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

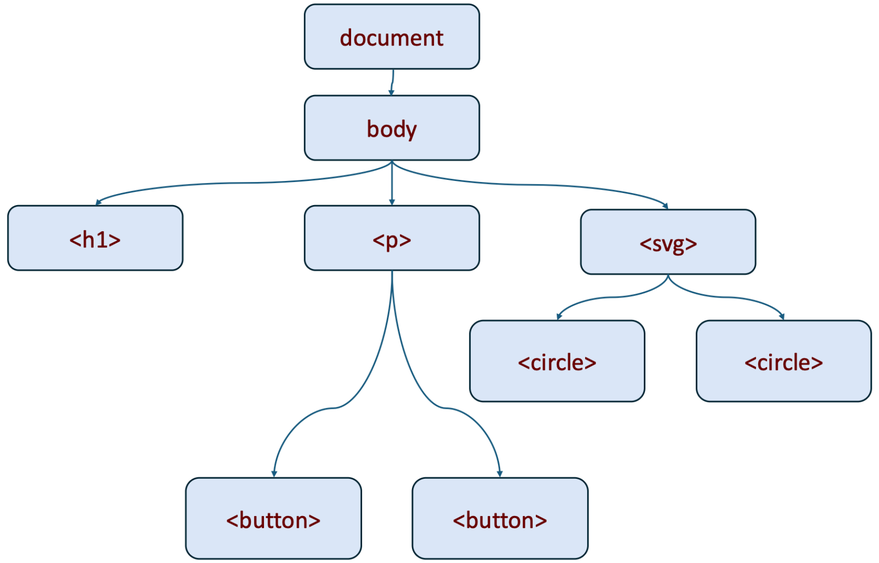
Anatomy of a web page

Hypertext Markup Language (HTML)

```
<!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>
```

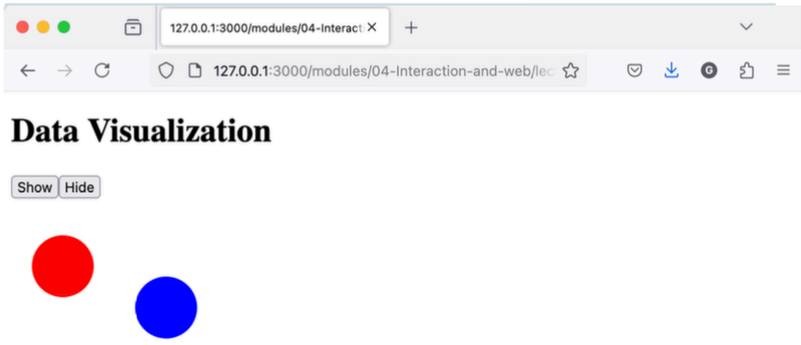
Parses to

Document Object Model (DOM)



Renders to

Browser display



Tree-structured *elements*

Hypertext Markup Language (HTML)

Document Object Model (DOM)

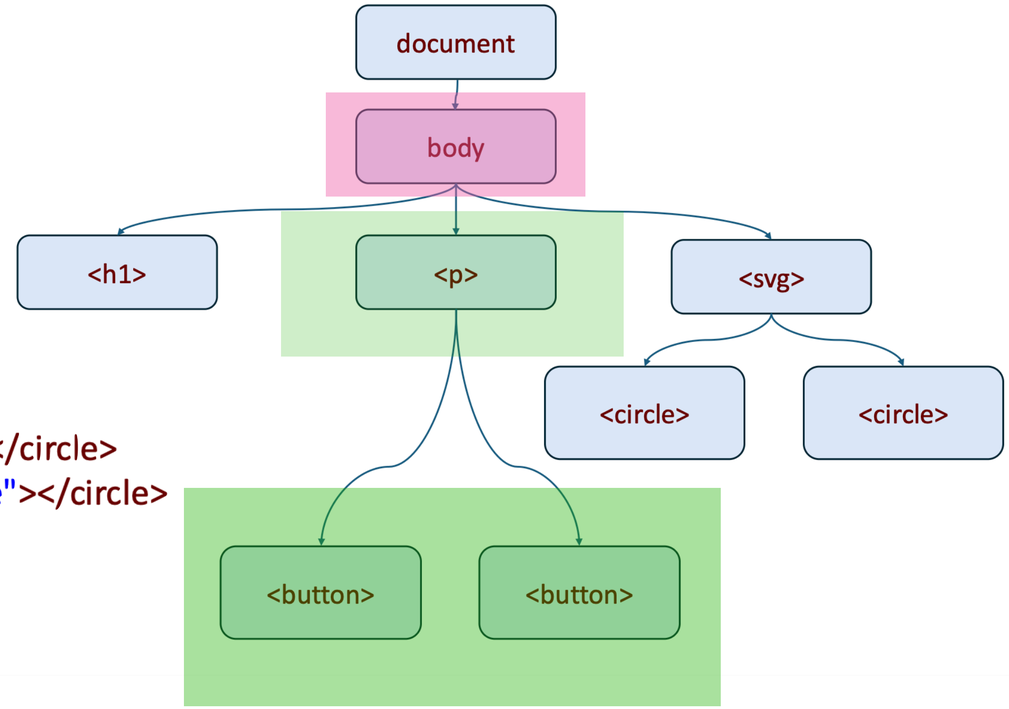
Page

Root element of page content

Parent element

Child elements

```
<!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>
```



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

Start tag:

End tag:

A single element

Hypertext Markup Language (HTML)

Document Object Model (DOM)

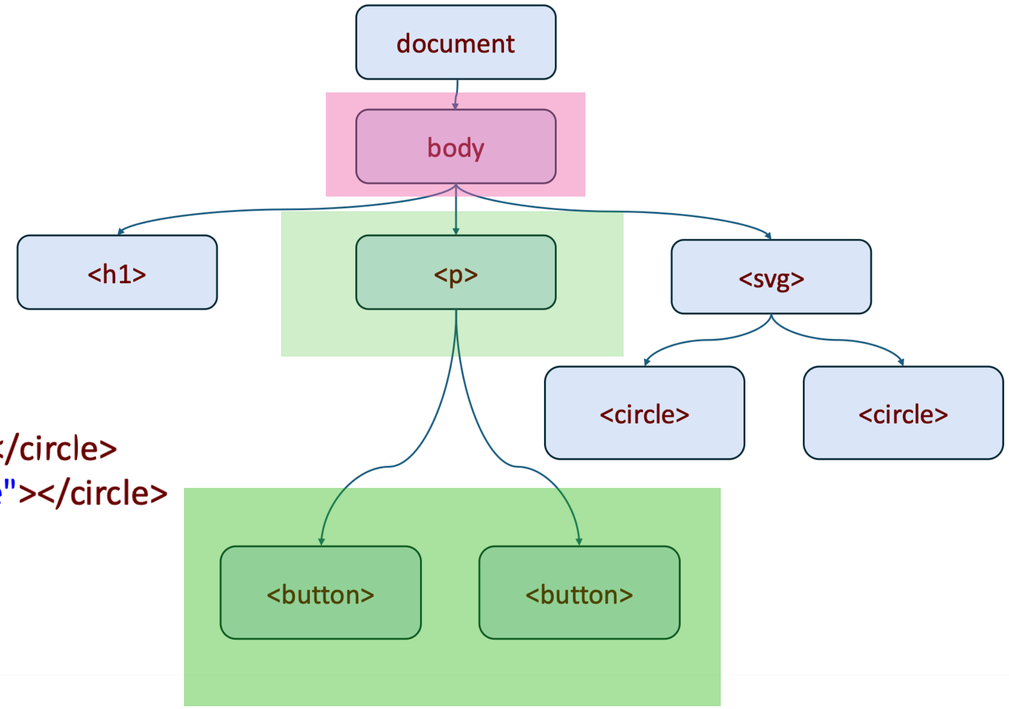
Page

Root element of page content

Parent element

Child elements

```
<!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>
```



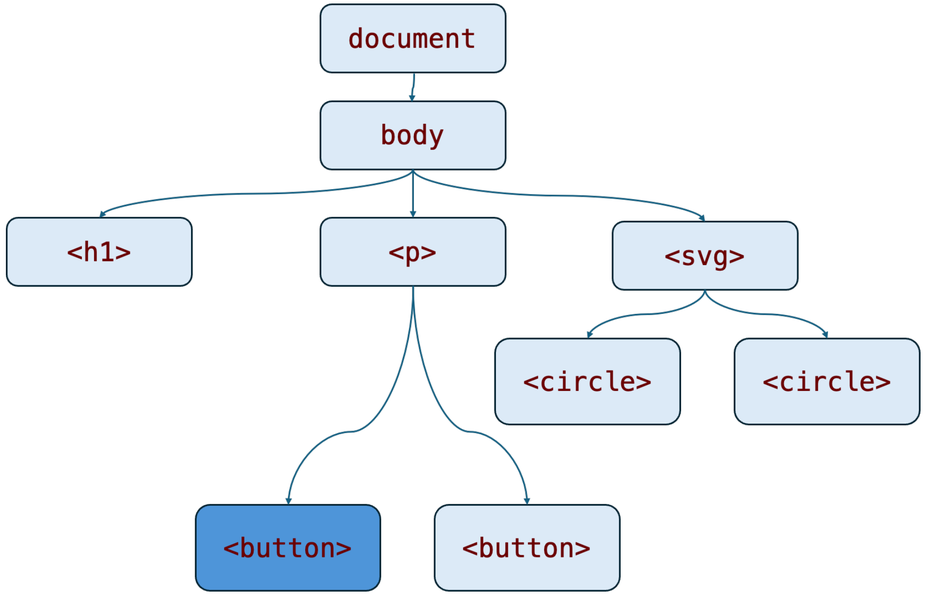
Children contained between *start* and *end tags* of parent

Start tag:

End tag:

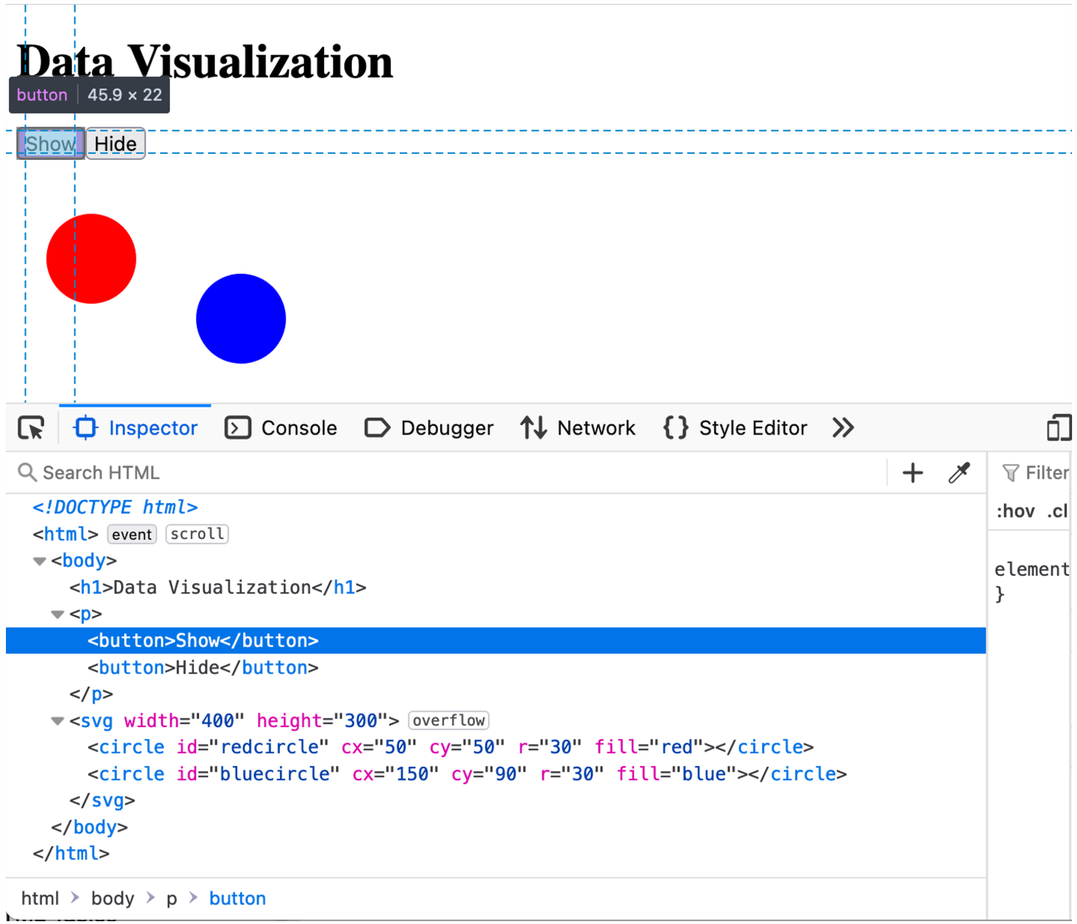
A single element

Document Object Model (DOM)



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

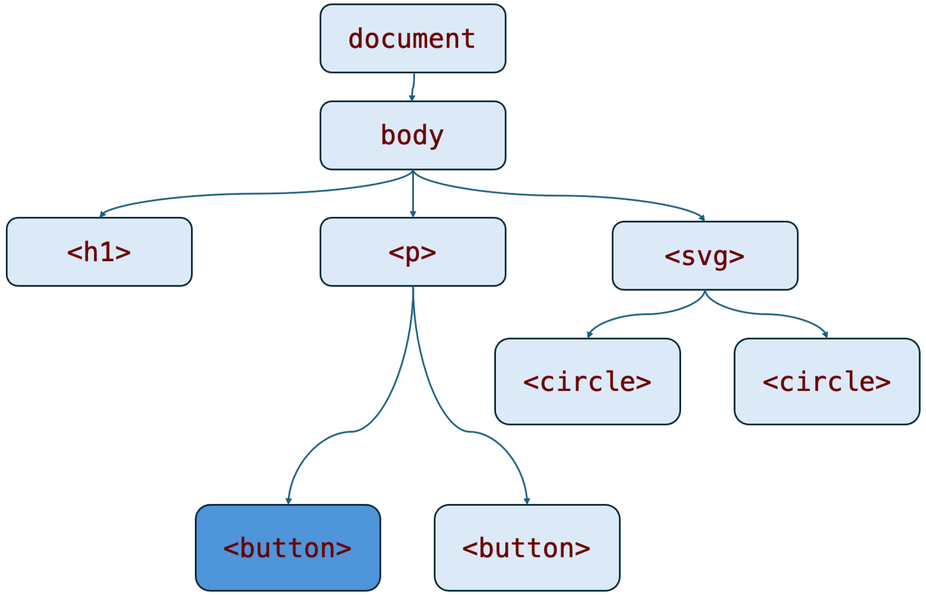
Browser display



Inspector: Browser's DOM visualization - *shown as HTML*

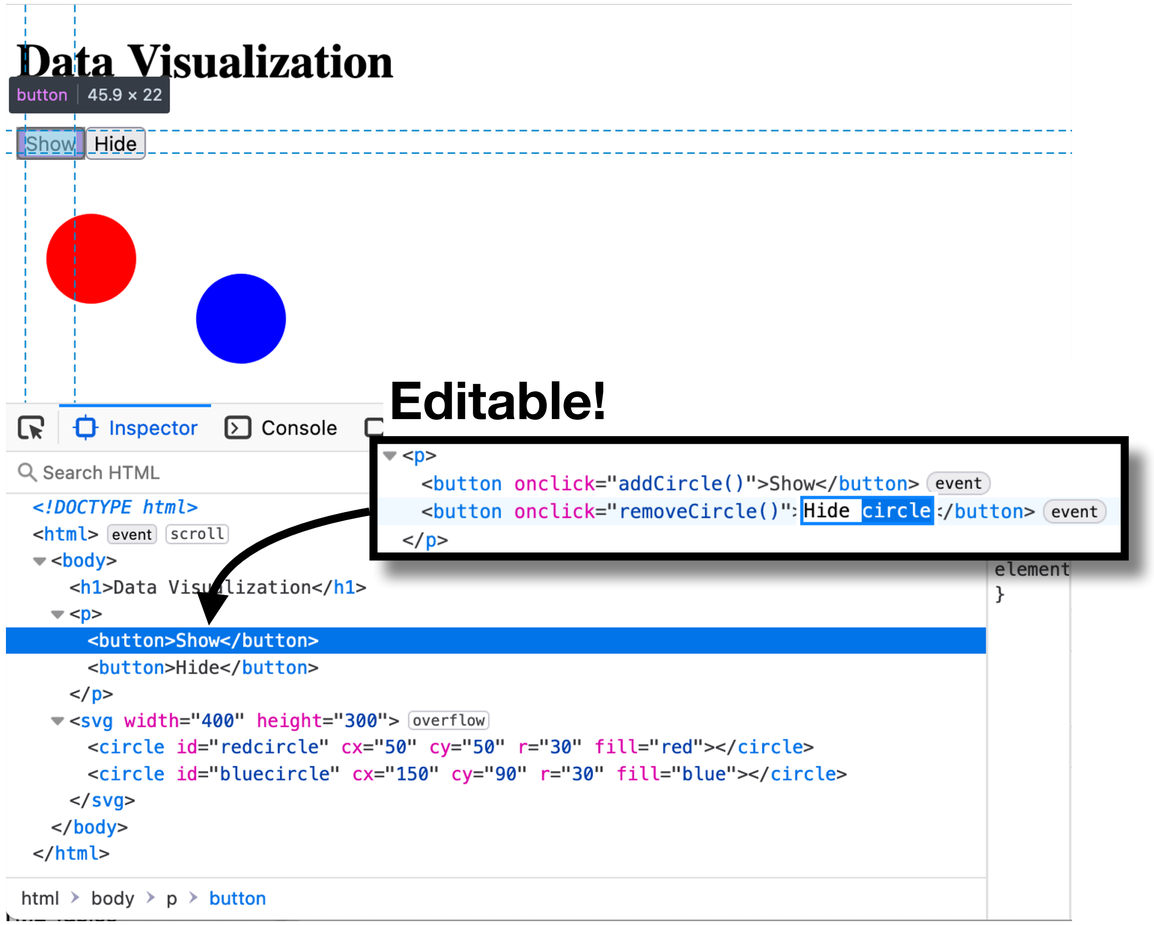
A single element

Document Object Model (DOM)



