

Web Basics

Disclaimer: This is not a course on Javascript and web development!

- We'll focus only on what's needed to accomplish our visualization goals
- There's **a lot** that we won't cover

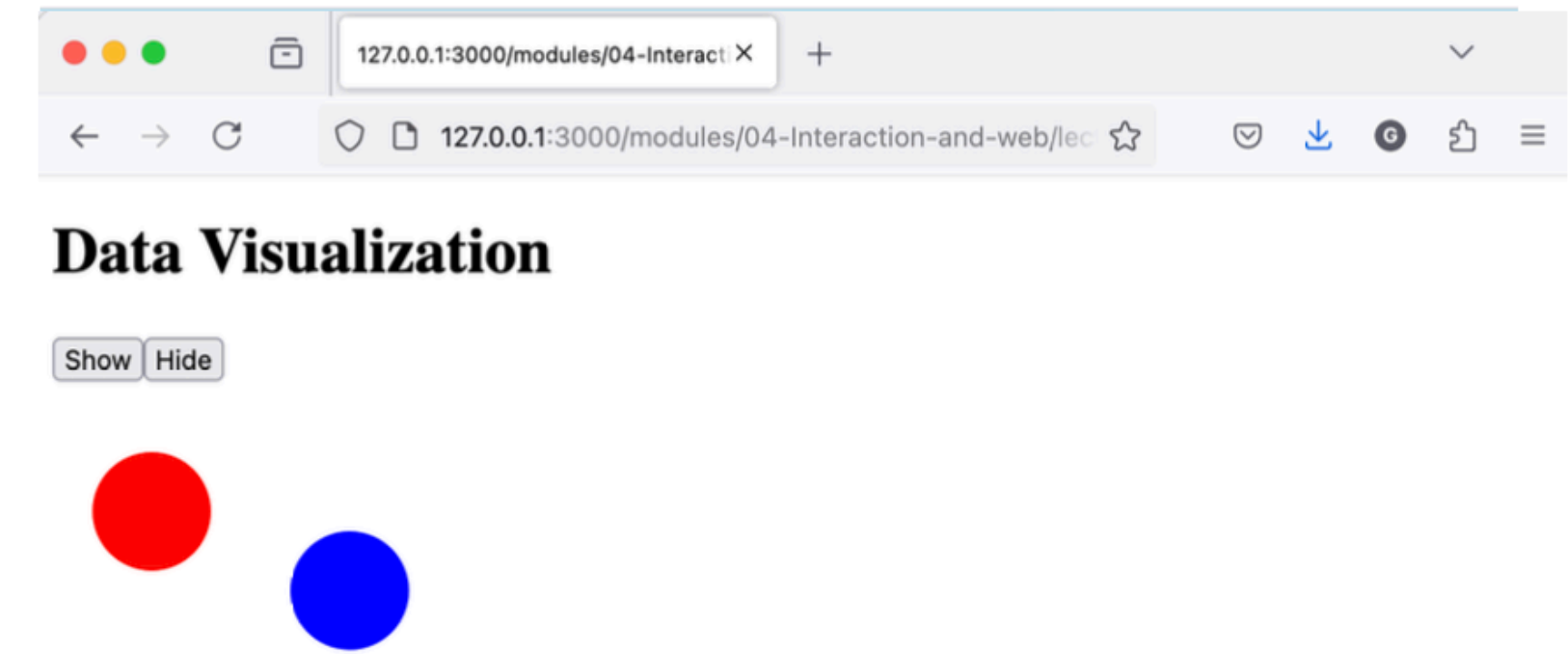
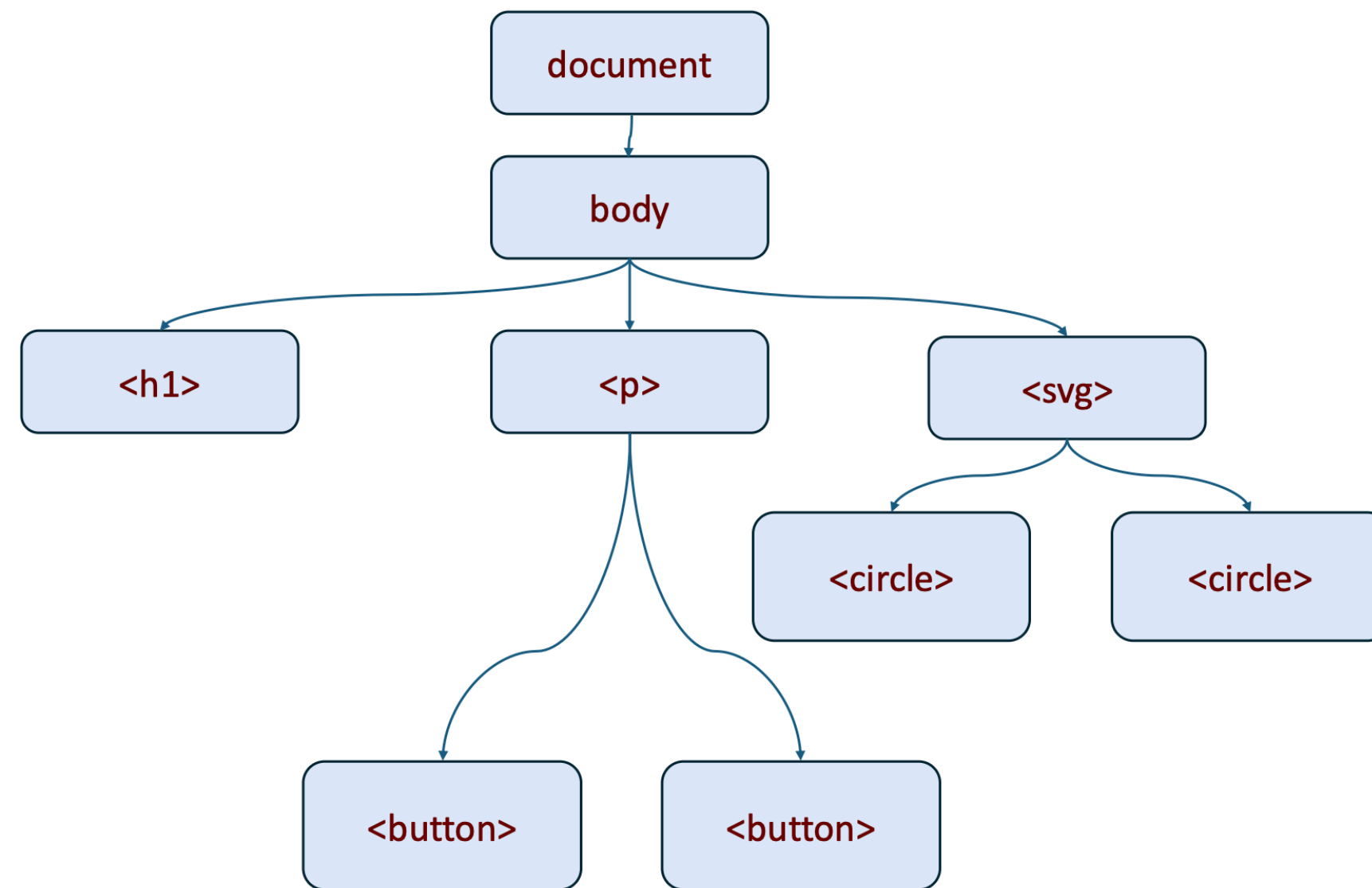
Anatomy of a web page

Hypertext Markup Language (HTML)

Document Object Model (DOM)

Browser display

```
<!DOCTYPE html>
<html>
<body>
  <h1>Data Visualization</h1>
  <p>
    <button>Show</button>
    <button>Hide</button>
  </p>
  <svg width="400" height="300">
    <circle id="redcircle" cx="50" cy="50" r="30" fill="red"></circle>
    <circle id="bluecircle" cx="150" cy="90" r="30" fill="blue"></circle>
  </svg>
</body>
</html>
```



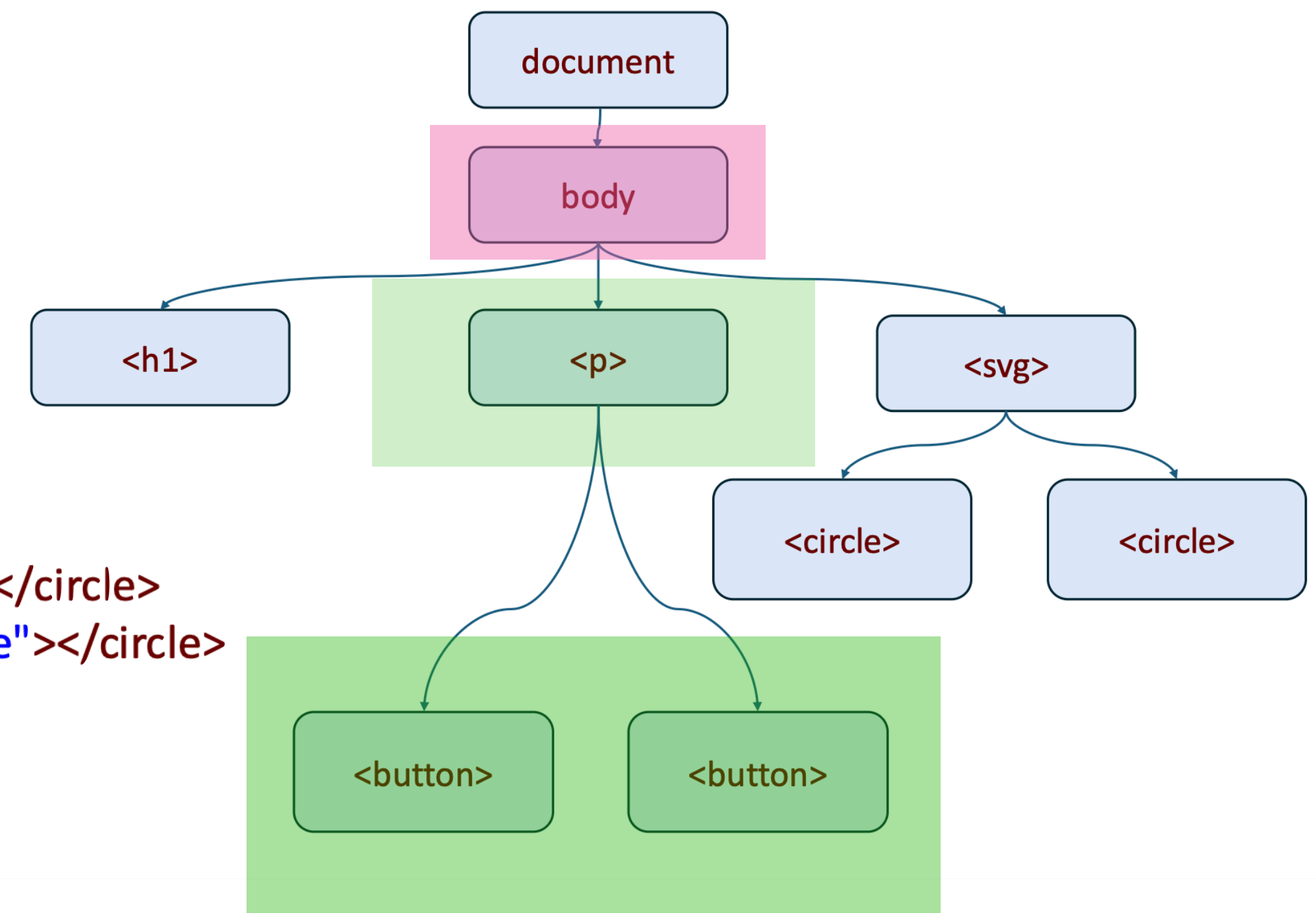
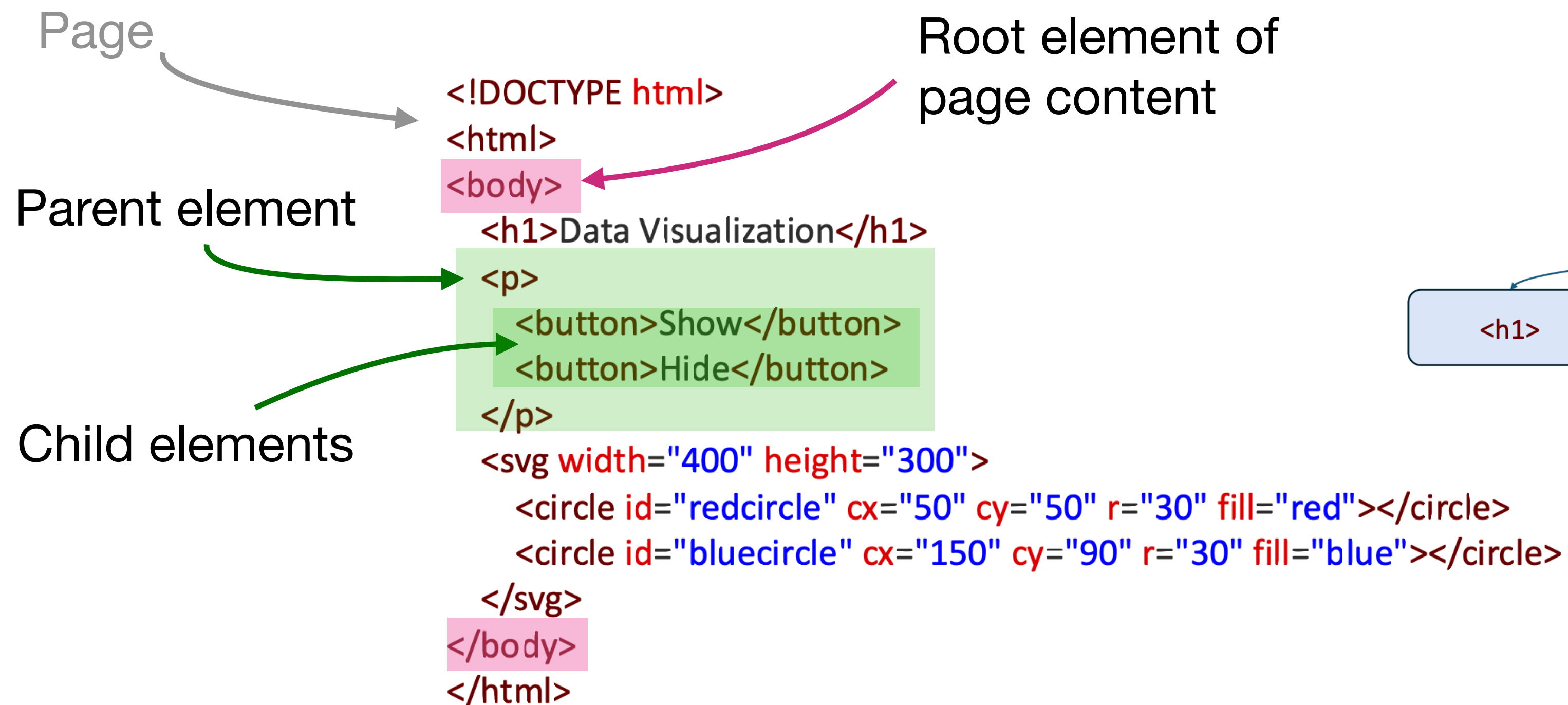
Parses to

Renders to

Tree-structured *elements*

Hypertext Markup
Language (HTML)

Document Object
Model (DOM)



Children contained between *start* and *end tags* of parent

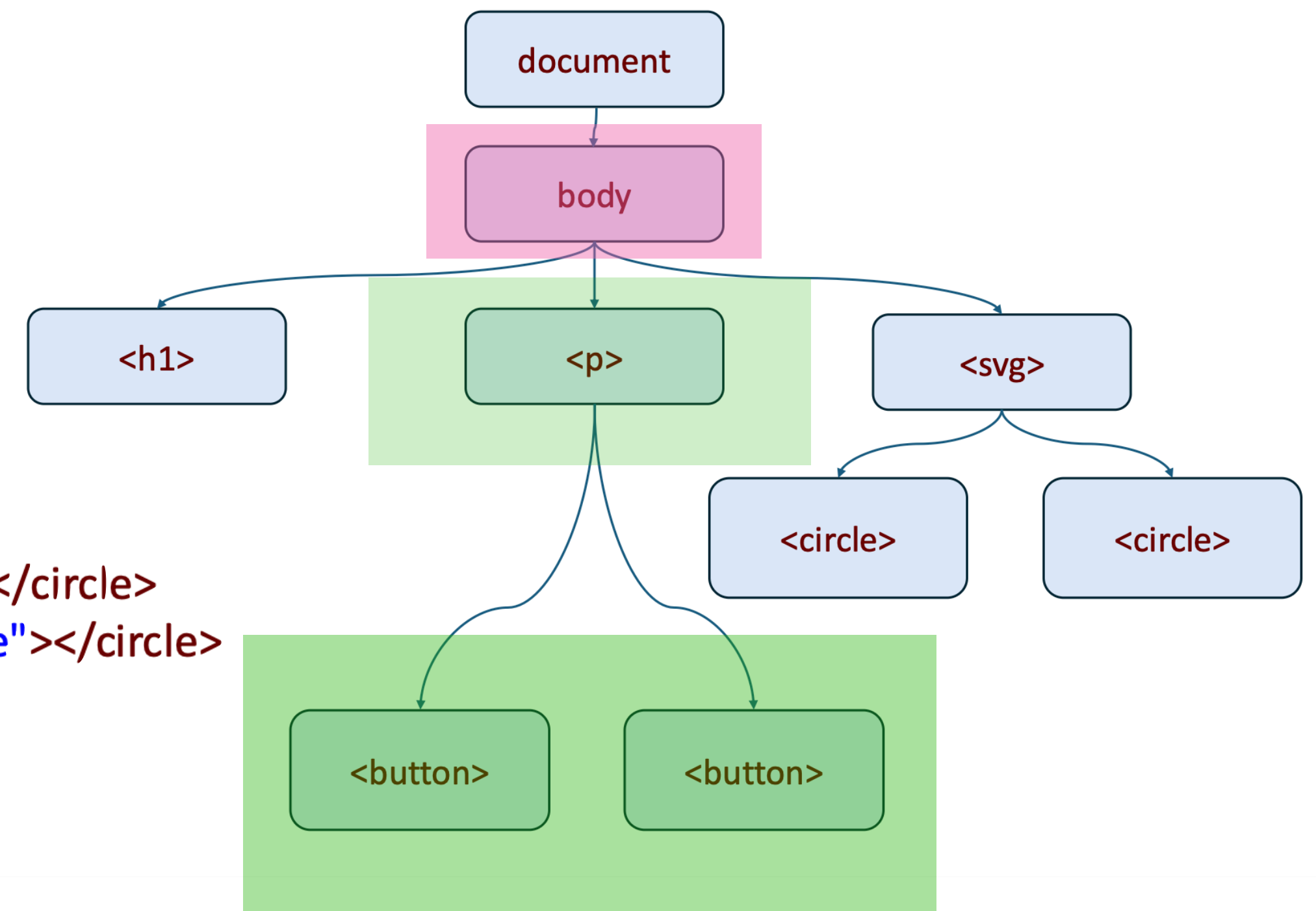
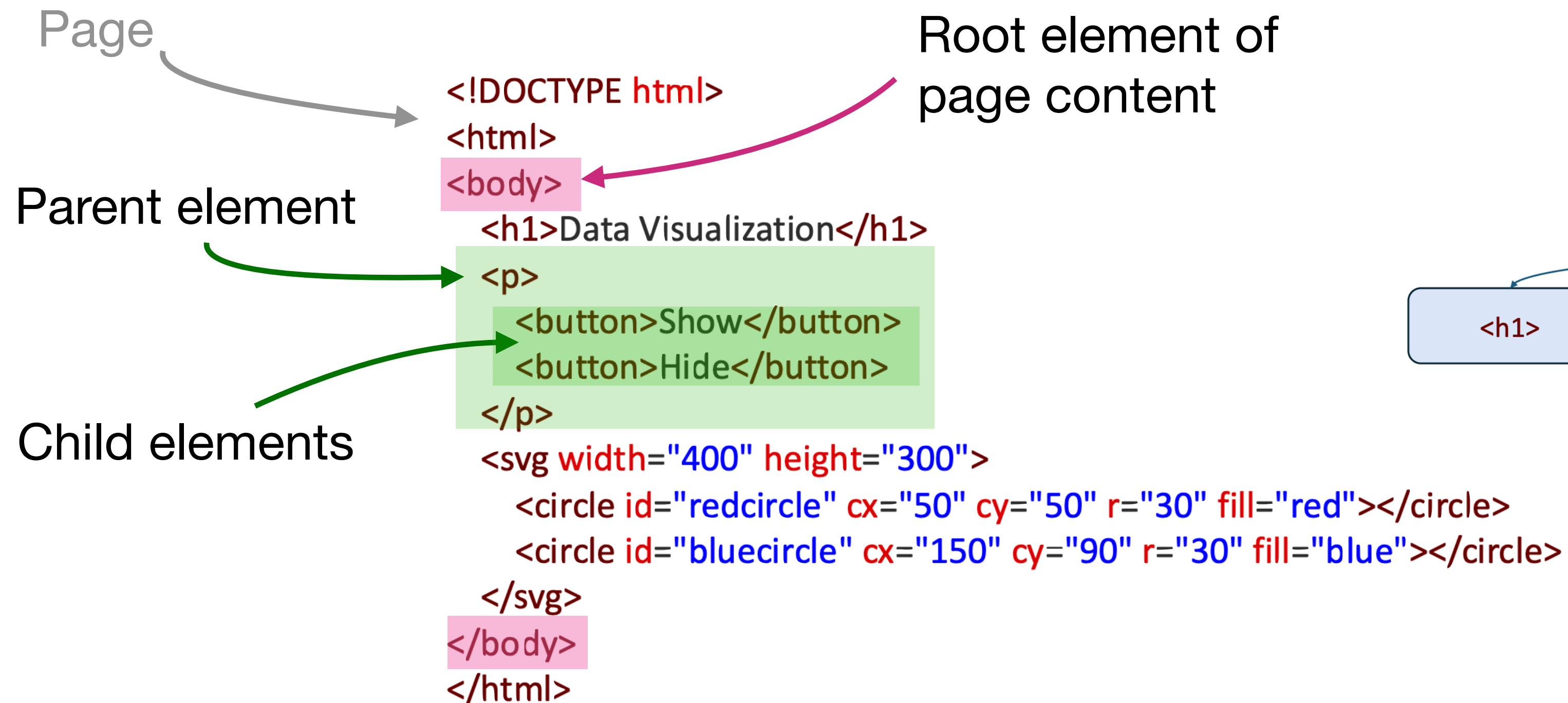
Start tag: `<p>`

End tag: `</p>`

A single element

Hypertext Markup Language (HTML)

Document Object Model (DOM)



