IIC2026
Visualización de Información

Hernán F. Valdivieso López
(2022 - 2 / Clase 13)

Antes de empezar... Revisión de contenidos (RC)

- ¡No se olviden de las actividades pendientes!
  - Hoy se finaliza el control sobre abstracción de tareas.
  - El otro martes finaliza la actividad para crear pregunta de Verdadero/Falso.
- Se acaba de publicar un mini control de alternativas **obligatorio** (no bonus) en Canvas sobre lo que **veremos en la clase de hoy**.
  - **Duración**: 2 semanas para realizarlo a partir de hoy.
  - **Intentos para responder**: ilimitados.
  - **Extensión**: 5 preguntas de 1 punto c/u.
  - **Condición para obtener el punto RC**: Al menos 4 puntos de 5.
  - Cada vez que respondan, verán el puntaje total logrado, pero no cuáles preguntas están correctas e incorrectas.

Temas de la clase - Layouts tabulares en D3

1. Profundización en diferentes tipos de escalas.
2. Uso de símbolos y líneas.
3. Creación de áreas.
4. Uso de disposición radial y creación de arcos.

Profundización en diferentes tipos de escalas

Profundización en diferentes tipos de escalas

- En clases pasadas vimos:
  - `scaleLinear` para datos cuantitativos.
  - `scaleBand` para datos categóricos.
- Vamos a ver otros tipos de escalas para datos cuantitativos.
- Vamos a profundizar en algunos parámetros de `scaleBand` y veremos otra escala más: `scalePoint`.

Profundización en diferentes tipos de escalas

Escalas para datos cuantitativos

- `scaleLinear`: escala lineal. Genera una función **`y=m*x+b`**.
  - Es la escala más intuitiva para el humano.
- `scaleLog`: escala logarítmica. Genera una función **`y=m*log(x)+b`.**
  - Ideal cuando un dato muy grande provoca que los datos pequeños no se vean bien.
- `scalePow`: escala potencia. Genera una función **$y=m*x^k+b$.** Donde k es un parámetro que nosotros indicamos.
  - Podemos construir una escala de raíz cuadrada con este comando. Ideal cuando trabajamos con el tamaño de círculos.

Profundización en diferentes tipos de escalas

Vamos al código 👨‍💻 👩‍💻

Profundización en diferentes tipos de escalas

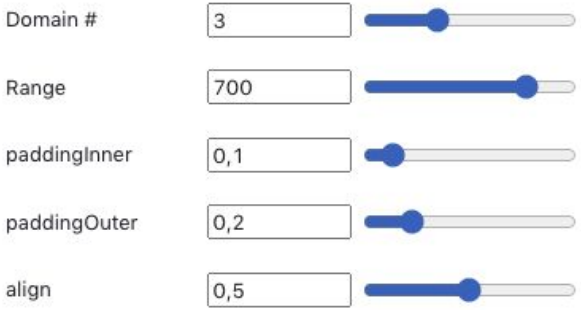
Escalas para datos categóricos

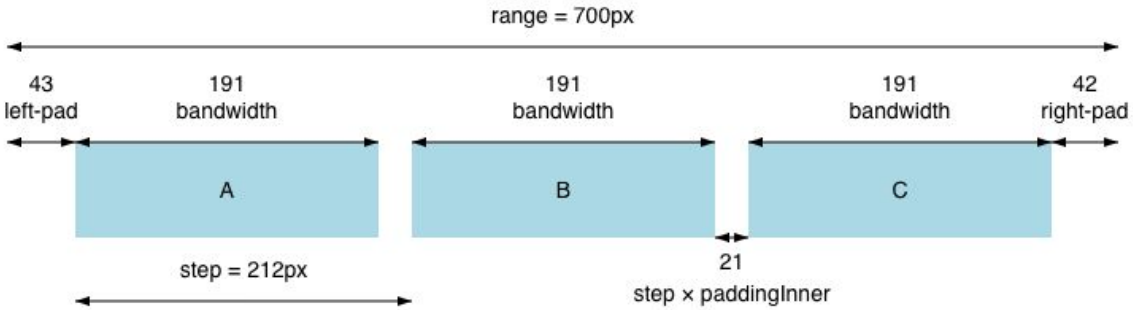
- `scaleBand`: genera una banda por categoría. El tamaño de dicha banda se calcula automáticamente.
    - Ideal para trabajar con gráfico de barras.
    - Parámetros `paddingInner`, `paddingOuter` y `align` permiten personalizar la escala.
- `scalePoint`: genera posiciones equidistantes entre cada categoría.
    - Ideal para trabajar con gráfico de puntos (no confundir con gráfico de dispersión).
    - Parámetros `padding` y `align` permiten personalizar la escala.

Profundización en diferentes tipos de escalas

Escalas para datos categóricos - scaleBand

- `paddingInner`: setea el espacio entre bandas.
- `paddingOuter`: setea los espacios externos.
- `align`: traslada las bandas a la derecha o izquierda. 0.5 para que esté centrado.
- Revisar este enlace: d3.scaleBand.



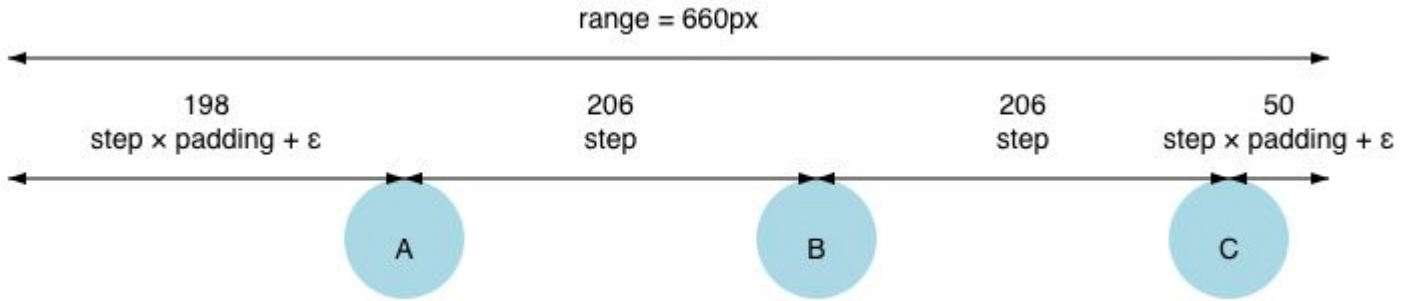


Profundización en diferentes tipos de escalas

Escalas para datos categóricos - scalePoint

- `padding`: setea los espacios externos.
- `align`: traslada las bandas a la derecha o izquierda. 0.5 para que esté centrado.
- Revisar este enlace: d3.scalePoint.

```
config = ({
  domain: ["A", "B", "C"], // 👀 change me!
  padding: 0.6, // 👀
  round: true, // 👀
  range: [40, Math.min(700, width - 40)] // 👀
})
```



Profundización en diferentes tipos de escalas

Vamos al código 👨‍💻 👩‍💻

Profundización en diferentes tipos de escalas

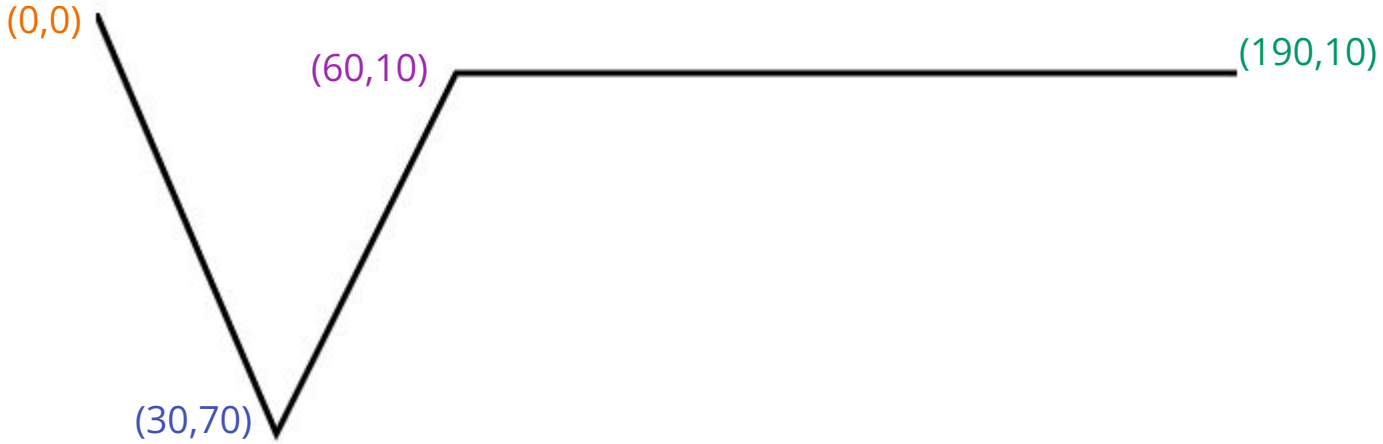
- 🤔 ¿Existe una escala para definir un *colomaps* cuantitativo ?
- 🤔 ¿Existe una escala para transformar categorías en colores?
- 🤔 Si digo que el dominio es `[100, 200]` ... y me llega un dato que es 250 ¿Puedo ajustar lo que sucederá en la misma escala?
- 🤔 ¿Si quiero una escala para datos temporales?
    - Todas esas respuestas en: Scale functions | D3 in Depth
    - Se recomienda **fuertemente** revisar dicho enlace en casa.
    - Si no tiene tiempo, guardenla en favoritos. Les ayudará en la tarea y examen.

Uso de símbolos y líneas

Uso de símbolos y líneas

- Hasta ahora las visualizaciones han sido de cuadrados y círculos.
- D3 provee un montón de facilidades para generar marcas de forma fácil y efectiva a partir de datos. Esto lo logra usando `<path></path>` y construyendo strings con el formato adecuado para el atributo d

```
<path d="M0,0L30,70L60,10L190,10" fill="none" stroke="black"></path>
```



Uso de símbolos y líneas

Símbolos

- `d3.symbol()` nos provee de un string para llenar el atributo “d” y crear diferentes figuras.

```
const simbolo = d3.symbol().size(30 * 30);

console.log(simbolo());
```

```
M16.925687506432688,0A16.925687506432688,16.925687506432688,0,1,1,-16.925687506432688,0A16.925687506432688,16.925687506432688,0,1,1,16.925687506432688,0
```

Uso de símbolos y líneas

Símbolos

```
const simbolo = d3.symbol().size(30 * 30);
simbolo.type(d3.symbolCross);
console.log(simbolo());
```

```
const simbolo = d3.symbol().size(30 * 30);
simbolo.type(d3.symbolStar);
console.log(simbolo());
```

```
const simbolo = d3.symbol().size(30 * 30);
simbolo.type(d3.symbolSquare);
console.log(simbolo());
```

¡Ahora podemos hacer un gráfico de dispersión cambiando la forma de cada punto!

Uso de símbolos y líneas

Símbolos - ¿cómo dibujarlo en D3?

```
const svg = d3
  .select("body")
  .append("svg")
  .attr("width", 1000)
  .attr("height", 200);

const simbolo = d3.symbol().size(30 * 30);
simbolo.type(d3.symbolCross);

svg
  .append("path")
  .attr("d", simbolo())
  .attr("transform", "translate(100, 100)");
```

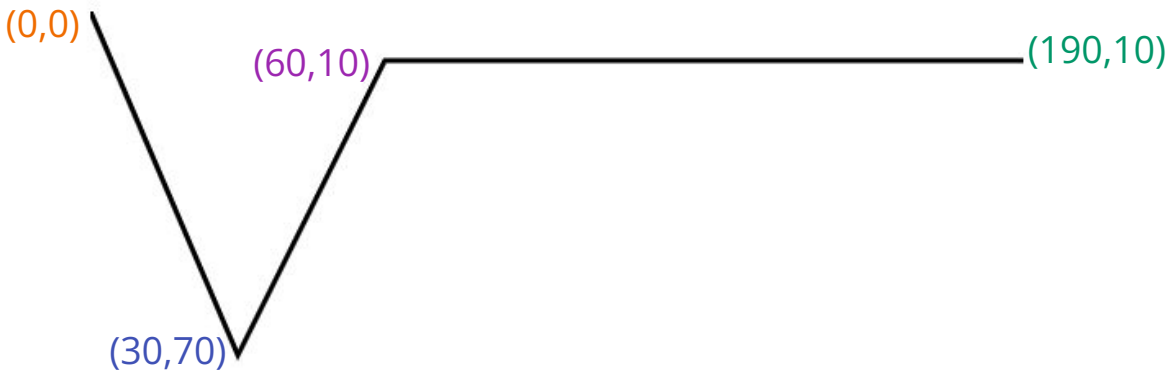
Uso de símbolos y líneas

Líneas

- `d3.line()` nos provee de un string para llenar el atributo “d” y crear líneas con la forma que necesitemos. Solo necesita de una lista de coordenadas (x,y) para ir guiando la creación de la línea.

```
const puntos = [[0, 0], [30, 70], [60, 10], [190, 10]];
const linea = d3.line();
console.log(linea(puntos));
```

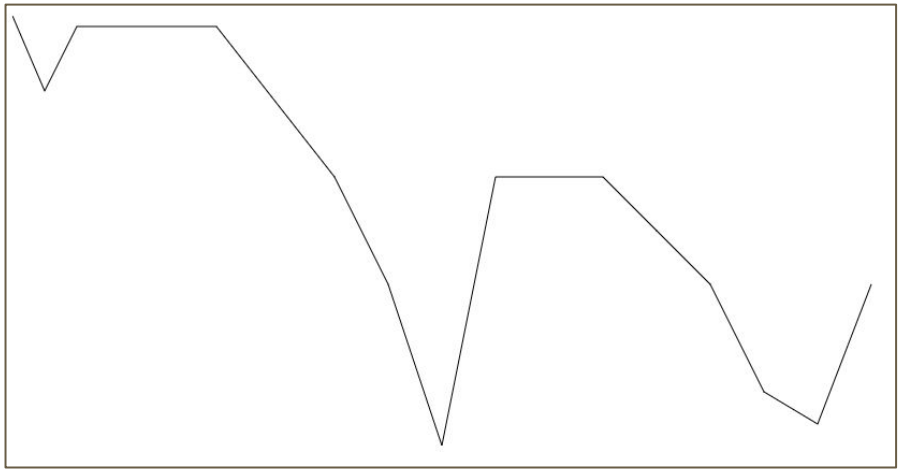
```
> M0,0L30,70L60,10L190,10
```



Uso de símbolos y líneas

Líneas - ¿cómo dibujarlo en D3?

```
const svg = d3
  .select("body")
  .append("svg")
  .attr("width", 1000)
  .attr("height", 200);

const puntos = [[0, 0], [30, 70], [60, 10], [190, 10]];
const linea = d3.line();

svg
  .append("path")
  .attr("fill", "transparent")
  .attr("stroke", "black")
  .attr("d", linea(puntos));
```

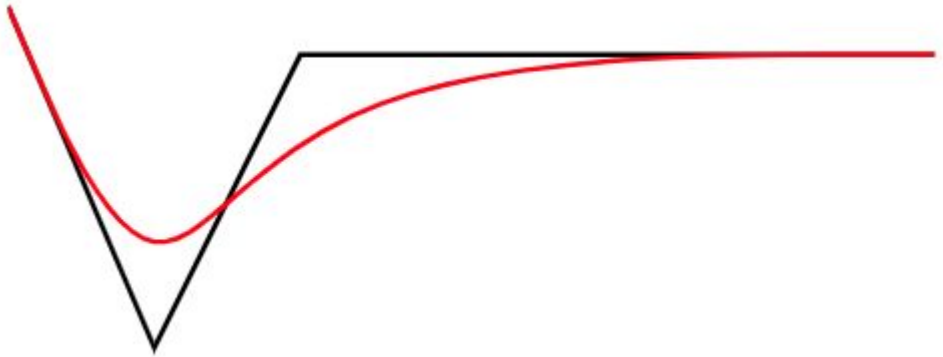


Uso de símbolos y líneas

Líneas

- Por defecto, `d3.line` crea líneas rectas para unir cada punto. Pero si usamos el método `curve`, podemos darle una curvatura con diferentes propiedades.

```
const puntos = [[0, 0], [30, 70], [60, 10], [190, 10]];
const linea = d3.line();
svg
  .append("path")
  .attr("fill", "transparent")
  .attr("stroke", "black")
  .attr("d", linea(puntos));
linea.curve(d3.curveBasic)
svg
  .append("path")
  .attr("fill", "transparent")
  .attr("stroke", "red")
  .attr("d", linea(puntos));
```



Uso de símbolos y líneas

Líneas

- Existen muchos tipos de curvas: [D3 Curve Explorer]

Uso de símbolos y líneas

Líneas

- 🤔 ¿Cómo combinar escalas y generador de líneas para transformar mis datos en coordenadas?
- `d3.line()` tiene 2 métodos: `x(...)` e `y(...)` que permiten personalizar como procesar el dato que será la coordenada X y la coordenada Y de cada punto respectivamente.

```
const datos = [
  { paso: 0, valor: 2.0603572936394787 }, { paso: 1, valor: 2.1258340075136997 },
]
const lineConEscalas = d3.line()
  .curve(d3.curveLinear)
  .x((d) => escalaX(d.paso)) // en el código veremos esta escala
  .y((d) => escalaY(d.valor)); // en el código veremos esta escala

lineConEscalas(datos)
```

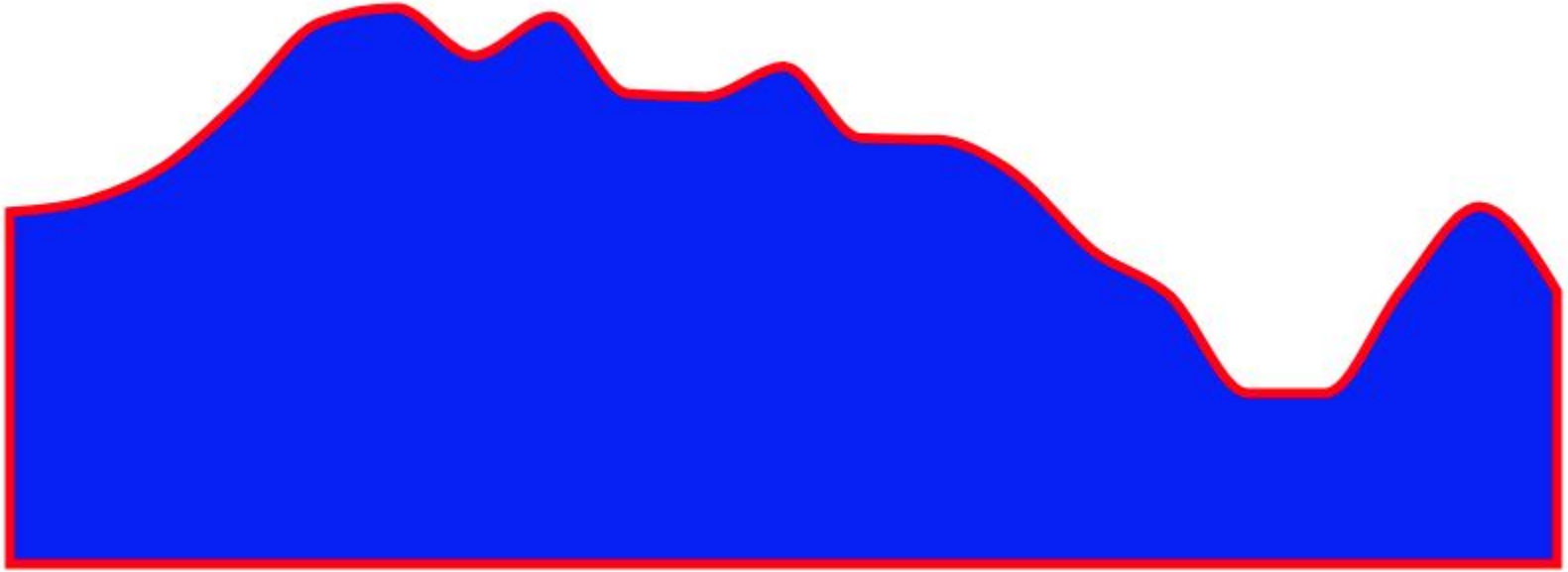
Uso de símbolos y líneas

Vamos al código 👨‍💻 👩‍💻

Creación de áreas

Creación de áreas

- `d3.area()` es un método para generar 2 líneas simultáneamente y pinta el área entre líneas.
- Útil cuando deseamos crear gráficos de área, streamgraph, barras apiladas, entre otros.



Creación de áreas

- `d3.area()` es un método para generar 2 líneas simultáneamente y pinta el área entre líneas.
- Se basa en definir 4 métodos `x0(...)`, `y0(...)`, `x1(...)` e `y1(...)`.
- Los `(x0,y0)` corresponden a la coordenada de la línea 0 y los `(x1,y1)` para la línea 1.
- Si `x0` y `x1` van a ser iguales, se puede usar `x(...)` para no duplicar código. Lo mismo ocurre para `y0` e `y1` que puedes usar `y(...)`.
- Se ocupa igual que `d3.line`. Solo basta llamar a la función para obtener un string con los datos para llenar el "d" de un `<path></path>`

Creación de áreas

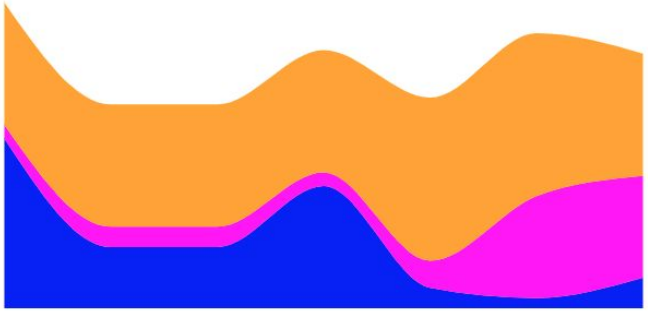
```
const datos = [
  { paso: 0, valor: 2.0603572936394787 }, { paso: 1, valor: 2.1258340075136997 },
]

const area = d3.area()
  .x(HEIGHT - 10) // x0 y x1 serán iguales
  .y0((d) => escalaX(d.paso));
  .y1((d) => escalaY(d.valor));

svg
  .append("path")
  .attr("fill", "blue")
  .attr("stroke", "red")
  .attr("stroke-width", 5)
  .attr("d", area(datos));
```

Creación de áreas

Vamos al código 👨‍💻 👩‍💻

Creación de áreas

- Las áreas suelen utilizarse en el contexto de gráficos de áreas apiladas.



- 😓 Esto requiere preprocesar datos, especialmente sumar valores unos detrás de otros para determinar luego las alturas correspondientes de las líneas que generan las áreas.
- D3 ya viene con funciones para facilitar este proceso: **`d3.stack()`**.
- Esta función espera recibir un arreglo de datos, y a partir de ellos generará una serie de sumas de valores de forma acumulada.

Creación de áreas

`d3.stack()`

- Se utiliza el método `keys([...])` para indicar las categorías a apilar y su orden.

```
const datos = [
  { dia: "Lunes", trabajar: 500, ver_anime_manga: 40, dormir: 360 },
  { dia: "Martes", trabajar: 180, ver_anime_manga: 60, dormir: 360 },
  { dia: "Miércoles", trabajar: 180, ver_anime_manga: 60, dormir: 360 },
  { dia: "Jueves", trabajar: 360, ver_anime_manga: 40, dormir: 360 },
  { dia: "Viernes", trabajar: 60, ver_anime_manga: 80, dormir: 480 },
];
const apilador = d3.stack().keys(["trabajar", "ver_anime_manga", "dormir"]);
const series = apilador(datos);
console.log(series)
```

```
0: (5) [Array(2), Array(2), …, key: 'trabajar', index: 0]
1: (5) [Array(2), Array(2), …, key: 'ver_anime_manga', index: 1]
2: (5) [Array(2), Array(2), …, key: 'dormir', index: 2]
```

Elemento 0 de cada lista (lunes):

```
0: (2) [0, 500, data: {…}]
0: (2) [500, 540, data: {…}]
0: (2) [540, 900, data: {…}]
```

Creación de áreas

Vamos al código 👨‍💻 👩‍💻

Uso de disposición radial y creación de arcos

Uso de disposición radial y creación de arcos

- D3 provee de `lineRadial()` y `areaRadial()` para crear líneas y áreas de con disposición circular.
- No son muy diferentes a `line()` y `area()`. Solo que en vez de usar coordenadas (x,y) funcionan a base de radio y ángulos.
- Las distancias horizontales (**eje X**) ahora se trabajan como **distancias angulares**. Necesitamos una escala cuyo `range` sea entre 0 y 2 Pi.
- Las distancias verticales (**eje Y**) ahora se trabajan como **distancias radiales**. Necesitamos una escala cuyo `range` sea entre 0 (centro del círculo) y R (radio máximo del círculo).

Uso de disposición radial y creación de arcos

`lineRadial()`

- El método `.angle(...)` permite procesar como se trabaja el ángulo de cada punto.
- El método `.radios(...)` permite procesar como se trabaja el radio de cada punto.
- Se ocupa igual que `d3.line`. Solo basta llamar a la función para obtener un string con los datos para llenar el "d" de un `<path></path>`

Uso de disposición radial y creación de arcos

```
const datos = [
  { paso: 0, valor: 2.0603572936394787 }, { paso: 1, valor: 2.1258340075136997 },
]

const lineaRadial = d3.lineRadial()
  .curve(d3.curveNatural)
  .angle((d) => escalaAngulo(d.paso))
  .radius((d) => escalaRadio(d.valor));

svg
  .append("path")
  .attr("fill", "transparent")
  .attr("stroke", "blue")
  .attr("stroke-width", 2)
  .attr("d", lineaRadial(datos));
```

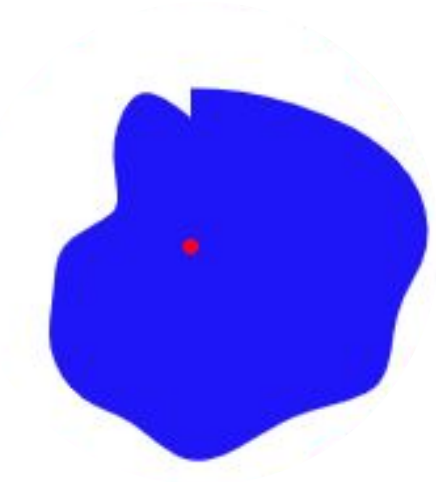
Uso de disposición radial y creación de arcos

`areaRadial()`

- El método `.areaRadial(...)` permite procesar cómo se trabaja el ángulo del área. Aquí **no hay** `areaRadial0` y `areaRadial1`, solo se define 1 vez el ángulo para ambas líneas.
- El método `.innerRadius(...)` permite procesar como se trabaja el radio **interno** del área.
- El método `.outerRadius(...)` permite procesar como se trabaja el radio **externo** del área.
- Se ocupa igual que d3`.area`. Solo basta llamar a la función para obtener un string con los datos para llenar el "d" de un `<path></path>`

Uso de disposición radial y creación de arcos

```
const datos = [
  { paso: 0, valor: 2.0603572936394787 }, { paso: 1, valor: 2.1258340075136997 },
]

const lineaRadial = d3.areaRadial()
  .curve(d3.curveNatural)
  .angle((d) => escalaAngulo(d.paso))
  .innerRadius(0)
  .outerRadius((d) => escalaRadio(d.valor));

svg
  .append("path")
  .attr("fill", "blue")
  .attr("stroke", "blue")
  .attr("stroke-width", 2)
  .attr("d", areaRadial(datos));
```



Uso de disposición radial y creación de arcos

Vamos al código 👨‍💻 👩‍💻

Uso de disposición radial y creación de arcos

`d3.arc()`

- Comando auxiliar de D3 para facilitar la creación de arcos.
- Solo necesitamos determinar el ángulo de inicio, de fin, radio interior y exterior.
- El método `.padAngle(...)` permite aplicar un padding a la figura resultante.
- El método `.cornerRadius(...)` permite redondear las esquinas del arco.

```
const arcos = d3.arc().padAngle((2 * Math.PI) / 100).cornerRadius(5);

const dString = arcos({
    innerRadius: 0, outerRadius: 100,
    startAngle: 0, endAngle: (2 * Math.PI) / 3,
})
```

Uso de disposición radial y creación de arcos

`d3.arc()`

- El método `.innerRadius(...)` permite fijar el radio **interno** del arco.
- El método `.outerRadius(...)` permite fijar el radio **externo** del arco.

```
const arcos = d3.arc().padAngle((2 * Math.PI) / 100).cornerRadius(5);
arcos.innerRadius(50).outerRadius(75);

const dString = arcos({
    startAngle: 0, endAngle: (2 * Math.PI) / 3,
})
```

🤔 Si quiero hacer un gráfico de torta ¿tendré que calcular los ángulos de cada segmento?

Uso de disposición radial y creación de arcos

`d3.pie()`

- Preprocesa los datos para asignarles un ángulo de inicio y término a cada arco/segmento del círculo.

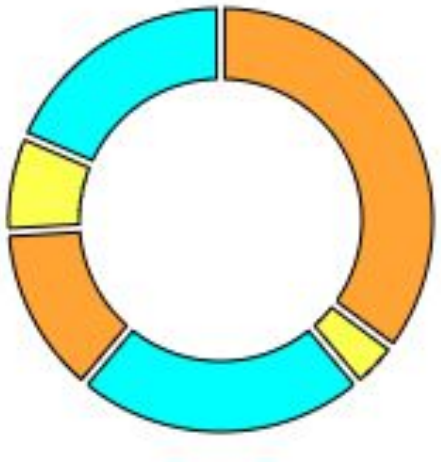
```
const datos = [
  { valor: 190, categoria: "orange" },
  { valor: 20, categoria: "yellow" },
  ...
];

const pie = d3.pie()
  .value((d) => d.valor)
  .sort(null);

const arcosCalculados = pie(datos);
console.log(arcosCalculados)
```

```
▼ Array(6) i
  ▼ 0:
    ▶ data: {valor: 190, categoria: 'orange'}
      endAngle: 2.210750385859484
      index: 0
      padAngle: 0
      startAngle: 0
      value: 190
    ▶ [[Prototype]]: Object
  ▼ 1:
    ▶ data: {valor: 20, categoria: 'yellow'}
      endAngle: 2.443460952792061
      index: 1
      padAngle: 0
      startAngle: 2.210750385859484
      value: 20
    ▶ [[Prototype]]: Object
```

Uso de disposición radial y creación de arcos

`d3.pie()`

- Preprocesa los datos para asignarles un ángulo de inicio y término a cada arco/segmento del círculo.



```
const arcosCalculados = pie(datos);
console.log(arcosCalculados)
```

```
const arcosPie = d3
  .arc()
  .innerRadius(50)
  .outerRadius(75)
  .padAngle((2 * Math.PI) / 200)
  .cornerRadius(0);

svg
  .selectAll("path")
  .data(arcosCalculados)
  .join("path")
  .attr("d", (d) => arcosPie(d))
```

Uso de disposición radial y creación de arcos

Vamos al código 👨‍💻 👩‍💻

Próximos eventos

Próxima clase

- Utilidades de D3 II (eventos y transiciones).
- Vamos a ver mucho código 💻 .

Ayudantía de mañana

- Cargar datos, uso de escalas y más :D

Tarea 2 📈 🖼️ 🌷

- Se publica este viernes.
- Evalúa contenidos de HTML, CSS, JS, SVG y d3 (hasta lo que veremos el jueves).
- La próxima semana haré ayudantía de la tarea. Al menos vengan con el enunciado leído 🥺 .

IIC2026
Visualización de Información

Hernán F. Valdivieso López
(2022 - 2 / Clase 13)