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

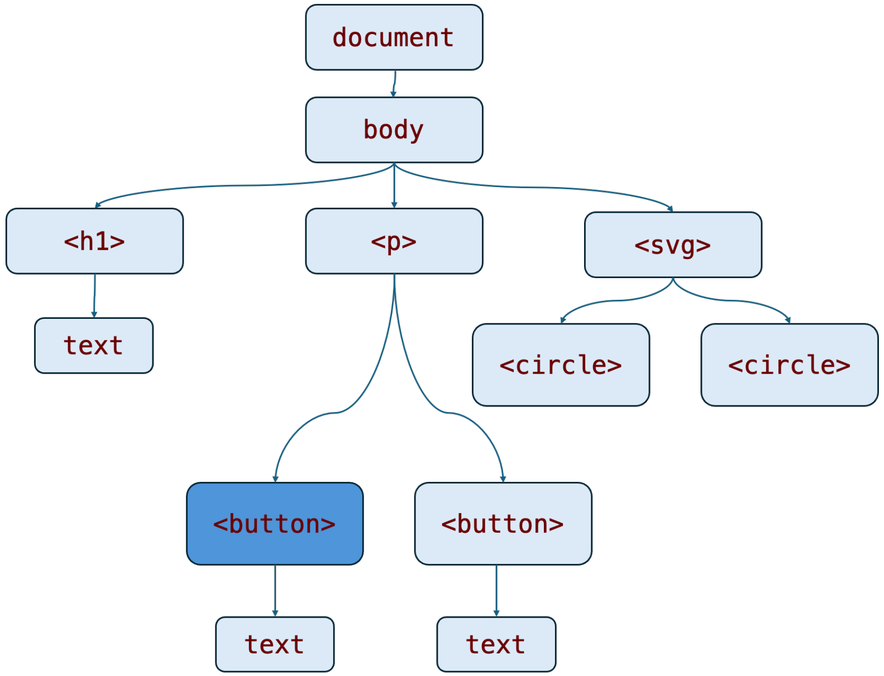
Browser display



Inspector: Browser's DOM visualization - *shown as HTML*

A single element

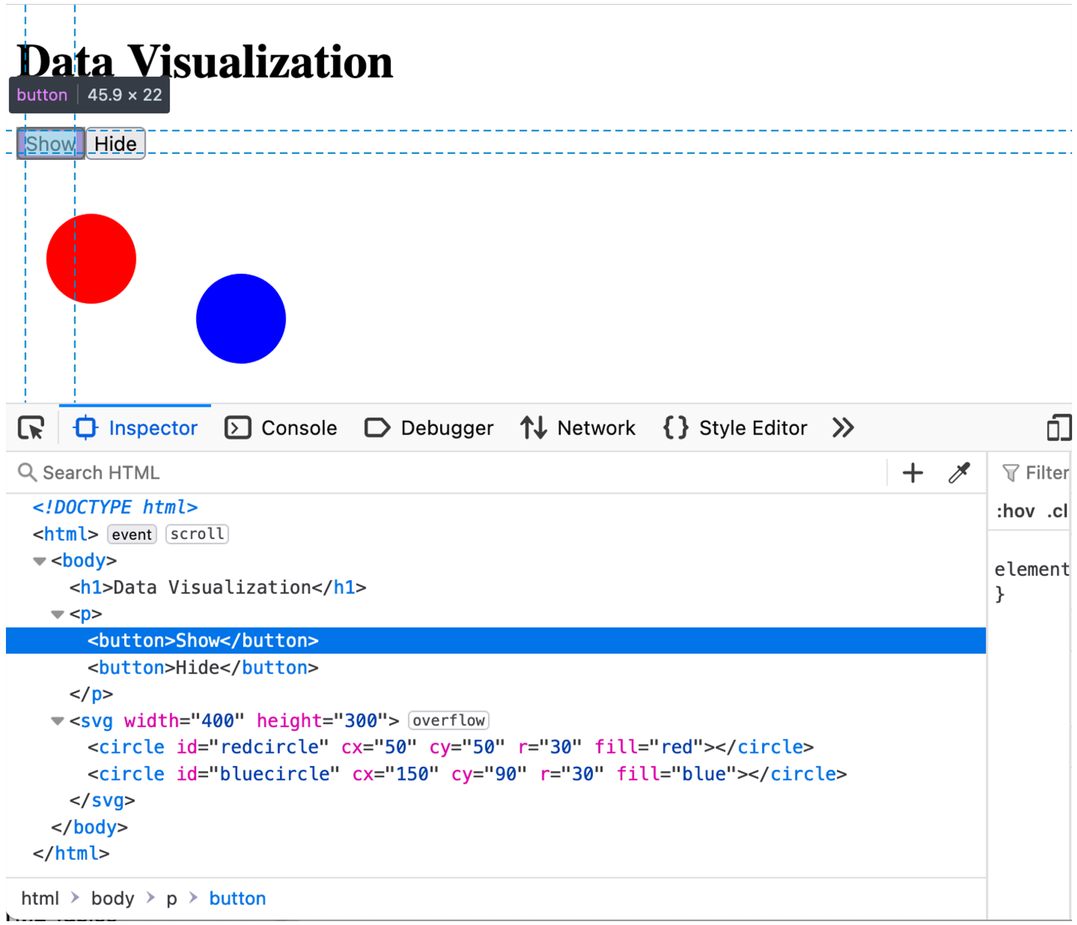
Document Object Model (DOM)



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

Omitted here for compactness

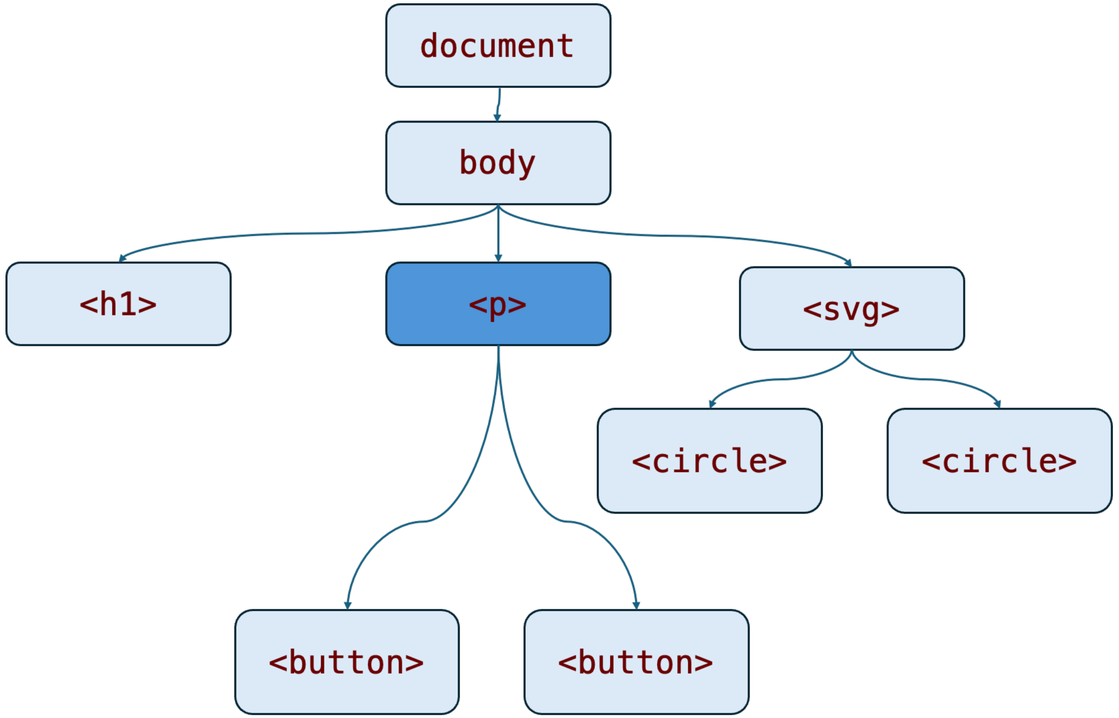
Browser display



Inspector: Browser’s DOM visualization - *shown as HTML*

Container elements

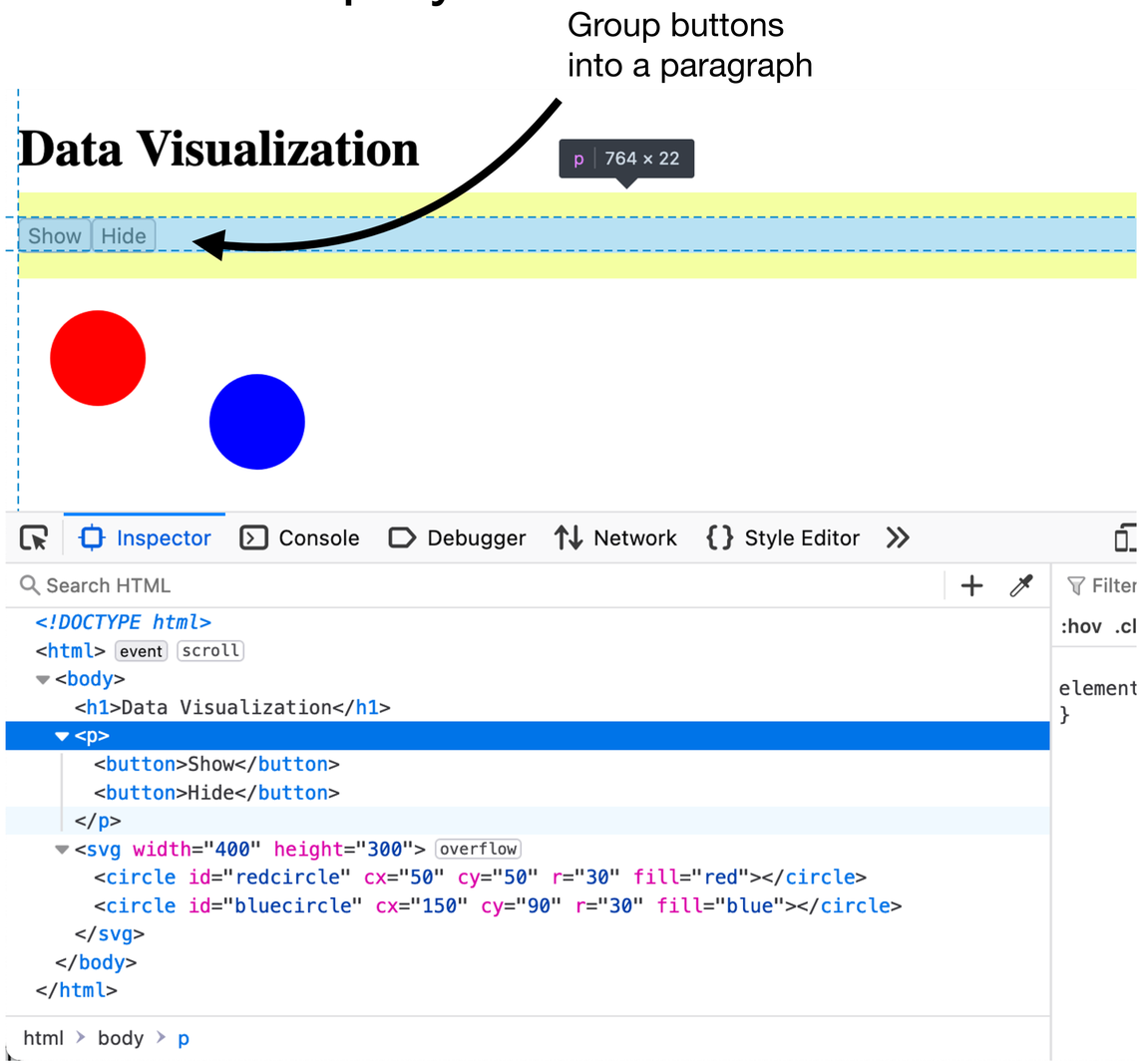
Document Object Model (DOM)



"Container" elements define the organization of UI elements

* … organizes its children as a paragraph*

Browser display



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

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

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

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

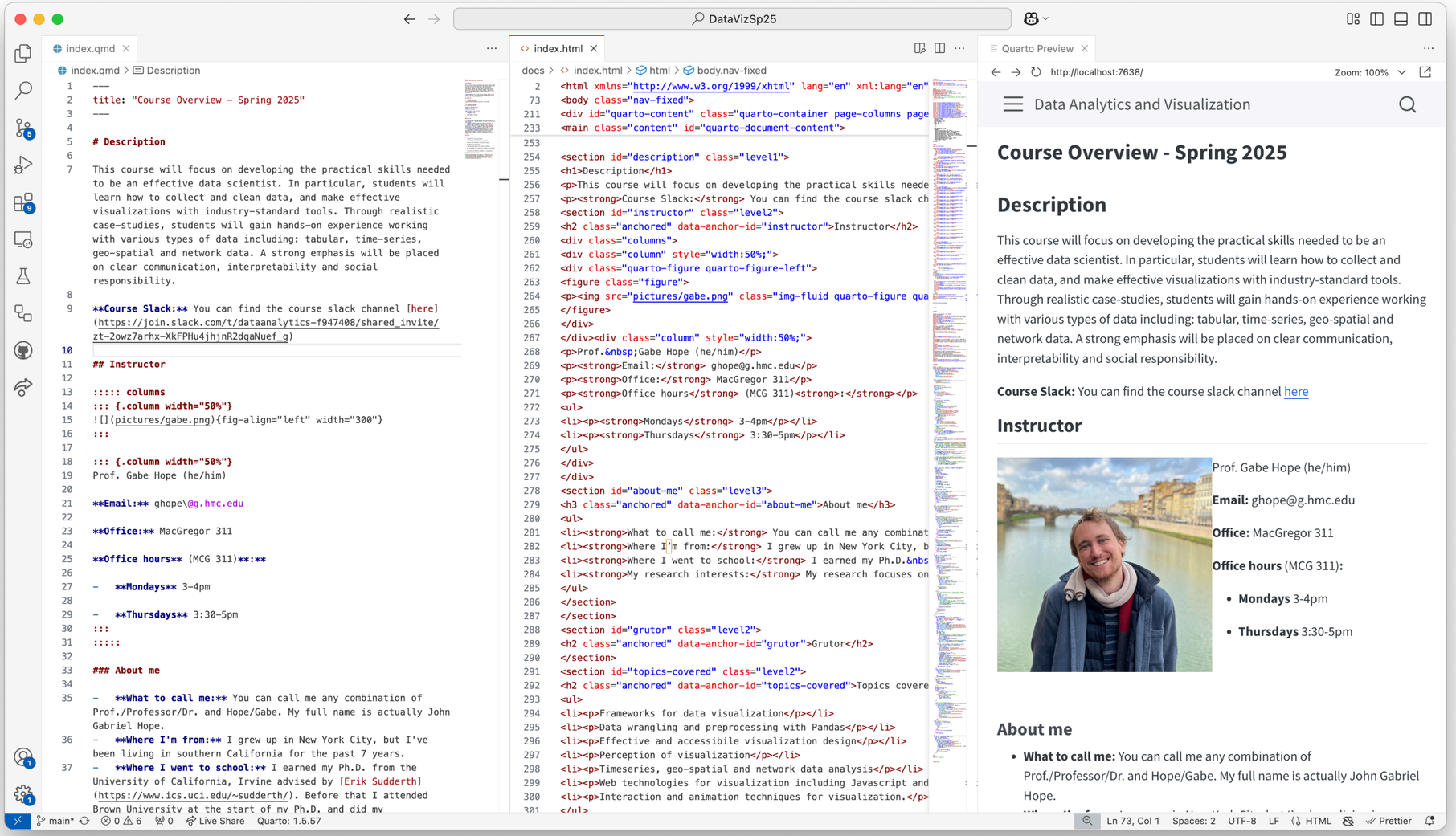
Input types

`type="checkbox"`	☐
`type="radio"`	◉
`type="file"`	Browse... No file selected.
`type="hidden"`	
`type="text"`	
`type="password"`	
`type="image"`	Login
`type="reset"`	Reset
`type="button"`	button
`type="submit"`	Submit Query

https://cheatsheets.zip/html

Quarto Can Handle Page Structure!