Children contained between *start* and *end tags* of parent

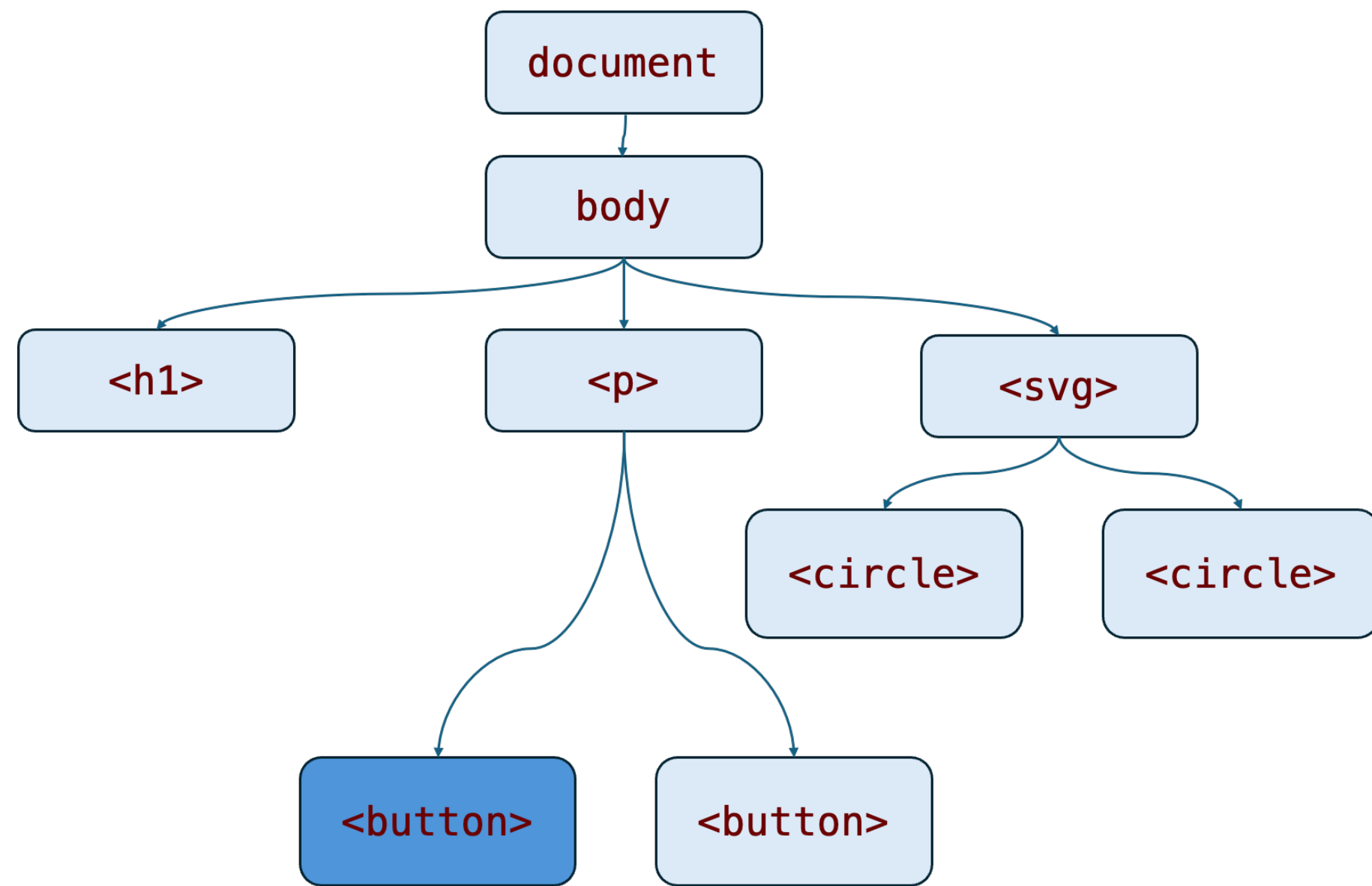
Start tag: `<p>`

End tag: `</p>`

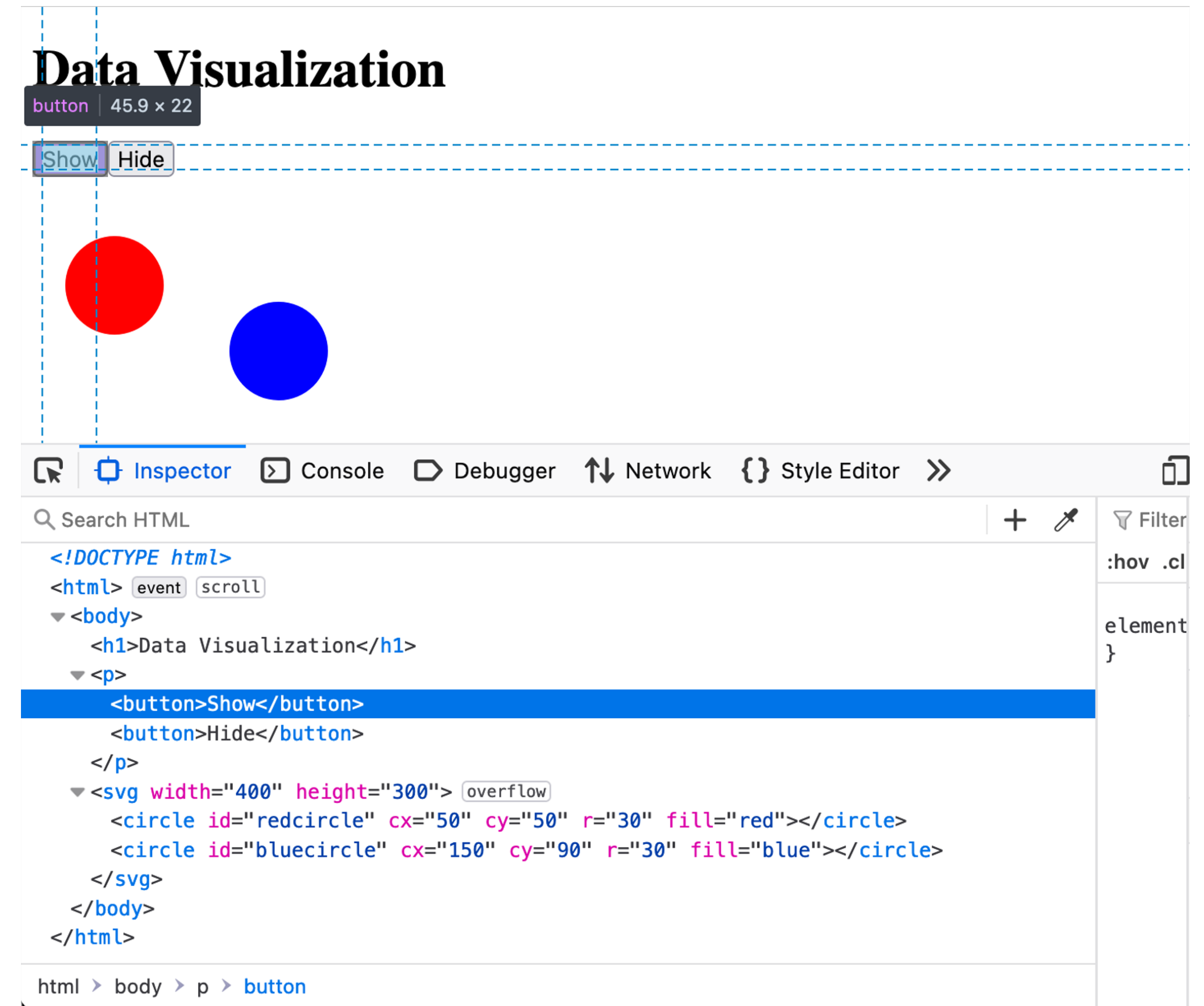
A single element

Document Object Model (DOM)

Browser display



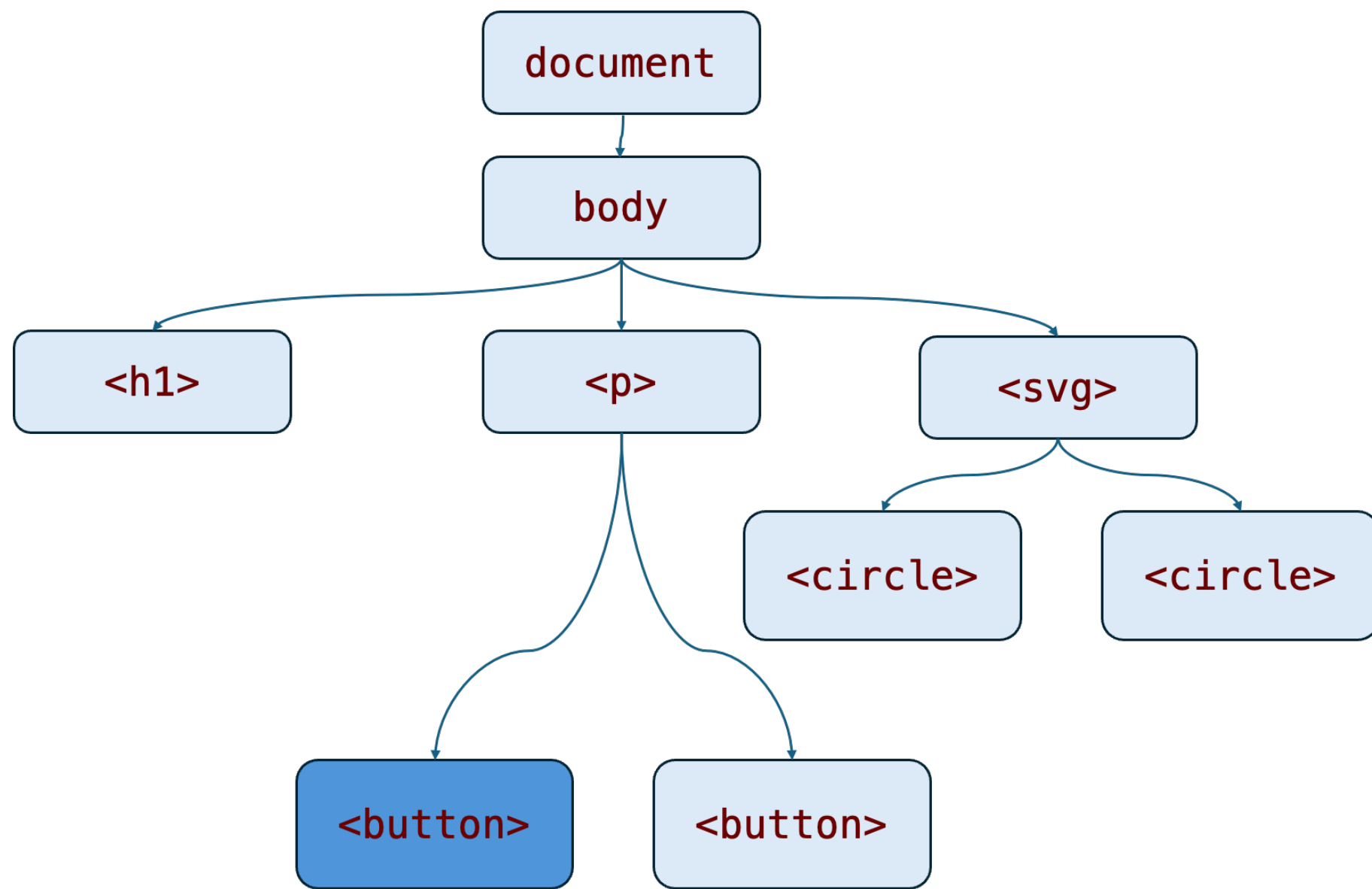
DOM Elements define the UI elements on the page: e.g. a **button**



Inspector: Browser's DOM visualization - shown as HTML

A single element

Document Object Model (DOM)



DOM Elements define the UI elements on the page: e.g. a **button**

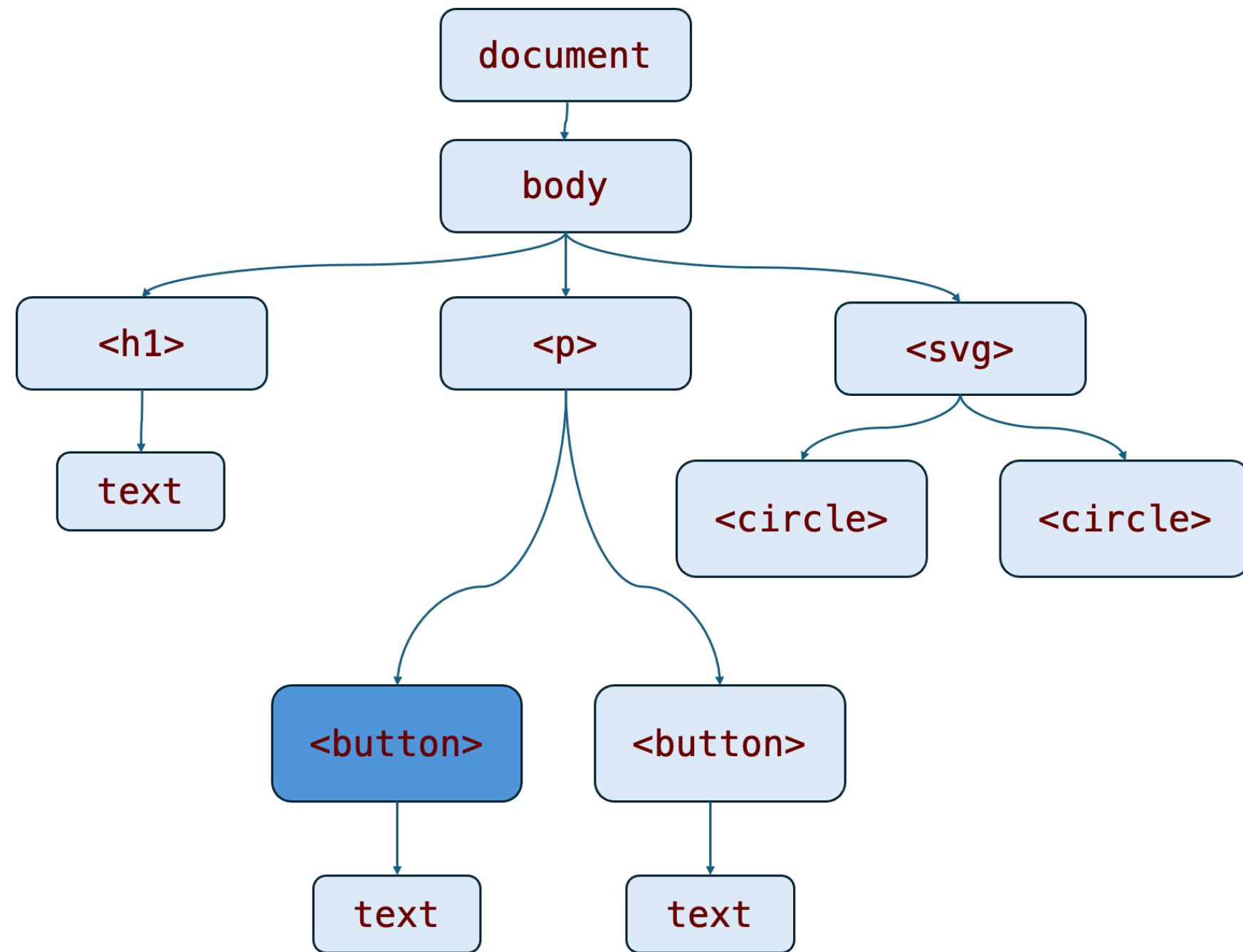
Browser display

Inspector: Browser's DOM visualization - shown as HTML

A single element

Document Object Model (DOM)

Browser display



Technically text elements (e.g. “show”) are also nodes in the tree!

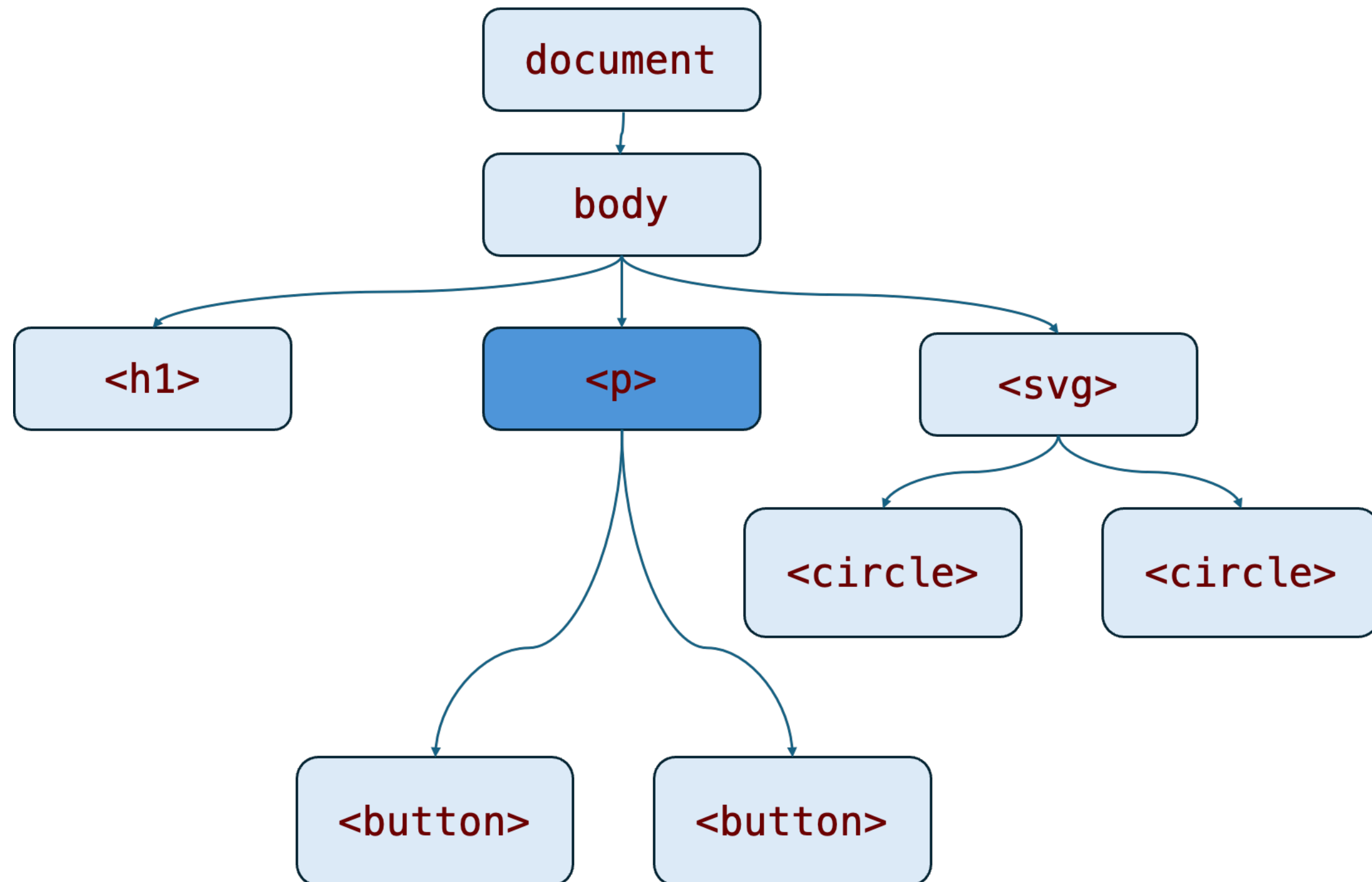
Omitted here for compactness

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Data Visualization</h1>
    <p>
      <button>Show</button>
      <button>Hide</button>
    </p>
    <svg width="400" height="300">
      <circle id="redcircle" cx="50" cy="50" r="30" fill="red"></circle>
      <circle id="bluecircle" cx="150" cy="90" r="30" fill="blue"></circle>
    </svg>
  </body>
</html>
```

Inspector: Browser's DOM visualization - shown as HTML

Container elements

Document Object Model (DOM)



“Container” elements define the organization of UI elements

`<p>...</p>` organizes its children as a paragraph

Browser display

Data Visualization

Group buttons into a paragraph

p 764 x 22

```
html > body > p
```

Common elements

Text Formatting Tags

```
<b>Bold Text</b>
<strong>This text is important</strong>
<i>Italic Text</i>
<em>This text is emphasized</em>
<u>Underline Text</u>
<pre>Pre-formatted Text</pre>
<code>Source code</code>
<del>Deleted text</del>
<mark>Highlighted text (HTML5)</mark>
<ins>Inserted text</ins>
<sup>Makes text superscripted</sup>
<sub>Makes text subscripted</sub>
<small>Makes text smaller</small>
<kbd>Ctrl</kbd>
<blockquote>Text Block Quote</blockquote>
```

Section Divisions

<code><div></div></code>	Division or Section of Page Content
<code></code>	Section of text within other content
<code><p></p></code>	Paragraph of Text
<code>
</code>	Line Break
<code><hr></code>	Basic Horizontal Line

These are the tags used to divide your page up into sections

Unordered list

```
<ul>
  <li>I'm an item</li>
  <li>I'm another item</li>
  <li>I'm another item</li>
</ul>
```

See: [The Unordered List element](#)

HTML Table Tags

<code><table></code>	Defines a table
<code><th></code>	Defines a header cell in a table
<code><tr></code>	Defines a row in a table
<code><td></code>	Defines a cell in a table
<code><caption></code>	Defines a table caption
<code><colgroup></code>	Defines a group of columns
<code><col></code>	Defines a column within a table
<code><thead></code>	Groups the header content
<code><tbody></code>	Groups the body content
<code><tfoot></code>	Groups the footer content

Input Attributes

The input tag is an empty element, identifying the particular type of field information to obtain from a user.

```
<input type="text" name="?" value="?" minlength="6" required />
```

<code>type="..."</code>	The type of data that is being input
<code>value="..."</code>	Default value
<code>name="..."</code>	Used to describe this data in the HTTP request
<code>id="..."</code>	Unique identifier that other HTML elements
<code>readonly</code>	Stops the user from modifying
<code>disabled</code>	Stops any interaction
<code>checked</code>	The radio or checkbox select or not

Input types

<code>type="checkbox"</code>	<input type="checkbox"/>
<code>type="radio"</code>	<input checked="" type="radio"/>
<code>type="file"</code>	<input type="file" value="Browse... No file selected."/>
<code>type="hidden"</code>	
<code>type="text"</code>	<input type="text"/>
<code>type="password"</code>	<input type="password"/>
<code>type="image"</code>	<input type="image" value="Login"/>
<code>type="reset"</code>	<input type="reset" value="Reset"/>
<code>type="button"</code>	<input type="button" value="button"/>
<code>type="submit"</code>	<input type="submit" value="Submit Query"/>

<https://cheatsheets.zip/html>

Quarto Can Handle Page Structure!

Quarto
markdown

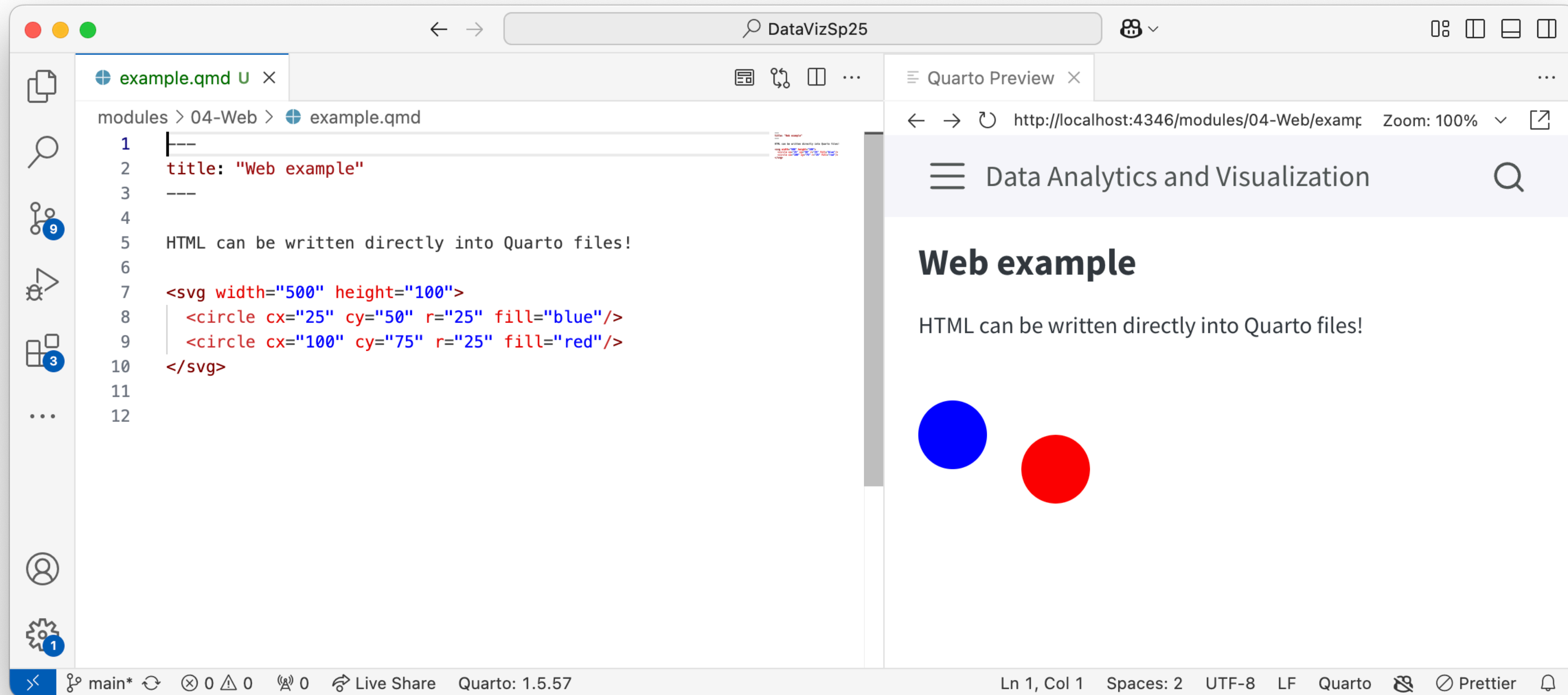
The screenshot displays the Quarto development environment with three main panels:

- index.qmd (Left):** Shows the source markdown file. It includes a title "Course Overview - Spring 2025", a description, an instructor section for Gabe Hope, and an "About me" section with details on contact information and background.
- index.html (Middle):** Shows the compiled HTML code, demonstrating how the markdown's multi-column layout is rendered using `<div class="column" style="width:50%;>` and `<div class="quarto-figure" style="width:50%;>`.
- Quarto Preview (Right):** Shows the rendered web page in a browser. The page features a navigation bar, a main heading, a description, an instructor profile with a photo, and an "About me" section with a bulleted list of topics covered.

Which can be
displayed in
browser

Compiles to HTML

HTML in Quarto



The image shows a Quarto editor window with a file named `example.qmd` open. The editor displays the following HTML code:

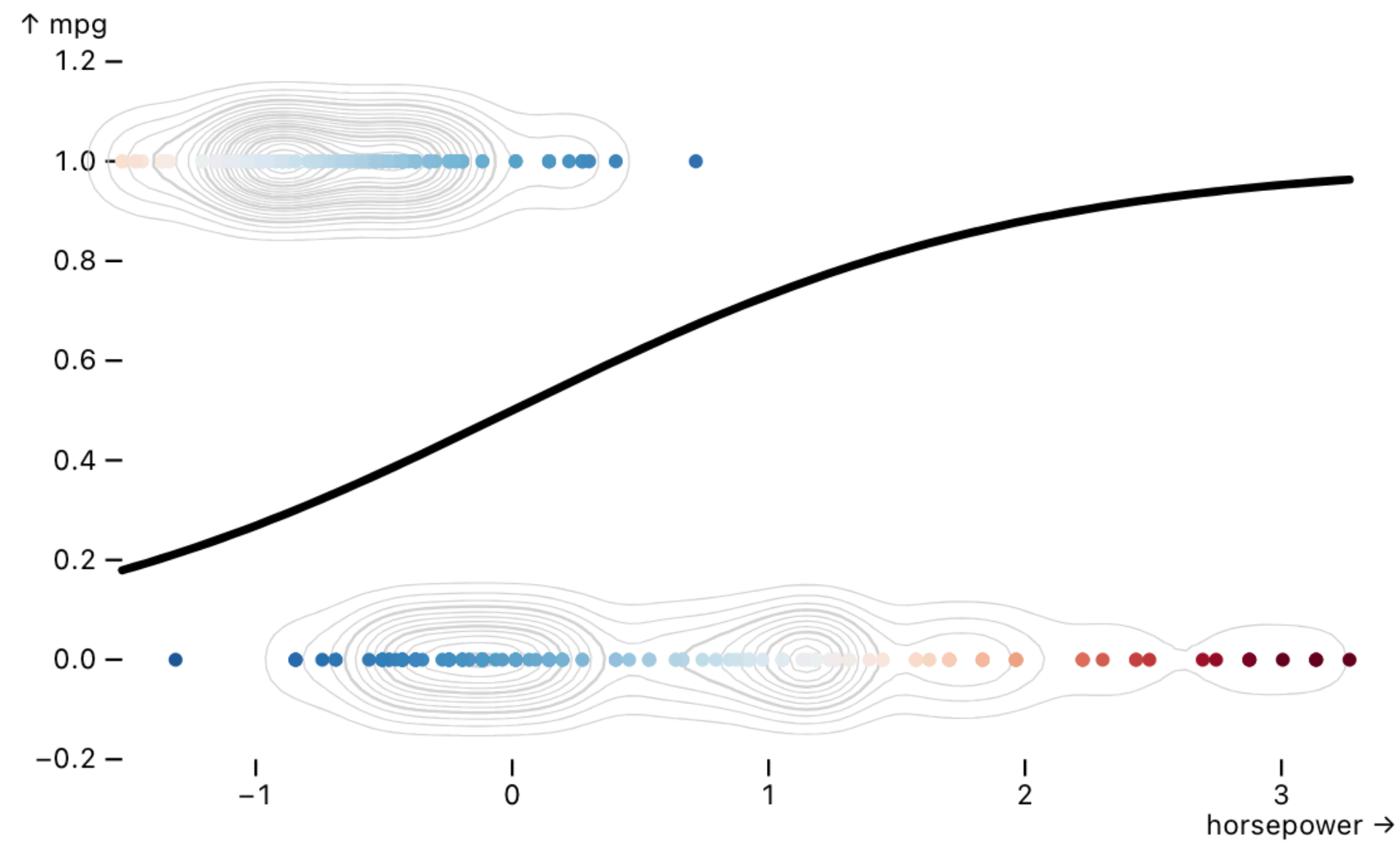
```
1 ---
2 title: "Web example"
3 ---
4
5 HTML can be written directly into Quarto files!
6
7 <svg width="500" height="100">
8   <circle cx="25" cy="50" r="25" fill="blue"/>
9   <circle cx="100" cy="75" r="25" fill="red"/>
10 </svg>
11
12
```

The rendered preview on the right shows the output of the code. It features a header with the title "Data Analytics and Visualization" and a search icon. Below the header, the title "Web example" is displayed, followed by the text "HTML can be written directly into Quarto files!". At the bottom of the preview, two circles are shown: a blue circle on the left and a red circle on the right.

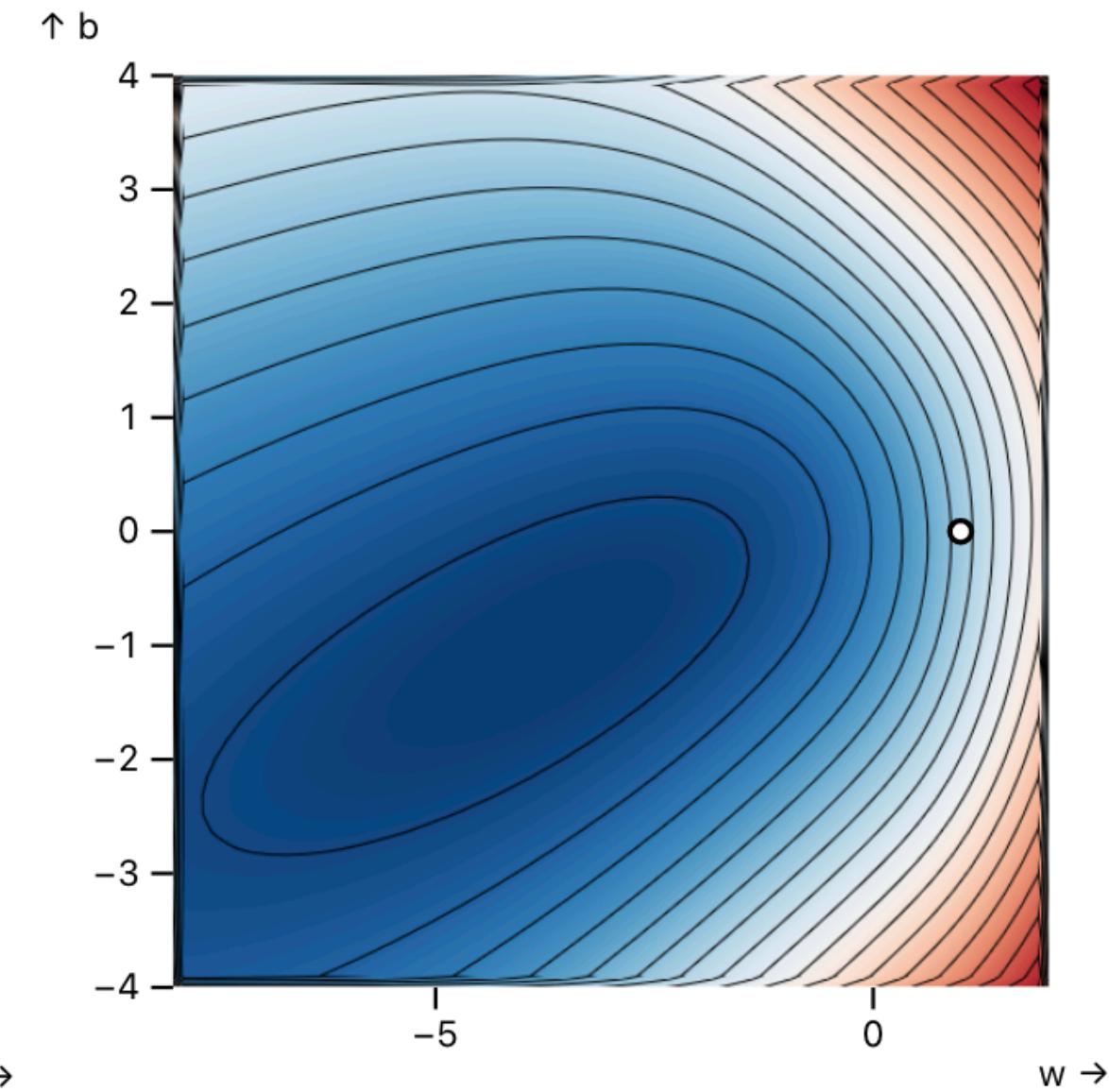
The status bar at the bottom of the editor indicates the current file is `main*`, the editor is running on `Quarto: 1.5.57`, and the current line and column are `Ln 1, Col 1`. The status bar also shows the file encoding as `UTF-8`, the line ending as `LF`, and the editor is using `Prettier` for formatting.

What about visualizations?

Prediction: $p(y = 1 | x) = \sigma(1.00x + 0.00)$



Loss = 1.129



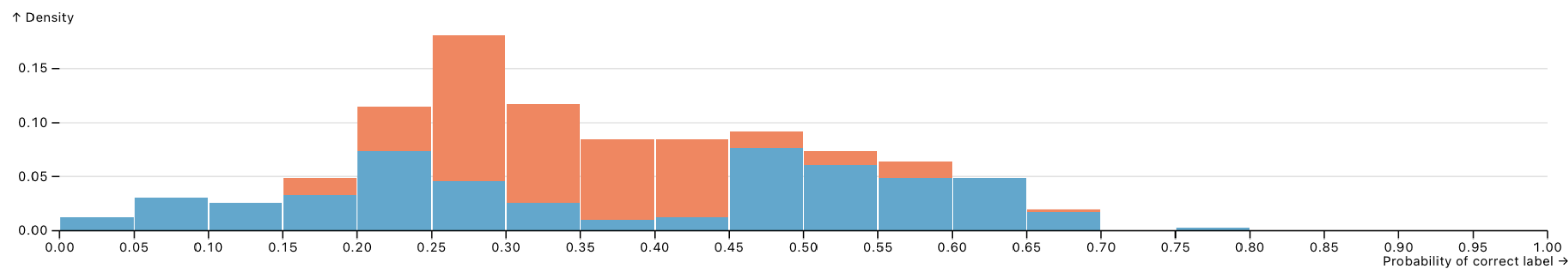
Learning rate



Steps

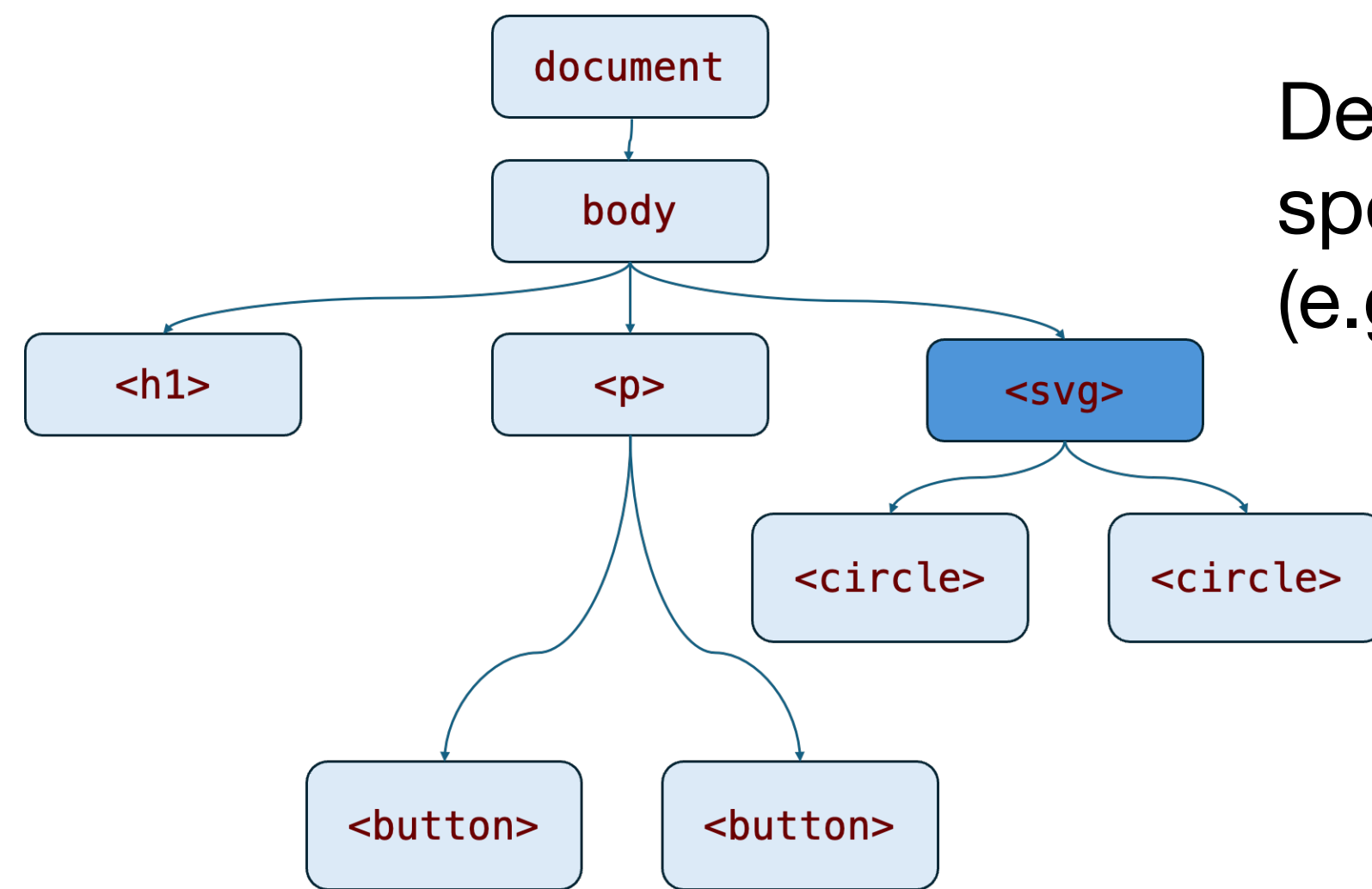


0 1



Scalable Vector Graphics (SVG)

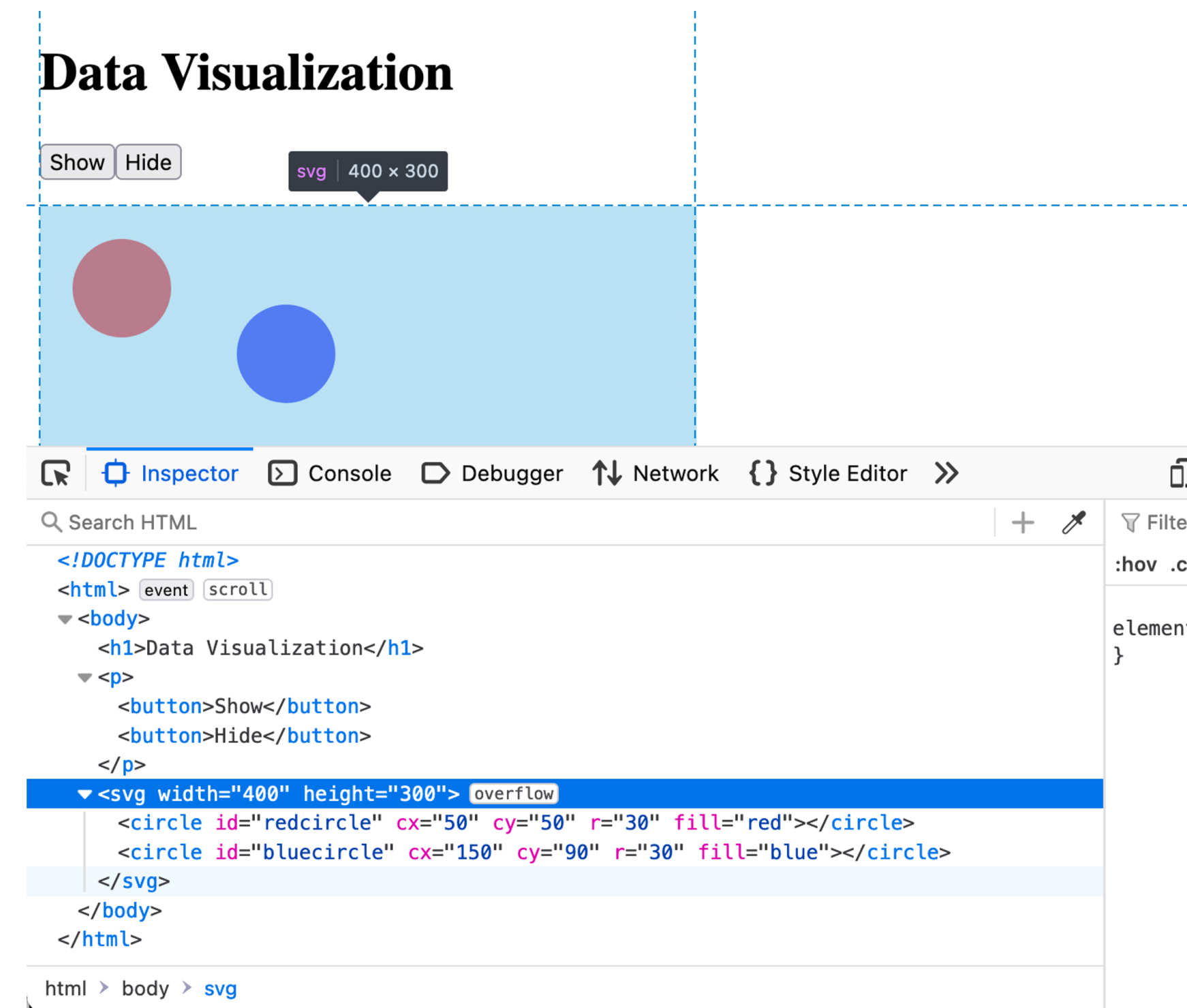
Document Object Model (DOM)



Descendants must be special drawing elements (e.g. *circle*, *rect*, *text*, etc.)

Special container element for drawing graphics (*like visualizations!*)

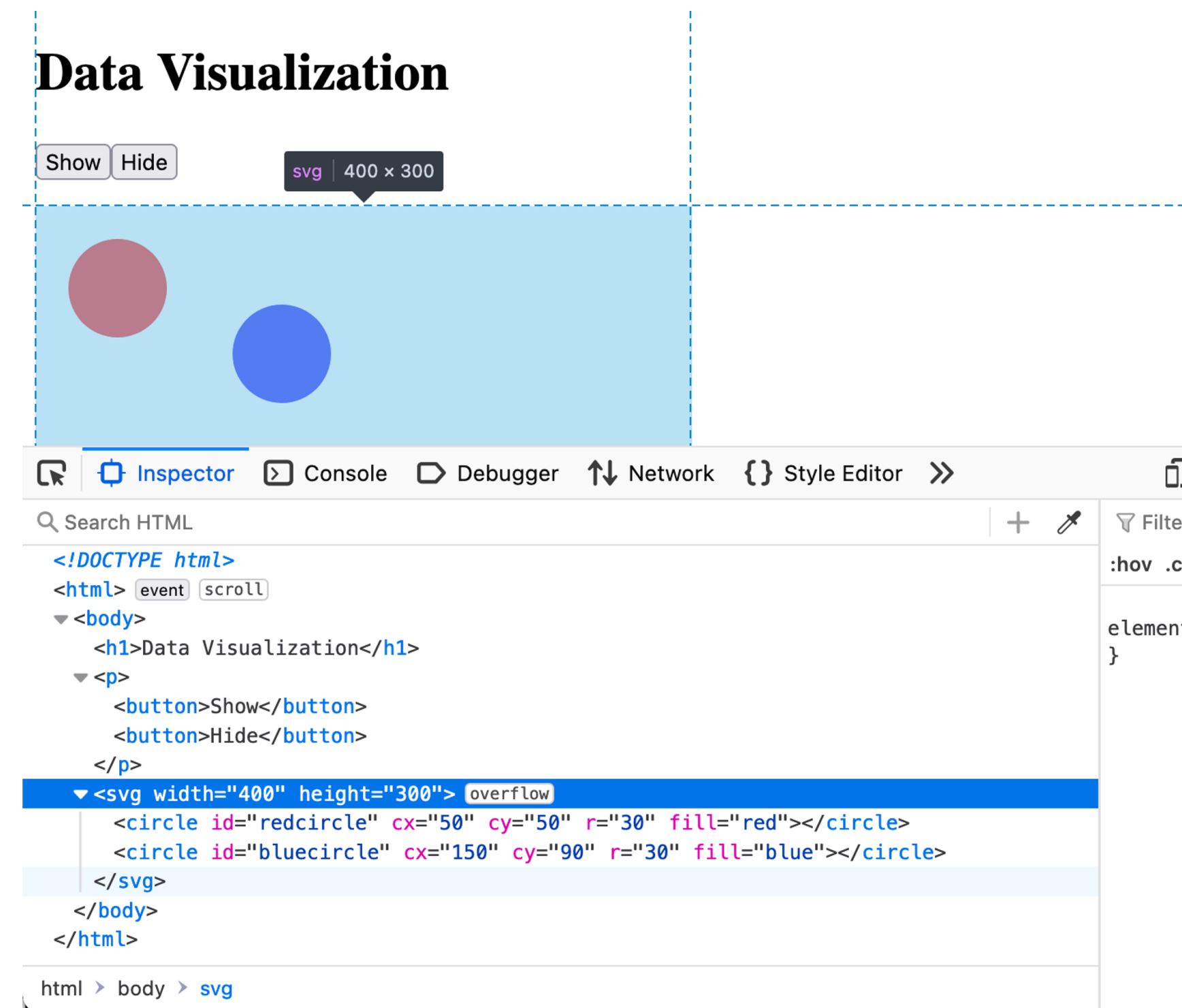
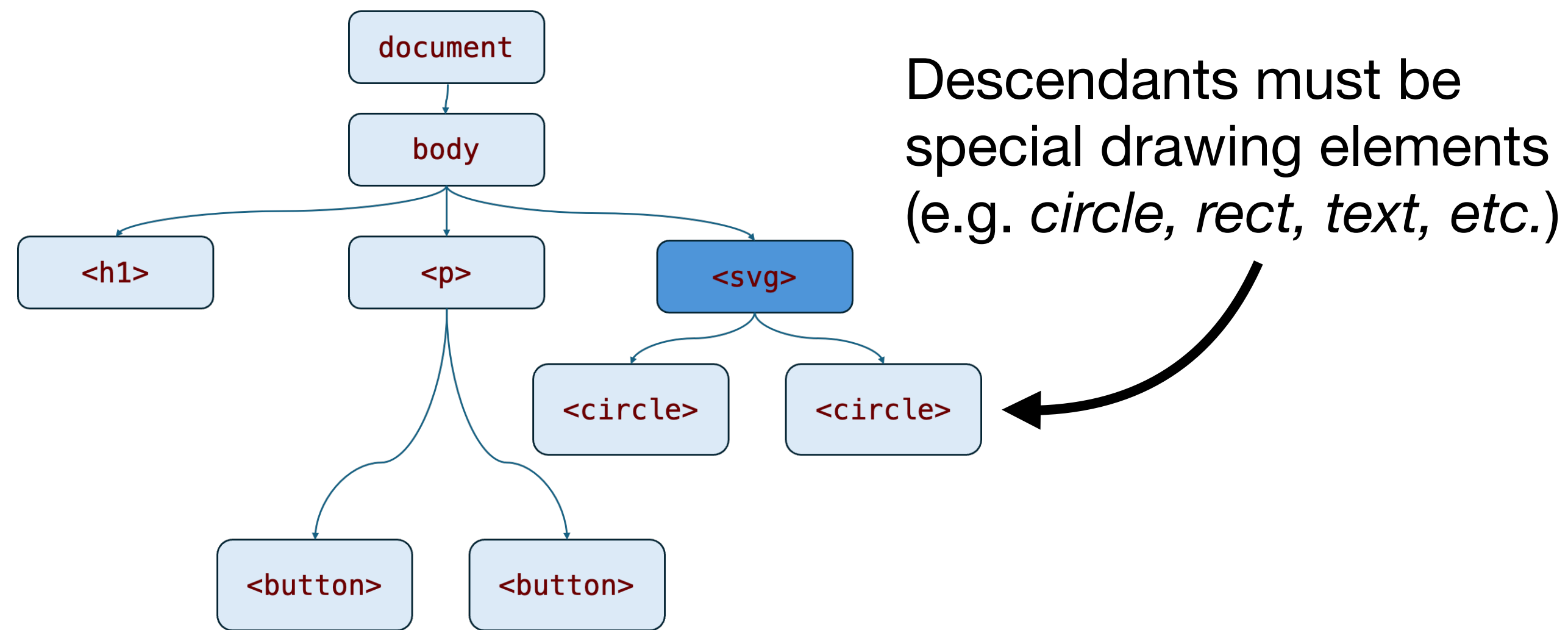
Browser display



Scalable Vector Graphics (SVG)

Document Object Model (DOM)

Browser display

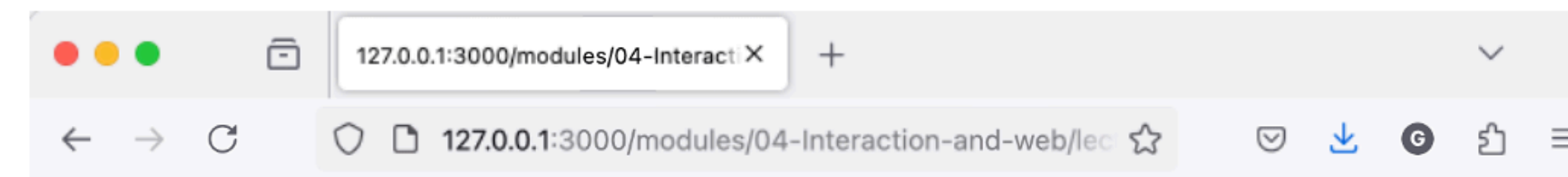
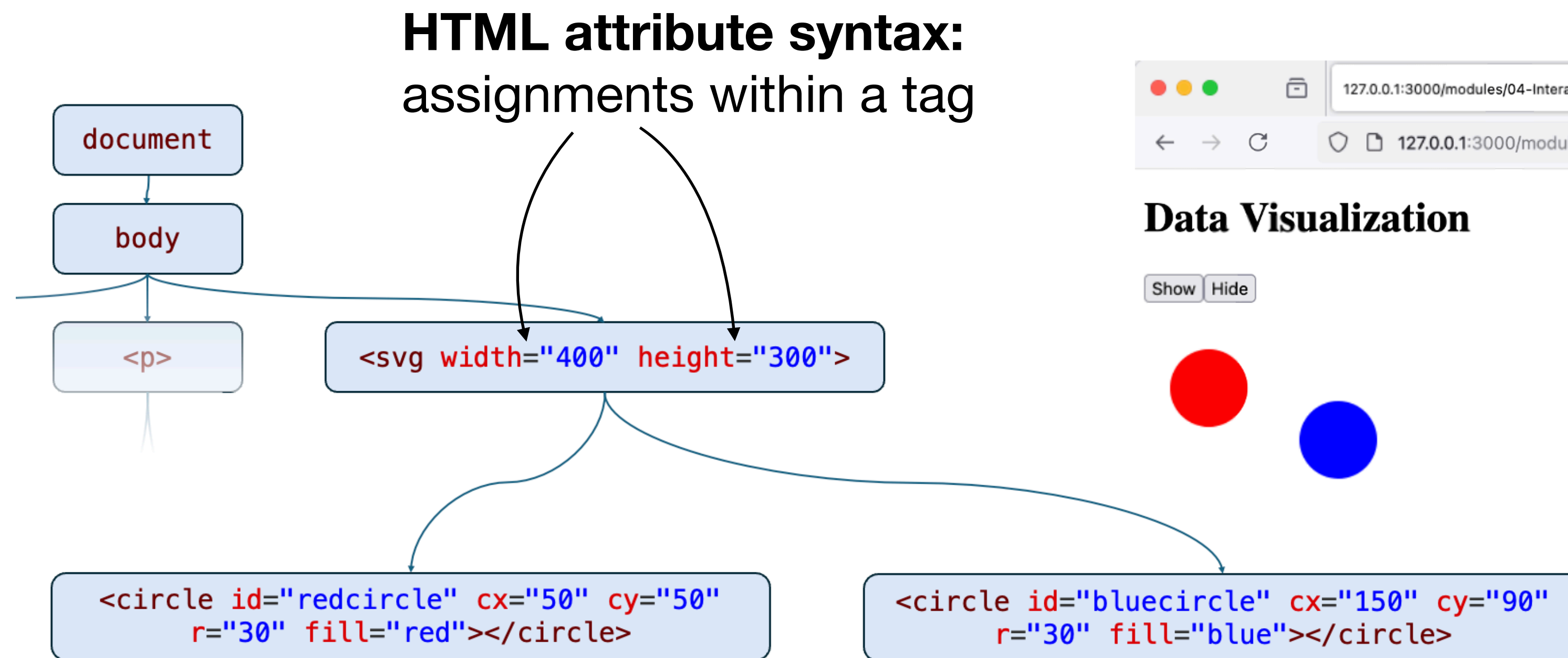


Special container element for drawing graphics (*like visualizations!*)

This will be our focus (mostly)!

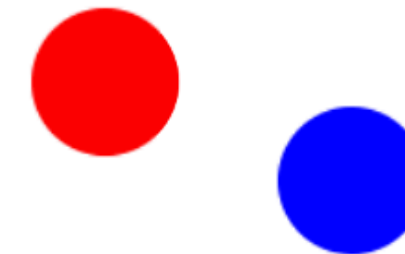
Element attributes

Control the properties of each element



Data Visualization

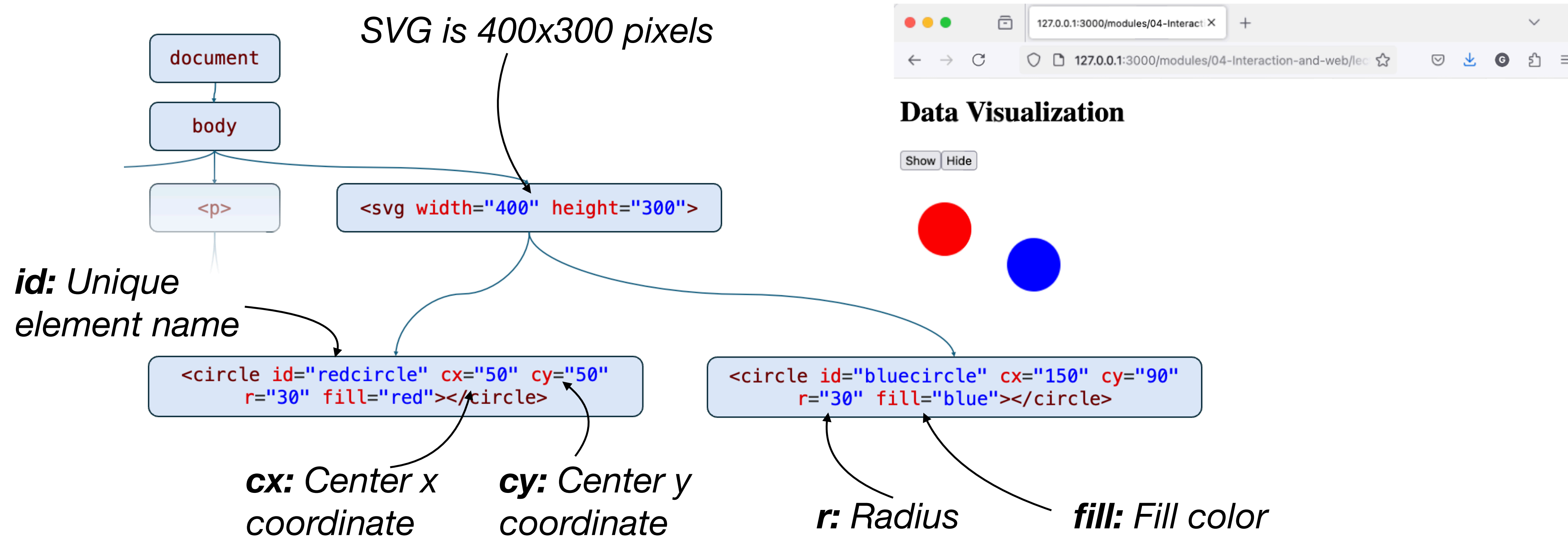
Show Hide



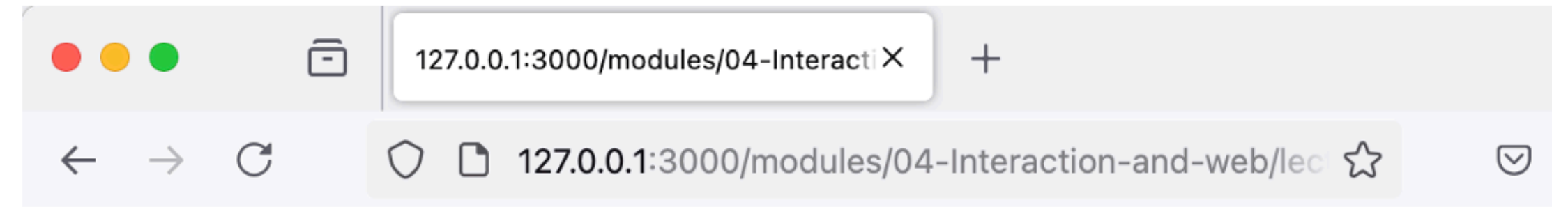
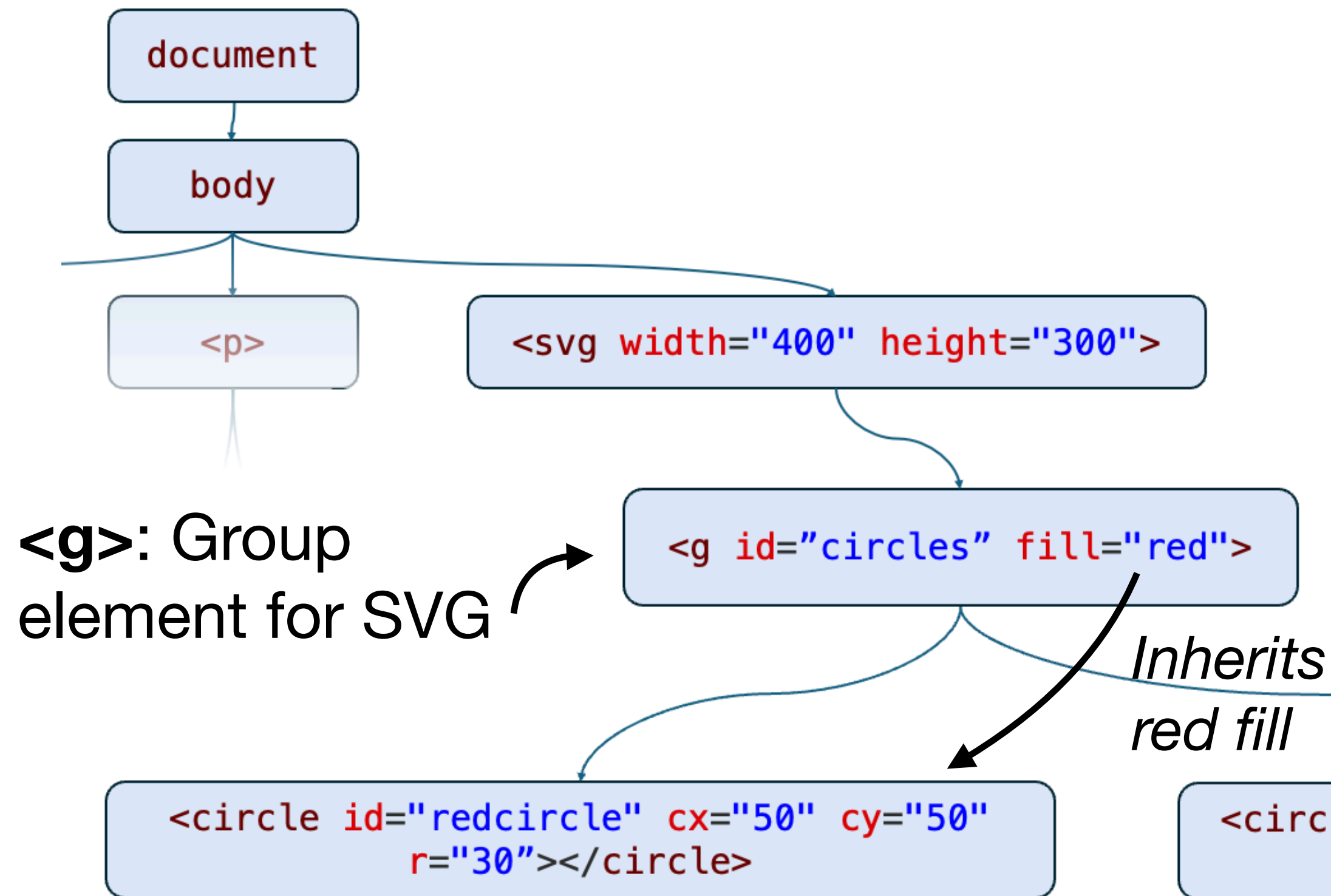
Stored as properties of nodes in the DOM

Element attributes

Control the properties of each element

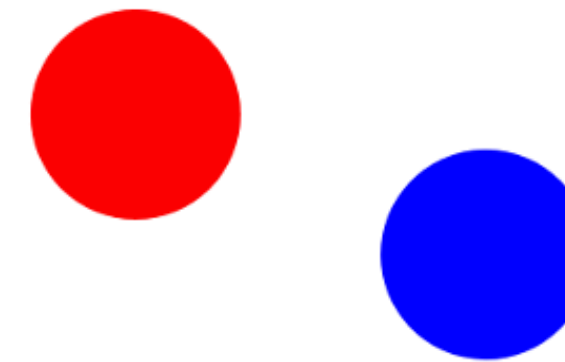


Attribute inheritance



Data Visualization

Show Hide

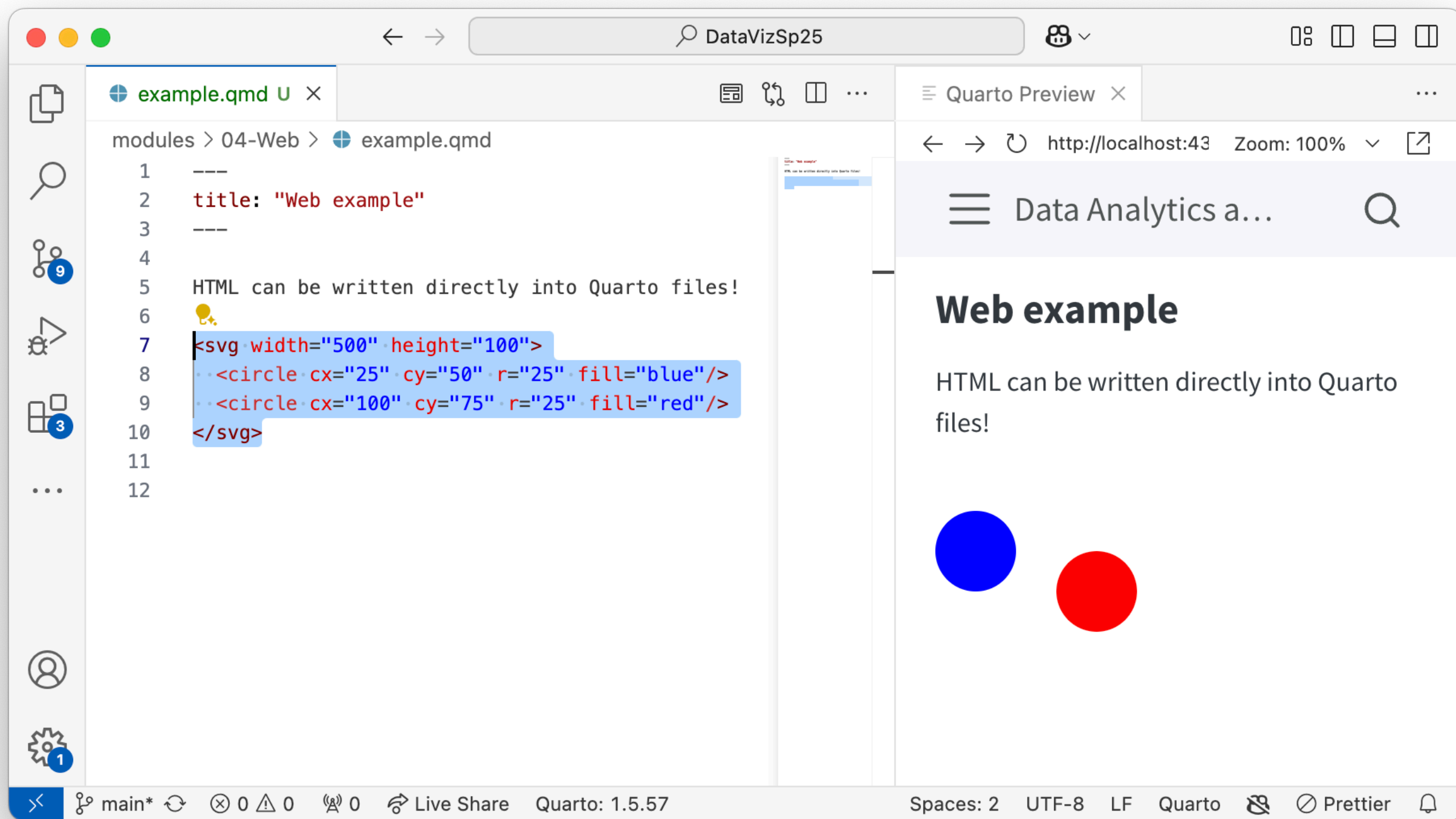


Inherits red fill

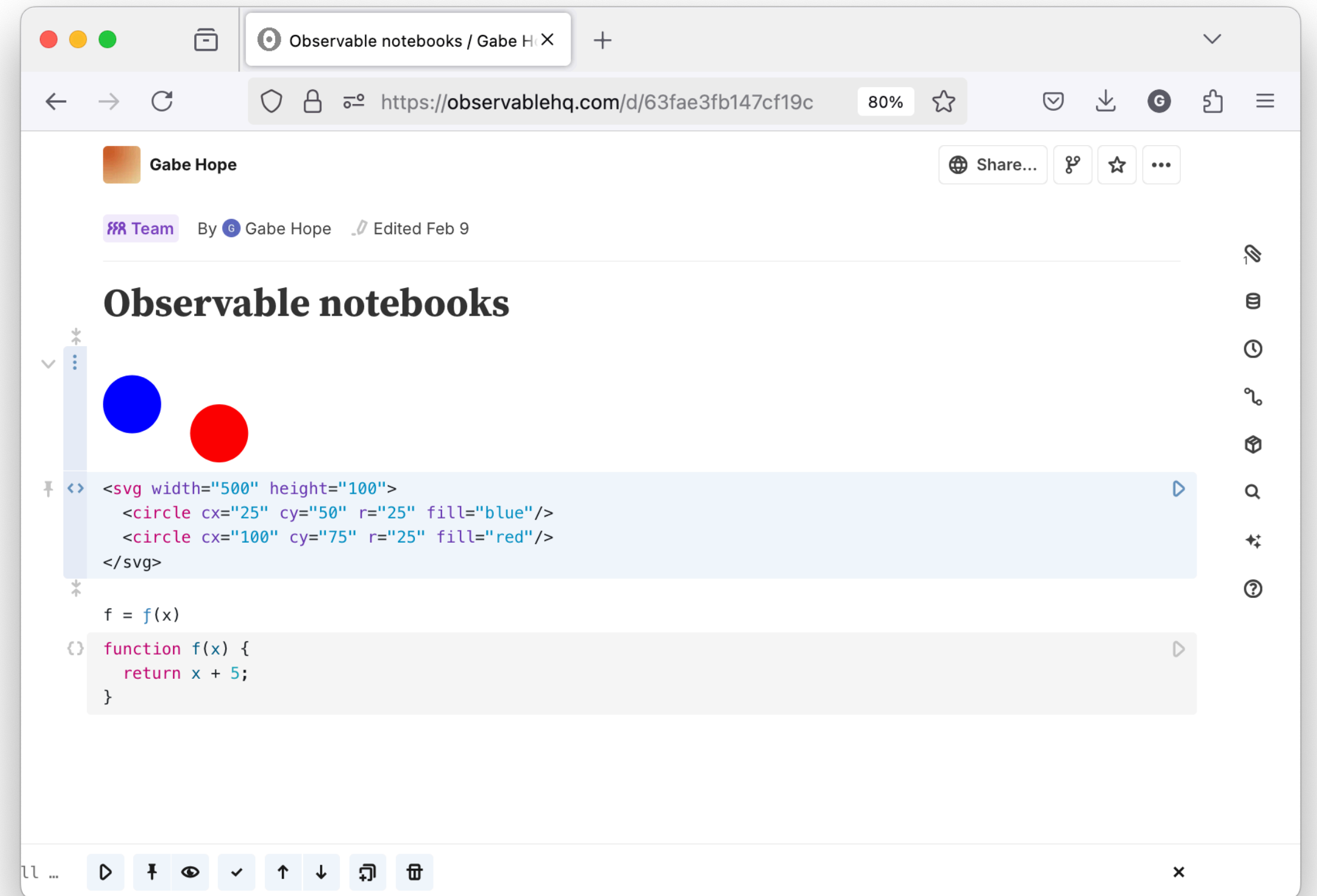
Overwrites with blue fill

Attributes are inherited from parents in the tree!

Trying it out!



Quarto



Observable Notebooks

Observable notebooks

Like Jupyter Notebooks, but for HTML and Javascript!

Output shown above cell!

Switch cell type

The screenshot shows a web browser window displaying an Observable notebook. The browser's address bar shows the URL `https://observablehq.com/d/63fae3fb147cf19c`. The notebook title is "Observable notebooks" by Gabe Hope, edited on Feb 9. The notebook content consists of two cells. The first cell is an HTML cell containing an SVG that renders two circles: a blue circle at (25, 50) and a red circle at (100, 75). The second cell is a Javascript cell containing the code `f = f(x)` and `function f(x) { return x + 5; }`. Annotations with arrows point to the rendered circles, the code editor of the HTML cell, and the code editor of the Javascript cell.

```
<svg width="500" height="100">
  <circle cx="25" cy="50" r="25" fill="blue"/>
  <circle cx="100" cy="75" r="25" fill="red"/>
</svg>
```

```
f = f(x)
function f(x) {
  return x + 5;
}
```

HTML cell

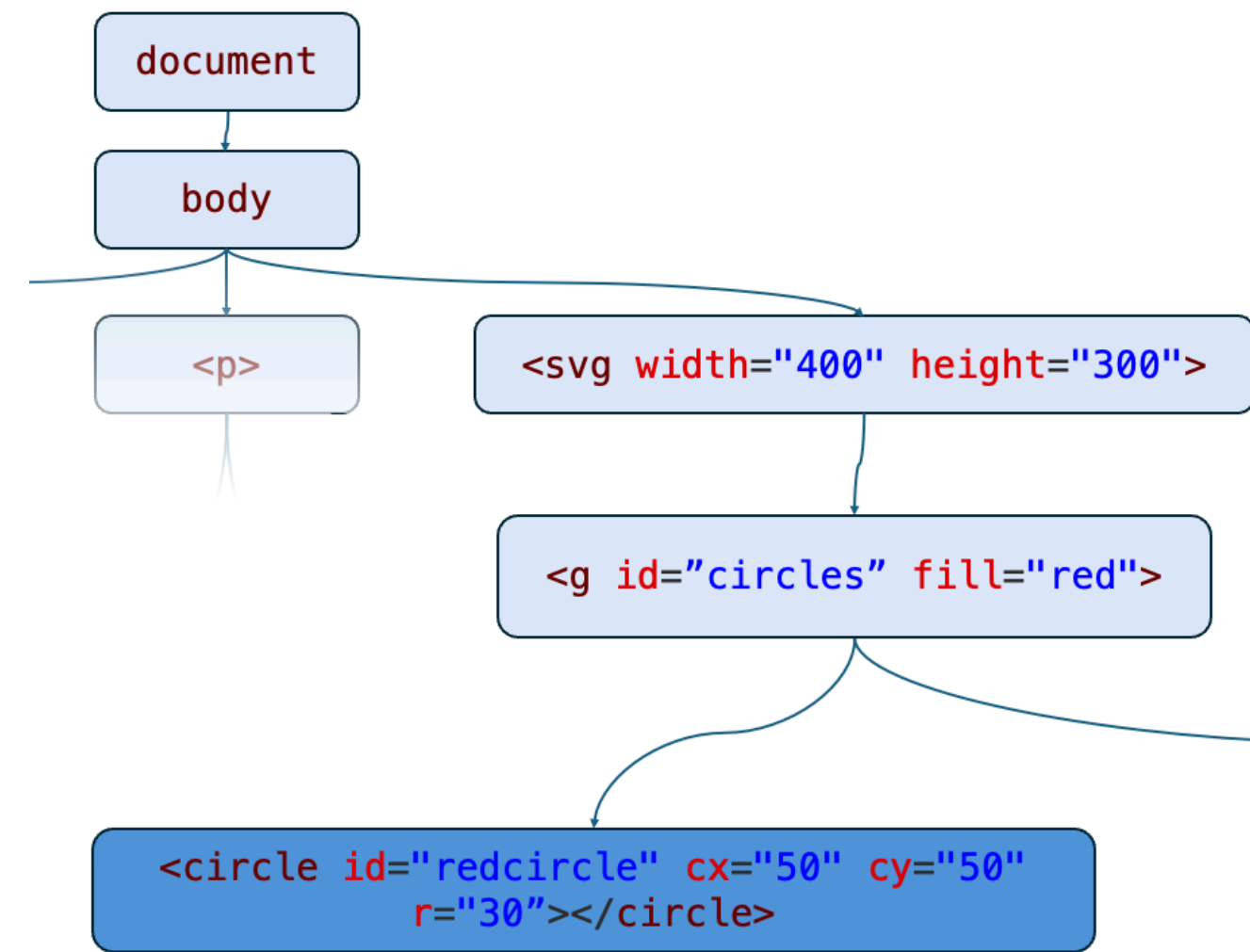
Javascript cell

Has some very special properties that we'll discuss later!

Exercise: SVG Elements

<https://observablehq.com/d/76b58a371719c6cf>

Javascript: Let's us manipulate the DOM!



document is a built-in global variable

Data Visualization

Show Hide

circle#redcircle 60 x 60

Blue

Inspector Console Debugger Network

Filter Output Errors Warnings

```
>> document.getElementById("redcircle")  
← <circle id="redcircle" cx="50" cy="50" r="30" fill="red">
```

Can execute Javascript in the web inspector console

```
document.getElementById("redcircle")
```

Step 1: Select an element

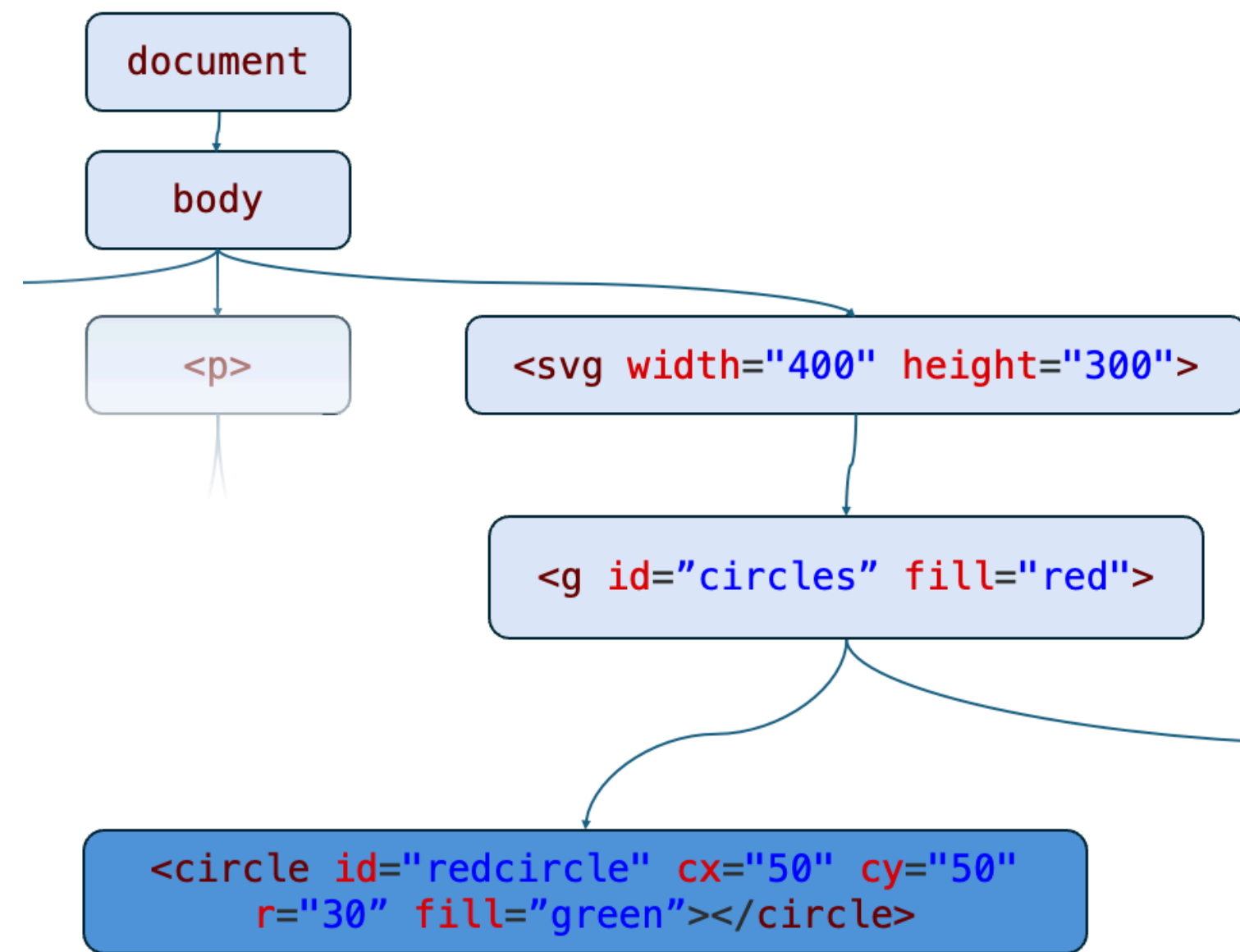
Search the subtree rooted at *document* for an element with *id="redcircle"*

Javascript: Let's us manipulate the DOM!

Step 1: Select an element

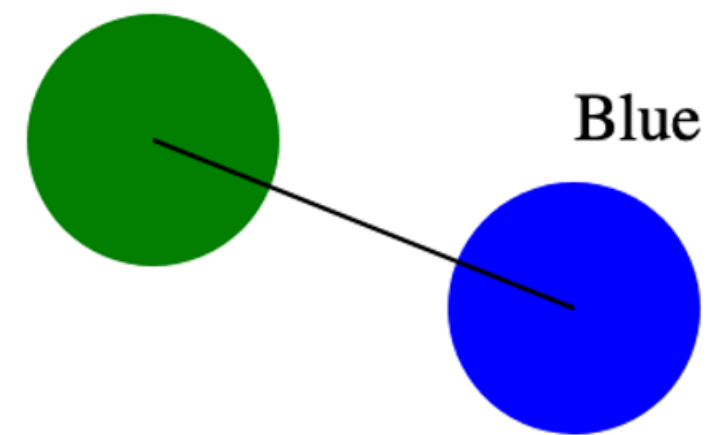
Step 2: Change properties

Many ways to select and modify elements are built into Javascript, but we'll introduce a library that makes it easy!



Data Visualization

Show Hide



```
Inspector Console Debugger Network Style Edi
Filter Output Errors Warnings Logs Info
>> document.getElementById("redcircle").setAttribute("fill", 'green')
← undefined
```

*Change the **fill** attribute to green*

Javascript in Observable

Observable notebooks



```
<> <svg width="500" height="100">  
  <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>  
  <circle id="redcircle" cx="100" cy="75" r="25" fill="red"/>  
</svg>
```

⋮

▶ SVGCircleElement {}

```
{ } document.getElementById('redcircle')
```

Selected element is represented with an *object* in Javascript

Observable notebooks



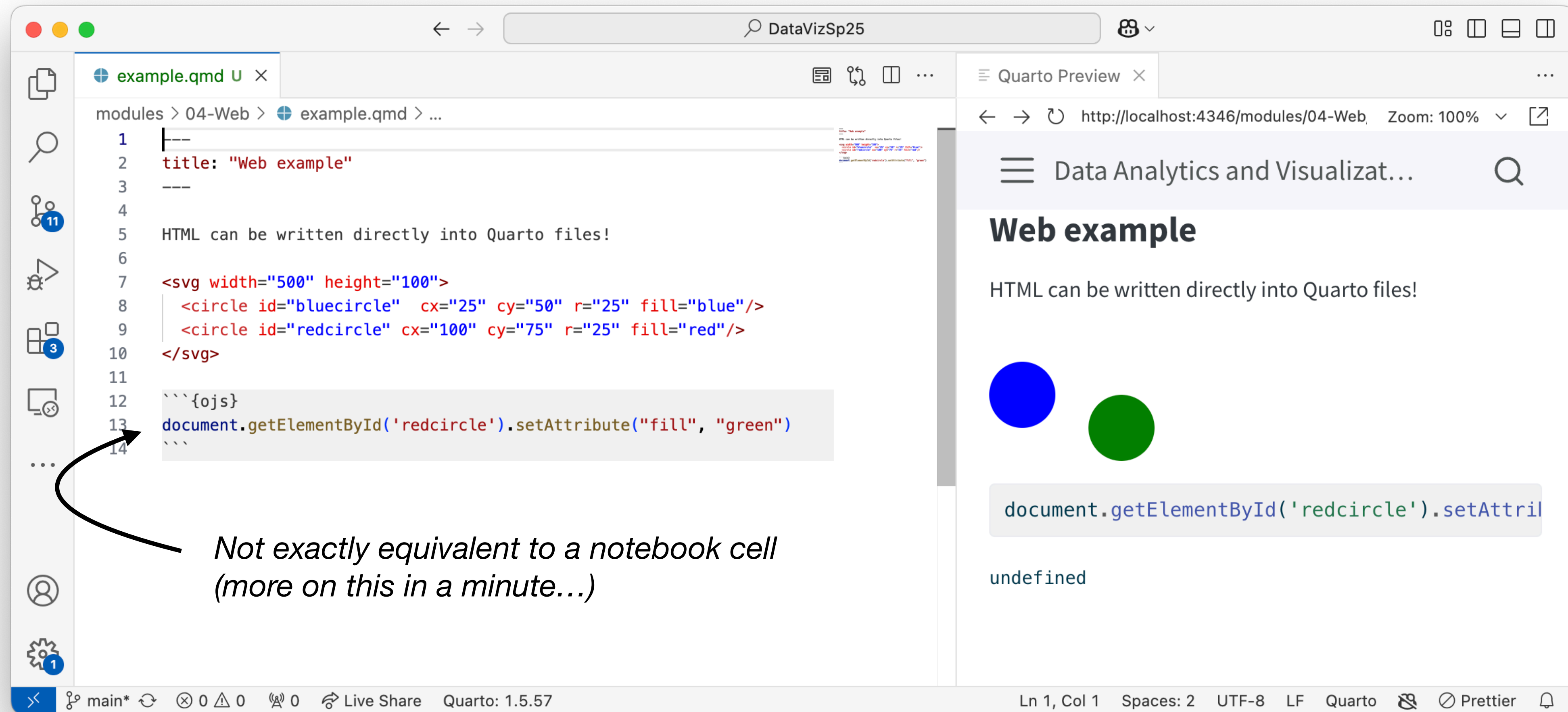
```
<> <svg width="500" height="100">  
  <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>  
  <circle id="redcircle" cx="100" cy="75" r="25" fill="red"/>  
</svg>
```

⋮

undefined

```
{ } document.getElementById('redcircle').setAttribute("fill", "green")
```

Quarto supports Javascript through *ojs* code blocks



The screenshot shows the Quarto editor interface. On the left, a code editor displays the following content:

```
1 ---  
2 title: "Web example"  
3 ---  
4  
5 HTML can be written directly into Quarto files!  
6  
7 <svg width="500" height="100">  
8   <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>  
9   <circle id="redcircle" cx="100" cy="75" r="25" fill="red"/>  
10 </svg>  
11  
12 ```{ojs}  
13 document.getElementById('redcircle').setAttribute("fill", "green")  
14 ```
```

An arrow points from the JavaScript code block to the rendered output on the right. The rendered output shows a blue circle and a green circle, with the JavaScript code block below them.

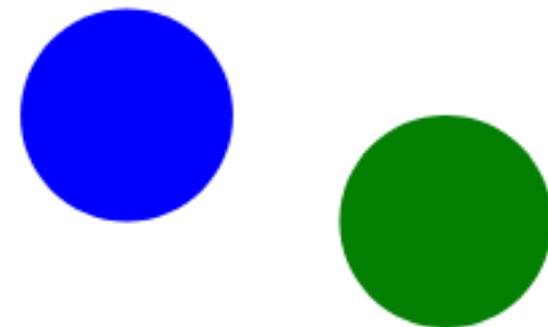
*Not exactly equivalent to a notebook cell
(more on this in a minute...)*

The bottom status bar shows: main* 0 0 0 Live Share Quarto: 1.5.57 Ln 1, Col 1 Spaces: 2 UTF-8 LF Quarto Prettier

Exercise: Can you change a property of your SVG with Javascript?

Why Javascript: Interaction

Observable notebooks



*Define a **function** to make our circle green*

```
<> <svg width="500" height="100">  
  <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>  
  <circle id="redcircle" cx="100" cy="75" r="25" fill="red"/>  
</svg>
```

```
makeGreen = f()
```

```
{ } function makeGreen() {  
  const redcircle = document.getElementById("redcircle");  
  redcircle.setAttribute("fill", "green");  
}
```

```
*  
: undefined
```

```
{ } document.getElementById("redcircle").addEventListener('click', makeGreen);
```

Call the function when a user clicks on the circle!

Exercise: Try it out!

makeGreen = f() *Javascript function syntax*

```
{ } function makeGreen() {  
  const redcircle = document.getElementById("redcircle");  
  redcircle.setAttribute("fill", "green");  
}
```

Declare a **constant** variable
(can't be reassigned)

```
*  
: undefined  
{ } document.getElementById("redcircle").addEventListener('click', makeGreen);  
.
```

Alternative syntax

Declare a variable
(**can** be reassigned)

```
document.getElementById("redcircle").addEventListener('click', () => {  
  let redcircle = document.getElementById("redcircle");  
  redcircle.setAttribute("fill", "green");  
});
```

Anonymous function
(not named, can't
reference later)

Back to Observable Notebooks

Basic Observable usage

In Observable, the output of a cell appears above its code.

3

```
{ } 1 + 2
```

Giving a cell a name allows us to reference its value elsewhere.

```
num = 3
```

```
{ } num = 1 + 2
```

12

```
{ } num * 4
```

Back to Observable Notebooks

Basic Observable usage

In Observable, the output of a cell appears above its code.

3

```
{ } 1 + 2
```

Giving a cell a name allows us to reference its value elsewhere.

```
num = 3
```

```
{ } num = 1 + 2
```

12

```
{ } num * 4
```

A cell in Observable only allows a **single expression**

3

```
{ } 1 + 2
```

+

```
⋮ | SyntaxError: Unexpected token
```

```
{ } 1 + 2;
```

```
  | 5 + 6;
```

Trying to write two expressions in the same cell results in an error!

Back to Observable Notebooks

Basic Observable usage

In Observable, the output of a cell appears above its code.

3

```
{ } 1 + 2
```

Giving a cell a name allows us to reference its value elsewhere.

```
num = 3
```

```
{ } num = 1 + 2
```

12

```
{ } num * 4
```

Normally we'd need to define a variable with *const* or *let*

A cell name is a special type of variable

```
function makeGreen() {  
  const redcircle = document.getElementById("redcircle");  
  redcircle.setAttribute("fill", "green");  
}
```

A function declaration also sets the cell name

Back to Observable Notebooks

Basic Observable usage

In Observable, the output of a cell appears above its code.

3

```
{ } 1 + 2
```

Giving a cell a name allows us to reference its value elsewhere.

```
num = 3
```

```
{ } num = 1 + 2
```

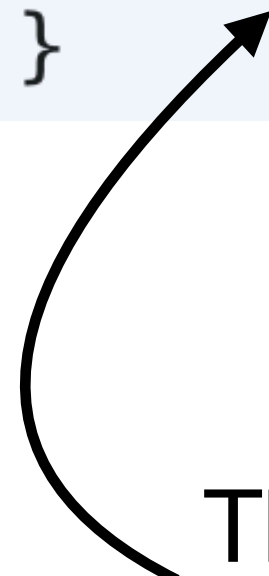
12

```
{ } num * 4
```

We can write normal Javascript by creating a block with curly braces

```
newNum = {  
  const a = 5;  
  const b = 6;  
  return a * b;  
}
```

The return value gets assigned to the cell variable



Observable reactivity

*Observable cells are **reactive!***

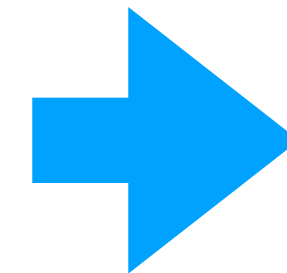
*When a cell's value changes,
other cells that use its value
are **automatically re-run!***

```
b = 60
```

```
b = a * 2
```

```
a = 30
```

```
a = 30
```



```
b = 100
```

```
b = a * 2
```

```
a = 50
```

```
a = 50
```

b is updated!

Change a to 50

Observable reactivity

Cells do *not* run top-to-bottom!

b = 100

b = a * 2

c = 150

c = a + b

a = 50

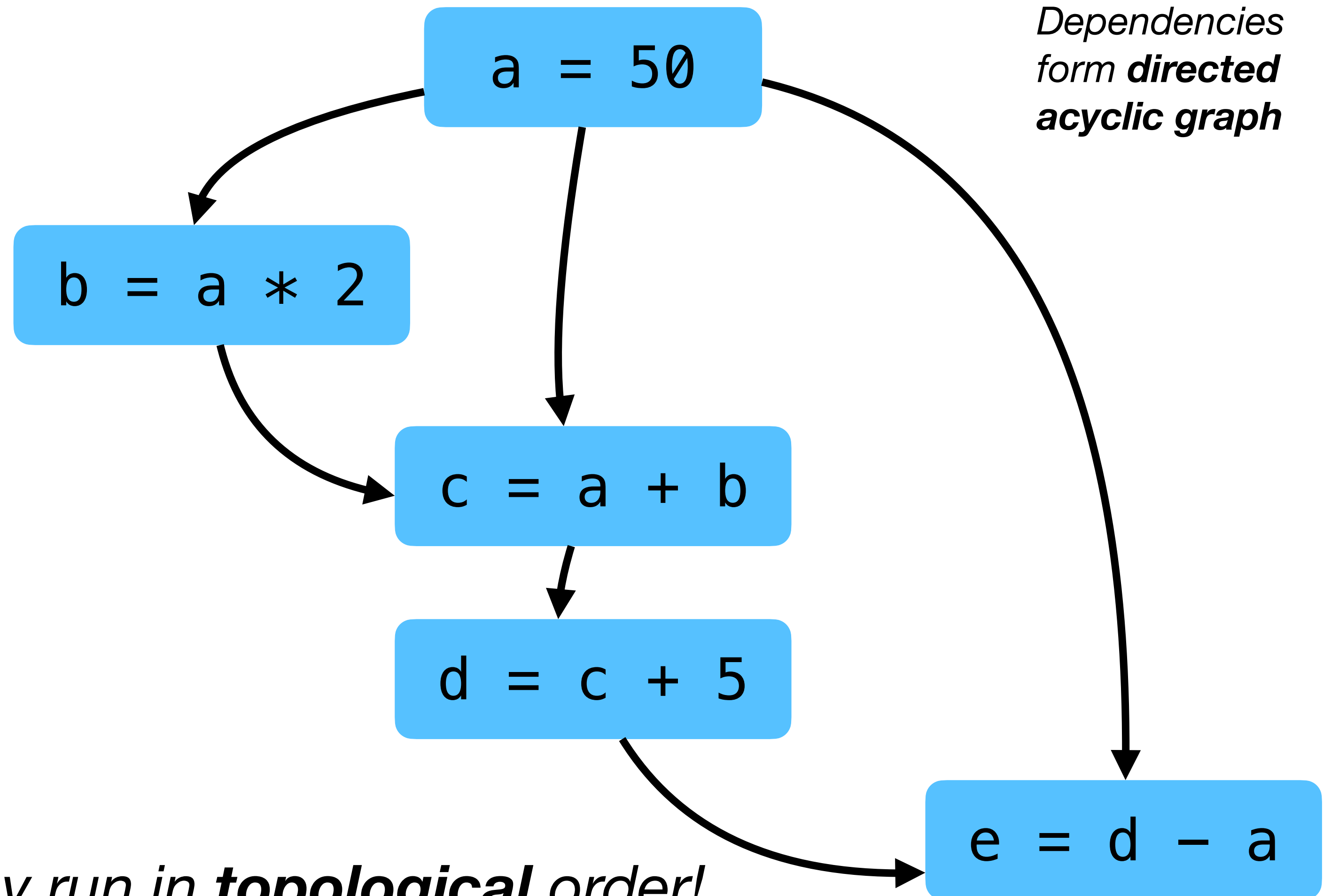
a = 50

d = 155

d = c + 5

e = 105

e = d - a



They run in **topological order**!

Observable reactivity

Cycles are not allowed!

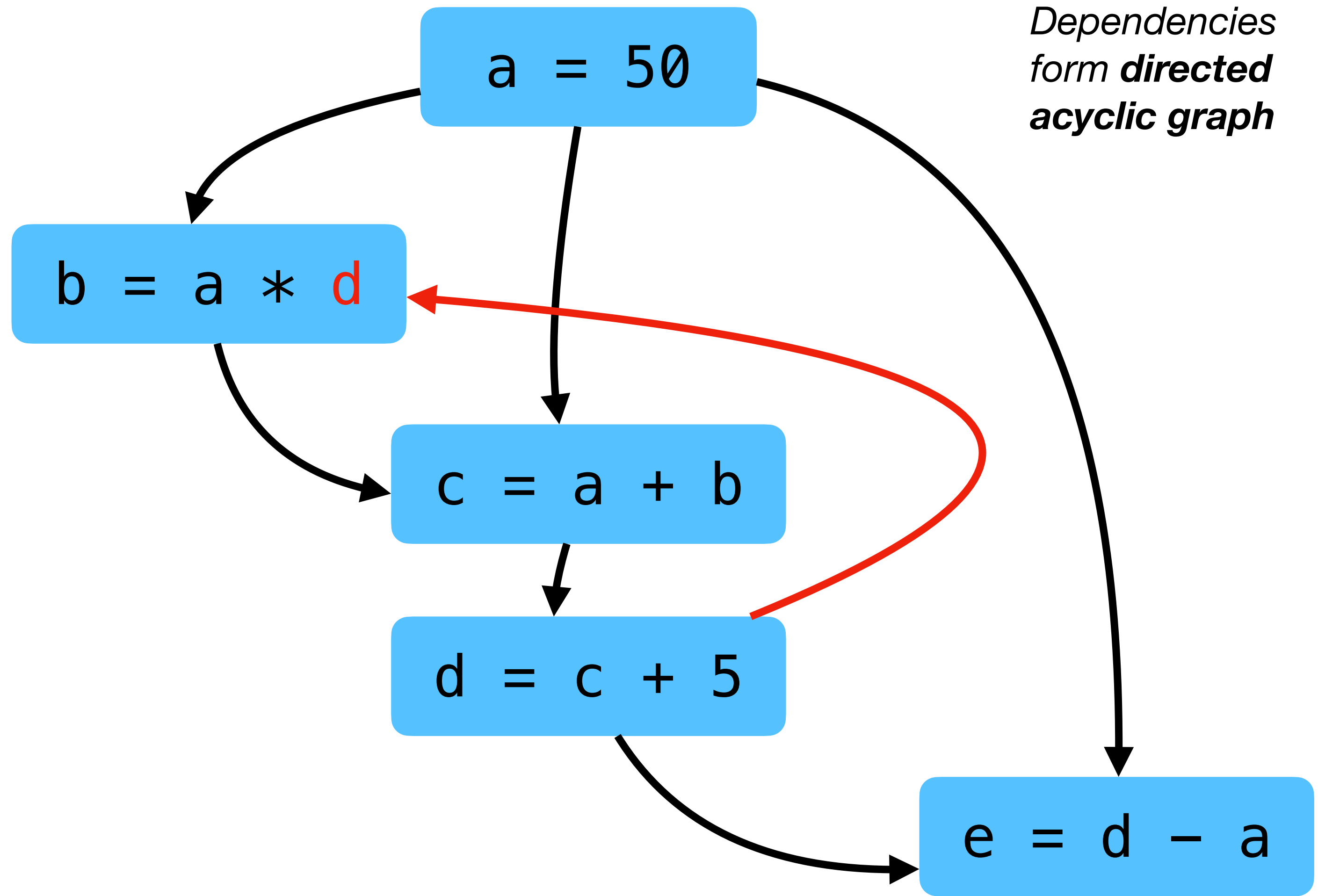
Dependencies form **directed acyclic graph**

```
⋮ | b = RuntimeError: circular definition
{ } | b = a * d
*
| c = RuntimeError: circular definition
{ } | c = a + b

a = 50
{ } | a = 50

| d = RuntimeError: circular definition
{ } | d = c + 5

| e = RuntimeError: circular definition
{ } | e = d - a
```



Observable reactivity

Inputs allow user actions to influence state

```
b = 6.4
```

```
{ } b = a / 10
```

```
a = 64
```

```
{ } a = r + 25
```

```
39
```

```
{ } viewof r = Inputs.range([0, 100], {step: 1})
```

Range: slider input

*Min/max values as Javascript Array
(like Python list)*

Keyword to watch for changes

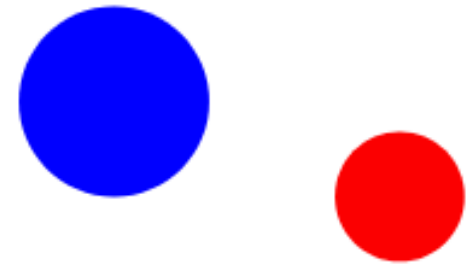
*Other options as Javascript object
(like Python dict)*

```
r = Input
```

```
a = r + 25
```

```
b = a / 10
```

Observable reactivity

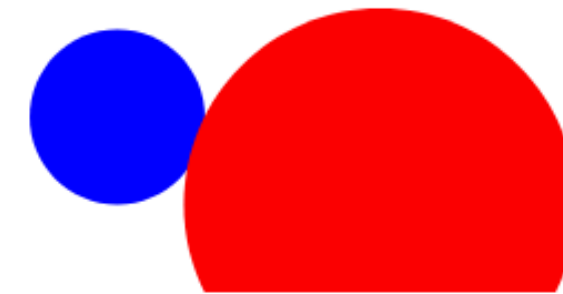


```
<> <svg width="500" height="100">  
  <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>  
  <circle id="redcircle" cx="100" cy="75" r=${r} fill="red"/>  
</svg>
```

17

```
viewof r = Inputs.range([0, 100], {step: 1})
```

HTML can react as well!



```
<> <svg width="500" height="100">  
  <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>  
  <circle id="redcircle" cx="100" cy="75" r=${r} fill="red"/>  
</svg>
```

56

```
viewof r = Inputs.range([0, 100], {step: 1})
```

Use Javascript value r

`r=${r}`

Exercise: Try it out!

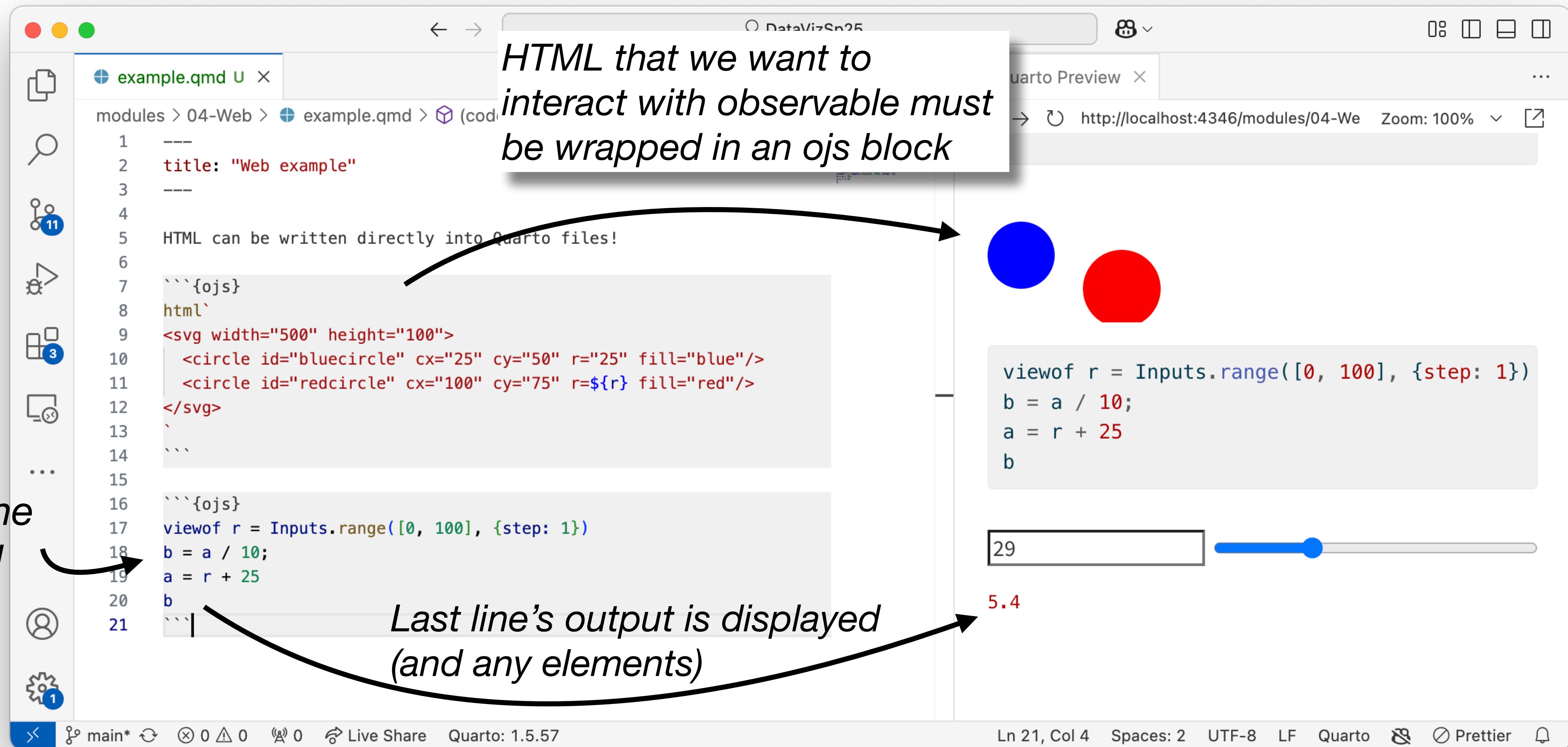
Observable in Quarto

HTML that we want to interact with observable must be wrapped in an ojs block

```
1 ---
2 title: "Web example"
3 ---
4
5 HTML can be written directly into Quarto files!
6
7 ```{ojs}
8 html`
9 <svg width="500" height="100">
10   <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>
11   <circle id="redcircle" cx="100" cy="75" r=${r} fill="red"/>
12 </svg>
13 `
14 ```
15
16 ```{ojs}
17 viewof r = Inputs.range([0, 100], {step: 1})
18 b = a / 10;
19 a = r + 25
20 b
21 ```
```

Each line is a cell

Last line's output is displayed (and any elements)



Quarto Preview X
http://localhost:4346/modules/04-We Zoom: 100%

viewof r = Inputs.range([0, 100], {step: 1})
b = a / 10;
a = r + 25
b

29

5.4

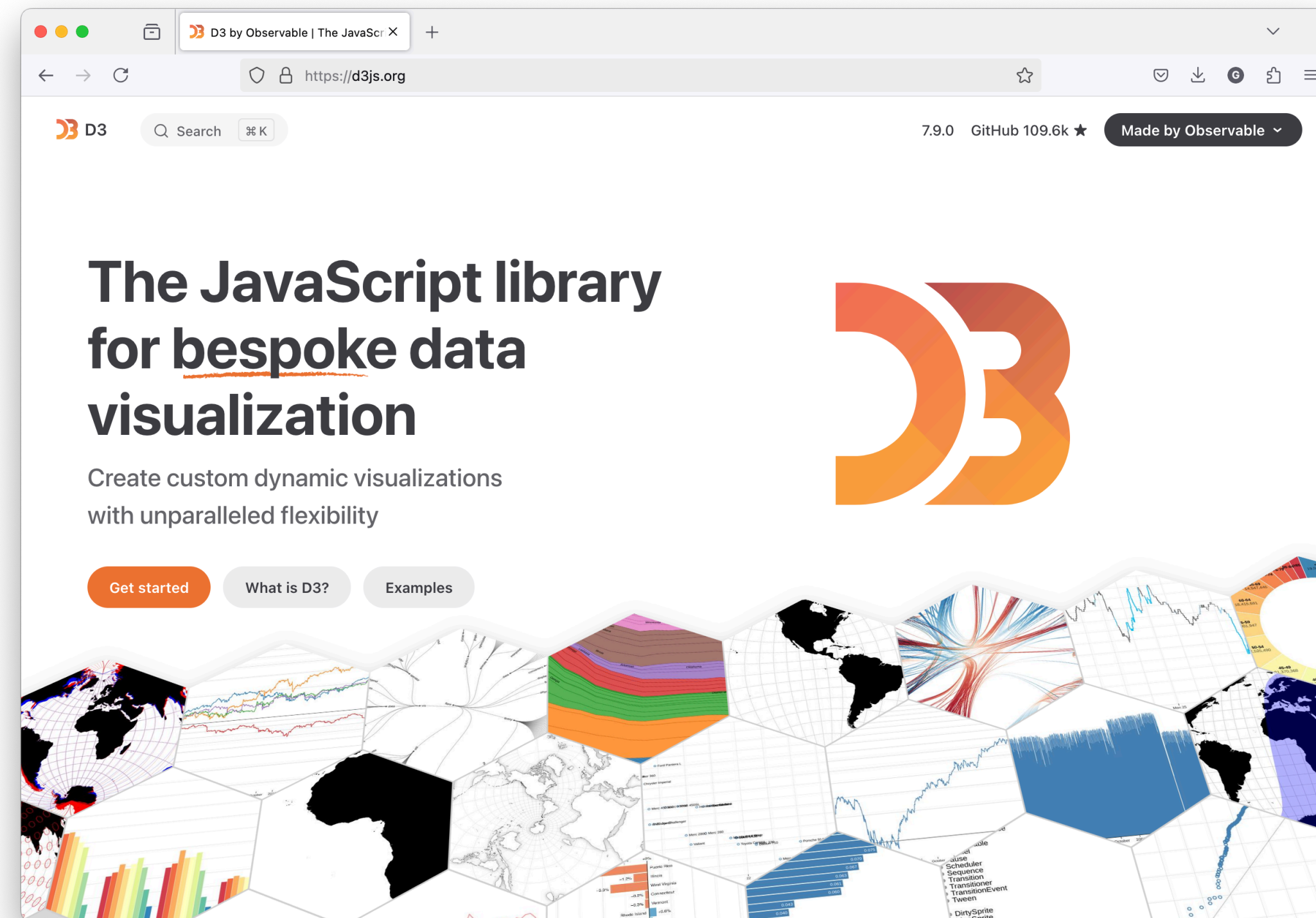
Ln 21, Col 4 Spaces: 2 UTF-8 LF Quarto Prettier

Next Steps: Javascript

- Javascript introduction: <https://observablehq.com/@nyuvis/javascript-basics?collection=@nyuvis/guides-and-examples>
- Javascript data wrangling: <https://observablehq.com/@nyuvis/data-transformation?collection=@nyuvis/guides-and-examples>

D3.js

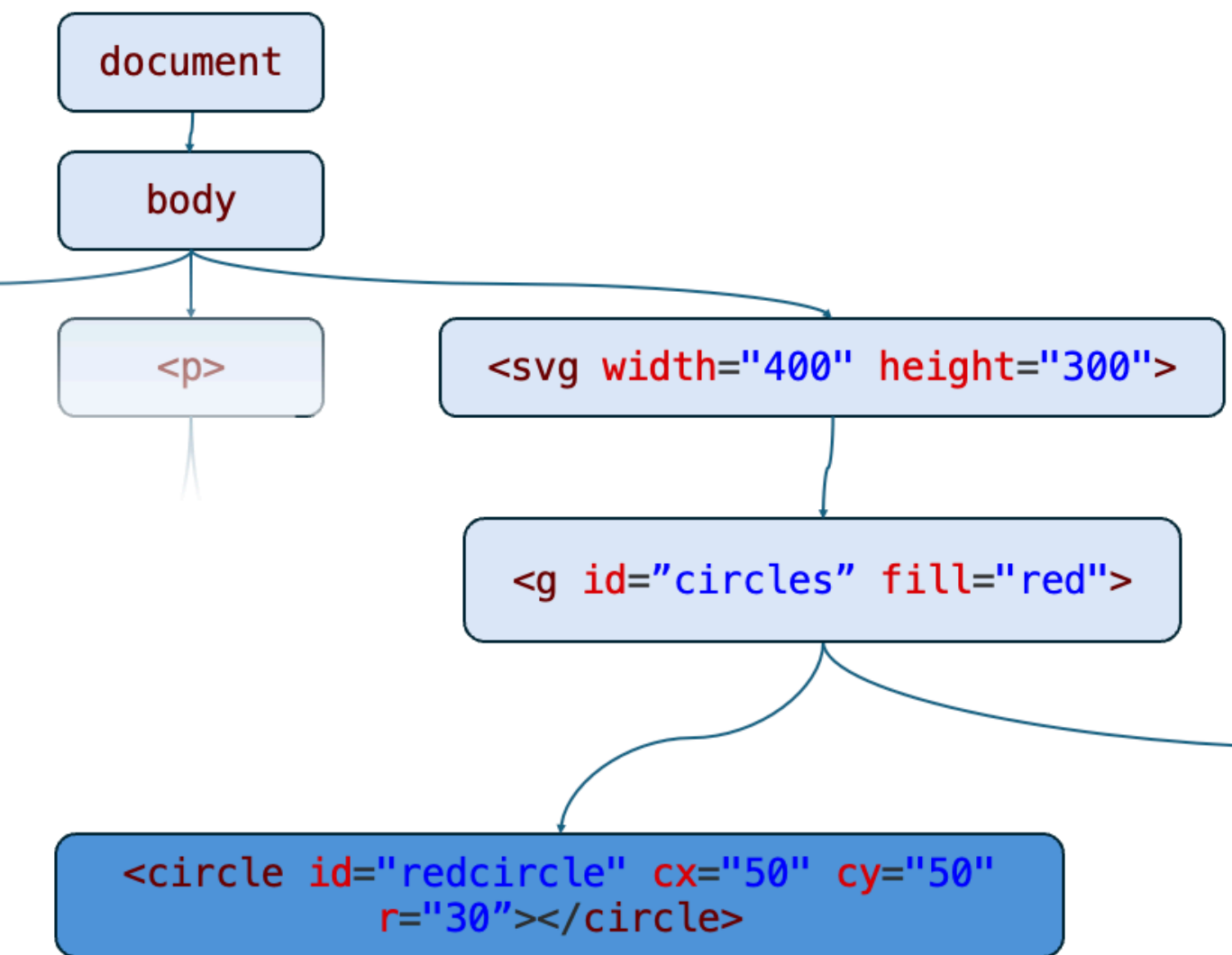
D3.js



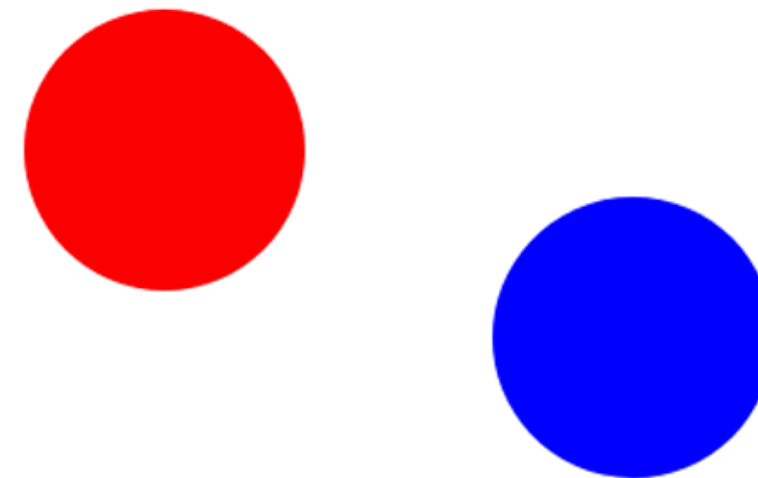
Industry-standard for data visualization on the web

D3: Let's us manipulate the DOM!

Data Visualization



Show Hide



```
Inspector Console Debugger Network
Filter Output Errors Warnin
>> d3.select('#redcircle')
← Object { _groups: (1) [...], _parents: (1) [...] }
```

d3 is loaded by default in observable!

'#' indicates id!

Step 1: Select an element

```
> d3.select('#redcircle')
```

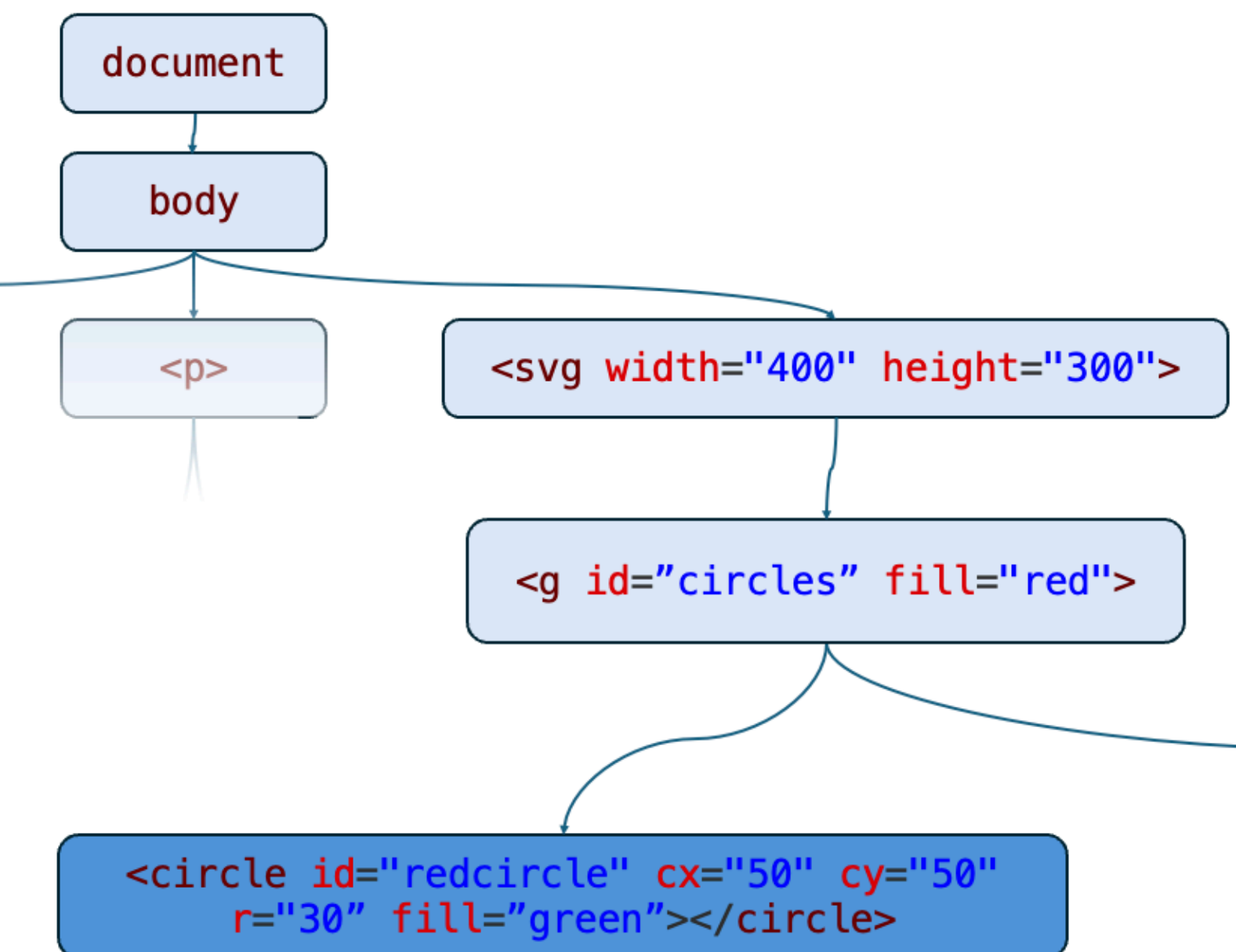
*Search for an element with **id="redcircle"***

D3: Let's us manipulate the DOM!

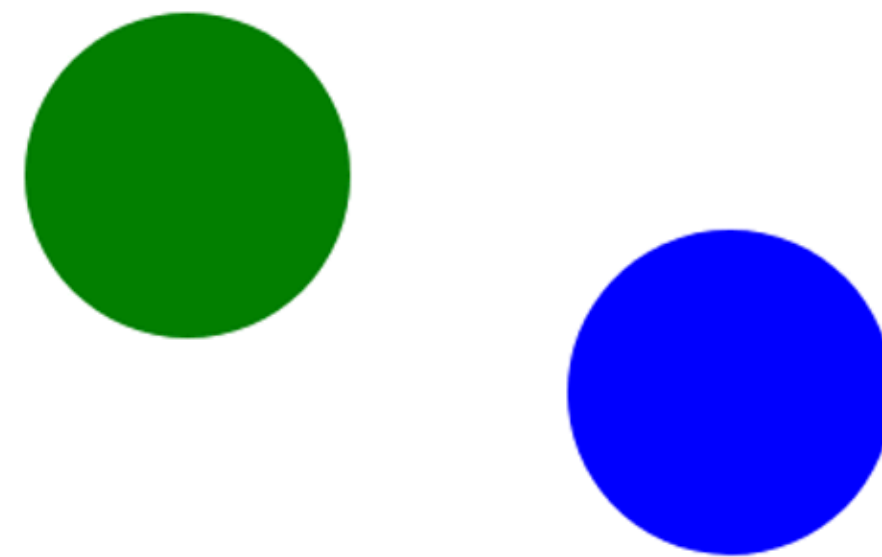
Data Visualization

Step 1: Select an element

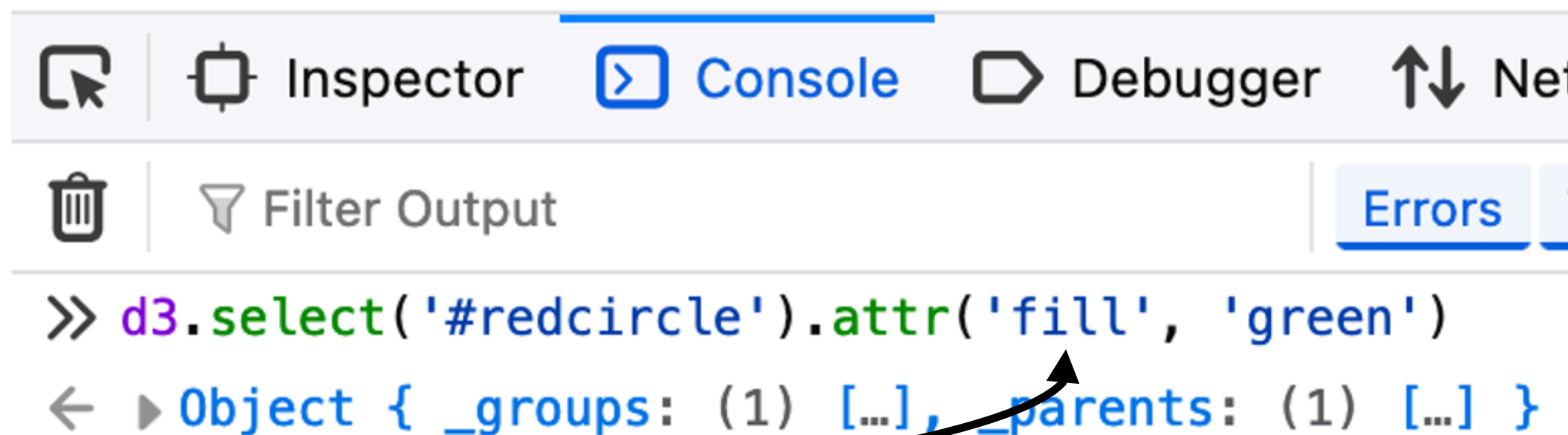
Step 2: Change properties



Show Hide

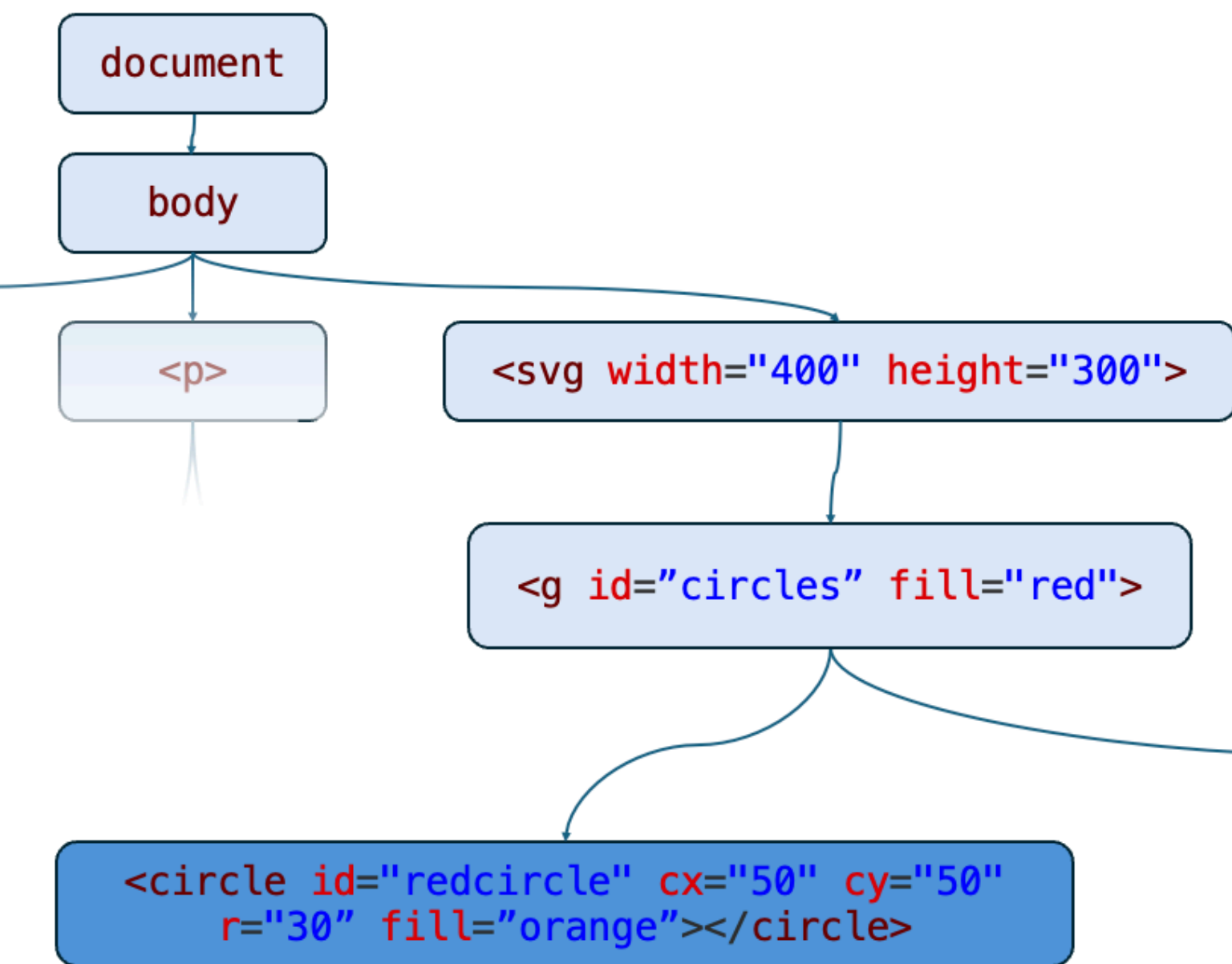


```
d3.select('#redcircle').attr('fill', 'green')
```



Change the **fill** attribute to green (using 'attr')

D3: Let's us manipulate the DOM!

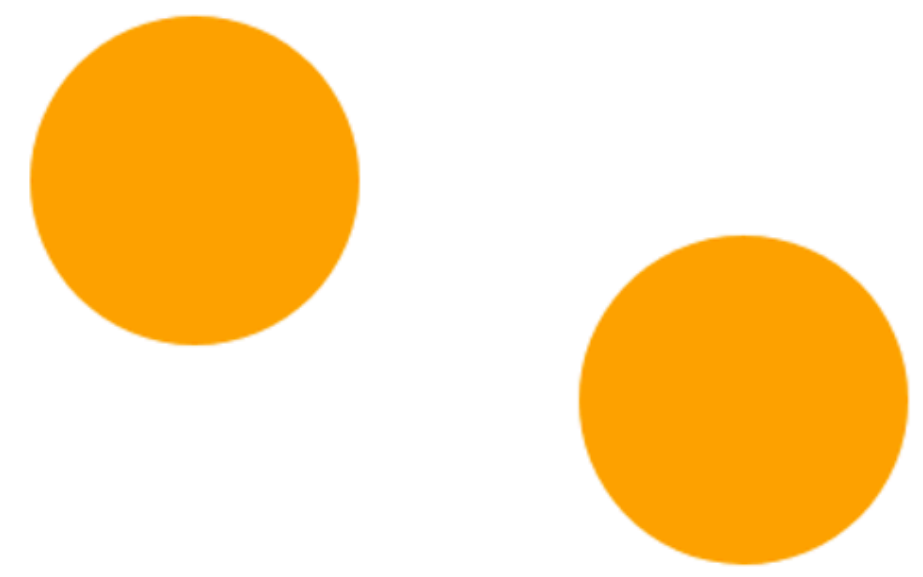


Set fill of all circles to 'orange'

Data Visualization

Show Hide

```
d3.selectAll('circle').attr('fill', 'orange')
```



*Select **all** elements with this property*

Element type is 'circle'

The screenshot shows the browser's developer console with the `Console` tab selected. The code `d3.selectAll('circle').attr('fill', 'orange')` has been executed, and the output is `Object { _groups: (1) [...], _parents: (1) [...] }`. The `Inspector` and `Debugger` tabs are also visible.

Creation with D3

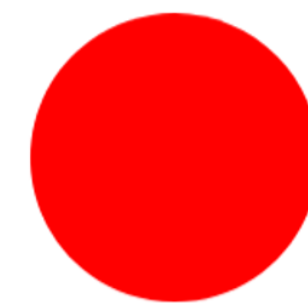
We haven't added it to the DOM so the svg is the root node

<svg>

```
<circle cx="100" cy="75" r="25" fill="red"></circle>
```

D3 wraps the actual element, `node()` gives us the element to show

```
const svg = d3.create("svg");
const circle = svg.append('circle');
circle.attr('cx', '100');
circle.attr('cy', '75');
circle.attr('r', '25');
circle.attr('fill', 'red');
return svg.node();
```



Create an svg element

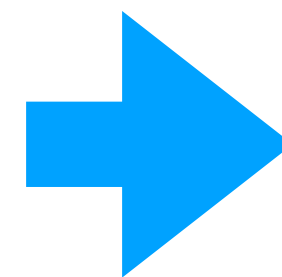
Create a circle element and add it as a child of svg

Method chaining



*Usually more convenient to **chain** methods*

```
{  
  const svg = d3.create("svg");  
  const circle = svg.append('circle');  
  circle.attr('cx', '100');  
  circle.attr('cy', '75');  
  circle.attr('r', '25');  
  circle.attr('fill', 'red');  
  return svg.node();  
}
```



```
{  
  const svg = d3.create("svg");  
  const circle = svg.append('circle')  
    .attr('cx', '100')  
    .attr('cy', '75')  
    .attr('r', '25')  
    .attr('fill', 'red');  
  return svg.node();  
}
```

***Append** returns the created element,
attr returns what it was called on*

***A few* Javascript tools**

Arrays and objects

Arrays: like Python lists

```
myarray = ▶ Array(5) [1, 2, 3, 4, 5]
```

```
myarray = (  
  [1, 2, 3, 4, 5]  
)
```

Getting and setting values

3

```
myarray[2]
```

7

```
myarray[2] = 7
```

*Can also access values
using property syntax!*



"penguin"

```
myobject.animal
```

"dolphin"

```
myobject.animal = 'dolphin'
```

Objects: (mostly) like Python dicts

```
myobject = ▶ Object {animal: "penguin", age: 3, height: 63}
```

```
myobject = (  
  {animal: "penguin", age: 3, height: 63}  
)
```

Getting and setting values

"penguin"

```
myobject['animal']
```

"dolphin"

```
myobject['animal'] = 'dolphin'
```

*Keys **must** be
strings, so omit
quotes*

Loops: more like C/C++

```
myarray = ▶ Array(5) [1, 2, 3, 4, 5]
```

```
myarray = (  
  [1, 2, 3, 4, 5]  
)
```

```
▶ Array(5) [2, 4, 14, 8, 10]
```

```
{  
  for (let i = 0; i < myarray.length; i++) {  
    myarray[i] *= 2  
  }  
  return myarray  
}
```

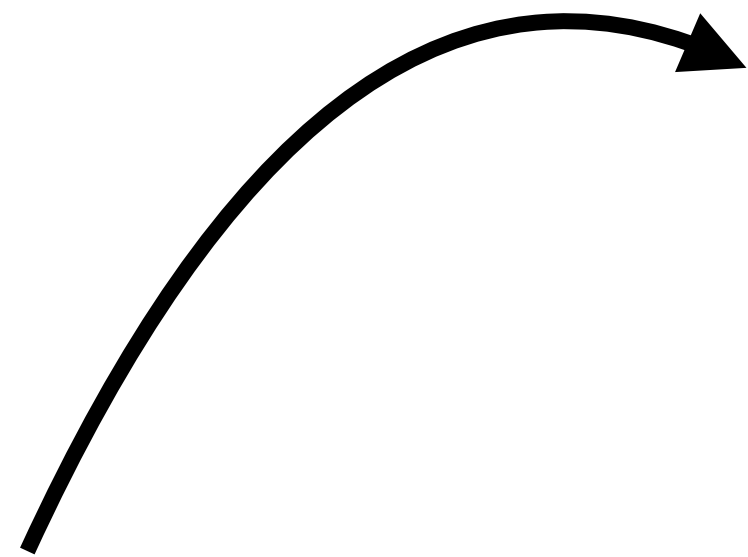
Get length of array



We will rarely write loops ourselves in this class!

Function syntax

Cell output



5

```
{
```

```
  function myfunction(x) {
```

```
    return x + 1;
```

```
  }
```

```
  return myfunction(4)
```

```
}
```

Passing functions as arguments

13

```
{  
  function myfunction(x, g) {  
    return x + 1 + g(x);  
  }  
  
  const factor = 2;  
  function multiply(x) {  
    return x * factor;  
  }  
  
  return myfunction(4, multiply)  
}
```

*Takes a function as
an argument*

*..and calls it
internally!*

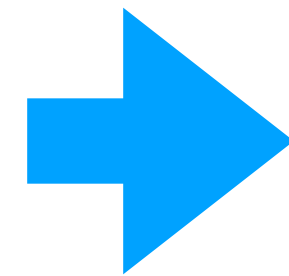
*Pass the multiply
function here*

*Functions can
reference external
variables (as in python)*

Anonymous functions

13

```
{  
  function myfunction(x, g) {  
    return x + 1 + g(x);  
  }  
  
  function multiply(x) {  
    return x * 2;  
  }  
  
  return myfunction(4, multiply)  
}
```



13

```
{  
  function myfunction(x, g) {  
    return x + 1 + g(x);  
  }  
  
  return myfunction(4, (x) => x * 2);  
}
```

Arguments

*Expression to
evaluate*

Anonymous functions

13

```
{  
  function myfunction(x, g) {  
    return x + 1 + g(x);  
  }  
  
  return myfunction(4, (x) => {let d = x * 2; return d;})  
}
```

*Use code block ({...}) if
function requires multiple
lines (or returns an object)*

Data Wrangling **Revisited**

Data representation in Javascript

Represent a data table as an array of objects

Pandas

Columns

Columns

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

```
▼ Array(344) [  
  0: ▼ Object {  
    species: "Adelie"  
    island: "Torgersen"  
    culmen_length_mm: 39.1  
    culmen_depth_mm: 18.7  
    flipper_length_mm: 181  
    body_mass_g: 3750  
    sex: "MALE"  
  }  
  1: ▼ Object {  
    species: "Adelie"  
    island: "Torgersen"  
    culmen_length_mm: 39.5  
    culmen_depth_mm: 17.4  
    flipper_length_mm: 186  
    body_mass_g: 3800  
    sex: "FEMALE"  
  }  
]
```

rows

ROWS

Javascript

Data representation in Javascript

Represent a data table as an array of objects

In Quarto

```
```{python}
import seaborn as sns

penguins = sns.load_dataset('penguins')
penguins_json = penguins.to_json(orient='records')
ojs_define(penguins=penguins_json)
```
```

*Convert to JS
representation*

```
```{ojs}
penguins
```
```

***ojs_define:** Pass value from Python to JS*

Columns

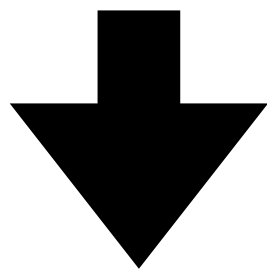
```
▼ Array(344) [
  0: ▼ Object {
    species: "Adelie"
    island: "Torgersen"
    culmen_length_mm: 39.1
    culmen_depth_mm: 18.7
    flipper_length_mm: 181
    body_mass_g: 3750
    sex: "MALE"
  }
  1: ▼ Object {
    species: "Adelie"
    island: "Torgersen"
    culmen_length_mm: 39.5
    culmen_depth_mm: 17.4
    flipper_length_mm: 186
    body_mass_g: 3800
    sex: "FEMALE"
  }
]
```

rows

Filtering data

species == "Adelie"

| species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex |
|-----------|-----------|----------------|---------------|-------------------|-------------|--------|
| Gentoo | Biscoe | 45.2 | 15.8 | 215.0 | 5300.0 | Male |
| Adelie | Dream | 40.6 | 17.2 | 187.0 | 3475.0 | Male |
| Adelie | Torgersen | 34.1 | 18.1 | 193.0 | 3475.0 | None |
| Adelie | Torgersen | 37.7 | 19.8 | 198.0 | 3500.0 | Male |
| Gentoo | Biscoe | 50.8 | 17.3 | 228.0 | 5600.0 | Male |
| Adelie | Biscoe | 38.2 | 20.0 | 190.0 | 3900.0 | Male |
| Adelie | Biscoe | 45.6 | 20.3 | 191.0 | 4600.0 | Male |
| Gentoo | Biscoe | 40.9 | 13.7 | 214.0 | 4650.0 | Female |
| Gentoo | Biscoe | 44.0 | 13.6 | 208.0 | 4350.0 | Female |
| Chinstrap | Dream | 51.3 | 18.2 | 197.0 | 3750.0 | Male |
| Adelie | Biscoe | 41.3 | 21.1 | 195.0 | 4400.0 | Male |
| Adelie | Torgersen | 33.5 | 19.0 | 190.0 | 3600.0 | Female |
| Gentoo | Biscoe | 46.2 | 14.5 | 209.0 | 4800.0 | Female |
| Adelie | Dream | 35.7 | 18.0 | 202.0 | 3550.0 | Female |
| Adelie | Torgersen | 40.9 | 16.8 | 191.0 | 3700.0 | Female |
| Chinstrap | Dream | 50.2 | 18.8 | 202.0 | 3800.0 | Male |
| Adelie | Dream | 41.1 | 17.5 | 190.0 | 3900.0 | Male |



| species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex |
|---------|--------|----------------|---------------|-------------------|-------------|--------|
| Gentoo | Biscoe | 45.2 | 15.8 | 215.0 | 5300.0 | Male |
| Gentoo | Biscoe | 50.8 | 17.3 | 228.0 | 5600.0 | Male |
| Gentoo | Biscoe | 40.9 | 13.7 | 214.0 | 4650.0 | Female |
| Gentoo | Biscoe | 44.0 | 13.6 | 208.0 | 4350.0 | Female |
| Gentoo | Biscoe | 46.2 | 14.5 | 209.0 | 4800.0 | Female |

Python

```
penguins.query("species == 'Adelie'")
```

Javascript (w/d3)

```
d3.filter(penguins, (d, i) => d.species == "Adelie")
```

Keep rows where expression is true

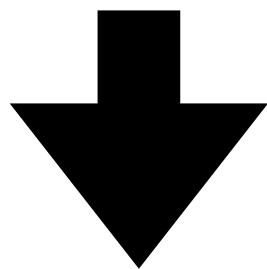
Row object *Index*

```
0: ▼ Object {  
  species: "Adelie"  
  island: "Torgersen"  
  culmen_length_mm: 39.1  
  culmen_depth_mm: 18.7  
  flipper_length_mm: 181  
  body_mass_g: 3750  
  sex: "MALE"  
}
```

Filtering data

species == "Adelie"

| species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex |
|-----------|-----------|----------------|---------------|-------------------|-------------|--------|
| Gentoo | Biscoe | 45.2 | 15.8 | 215.0 | 5300.0 | Male |
| Adelie | Dream | 40.6 | 17.2 | 187.0 | 3475.0 | Male |
| Adelie | Torgersen | 34.1 | 18.1 | 193.0 | 3475.0 | None |
| Adelie | Torgersen | 37.7 | 19.8 | 198.0 | 3500.0 | Male |
| Gentoo | Biscoe | 50.8 | 17.3 | 228.0 | 5600.0 | Male |
| Adelie | Biscoe | 38.2 | 20.0 | 190.0 | 3900.0 | Male |
| Adelie | Biscoe | 45.6 | 20.3 | 191.0 | 4600.0 | Male |
| Gentoo | Biscoe | 40.9 | 13.7 | 214.0 | 4650.0 | Female |
| Gentoo | Biscoe | 44.0 | 13.6 | 208.0 | 4350.0 | Female |
| Chinstrap | Dream | 51.3 | 18.2 | 197.0 | 3750.0 | Male |
| Adelie | Biscoe | 41.3 | 21.1 | 195.0 | 4400.0 | Male |
| Adelie | Torgersen | 33.5 | 19.0 | 190.0 | 3600.0 | Female |
| Gentoo | Biscoe | 46.2 | 14.5 | 209.0 | 4800.0 | Female |
| Adelie | Dream | 35.7 | 18.0 | 202.0 | 3550.0 | Female |
| Adelie | Torgersen | 40.9 | 16.8 | 191.0 | 3700.0 | Female |
| Chinstrap | Dream | 50.2 | 18.8 | 202.0 | 3800.0 | Male |
| Adelie | Dream | 41.1 | 17.5 | 190.0 | 3900.0 | Male |



| species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex |
|---------|--------|----------------|---------------|-------------------|-------------|--------|
| Gentoo | Biscoe | 45.2 | 15.8 | 215.0 | 5300.0 | Male |
| Gentoo | Biscoe | 50.8 | 17.3 | 228.0 | 5600.0 | Male |
| Gentoo | Biscoe | 40.9 | 13.7 | 214.0 | 4650.0 | Female |
| Gentoo | Biscoe | 44.0 | 13.6 | 208.0 | 4350.0 | Female |
| Gentoo | Biscoe | 46.2 | 14.5 | 209.0 | 4800.0 | Female |

Python

```
penguins.query("species == 'Adelie'")
```

Javascript (w/d3)

```
d3.filter(penguins, (d, i) => d.species == "Adelie")
```

Keep rows where expression is true

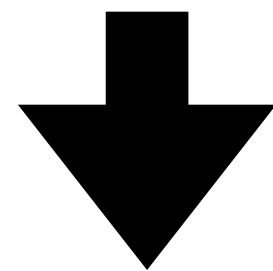
Row object *Index*

```
0: ▼Object {  
  species: "Adelie"  
  island: "Torgersen"  
  culmen_length_mm: 39.1  
  culmen_depth_mm: 18.7  
  flipper_length_mm: 181  
  body_mass_g: 3750  
  sex: "MALE"  
}
```

Sorting data

By body_mass_g

| species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex |
|-----------|-----------|----------------|---------------|-------------------|-------------|--------|
| Gentoo | Biscoe | 45.2 | 15.8 | 215.0 | 5300.0 | Male |
| Adelie | Dream | 40.6 | 17.2 | 187.0 | 3475.0 | Male |
| Adelie | Torgersen | 34.1 | 18.1 | 193.0 | 3475.0 | None |
| Adelie | Torgersen | 37.7 | 19.8 | 198.0 | 3500.0 | Male |
| Gentoo | Biscoe | 50.8 | 17.3 | 228.0 | 5600.0 | Male |
| Adelie | Biscoe | 38.2 | 20.0 | 190.0 | 3900.0 | Male |
| Adelie | Biscoe | 45.6 | 20.3 | 191.0 | 4600.0 | Male |
| Gentoo | Biscoe | 40.9 | 13.7 | 214.0 | 4650.0 | Female |
| Gentoo | Biscoe | 44.0 | 13.6 | 208.0 | 4350.0 | Female |
| Chinstrap | Dream | 51.3 | 18.2 | 197.0 | 3750.0 | Male |
| Adelie | Biscoe | 41.3 | 21.1 | 195.0 | 4400.0 | Male |
| Adelie | Torgersen | 33.5 | 19.0 | 190.0 | 3600.0 | Female |
| Gentoo | Biscoe | 46.2 | 14.5 | 209.0 | 4800.0 | Female |
| Adelie | Dream | 35.7 | 18.0 | 202.0 | 3550.0 | Female |
| Adelie | Torgersen | 40.9 | 16.8 | 191.0 | 3700.0 | Female |
| Chinstrap | Dream | 50.2 | 18.8 | 202.0 | 3800.0 | Male |
| Adelie | Dream | 41.1 | 17.5 | 190.0 | 3900.0 | Male |



| species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex |
|-----------|-----------|----------------|---------------|-------------------|-------------|--------|
| Gentoo | Biscoe | 50.8 | 17.3 | 228.0 | 5600.0 | Male |
| Gentoo | Biscoe | 45.2 | 15.8 | 215.0 | 5300.0 | Male |
| Gentoo | Biscoe | 46.2 | 14.5 | 209.0 | 4800.0 | Female |
| Gentoo | Biscoe | 40.9 | 13.7 | 214.0 | 4650.0 | Female |
| Adelie | Biscoe | 45.6 | 20.3 | 191.0 | 4600.0 | Male |
| Adelie | Biscoe | 41.3 | 21.1 | 195.0 | 4400.0 | Male |
| Gentoo | Biscoe | 44.0 | 13.6 | 208.0 | 4350.0 | Female |
| Adelie | Biscoe | 38.2 | 20.0 | 190.0 | 3900.0 | Male |
| Adelie | Dream | 41.1 | 17.5 | 190.0 | 3900.0 | Male |
| Chinstrap | Dream | 50.2 | 18.8 | 202.0 | 3800.0 | Male |
| Chinstrap | Dream | 51.3 | 18.2 | 197.0 | 3750.0 | Male |
| Adelie | Torgersen | 40.9 | 16.8 | 191.0 | 3700.0 | Female |
| Adelie | Torgersen | 33.5 | 19.0 | 190.0 | 3600.0 | Female |
| Adelie | Dream | 35.7 | 18.0 | 202.0 | 3550.0 | Female |
| Adelie | Torgersen | 37.7 | 19.8 | 198.0 | 3500.0 | Male |
| Adelie | Dream | 40.6 | 17.2 | 187.0 | 3475.0 | Male |
| Adelie | Torgersen | 34.1 | 18.1 | 193.0 | 3475.0 | None |

Python

```
penguins.sort_values(by = "body_mass_g")
```

Javascript (w/d3)

More robust than "a.body_mass_g < b.body_mass_g"

```
d3.sort(penguins, (a, b) => d3.ascending(a.body_mass_g, b.body_mass_g))
```

Function to compare if $a < b$

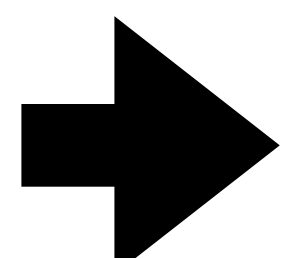
Selecting variables

Python

```
penguins[['species', 'island', 'body_mass_g']]
```

Javascript (w/d3)

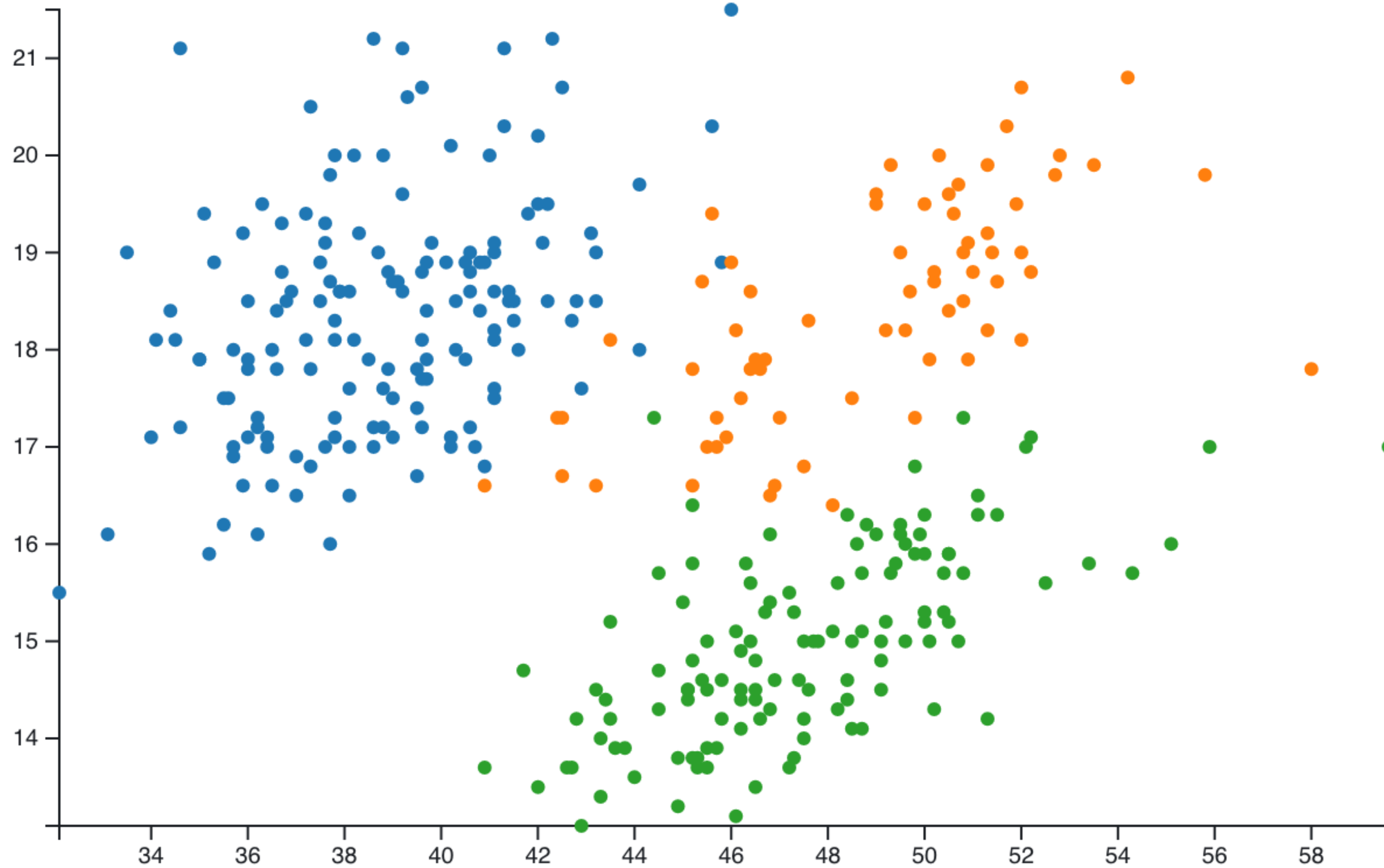
```
d3.map(penguins, (d) => {return {species: d.species, island: d.island, body_mass_g: d.body_mass_g}})
```



| species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex |
|-----------|-----------|----------------|---------------|-------------------|-------------|--------|
| Gentoo | Biscoe | 45.2 | 15.8 | 215.0 | 5300.0 | Male |
| Adelie | Dream | 40.6 | 17.2 | 187.0 | 3475.0 | Male |
| Adelie | Torgersen | 34.1 | 18.1 | 193.0 | 3475.0 | None |
| Adelie | Torgersen | 37.7 | 19.8 | 198.0 | 3500.0 | Male |
| Gentoo | Biscoe | 50.8 | 17.3 | 228.0 | 5600.0 | Male |
| Adelie | Biscoe | 38.2 | 20.0 | 190.0 | 3900.0 | Male |
| Adelie | Biscoe | 45.6 | 20.3 | 191.0 | 4600.0 | Male |
| Gentoo | Biscoe | 40.9 | 13.7 | 214.0 | 4650.0 | Female |
| Gentoo | Biscoe | 44.0 | 13.6 | 208.0 | 4350.0 | Female |
| Chinstrap | Dream | 51.3 | 18.2 | 197.0 | 3750.0 | Male |
| Adelie | Biscoe | 41.3 | 21.1 | 195.0 | 4400.0 | Male |
| Adelie | Torgersen | 33.5 | 19.0 | 190.0 | 3600.0 | Female |
| Gentoo | Biscoe | 46.2 | 14.5 | 209.0 | 4800.0 | Female |
| Adelie | Dream | 35.7 | 18.0 | 202.0 | 3550.0 | Female |
| Adelie | Torgersen | 40.9 | 16.8 | 191.0 | 3700.0 | Female |
| Chinstrap | Dream | 50.2 | 18.8 | 202.0 | 3800.0 | Male |
| Adelie | Dream | 41.1 | 17.5 | 190.0 | 3900.0 | Male |

| species | island | body_mass_g |
|-----------|-----------|-------------|
| Gentoo | Biscoe | 5300.0 |
| Adelie | Dream | 3475.0 |
| Adelie | Torgersen | 3475.0 |
| Adelie | Torgersen | 3500.0 |
| Gentoo | Biscoe | 5600.0 |
| Adelie | Biscoe | 3900.0 |
| Adelie | Biscoe | 4600.0 |
| Gentoo | Biscoe | 4650.0 |
| Gentoo | Biscoe | 4350.0 |
| Chinstrap | Dream | 3750.0 |
| Adelie | Biscoe | 4400.0 |
| Adelie | Torgersen | 3600.0 |
| Gentoo | Biscoe | 4800.0 |
| Adelie | Dream | 3550.0 |
| Adelie | Torgersen | 3700.0 |
| Chinstrap | Dream | 3800.0 |

Creating a plot with D3



Creating a plot with D3

Code

```
chart = {
  const width = 640;
  const height = 400;
  const margin = {top: 20, right: 30, bottom: 30, left: 40};

  const x = d3.scaleLinear()
    .domain(d3.extent(penguins, d => d.culmen_length_mm))
    .range([margin.left, width - margin.right]);

  const y = d3.scaleLinear()
    .domain(d3.extent(penguins, d => d.culmen_depth_mm))
    .range([height - margin.bottom, margin.top]);

  const color = d3.scaleOrdinal(d3.schemeCategory10)
    .domain(d3.map(penguins, d => d.species));

  const svg = d3.create("svg")
    .attr("width", width)
    .attr("height", height);

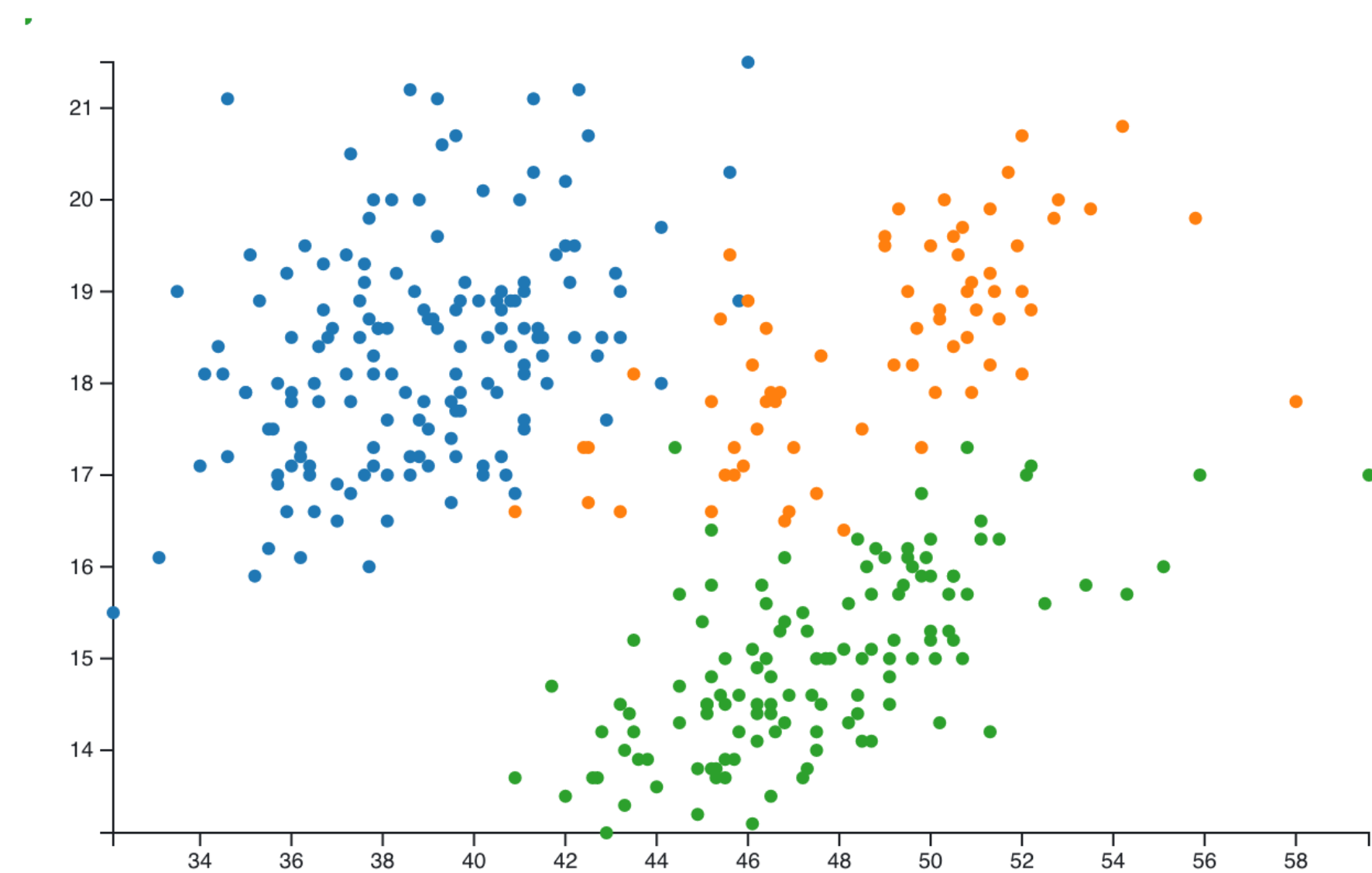
  svg.append("g")
    .attr("transform", `translate(0,${height - margin.bottom})`)
    .call(d3.axisBottom(x));

  svg.append("g")
    .attr("transform", `translate(${margin.left},0)`)
    .call(d3.axisLeft(y));

  svg.selectAll('circle')
    .data(penguins)
    .enter().append('circle')
    .attr('fill', d => color(d.species))
    .attr('cx', d => x(d.culmen_length_mm))
    .attr('cy', d => y(d.culmen_depth_mm))
    .attr('r', 3)

  return svg.node();
}
```

Output



Creating the SVG Element

```
const width = 640;
const height = 400;
const margin = {top: 20, right: 30, bottom: 30, left: 40};

const svg = d3.create("svg")
  .attr("width", width)
  .attr("height", height);
```

```
<svg width="640" height="400" >
```

We've already seen this!

Creating scales

Domain: What is the extent of our data?

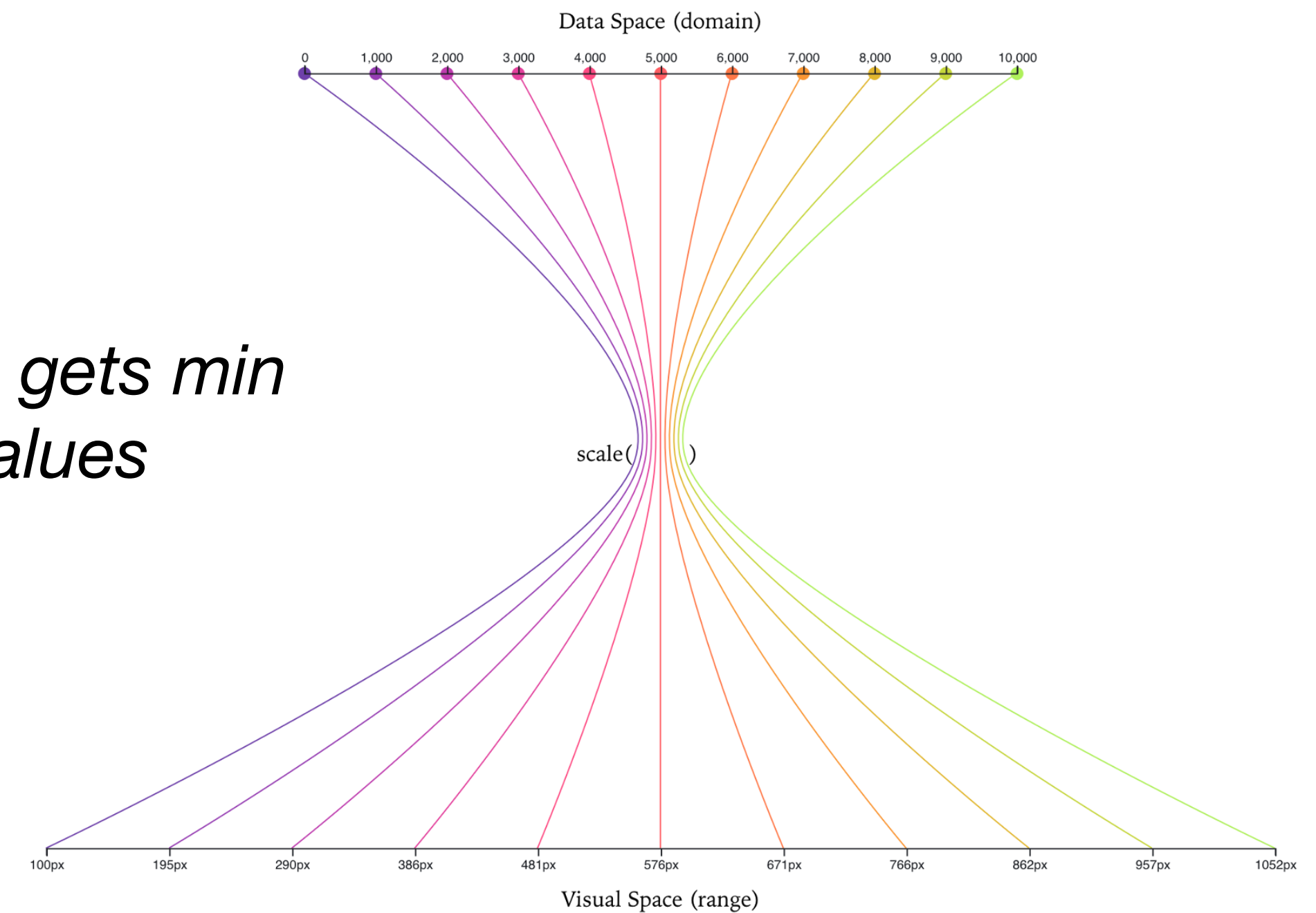
```
const x = d3.scaleLinear()  
  .domain(d3.extent(penguins, d => d.culmen_length_mm))  
  .range([margin.left, width - margin.right]);  
  
const y = d3.scaleLinear()  
  .domain(d3.extent(penguins, d => d.culmen_depth_mm))  
  .range([height - margin.bottom, margin.top]);
```

scaleLinear: Gives us a function!

d3.extent: gets min and max values

Range: How big is our figure?

Transition

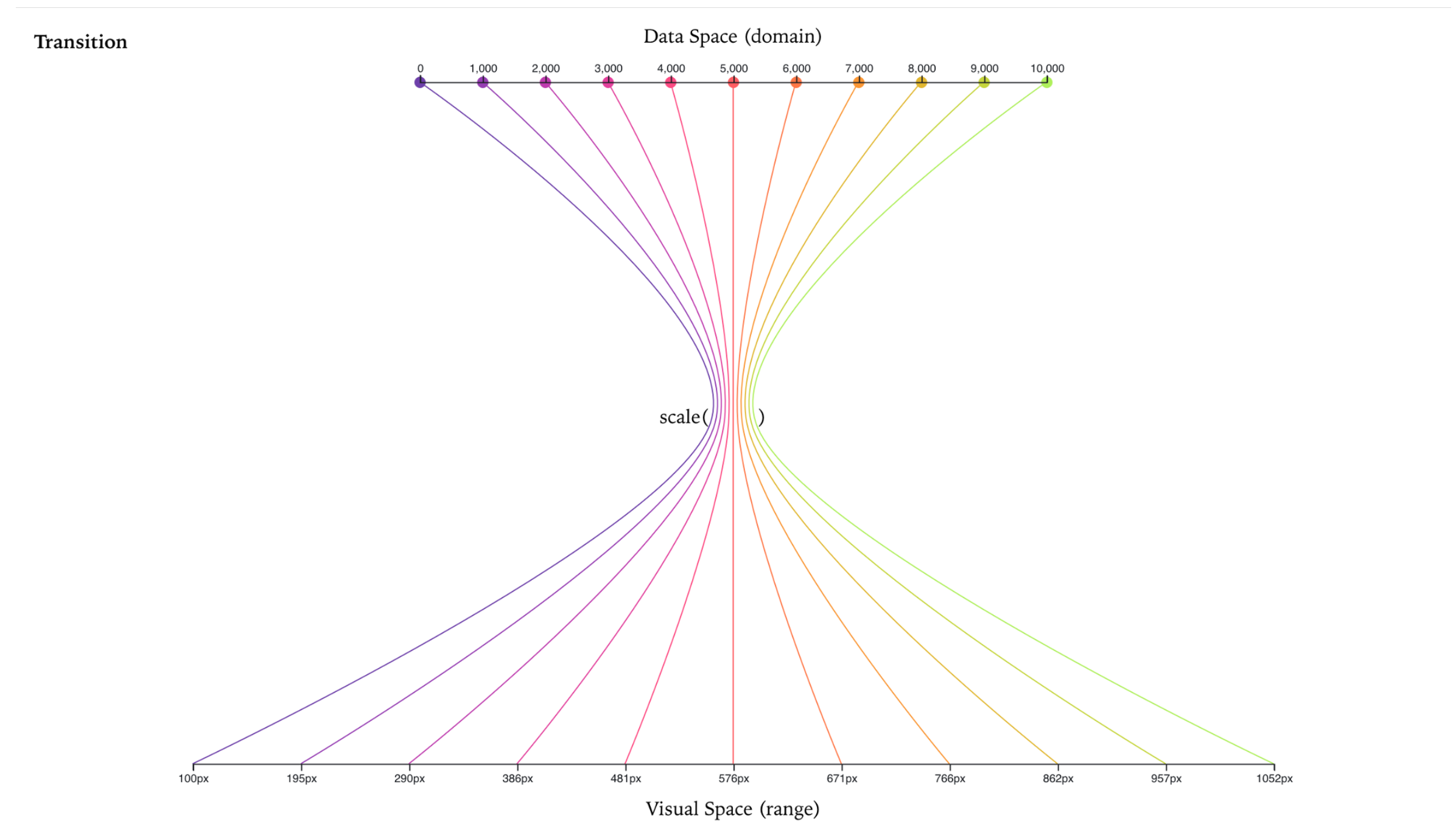


Creating color palettes

Range: What's our color scheme

```
const color = d3.scaleOrdinal(d3.schemeCategory10)  
  .domain(d3.map(penguins, d => d.species));
```

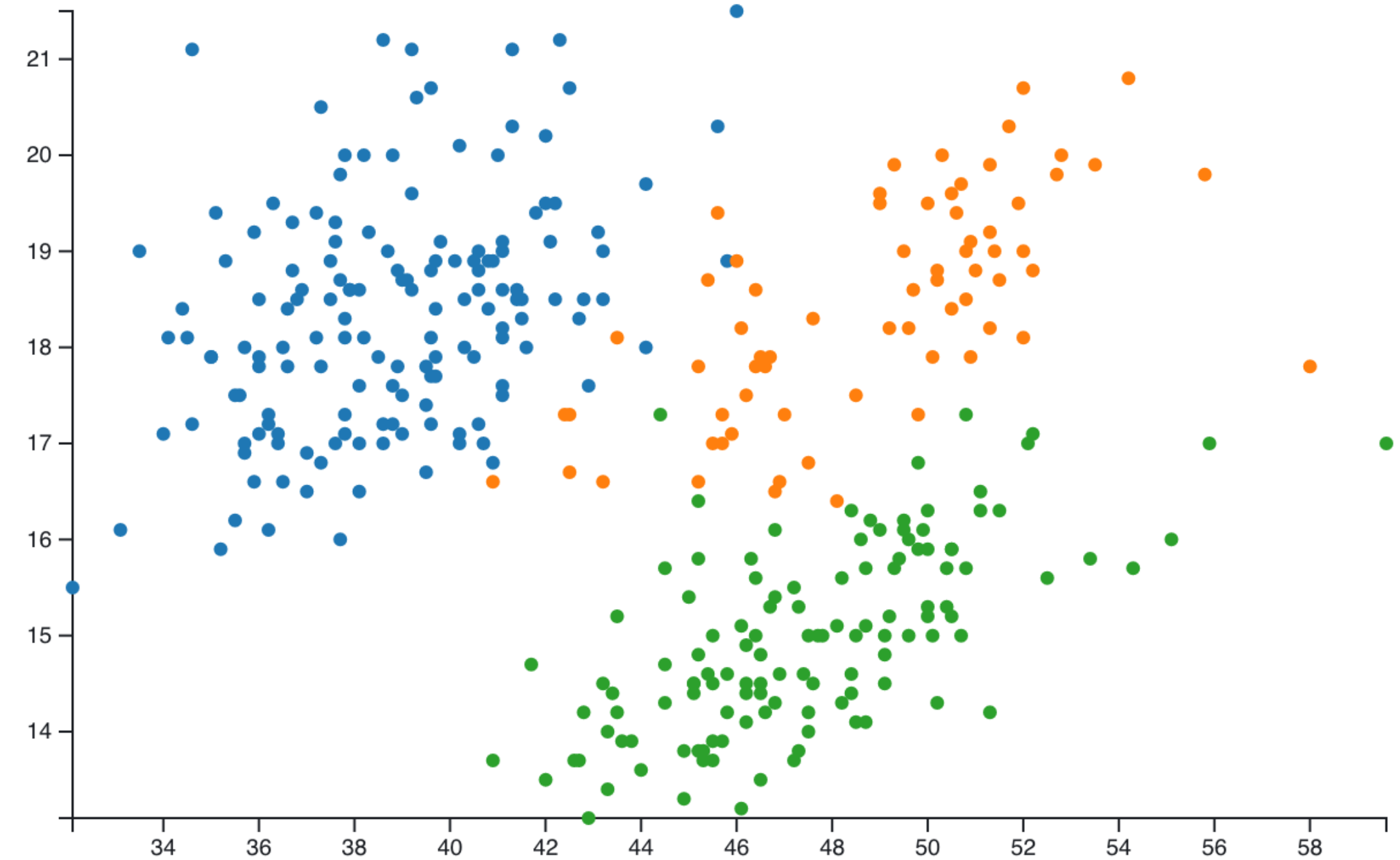
Domain: List of possible values for categorical features



Axis elements

Drawing the axes (will come back to this)

```
svg.append("g")  
  .attr("transform", `translate(0,${height - margin.bottom})`)  
  .call(d3.axisBottom(x));  
  
svg.append("g")  
  .attr("transform", `translate(${margin.left},0)`)  
  .call(d3.axisLeft(y));
```



Drawing the geometry

Set our dataset

Select any existing circles

```
svg.selectAll('circle')  
  .data(penguins)  
  .enter().append('circle')  
  .attr('fill', d => color(d.species))  
  .attr('cx', d => x(d.culmen_length_mm))  
  .attr('cy', d => y(d.culmen_depth_mm))  
  .attr('r', 3)
```

Add a circle for each row

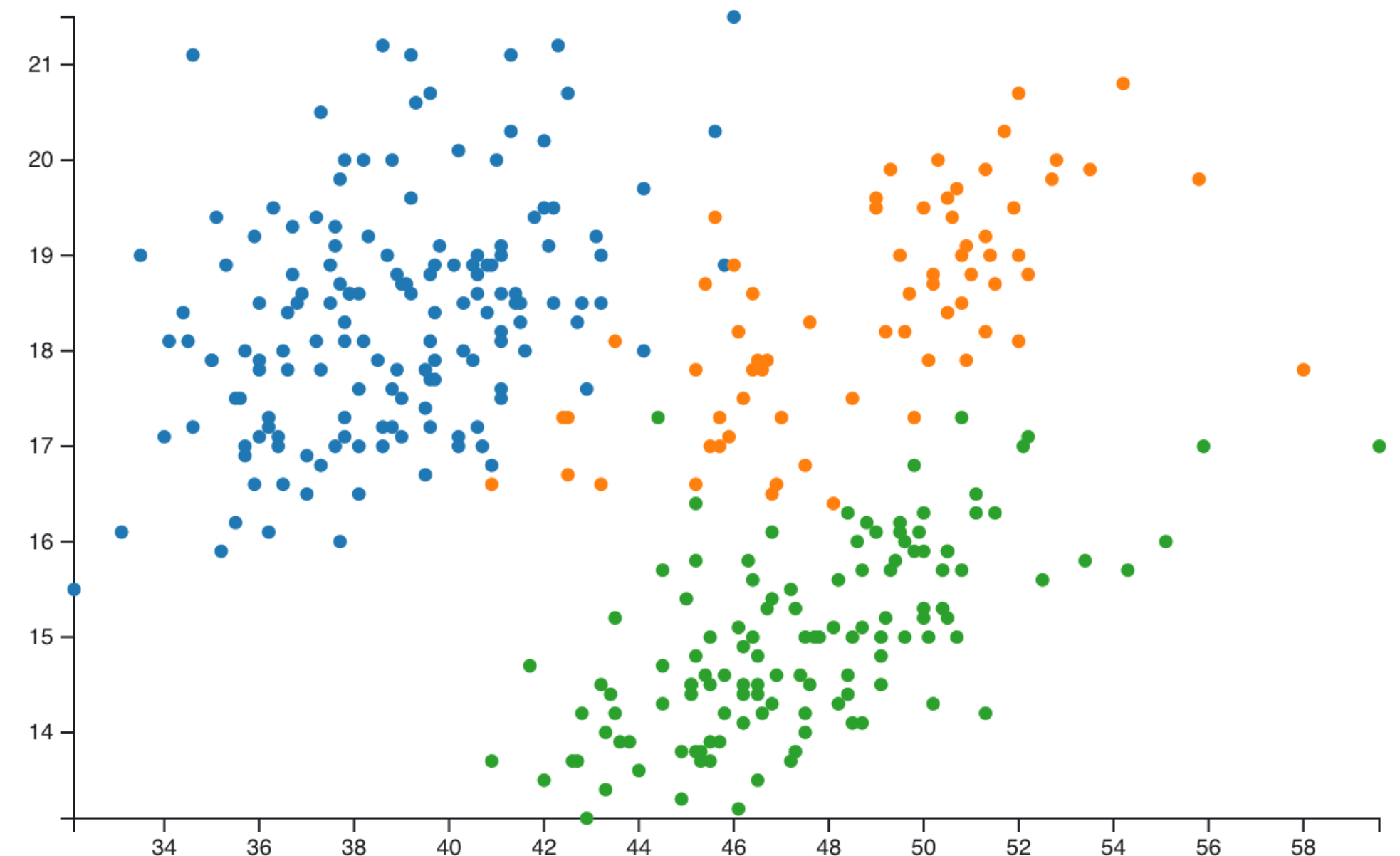
Specify the attributes of each circle

```
<svg width="640" height="400" >
```

```
<circle cx=...>
```

```
<circle cx=...>
```

```
<circle cx=...>
```



Drawing the geometry

Set our dataset

*Select any
existing circles*

```
svg.selectAll('circle')  
  .data(penguins)  
  .enter().append('circle')  
  .attr('fill', d => color(d.species))  
  .attr('cx', d => x(d.culmen_length_mm))  
  .attr('cy', d => y(d.culmen_depth_mm))  
  .attr('r', 3)
```

*Add a circle for
each row*

*Specify the attributes
of each circle*

As pseudo-code:

```
// Iterate through dataset  
for row in penguins {  
  // Create a geometry  
  add circle to svg  
  
  // Set visual aesthetics  
  set circle fill to row.species  
  set circle cx to row.culmen_length_mm  
  set circle cy to row.culmen_depth_mm  
}
```

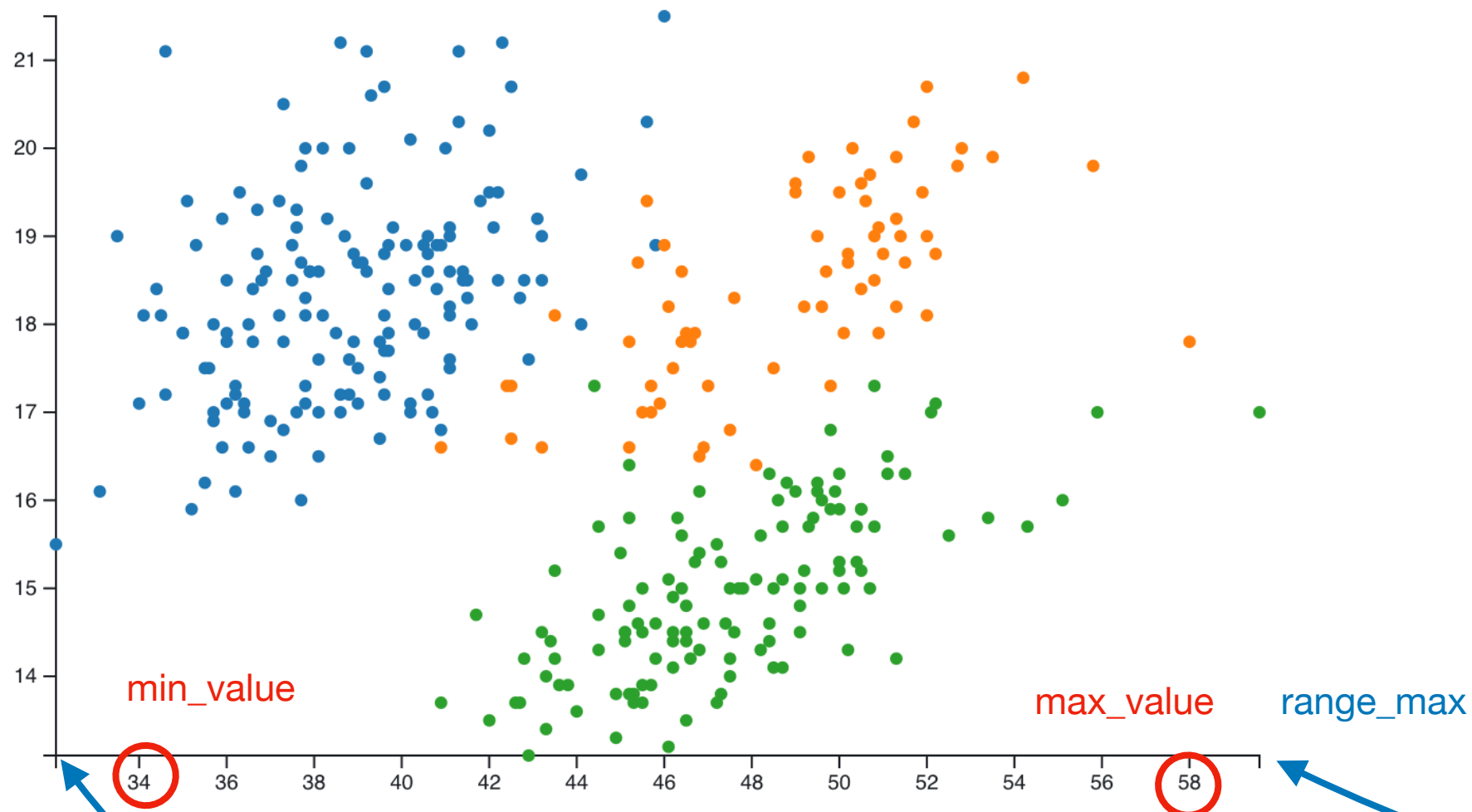
Drawing the geometry

Plot

```
svg.selectAll('circle')  
  .data(penguins)  
  .enter().append('circle')  
  .attr('fill', d => color(d.species))  
  .attr('cx', d => x(d.culmen_length_mm))  
  .attr('cy', d => y(d.culmen_depth_mm))  
  .attr('r', 3)
```

Reminder: creating a scale

```
const x = d3.scaleLinear()  
  .domain(d3.extent(penguins, d => d.culmen_length_mm))  
  .range([margin.left, width - margin.right]);
```



As pseudo-code:

```
function x(value) {  
  scaled_value = (value - max_value) / (max_value - min_value)  
  return value * (range_max - range_min) + range_min  
}
```

Exercise: