body>
```

```
</html>
```

Analicemos lo importante:

```
d3.csv("comida.csv", function(data) {
data.forEach(function(d){
d3.select("body").append("p").text(d.Comida + " " + d. Preferencia);
})
})
```

La función CSV acepta **dos atributos**:

- El primero es la **ruta al archivo**, es decir, que si el archivo no está en la misma carpeta del HTML, posiblemente dará error. Es importante recordar estos pequeños detalles.
- La segunda es la **función callback**, que ya hemos ido viendo en otros temas. Si la función no encuentra el archivo CSV, la función callback nunca será llamada.

Lo que hacemos a continuación es imprimir todos los valores de nuestro CSV, observa que no es una matriz de valores. Accede a los valores iterando por las filas y luego llama a los valores basándote en las cabeceras.

JSON

De manera muy similar, podemos trabajar con archivos JSON. Crearemos un ejemplo de coordenadas. El nuevo archivo lo llamaremos «coordenadas.json».

```
[{
"x_axis": 30,
"y_axis": 30,
"radius": 20,
"color": "green"
}, {
"x_axis": 70,
"y_axis": 70,
"radius": 20,
"color": "purple"
```

```
}, {
"x_axis": 110,
"y_axis": 100,
"radius": 20,
"color": "red"
}]
```

Leeremos el objeto y lo recorreremos de forma muy similar al CSV:

```
d3.json("coordenadas.json", function(data) {
data.forEach(function(d){
d3.select("body").append("p").text(d.x_axis + " " + d.radius+ " "
+ d.y_axis + " " + d.color);
})
})
```

En este nuevo ejemplo, fíjate que utilizamos la función d3.json para leer el fichero en lugar de la función d3.csv.

4.3. Generar y dibujar con SVG

D3 es mucho más útil cuando lo utilizamos para gestionar objetos gráficos como los elementos SVGs (*Scalable Vector Graphic*). También podemos crear visualizaciones con elementos div, pero es mucho más efectivo trabajar con estos elementos para evitar problemas entre diferentes tipos de dispositivos y exploradores web y asegurar así, la **máxima compatibilidad**.

Empezamos con un ejemplo. Pon en el *body* del HTML el siguiente código:

```
<svg width="50" height="50">
<circle cx="25" cy="25" r="22" fill="blue" stroke="gray" stroke-width="2"/>
</svg>
```

Obtendremos el siguiente resultado:

Figura 1. Resultado objeto SVG.

Se recomienda siempre generar los elementos SVG con sus **atributos *width* y *height*.**

La unidad por defecto es píxel, por esa misma razón veremos que las medidas de nuestro ejemplo son 50 y no 50px. También podremos jugar con la posición dentro de nuestra pantalla. Recuerda que el **elemento 0,0** siempre es la esquina superior izquierda.

```
<rect x="100" y="0" width="200" height="50"/>
```

Si te fijas en la diferencia, cuando hemos dibujado un **círculo**, las coordenadas cx y cy indican las coordenadas del centro del círculo, además se define un radio para

definir el tamaño, mientras que en el **rectángulo**, indicarían la posición de la esquina superior izquierda.

Podemos dibujar una **elipse**:

```
<ellipse cx="250" cy="25" rx="100" ry="25"/>
```

También es importante saber cómo dibujar una **línea**. Aquí definiremos el punto inicial y el punto final y elemento que les une: *stroke*.

```
<line x1="0" y1="0" x2="500" y2="50" stroke="black"/>
```

Podemos **renderizar** elementos gráficos:

```
<text x="250" y="25" font-family="serif" font-size="25" fill="gray"></text>
```

Como en casi cualquier elemento HTML, puedes definir el **estilo**:

- Fill: si lo quieres rellenar. Puedes utilizar los mismos valores que en CSS para los colores.
- Stroke: el color de la línea.
- Stroke-width: anchura de la línea.
- Opacity: un número ente 0.0 y 1.0.

También se puede **utilizar CSS**. Por ejemplo, especificamos la clase en el SVG:

```
<circle cx="25" cy="25" r="22" class="pumpkin"/>
```

Y la definimos:

```
.pumpkin {
fill: yellow;
stroke: orange;
stroke-width: 5;
}
```

Es importante saber que en SVG no existe el concepto capas ni manera de definir la posición del eje z, ya que nos movemos en un plano bidimensional. Por ejemplo, si queremos causar un efecto como el de la figura 2:

Figura 2. Rectángulos superpuestos en plano bidimensional.

Deberemos jugar con el orden y la posición de la x y la y:

```
<rect x="0" y="0" width="30" height="30" fill="purple"/>
<rect x="20" y="5" width="30" height="30" fill="blue"/>
<rect x="40" y="10" width="30" height="30" fill="green"/>
<rect x="60" y="15" width="30" height="30" fill="yellow"/>
<rect x="80" y="20" width="30" height="30" fill="red"/>
```

Si quieres jugar con las **transparencias**, puedes utilizar los colores en formato RGBA, donde el último valor indica la transparencia:

```
<circle cx="25" cy="25" r="20" fill="rgba(128, 0, 128, 1.0)"/>
<circle cx="50" cy="25" r="20" fill="rgba(0, 0, 255, 0.75)"/>
<circle cx="75" cy="25" r="20" fill="rgba(0, 255, 0, 0.5)"/>
<circle cx="100" cy="25" r="20" fill="rgba(255, 255, 0, 0.25)"/>
<circle cx="125" cy="25" r="20" fill="rgba(255, 0, 0, 0.1)"/>
```

Da el siguiente resultado:

Figura 3. Círculos superpuestos con transparencia.

Pero también podemos jugar todavía con el elemento de la **opacidad** que hemos comentado antes. Por ejemplo:

```
<circle cx="25" cy="25" r="20" fill="purple" stroke="green" stroke-width="10"
```

```
opacity="0.9"/>
<circle cx="65" cy="25" r="20" fill="green" stroke="blue" stroke-width="10"
opacity="0.5"/>
<circle cx="105" cy="25" r="20" fill="yellow" stroke="red" stroke-width="10"
opacity="0.1"/>
```

Da como resultado:

Figura 4. Círculos superpuestos con transparencia y opacidad.

4.4. Bar Chart y Scatter Plot desde cero

Una vez explicados los elementos SVG, explicaremos cómo hacer un *Bar Chart* y un *Scatter Plot* desde cero. En un principio no utilizaremos elementos SVG, sino que usaremos elementos div para trabajar con ellos. Trabajaremos con una estructura simple de datos:

```
var dataset = [ 5, 10, 15, 20, 25 ];
```

Como ejercicio intenta trabajar con la función de lectura del CSV y sigue los pasos con el CSV en lugar de este *array*.

Bar Chart

¿Cómo podemos dibujar una barra con el uso exclusivo del elemento *div*?

Prueba este código:

```
<div style="display: inline-block;
width: 20px;
height: 75px;
background-color: teal;">
</div>
```

Como podemos observar, lo logramos definiendo el estilo como display: inline-block; y se obtiene lo siguiente:

Figura 5. Barra creada con elementos div.

Para poder aplicar este estilo a diferentes *div*, crearemos nuestra clase en CSS:

```
div.bar {
display: inline-block;
width: 20px;
height: 75px; /* We'll override height later */
background-color: teal;
}
```

Por lo tanto, ahora solo tendremos que referenciarlo de la siguiente manera obteniendo el mismo resultado:

```
<div class="bar"></div>
```

Intentamos hacer lo mismo con D3, por cada elemento del *array dataset* que hemos creado antes:

```
function read(){
d3.select("body").selectAll("div")
.data(dataset)
.enter()
.append("div")
.attr("class", "bar");
}
```

Verás que ahora tenemos cinco barras con la misma altura y concatenadas unas a otras, lo que no es muy útil, ya que la altura debería variar según el contenido del *array*.

Para conseguir el comportamiento deseado hemos de sobrescribir el valor *height* de nuestro CSS. Para ello solo tenemos que añadir una pequeña sentencia después del código D3:

```
function read(){
d3.select("body").selectAll("div")
.data(dataset)
.enter()
.append("div")
```

```
.attr("class", "bar")
.style("height", function(d) {
return d + "px";
})
}
```

Si quieres ver el comportamiento con **más valores** en el *array,* puedes probar a generar el *array* con valores aleatorios.

```
var dataset = []; //Inicializar array vacio
for (var i = 0; i < 10; i++) { //Loop 10 veces
var newNumber = Math.floor(Math.random() * 30); //Números aleatorios (0-30)
dataset.push(newNumber); //Añadir al array
}
```

Este es el ejemplo que se da para 10 valores. Solo tienes que modificar ese número si quieres más o menos valores.

El código completo quedaría de la siguiente manera:

```
<html>
<head>
<script language="javascript" type="text/javascript"
src="http://d3js.org/d3.v3.min.js"></script>
<style>
.bar {
display: inline-block;
width: 50px;
height: 75px; /* Lo sobreescribiremos */
background-color: teal;
}
</style>
<script>
var dataset = []; //Inicializar array vacío
for (var i = 0; i < 10; i++) { //Loop 10 veces
var newNumber = Math.floor(Math.random() * 30);
dataset.push(newNumber); //Añadir al array
}
function read(){
d3.select("body").selectAll("div")
.data(dataset)
.enter()
.append("div")
.attr("class", "bar")
```

```
.style("height", function(d) {
return d + "px";
})
}
</script>
</head>
<body onload="read()">
</body>
</html>
```

El resultado con 10 valores aleatorios es el siguiente:

Figura 6. Barras aleatorias en D3.

Ya lo sabemos hacer con elementos *div*. ¿Qué tal si lo intentamos con elementos SVG?

¿Cómo podemos dibujar una barra con elementos SVG?

Primero definiremos el **tamaño** de nuestro SVG:

```
//Width and height
var w = 500;
var h = 100;
```

Luego crearemos nuestro **elemento** SVG:

```
//Create SVG element
var svg = d3.select("body")
.append("svg")
.attr("width", w)
.attr("height", h);
```

Podemos ver que lo que realmente crea el elemento SVG es la llamada a la función *append("svg").* Luego se generan las barras creando rectángulos.

```
svg.selectAll("rect")
```

```
.data(dataset)
.enter()
.append("rect")
.attr("x", 0)
.attr("y", 0)
.attr("width", 20)
.attr("height", 100);
```

Si ejecutamos esto, veremos que solo tenemos una barra negra del mismo tamaño.

Hemos creado cinco barras, pero todas están en la misma X e Y, y todas tienen la misma altura. ¿Cómo lo arreglamos?

```
svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("x", function(d, i) {
return i * (w / dataset.length);
})
.attr("y", 0)
.attr("width", w / dataset.length - 1)
.attr("height", function(d) {
return d;
});
```

Atención a lo que hacemos con los valores *x, height* y *width*. El -1 es para hacer la separación entre barra y barra, pero si ejecutas esto en Brackets, ¿cuál será el problema?

El gráfico te sale invertido. Bueno hemos de recordar que empezamos a dibujar en la coordenada 0,0 que está en la esquina superior izquierda, por lo que la altura definida al rectángulo es hacia abajo.

Figura 7. Barras aleatorias en D3 con SVG.

Para que todo salga bien, tienes que cambiar el valor de la "y" en cada elemento.

```
svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("x", function(d, i) {
return i * (w / dataset.length);
})
.attr("y", function(d) {
return h - d; // Altura menos el valor del dato
})
.attr("width", w / dataset.length - 1)
.attr("height", function(d) {
return d;
});
```

El cambio que hacemos en la "y" no es para decirle que «dibuje» hacia arriba, sino que desplazamos la "y" para que, cuando empiece a dibujar, se queden todos alineados en el eje virtual de las "x". Pero el dibujo sigue estando hacia abajo.

Añadiendo el atribute *fill*, le podemos dar un poco de **color**:

```
.attr("fill", "teal");
```

Hasta le podemos poner **gradiente**:

```
.attr("fill", function(d) {
return "rgb(0, 0, " + (d * 10) + ")";
})
```

Y ahora pondremos el **valor encima de las barras**, añadiendo el valor que le toca:

```
svg.selectAll("text")
.data(dataset)
.enter()
.append("text")
.text(function(d) {
return d;
}
.attr("x", function(d, i) {
return i * (w / dataset.length);
})
.attr("y", function(d) {
```

```
return h - (d);
});
```

El resultado final sería algo tal que así:

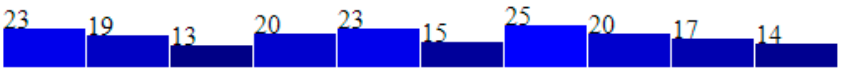


Figura 8. Barras aleatorias en D3 con SVG, gradiente de color y valor.

Scatter Plot

¿Cómo podríamos hacer un Scatter Plot de manera similar? Trabajaremos con otro tipo de *array*:

```
var dataset = [
[5, 20], [480, 90], [250, 50], [100, 33], [330, 95], [410, 12], [475, 44],
[25, 67], [85, 21], [220, 88]
];
```

La función es muy parecida a la que utilizamos para el *Bar Chart*:

```
svg.selectAll("circle") // <-- Cambiamos donde antes ponía "rect"
.data(dataset)
.enter()
.append("circle")
.attr("cx", function(d) {
return d[0];
})
.attr("cy", function(d) {
return d[1];
})
.attr("r", 5);
```

Y ya deberíamos tener un resultado similar a este:

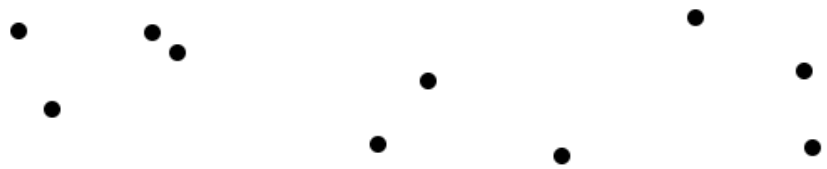


Figura 9. *Scatter Plot*.

Evidentemente, nos queda aún dibujar los ejes y demás elementos, pero todo es comenzar.

Para ejercitar lo visto este tema, se recomienda hacer este proceso con un archivo CSV y otro JSON para ambos casos. Si tienes problemas con los navegadores en local, que no permiten acceder a ficheros externos, introduce los datos en el código, aunque no sea una situación tan real como cuando tenemos a nuestra disposición un servidor donde publicar nuestros gráficos.

Por último, recomendamos los siguientes tutoriales prácticos:

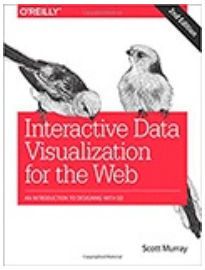
- w3schools.com | SVG Tutorial: https://www.w3schools.com/graphics/svg_intro.asp
- W3C | SCALABLE VECTOR GRAPHICS (SVG): https://www.w3.org/Graphics/SVG/
- DesarrolloWeb.com | Qué es SVG: https://desarrolloweb.com/articulos/que-es-svg.html

4.5. Referencias bibliográficas

Brainy Quote. *Confucius Quotes*. https://www.brainyquote.com/quotes/confucius_104563

Interactive Data Visualization for the Web

Murray, S. (2017). *Interactive Data Visualization for the Web.* [S. l.]: O'Reilly.



Para este tema son interesante las secciones del libro que contienen información sobre los SVGs y explican cómo dibujar un *Bar Chart* y un *Scatter Plot*.

Teaching an old dog new tricks. How data visualization & design can be used by everyone

Morris, C. y Wocknitz, S. (2013). Teaching an old dog new tricks. How data visualization & design can be used by everyone. *Proceedings of Annual Conference 2013*. http://www.samra.co.za/wp-content/uploads/2013/05/Wocknitz-Morris_Teaching-an-old-dog-new-tricks-Research-Paper.pdf

En este documento se muestra cómo la información puede ser mucho más accesible a través de la visualización y el diseño.

Film Brazil

Pommella, D. y Carelli, R. (2013). *Film Brazil* [Vídeo]: Vimeo. https://vimeo.com/68577610

Un vídeo, ejemplo de inspiración, sobre cómo utilizar infografía.

Test

1. ¿Podemos utilizar clases CSS en SVG?

A. Sí.

B. No, no son compatibles con HTML.

C. No, no son necesarias.

D. No, porque D3 no incorpora clases CSS.

2. Si queremos dibujar una elipse, ¿debemos utilizar el SVG elemento de e indicar que el radio es variante?

A. Sí.

B. No, requiere el *elemento* y cuatro parámetros: cx, cy, rx, ry.

C. No, requiere el *elemento* y dos parámetros: cx, cy.

D. No, requiere el *elemento* y tres parámetros: cx, cy, r.

3. Para dibujar una línea debemos definir los puntos de partida y de fin, y qué tipo de línea queremos que les una:

A. Sí.

B. No.

C. No porque depende de si queremos líneas continuas o a trazos.

D. No existen líneas en D3.

4. Si ponemos el valor de la altura en negativo cuando vamos a dibujar un rectángulo, ¿el rectángulo se dibujará hacia arriba?

A. Sí, porque valores positivos apuntan hacia abajo y negativos hacia arriba.

B. No.

C. Sí, porque las coordenadas empiezan en 0,0.

D. Sí, porque las coordenadas tienen su origen en el margen superior izquierdo.

5. Cuando dibujamos con SVG, ¿qué diferencia hay entre cx y x?

A. cx indica el centro del círculo y x indica el comienzo a dibujar, por ejemplo, en un rectángulo la esquina superior izquierda.

B. cx indica el comienzo a dibujar, por ejemplo, en un rectángulo la esquina superior izquierda, y x indica el centro del círculo.

C. Ninguna.

D. No existen el parámetro x.

6. ¿Qué función hemos de utilizar para cargar un archivo CSV?

A. D3.loadcsv.

B. D3.csv.

C. D3.loadercsv.

D. D3.csvload.

7. ¿Qué función hemos de utilizar para cargar un archivo JSON?

A. D3.jso

B. D3.loadjson

C. D3.loaderjson

D. D3.jsonload

8. ¿Qué es lo que hace este código?

```
var dataset = [];
for (var i = 0; i < 50; i++) {
var newNumber = Math.random() * 30;
dataset.push(newNumber);
}
```

A. Nos genera un *array* de 50 números del 0 al 30.

B. Nos genera un *array* de 30 números del 0 al 50.

C. Dibuja 50 números al azar.

D. Dibuja 30 números del 0 al 50.

9. ¿Qué nos dibujará el siguiente código?

```
var dataset = [ 5, 10, 15, 20, 25 ];
svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("x", function(d, i) {
return i * (w / dataset.length);
})
.attr("y", function(d) {
return h - d;
})
.attr("width", w / dataset.length - 1)
.attr("height", function(d) {
return d;
});
```

A. Cinco barras con color gradiente azul con el texto encima de las barras.

B. Diez barras con color gradiente azul con el texto encima de las barras.

C. Cinco barras con la dirección invertida.

D. Ninguna de las anteriores.

10. ¿Qué nos dibujará el siguiente código?

```
var dataset = [
[5, 20], [480, 90], [250, 50], [100, 33], [330, 95],
[410, 12], [475, 44], [25, 67], [85, 21], [220, 88]
]

svg.selectAll("circle")
.data(dataset)
.enter()
.append("circle")
.attr("cx", function(d) {
return d[0];
})
.attr("cy", function(d) {
return d[1];
})
.attr("r", 5);
```

A. Diez círculos con radio 5 en posición *random* en nuestro SVG.

B. Nueve círculos con radio 5 en posición definida por el dataset.

C. Una gráfica *Scatter Plot* con diez barras definidas por el dataset.

D. Ninguna de las anteriores.