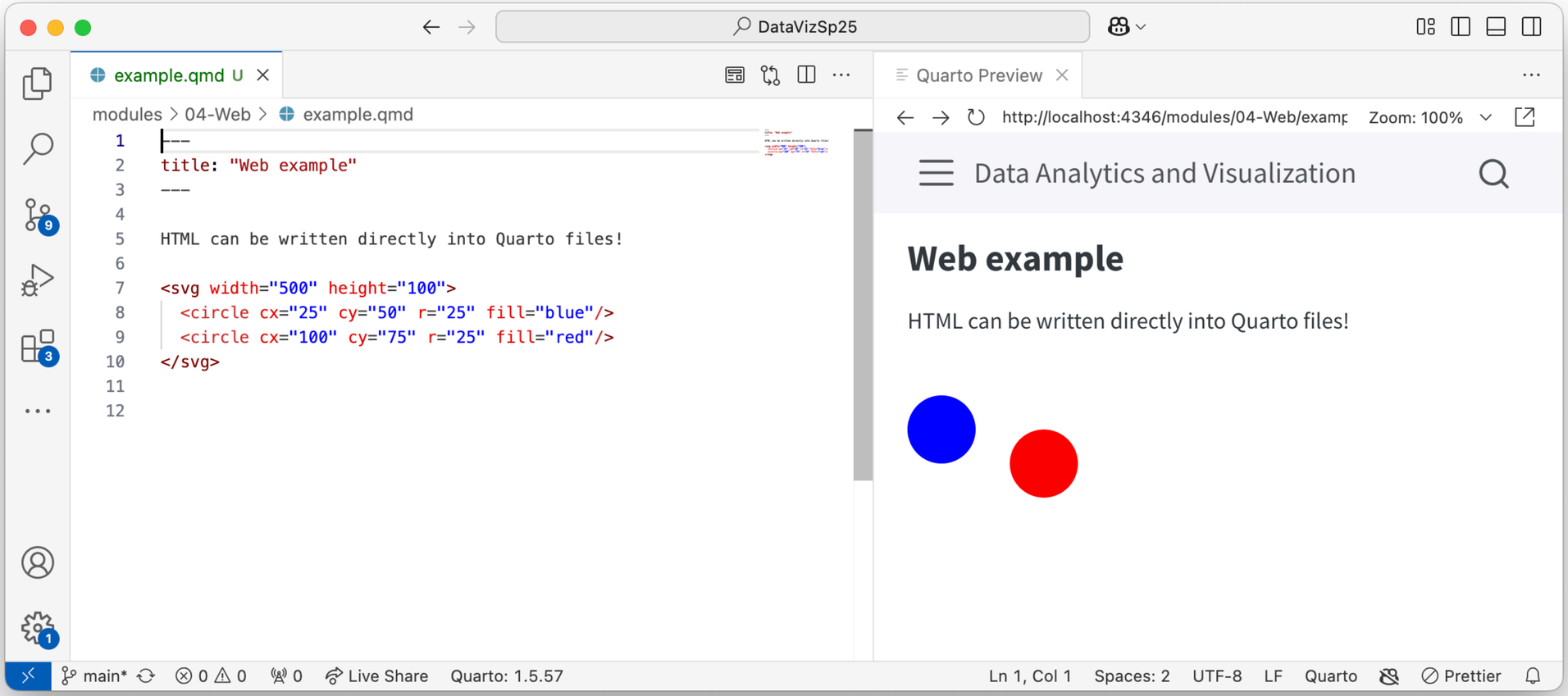
Quarto markdown



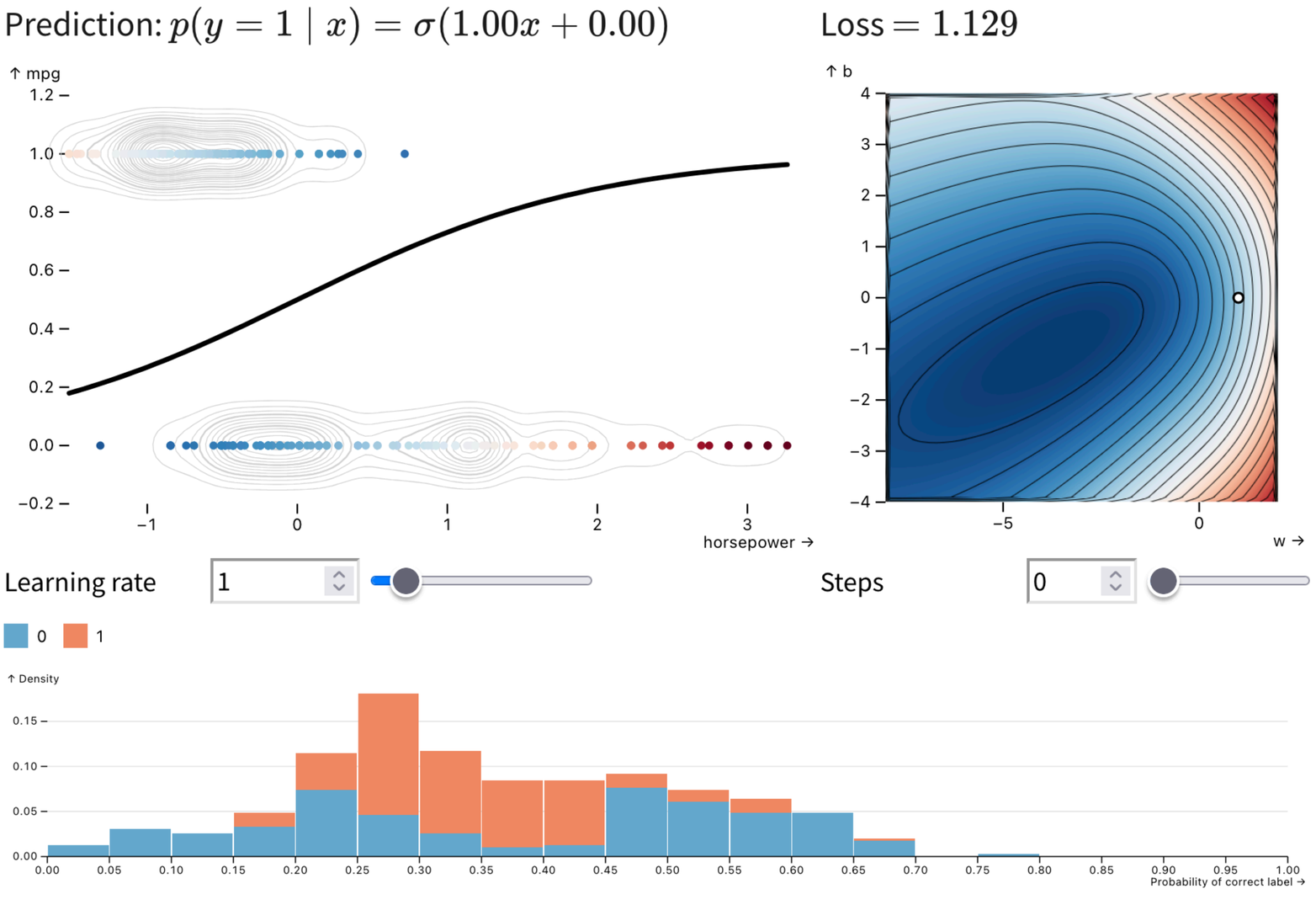
Which can be displayed in browser

Compiles to HTML

HTML in Quarto

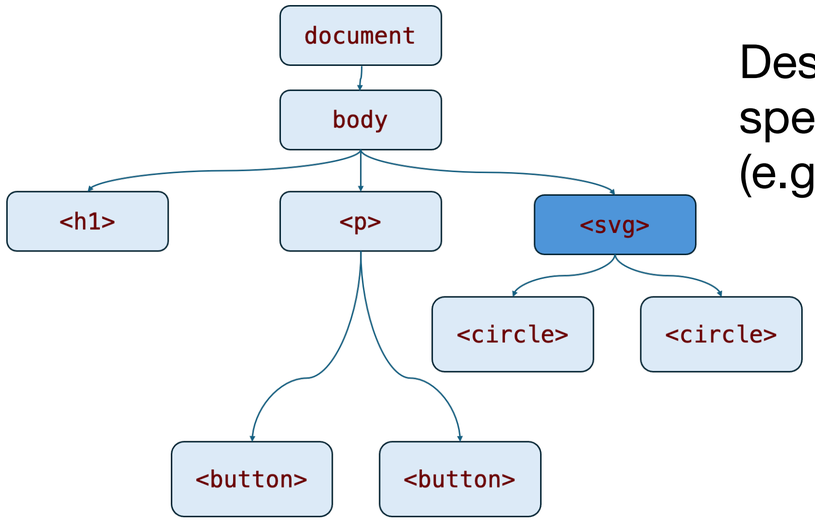


What about visualizations?



Scalable Vector Graphics (SVG)

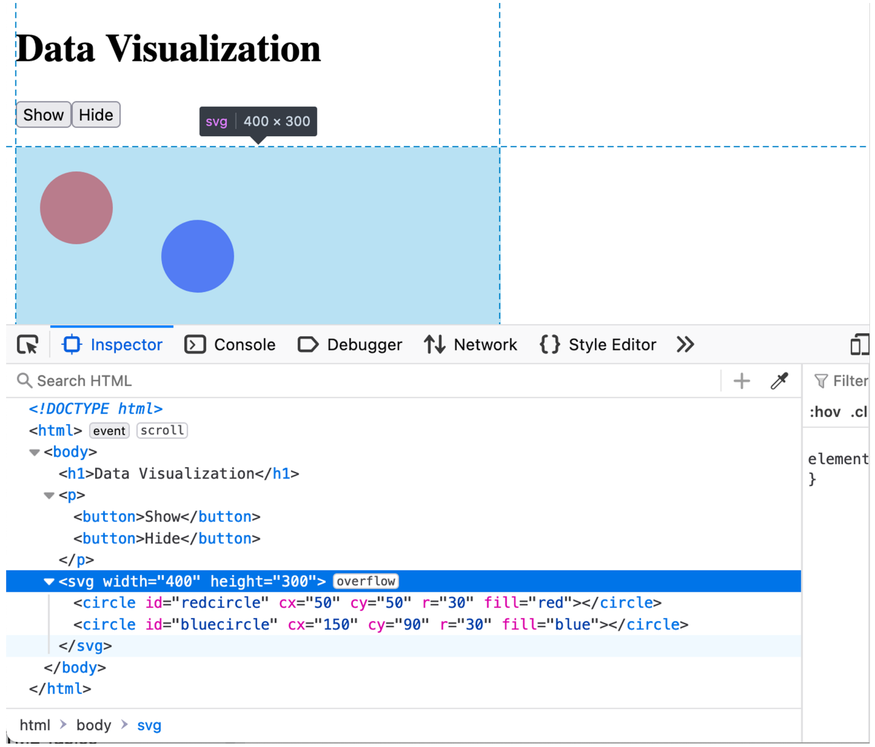
Document Object Model (DOM)



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

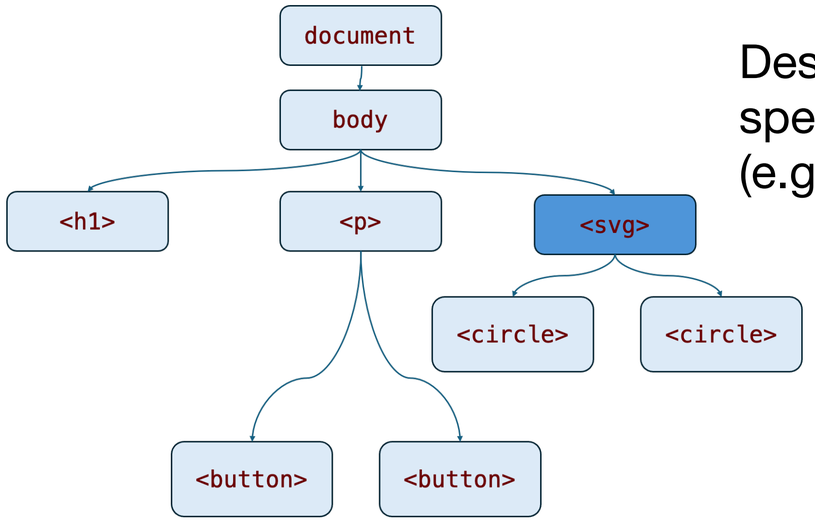
Special container element for drawing graphics (*like visualizations!*)

Browser display



Scalable Vector Graphics (SVG)

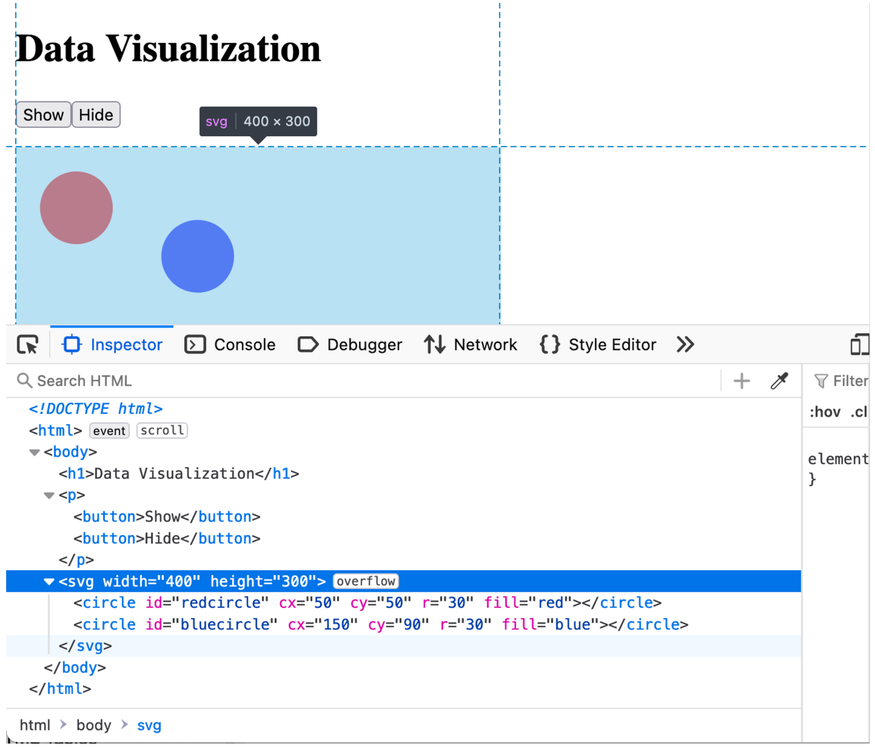
Document Object Model (DOM)



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

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

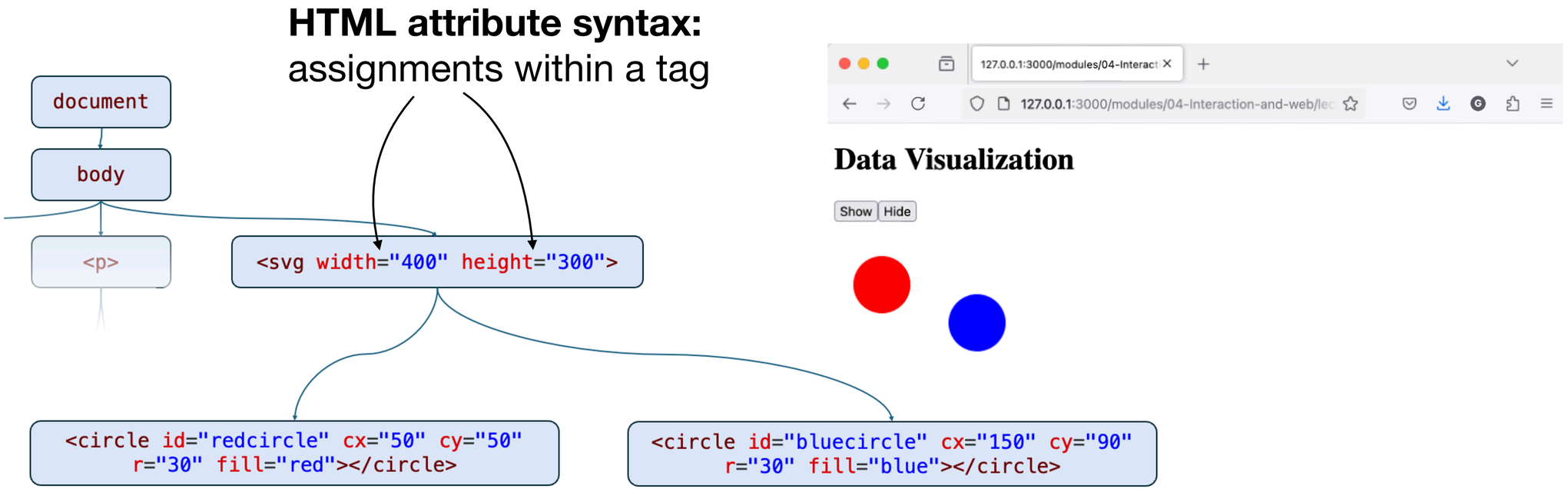
Browser display



This will be our focus (mostly)!

Element attributes

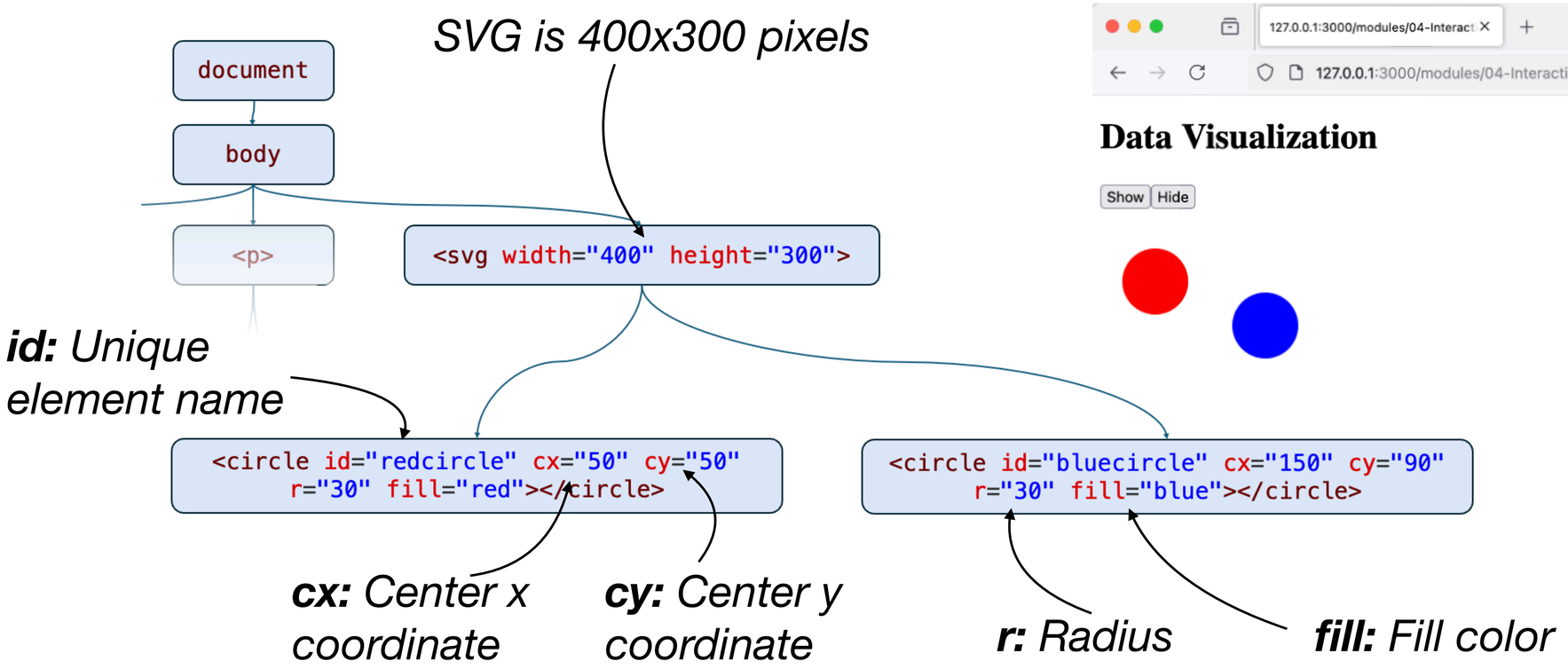
Control the properties of each element



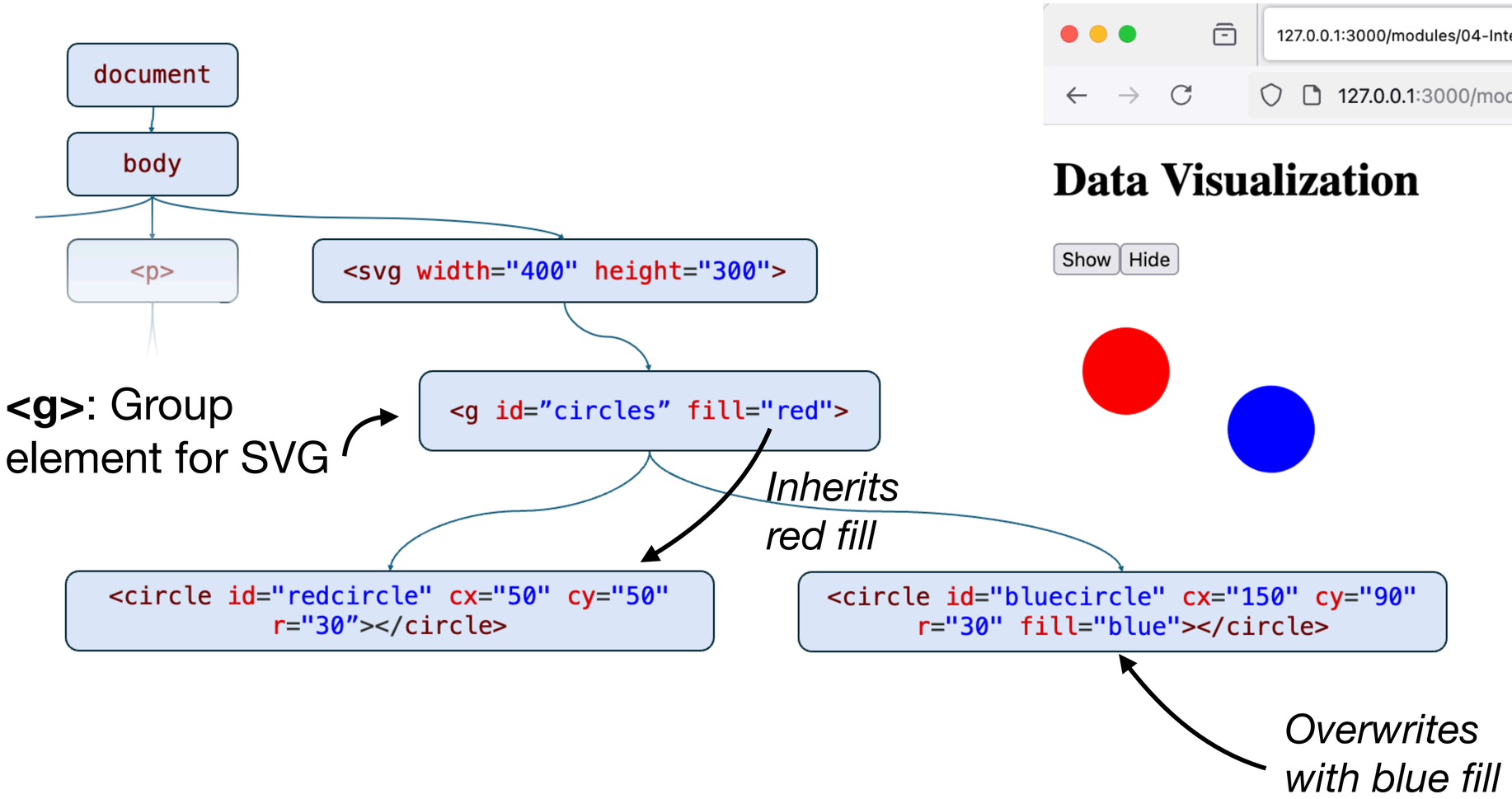
Stored as properties of nodes in the DOM

Element attributes

Control the properties of each element

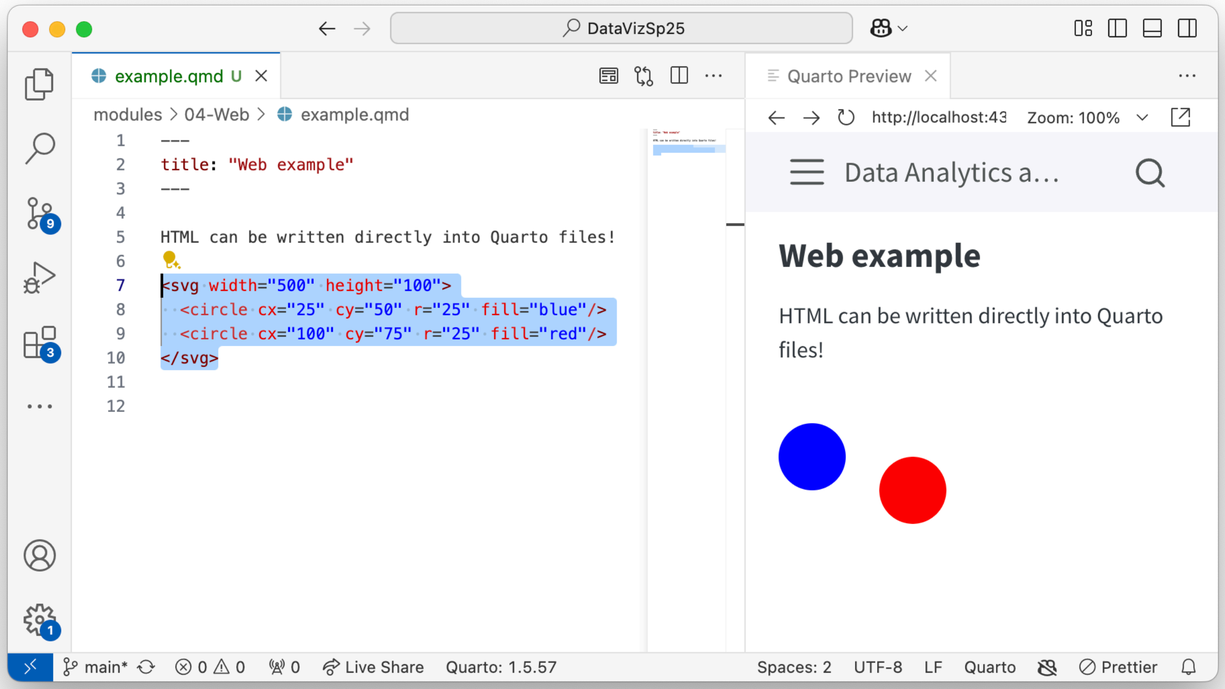


Attribute inheritance

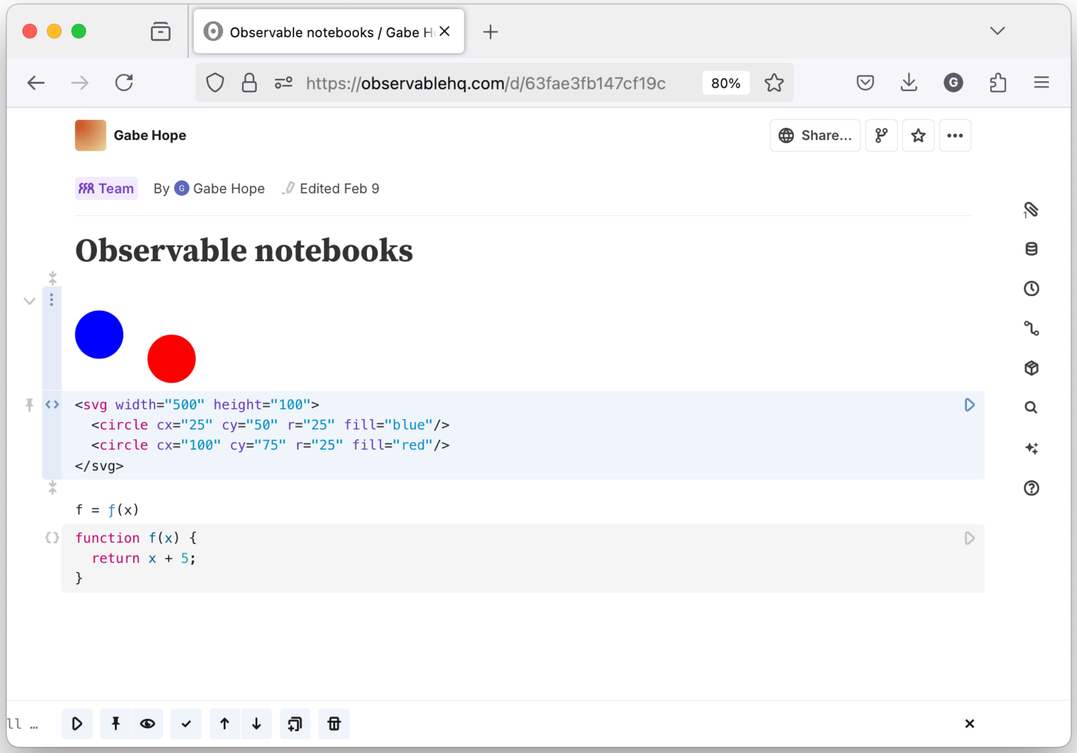


Attributes are inherited from parents in the tree!

Trying it out!



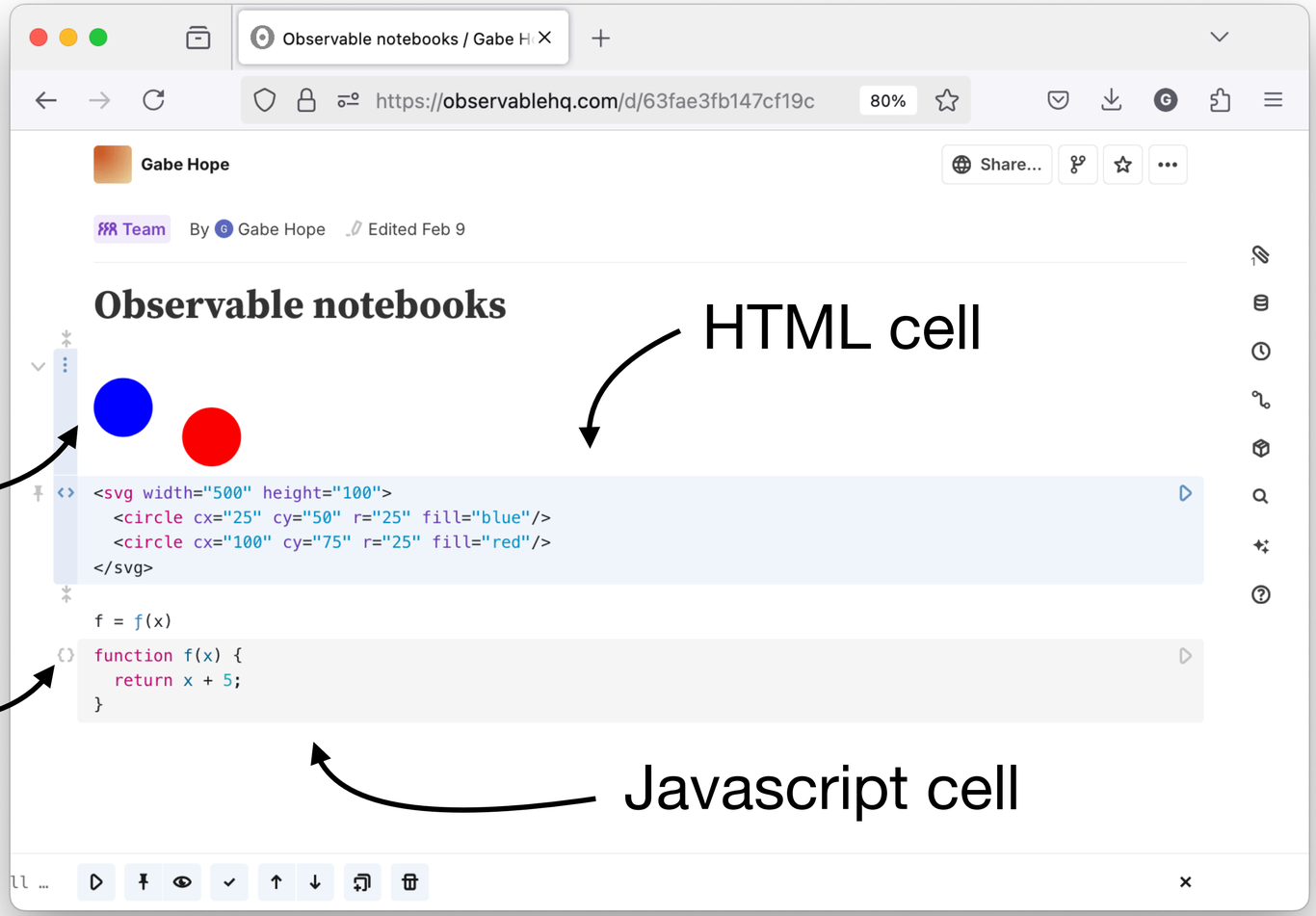
Quarto



Observable Notebooks

Observable notebooks

Like Jupyter Notebooks, but for HTML and Javascript!



Output shown above cell!

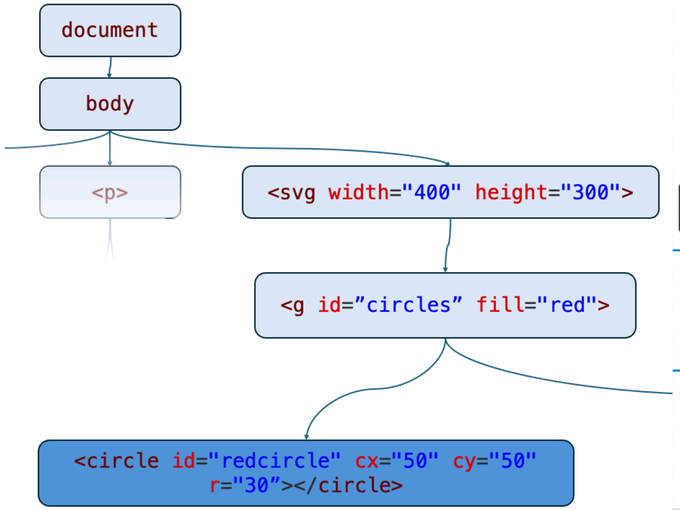
Switch cell type

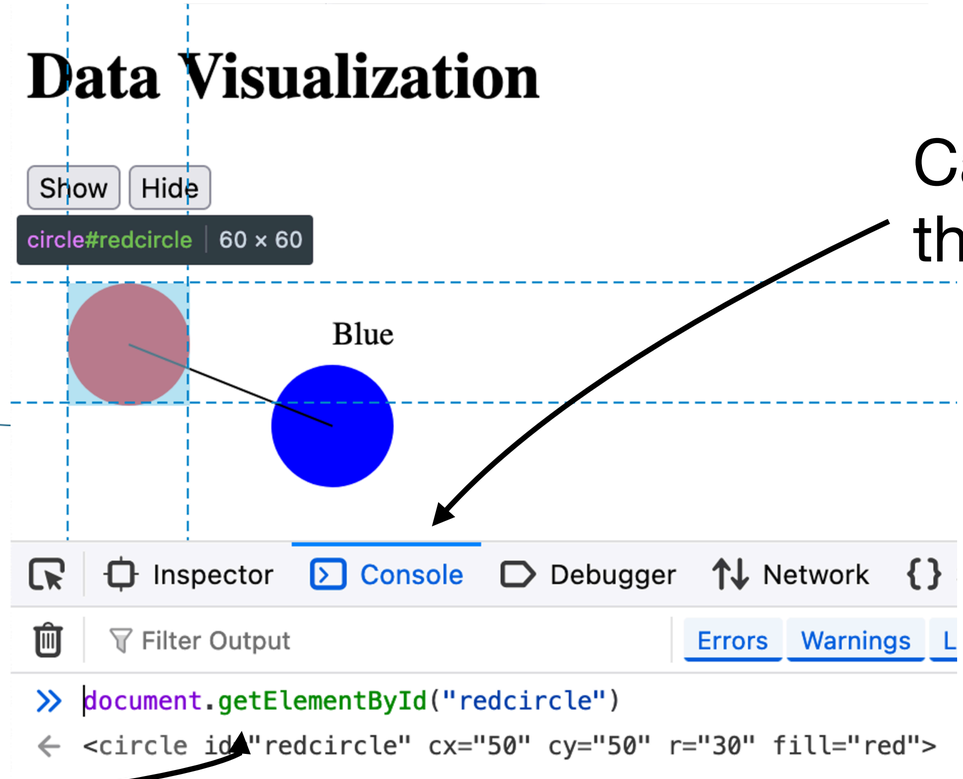
Has some very special properties that we'll discuss later!

Exercise: SVG Elements

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

Javascript: Let's us manipulate the DOM!





Can execute Javascript in the web inspector console

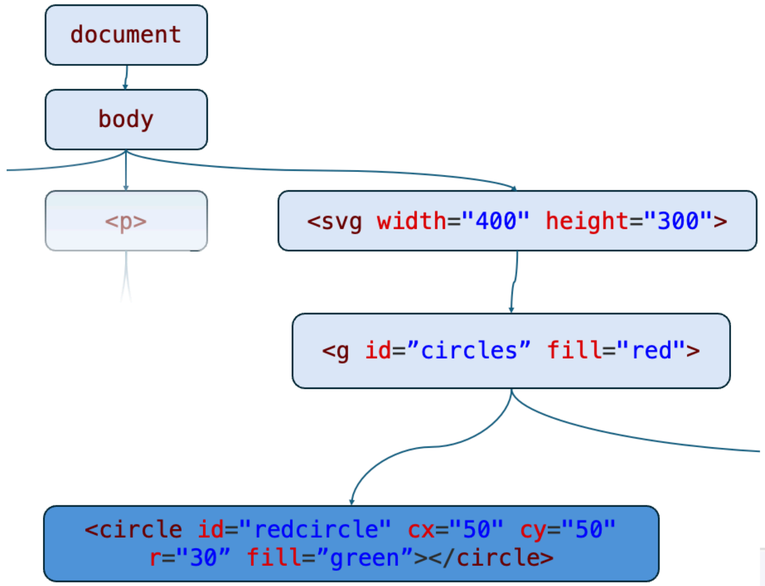
document *is a built-in global variable*

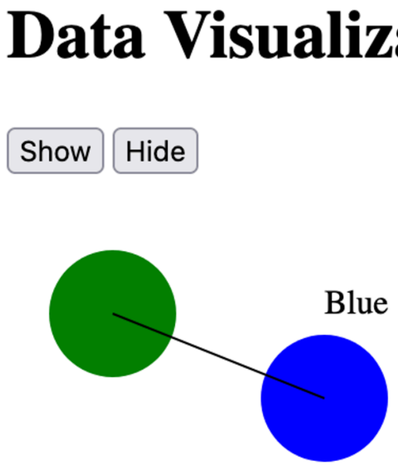
```
document.getElementById("redcircle")
```

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

Step 1: Select an element

Javascript: Let's us manipulate the DOM!

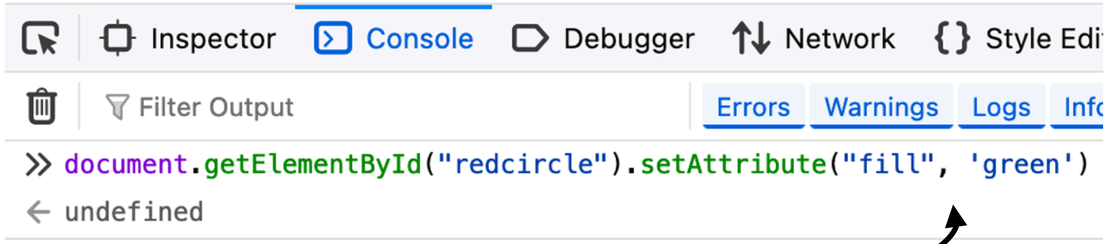




Step 1: Select an element

Step 2: Change properties

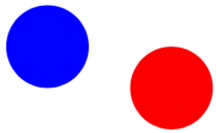
Many ways to select and modify elements are built into Javascript, but we'll introduce a library that makes it easy!



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

Javascript in Observable

Observable notebooks



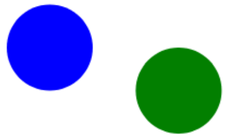
```
<svg width="500" height="100">
  <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>
  <circle id="redcircle" cx="100" cy="75" r="25" fill="red"/>
</svg>
```

▸ SVGCircleElement {}

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

Selected element is represented with an *object* in Javascript

Observable notebooks

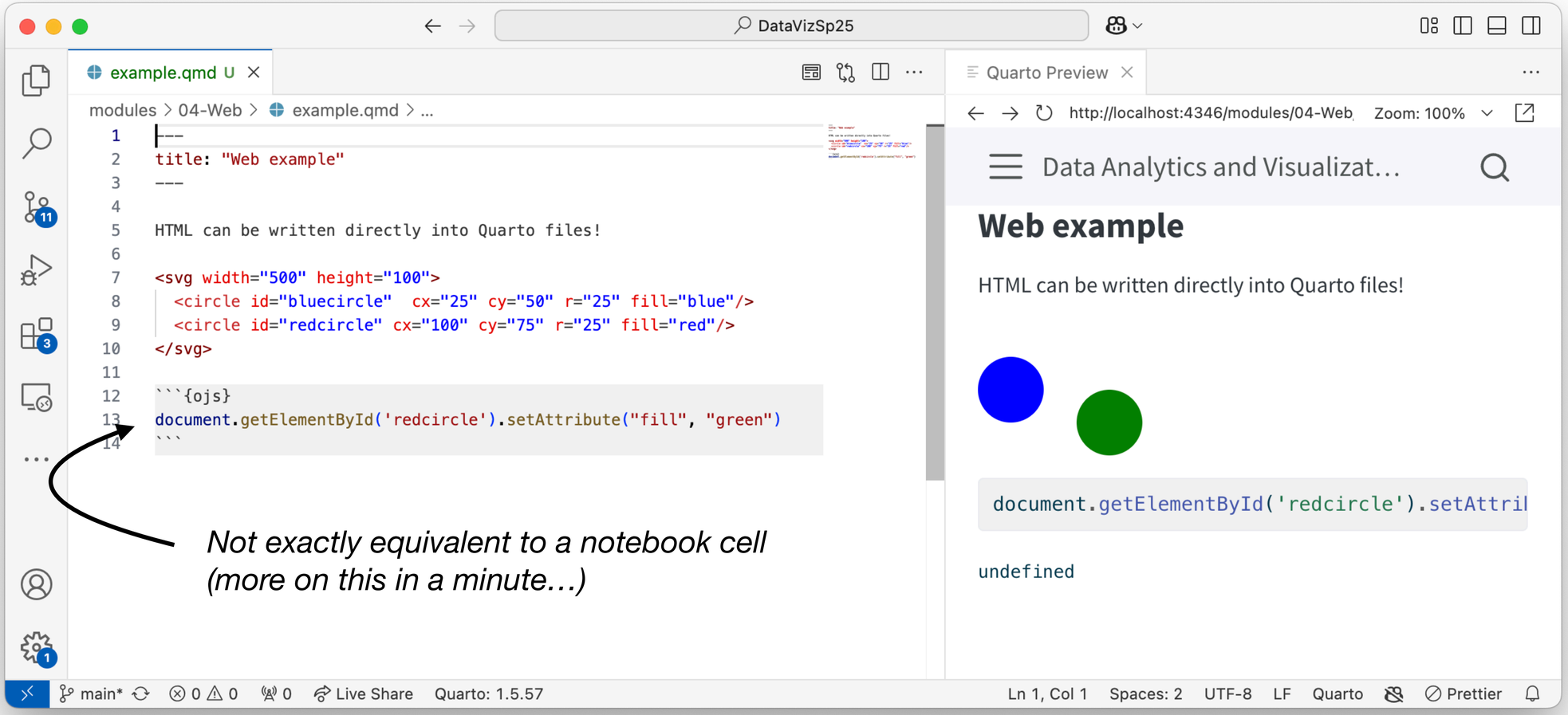


```
<svg width="500" height="100">
  <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>
  <circle id="redcircle" cx="100" cy="75" r="25" fill="red"/>
</svg>
```

undefined

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

Quarto supports Javascript through *ojs code blocks*

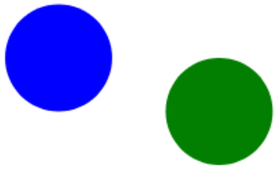


Exercise: Can you change a property of your SVG with Javascript?

Why Javascript: Interaction

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

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>
```

```
makeGreen = ƒ()
```

```
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!

Javascript function syntax

```
makeGreen = f()
```

```
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)*

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

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

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
```

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

```
12
```

```
num * 4
```

*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

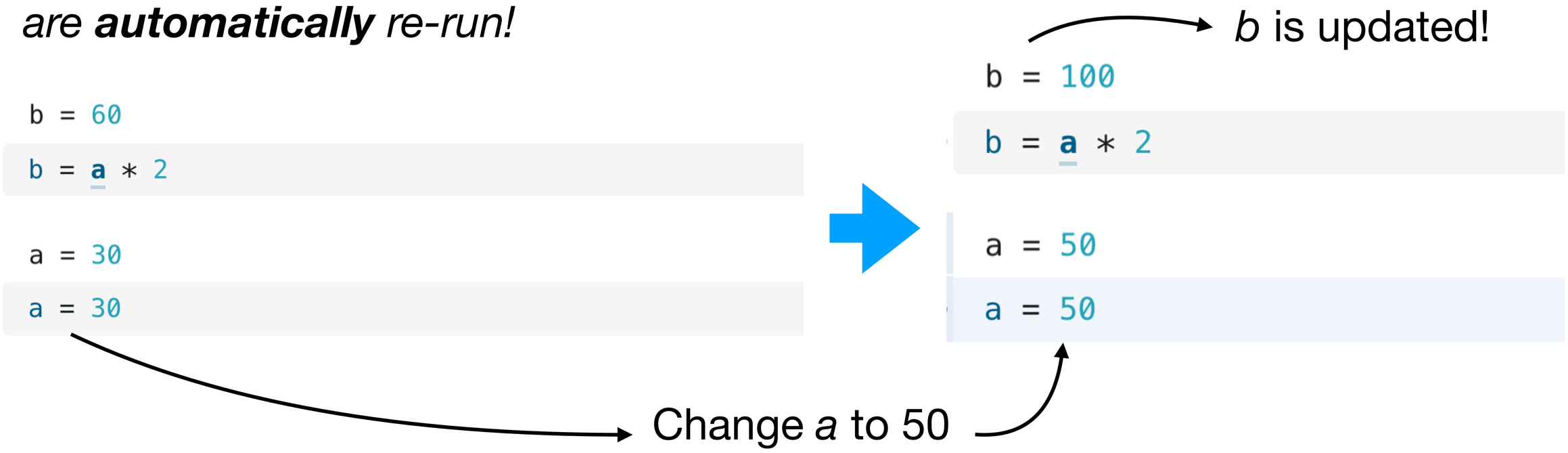
```
newNum = {
  const a = 5;
  const b = 6;
  return a * b;
}
```

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 it's value are **automatically** re-run!*



Observable reactivity

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

```
b = 100
```

```
b = a * 2
```

```
c = 150
```

```
c = a + b
```

```
a = 50
```

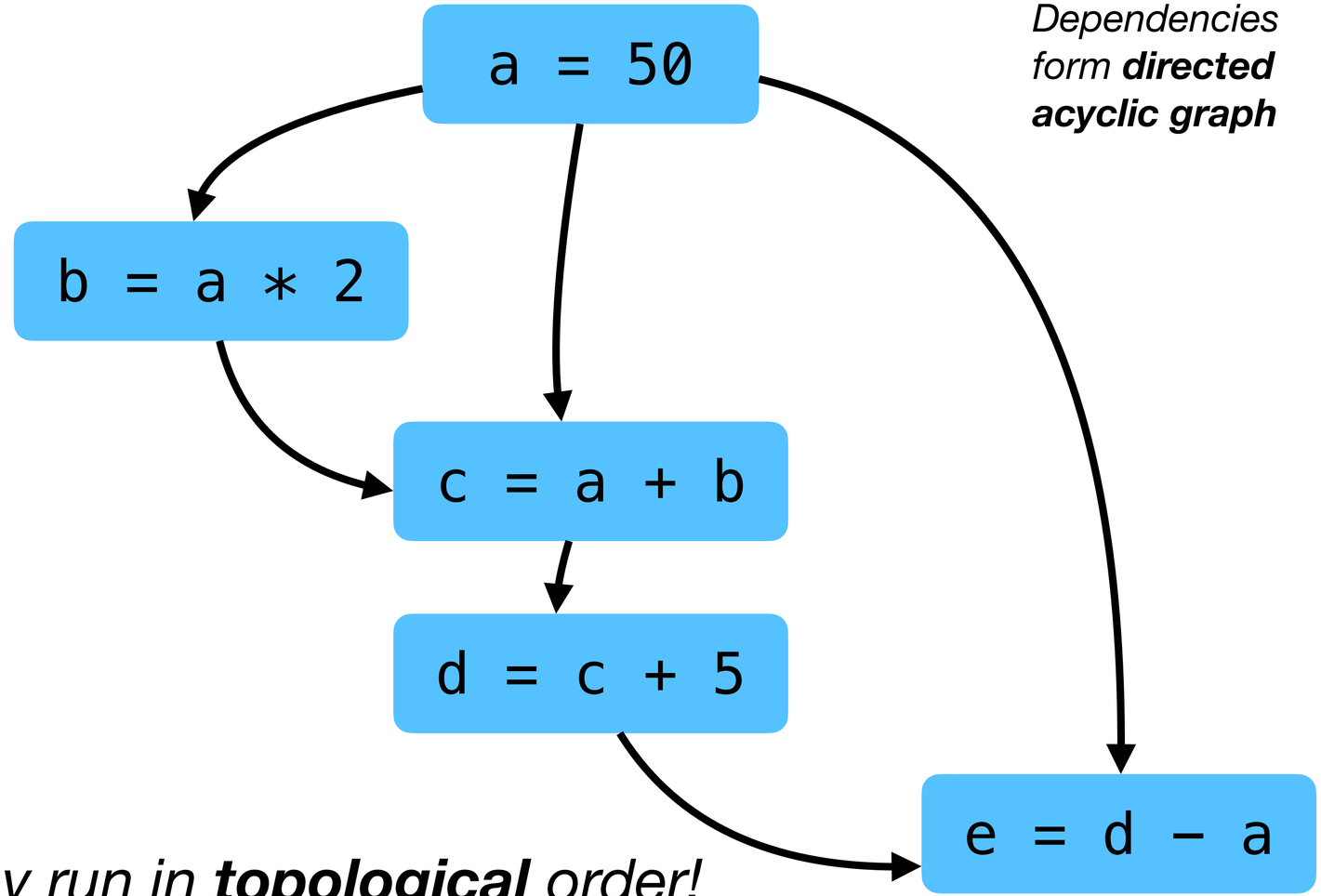
```
a = 50
```

```
d = 155
```

```
d = c + 5
```

```
e = 105
```

```
e = d - a
```



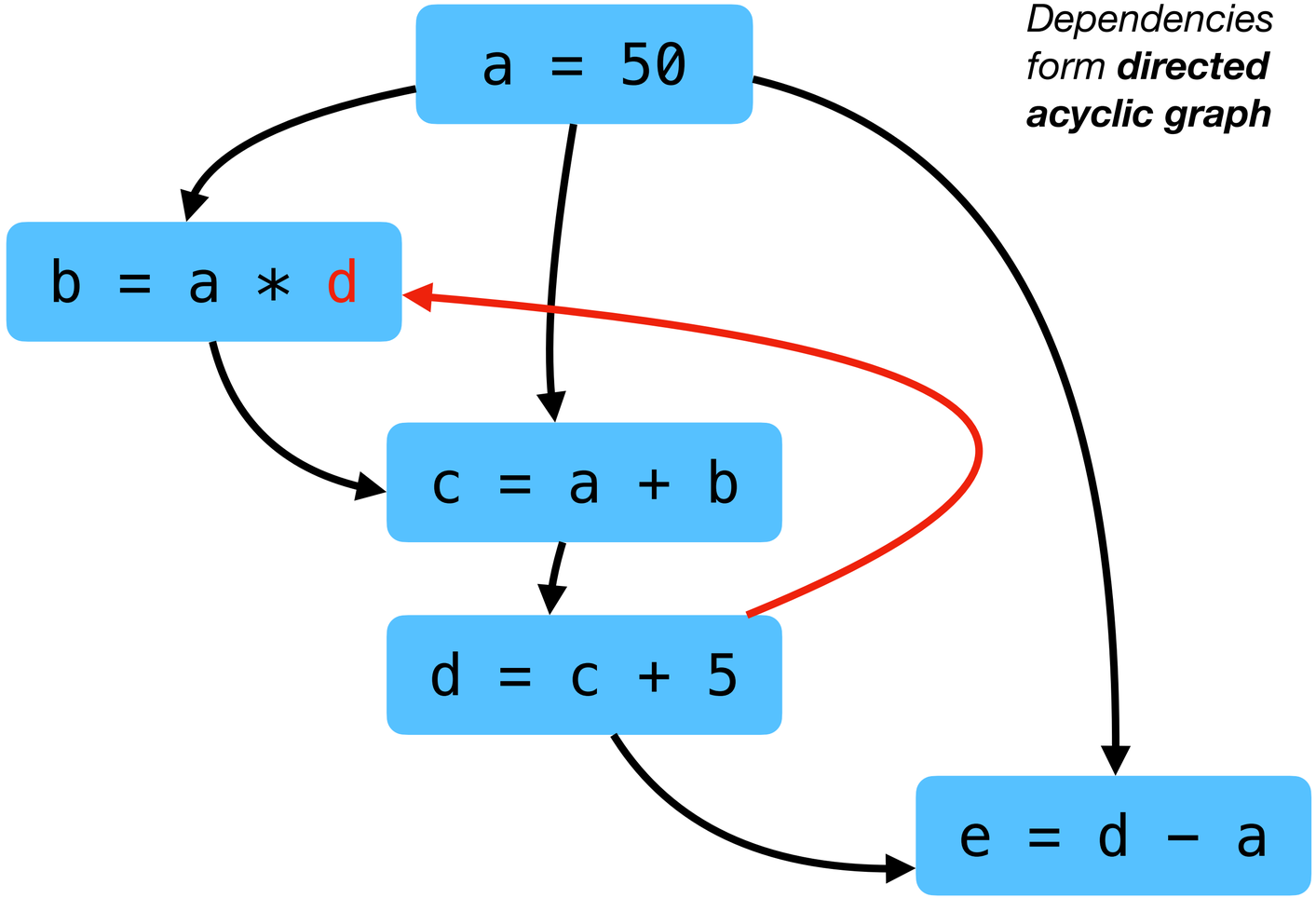
*Dependencies form **directed acyclic graph***

*They run in **topological** order!*

Observable reactivity

Cycles are not allowed!

```
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
```



*Dependencies form **directed acyclic graph***

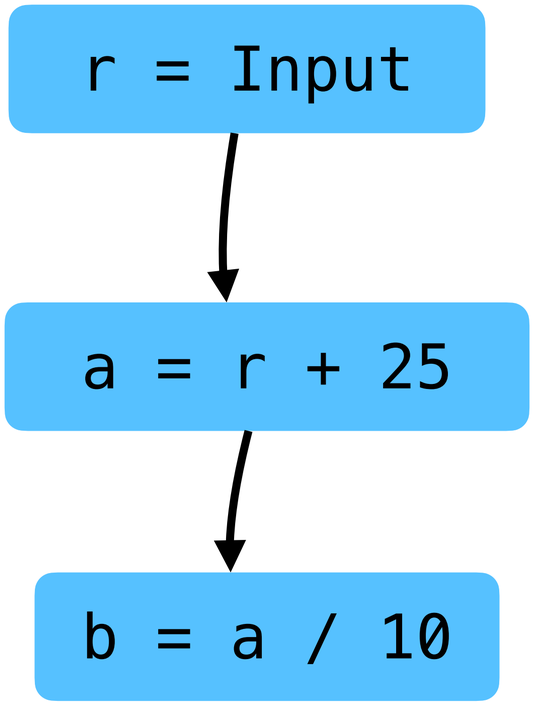
Observable reactivity

***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})
```



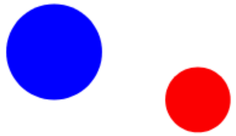
***Range:** slider input*

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

Keyword to watch for changes

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

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>
```

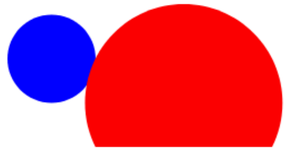
17

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

HTML can react as well!

`r=${r}`

Use Javascript value r



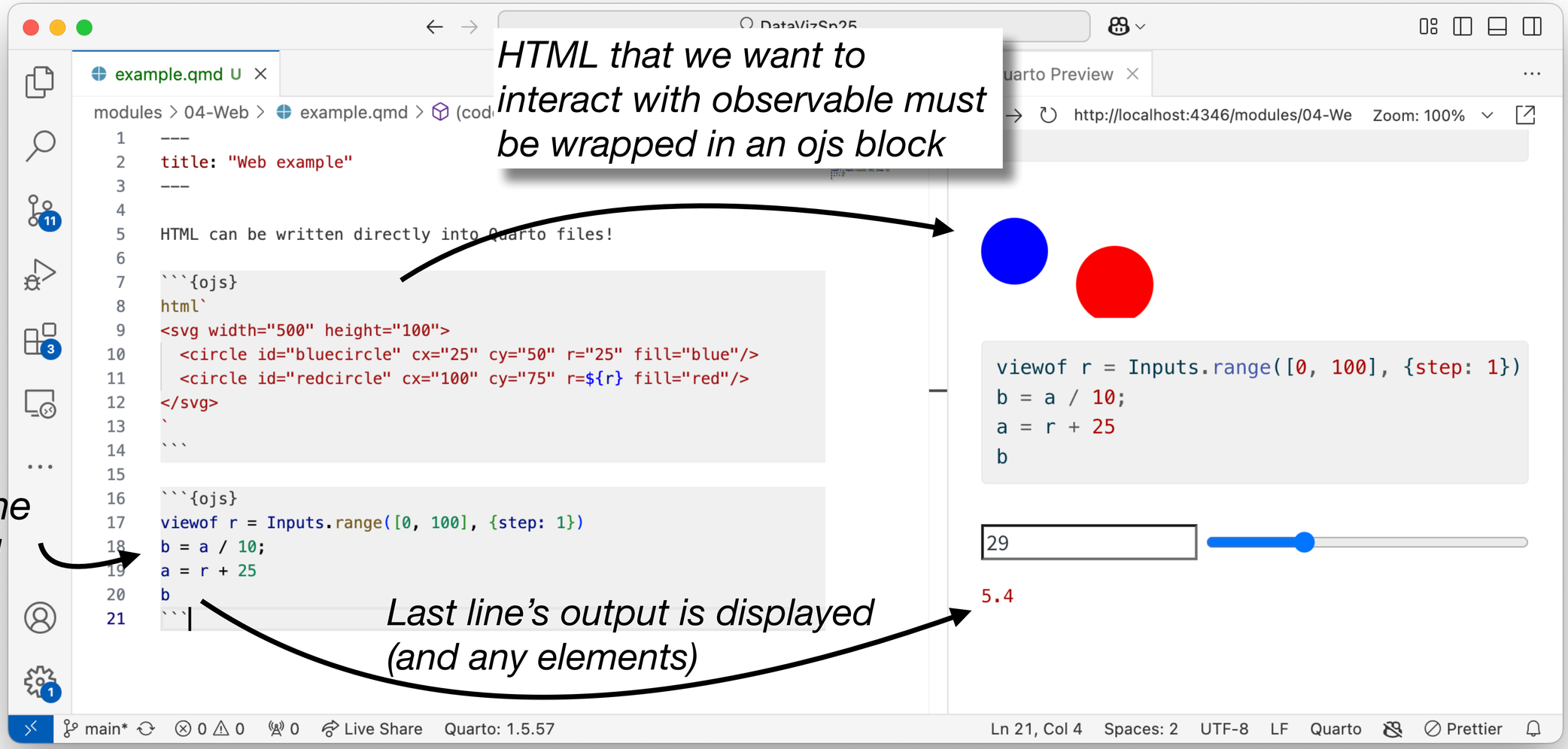
```
<svg width="500" height="100">
  <circle id="bluecircle" cx="25" cy="50" r="25" fill="blue"/>
  <circle id="redcircle" cx="100" cy="75" r=${r} fill="red"/>
</svg>
```

56

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

Exercise: Try it out!

Observable in Quarto

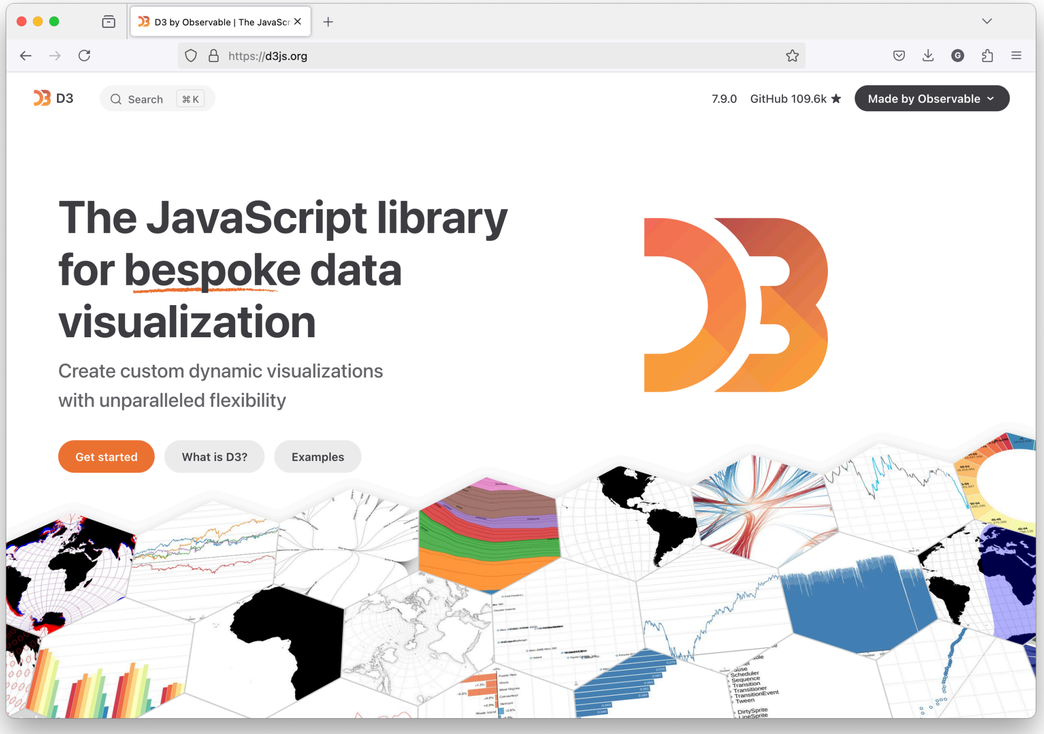


Next Steps: Javascript

- 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

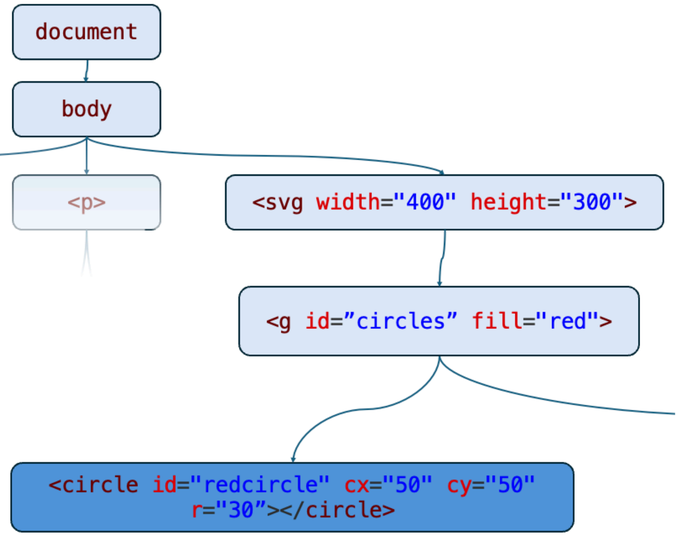
D3.js

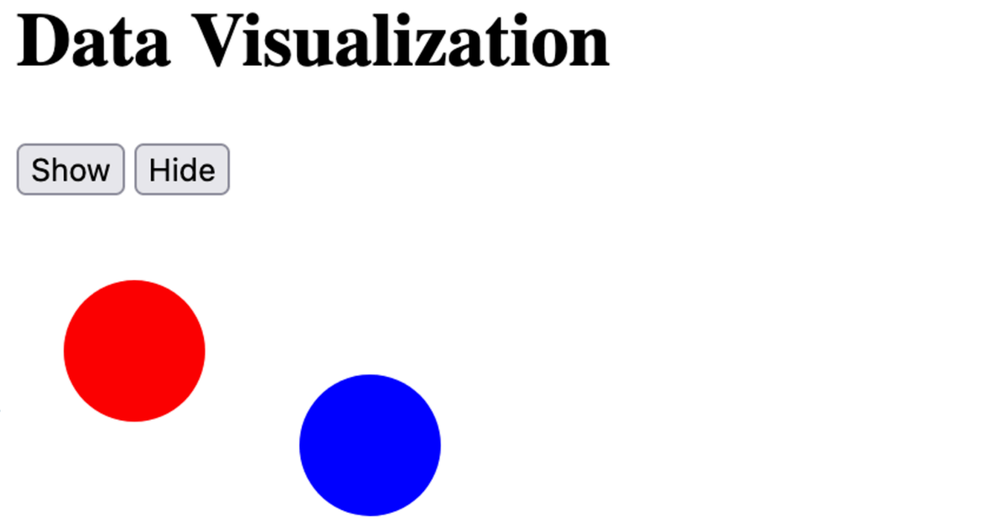
D3.js



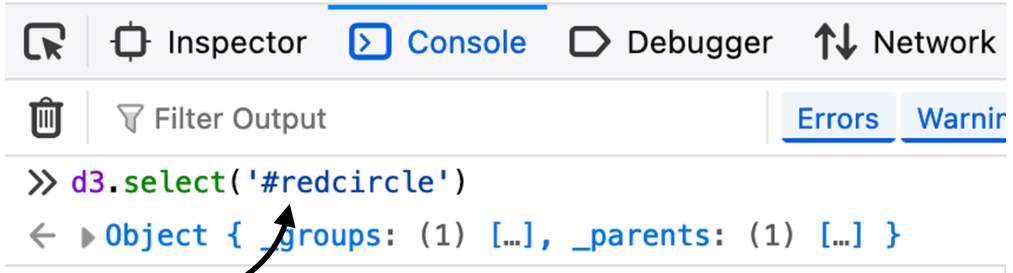
Industry-standard for data visualization on the web

D3: Let's us manipulate the DOM!





d3 is loaded by default in observable!



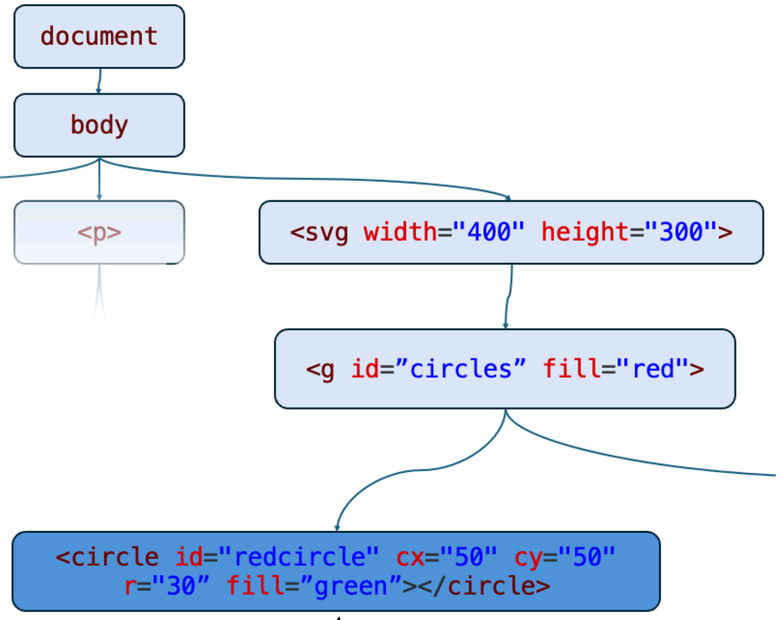
'#' indicates id!

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

Step 1: Select an element

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

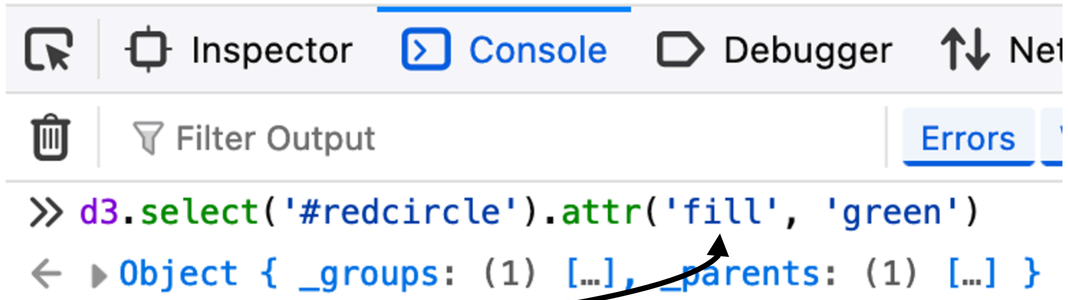
D3: Let's us manipulate the DOM!



Step 1: Select an element

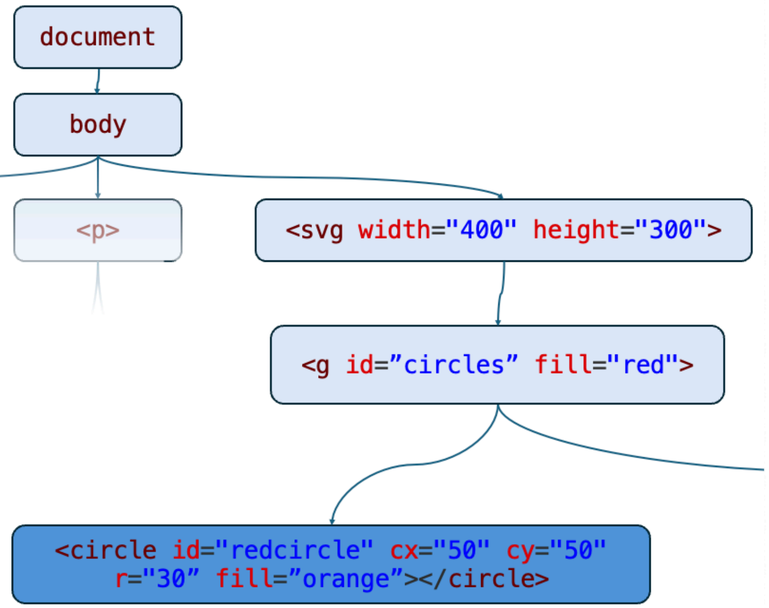
Step 2: Change properties

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

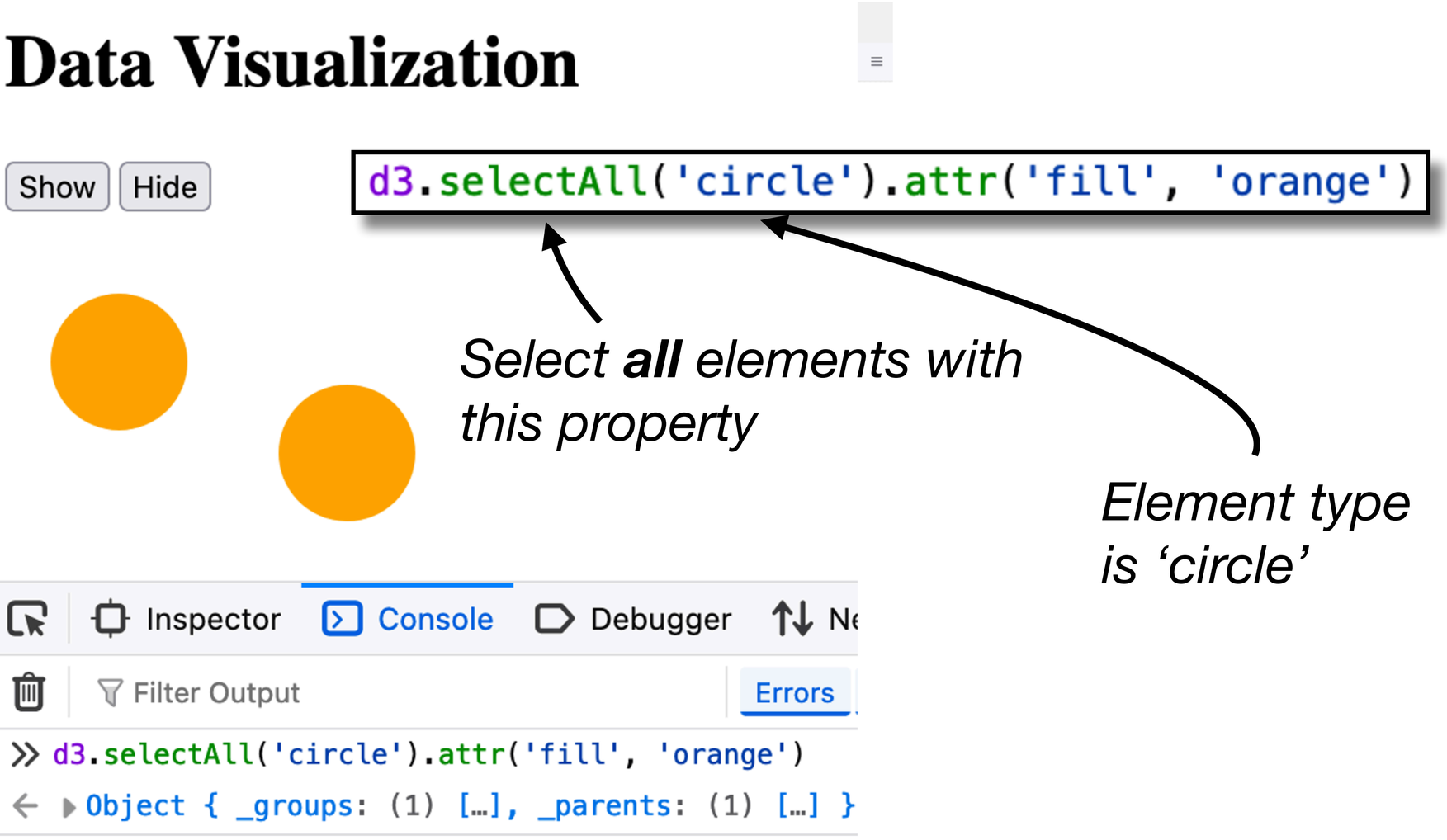


*Change the **fill** attribute to green (using 'attr')*

D3: Let's us manipulate the DOM!



Set fill of all circles to 'orange'

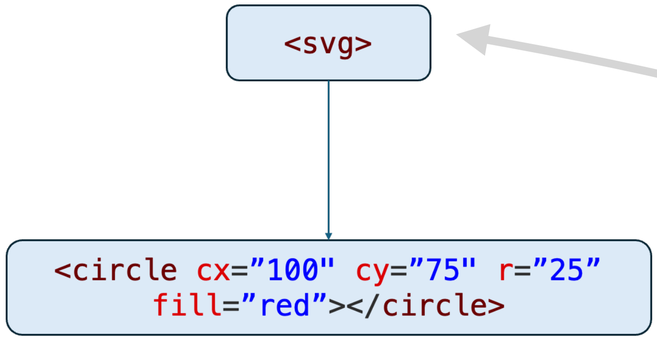


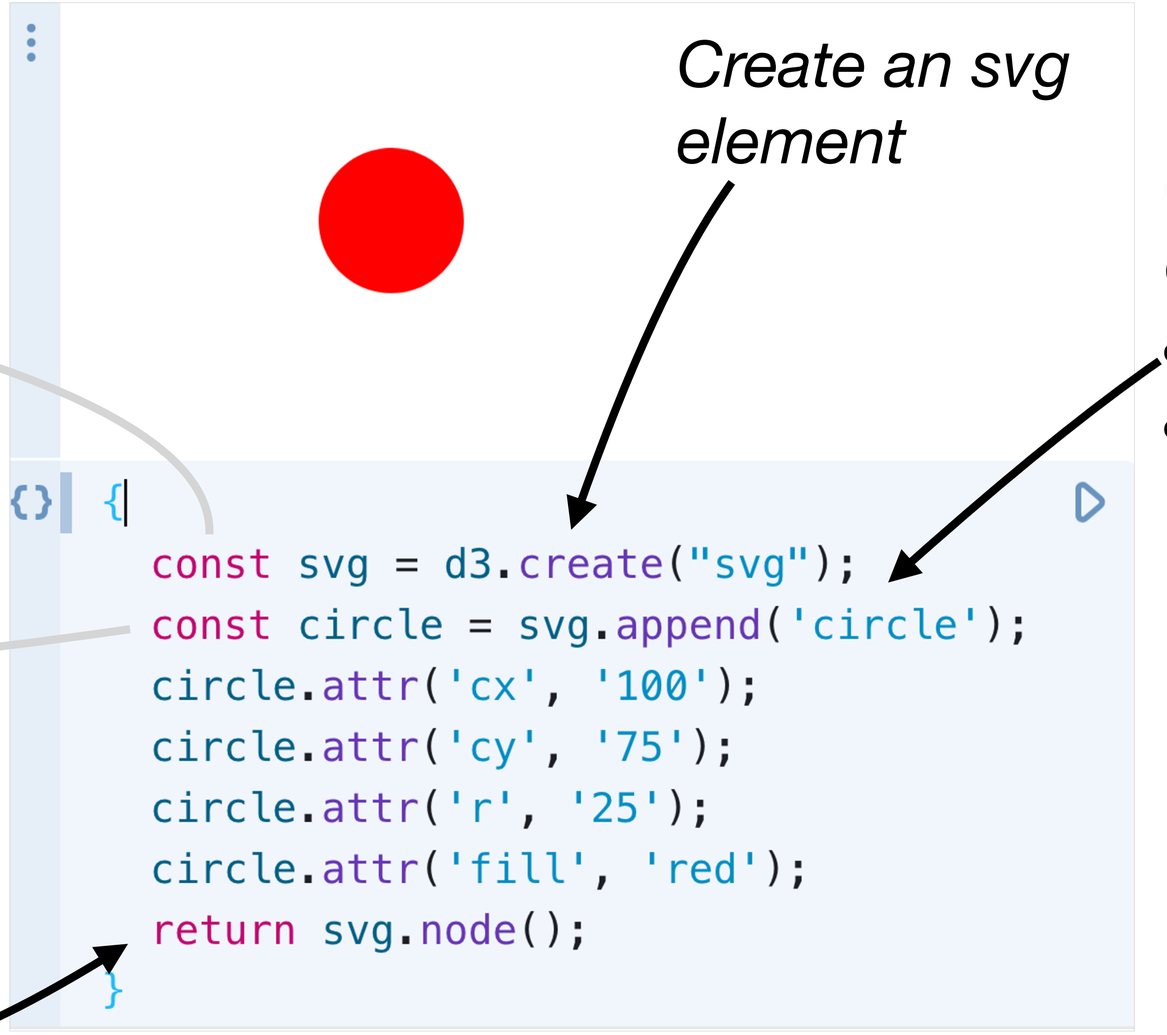
*Select **all** elements with this property*

Element type is 'circle'

Creation with D3

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





Create an svg element

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

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

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
```

Objects: (mostly) like Python dicts

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

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

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

Getting and setting values

```
"penguin"
```

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

```
"dolphin"
```

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

Can also access values using property syntax!

```
"penguin"
```

```
myobject.animal
```

```
"dolphin"
```

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

Loops: more like C/C++

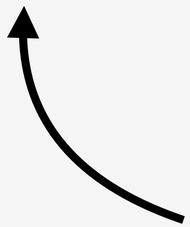
```
myarray = ▶ Array(5) [1, 2, 3, 4, 5]
```

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

```
▶ Array(5) [2, 4, 14, 8, 10]
```

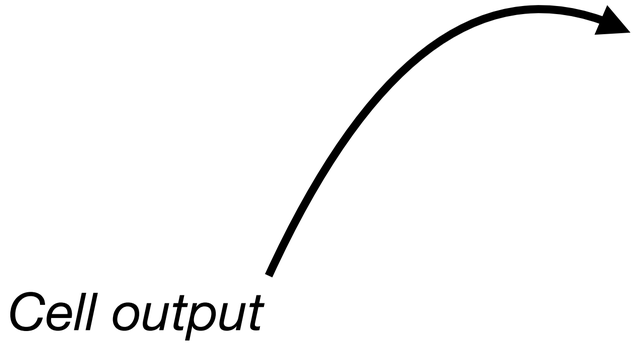
We will rarely write loops ourselves in this class!

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



Get length of array

Function syntax




```
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

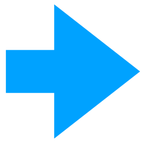
..and calls it internally!

Functions can reference external variables (as in python)

Pass the multiply function here

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

	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

Columns

rows

```
▼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

Javascript

Data representation in Javascript

Represent a data table as an array of objects

rows

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"
  }
```

In Quarto

Convert to JS representation

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

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

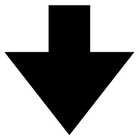
````
```{ojs}
penguins
```
````

ojs_define: *Pass value from Python to JS*

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'")
```

Keep rows where expression is true

Javascript (w/d3)

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

Row object

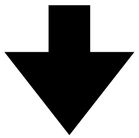
Index

```
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'")
```

Keep rows where expression is true

Javascript (w/d3)

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

Row object

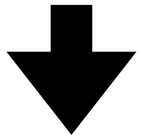
Index

```
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

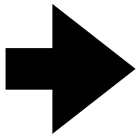
Selecting variables

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

Python

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

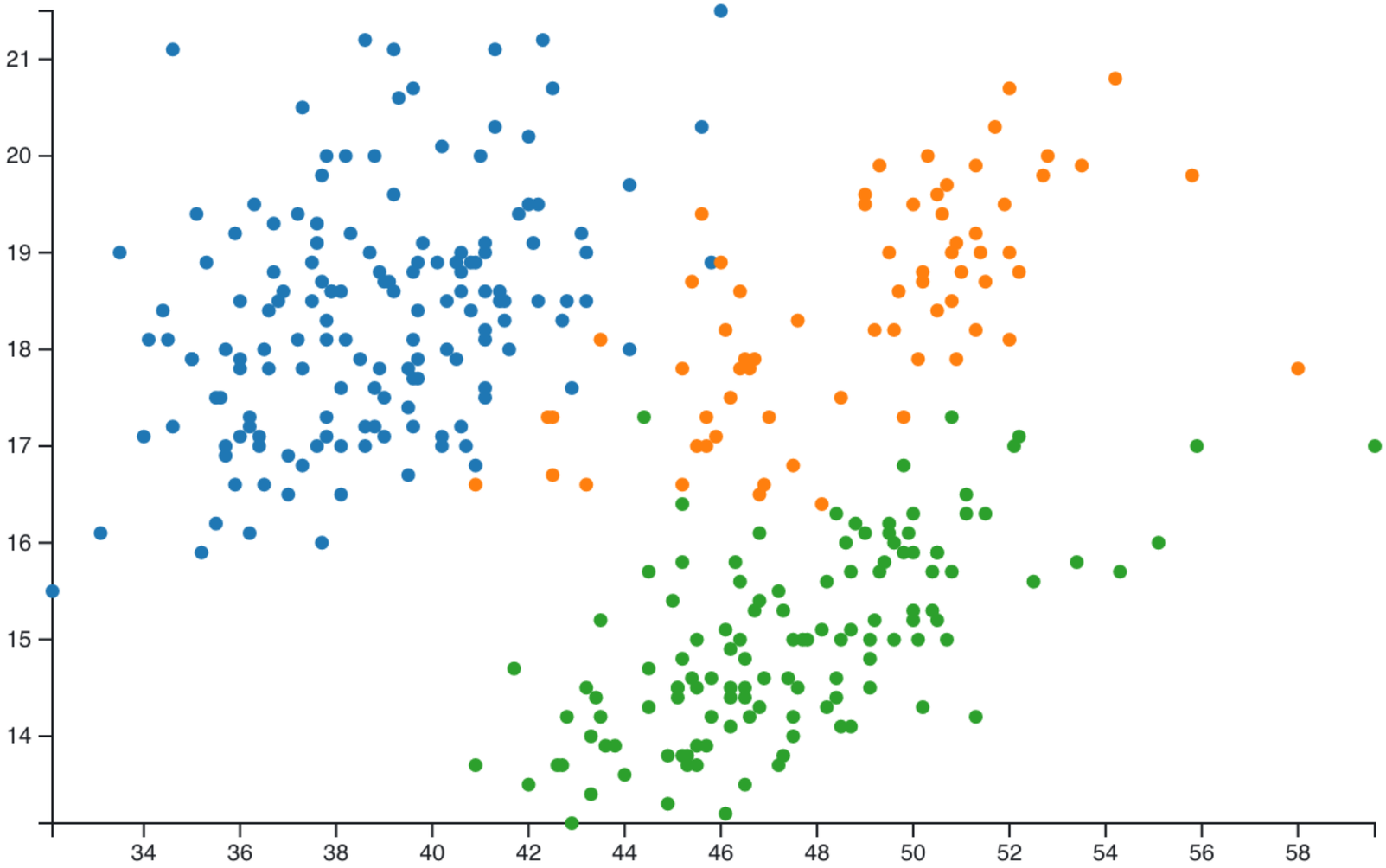
Javascript (w/d3)



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

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

Creating a plot with D3

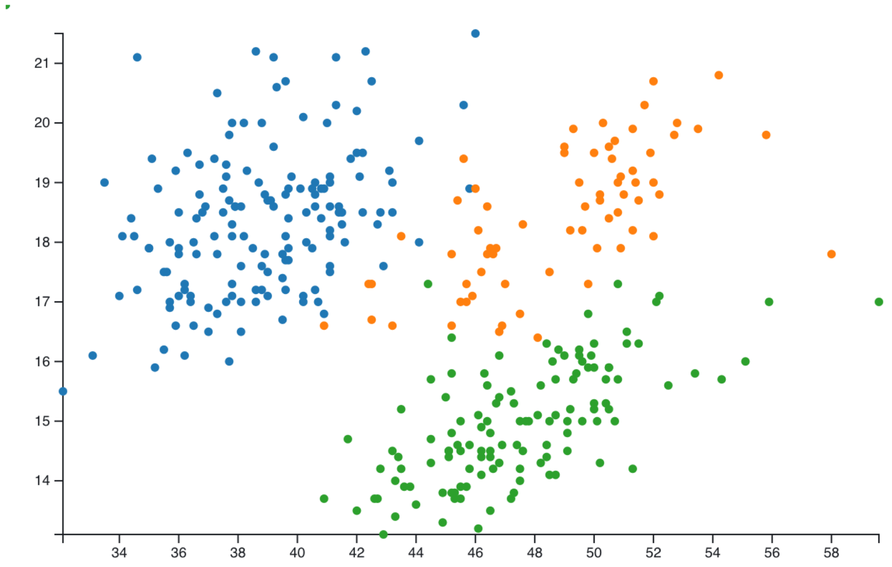


Creating a plot with D3

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!

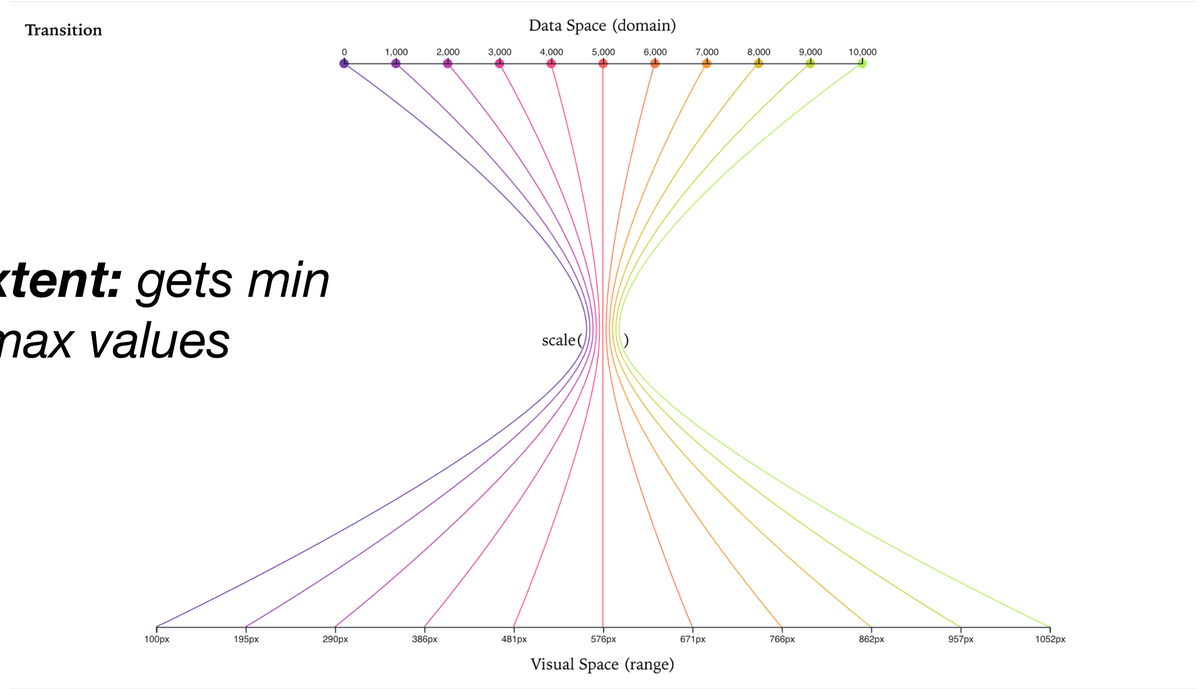
Creating scales

Domain: *What is the extent of our data?*

scaleLinear: *Gives us a function!*

```
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]);
```

d3.extent: *gets min and max values*



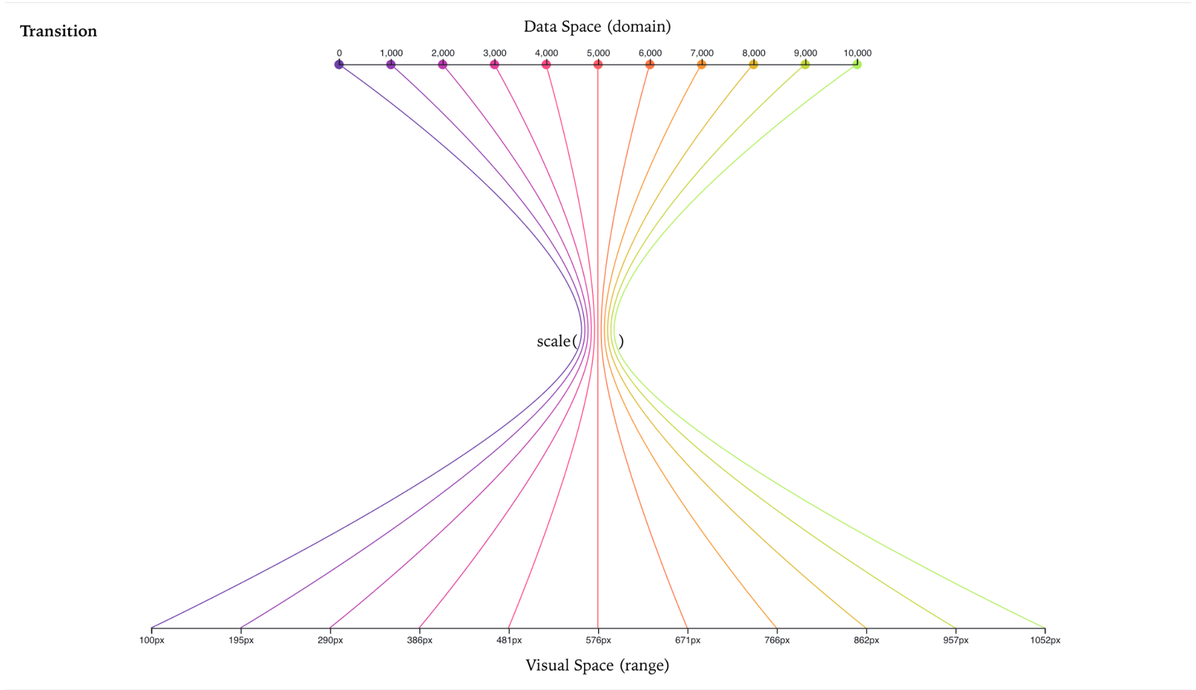
Range: *How big is our figure?*

Creating color palettes

Range: *What's our color scheme*

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

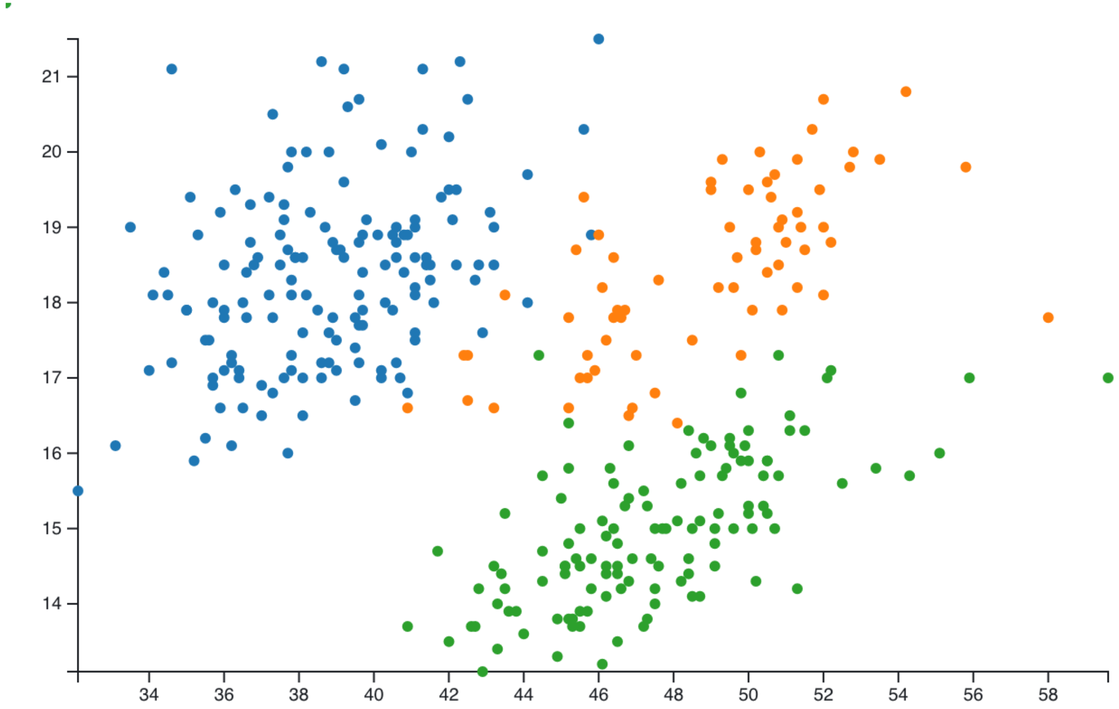
Domain: *List of possible values for categorical features*



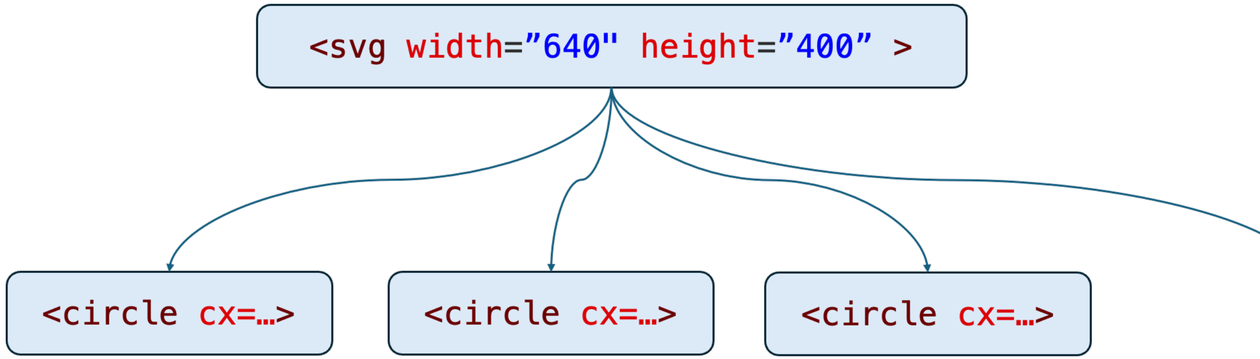
Axis elements

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



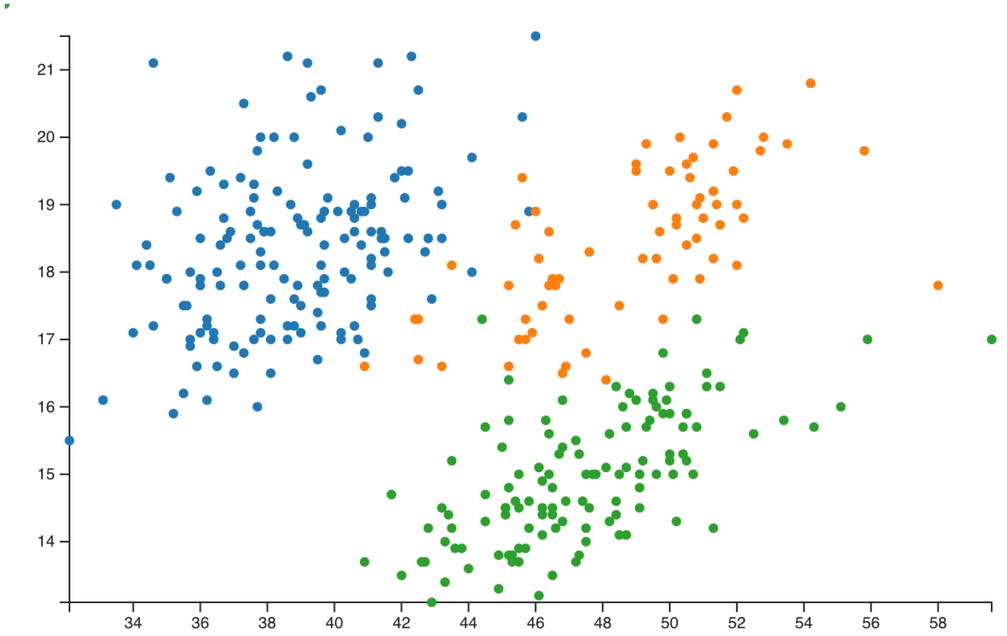
Set our dataset

Select any existing circles

Add a circle for each row

```
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)
```

Specify the attributes of each circle



Drawing the geometry

Set our dataset

Select any existing circles

Add a circle for each row

```
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)
```

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 aethetics
    set circle fill to row.species
    set circle cx to row.culmen_length_mm
    set circle cy to row.culmen_depth_mm
}
```

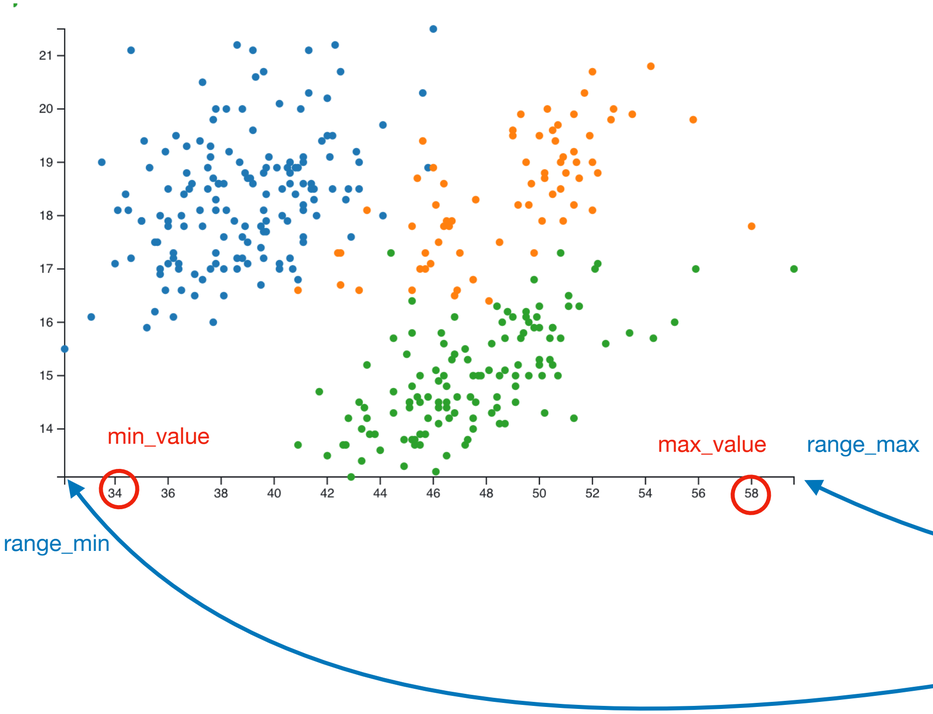
Drawing the geometry

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: