Front End

Basic Fundamentals

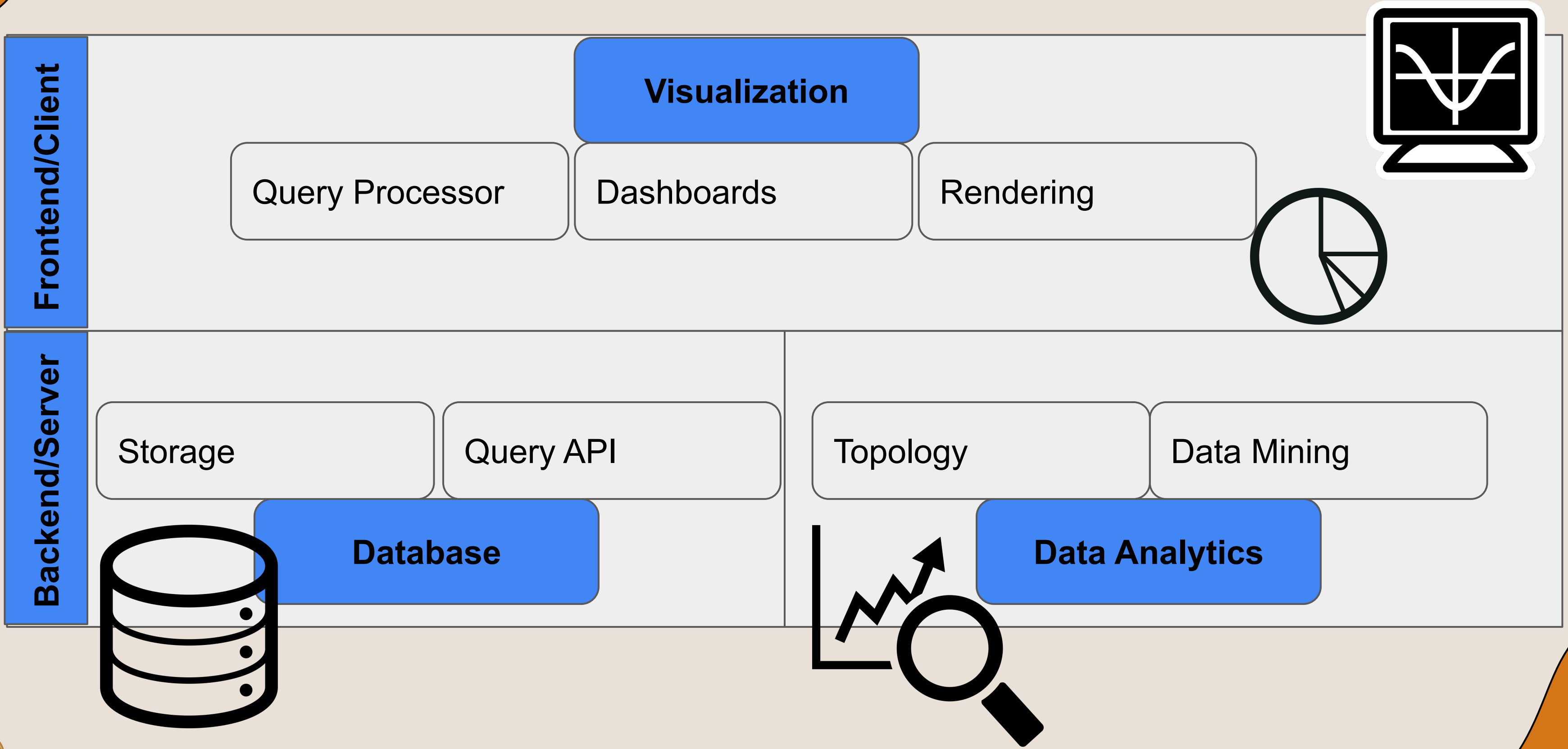
BUDGET COLLECTOR

Table of Contents

What Budget Collector Uses?

- Front-end:
  - JavaScript, HTML, CSS
  - built within the wordpress ecosystem
- Back-end:
  - Airtable
    - Contains art information (artist names, images, region, etc)
    - CANNOT manipulate data within airtable
  - AWS EC2
    - Backend server and database for website
    - CAN post manipulated data or run backend code
- Your Github Repo:
  - Needs to detail:
    - how to run it locally
    - what front-end & backend technologies that your using (flask, django, etc)

Visualization System



Visualization System

- Why is this type of architecture the norm?
  - Modular system allows for:
    - Rapid changes
    - Specialization
  - Easily understood in a client-server concept
  - Easy deployment

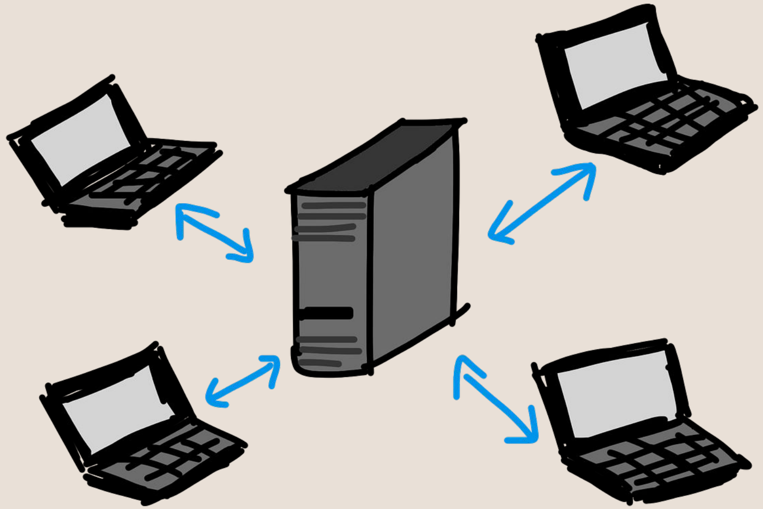
Website Basics: Client and Server

Client

- Any internet connected device or software (e.g. iphone, web browser, etc)
- Makes HTTP requests

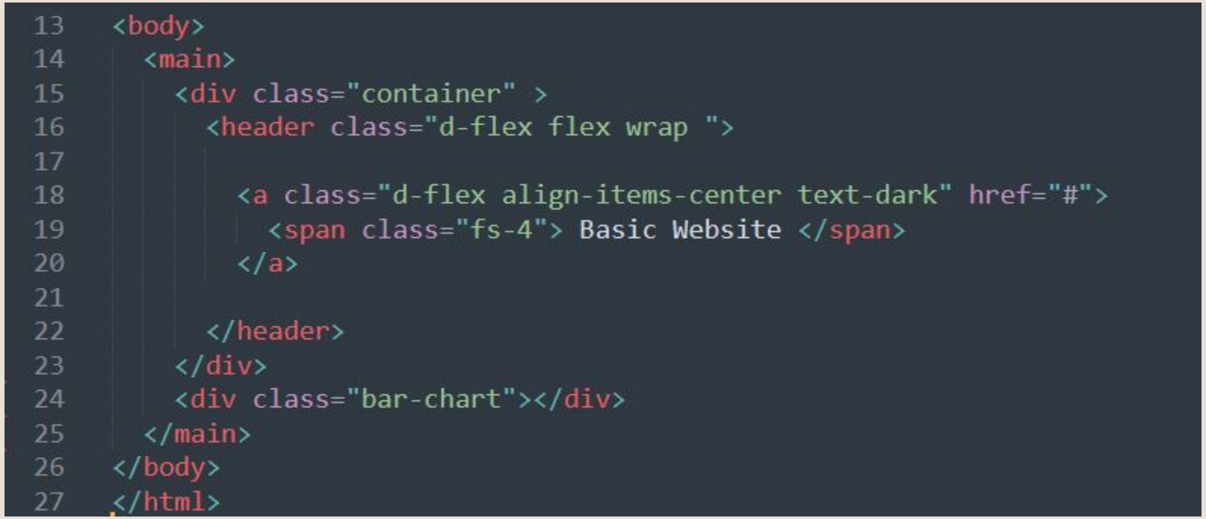
Server

- A device that stores web server software and component files (e.g. html, cvs, etc)
- Responds to clients with component files (i.e. data)



Website Basics: Markup Language + Style Sheets

- Most Used: HTML, CSS
- Interpreted by the Client Side
- Statically structure web pages



```html
<body>
  <main>
    <div class="container" >
      <header class="d-flex flex wrap ">

        <a class="d-flex align-items-center text-dark" href="#">
          <span class="fs-4"> Basic Website </span>
        </a>

      </header>
    </div>
    <div class="bar-chart"></div>
  </main>
</body>
</html>
```

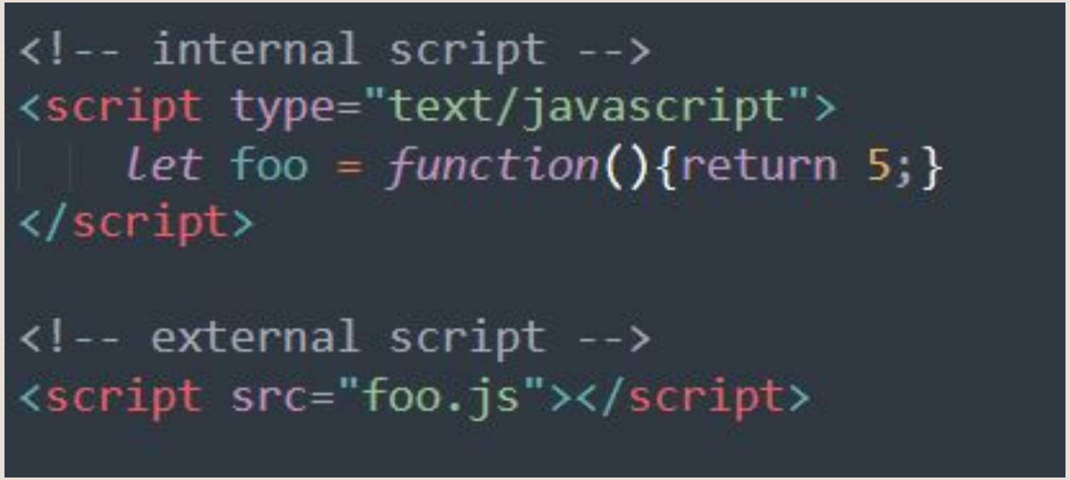
JavaScript Basics

Javascript:

- Client side language
- Loosely typed, object oriented
  - Variable type does not need to be specified
- Easy way to make dynamic pages
- Can be integrated w/ other frameworks and libraries
  - D3
  - ReactJS

JavaScript Basics cont.

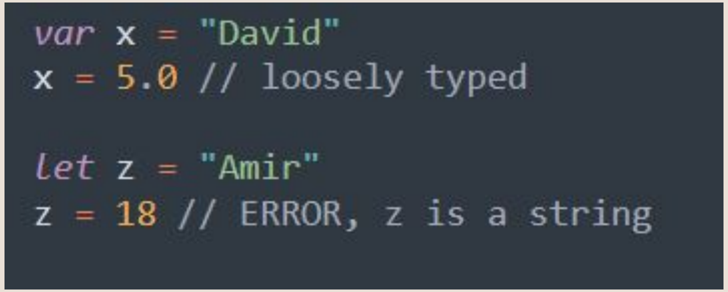
- Inserting JavaScript into HTML document
  - As an external file
  - With the HTML tag <script>



```
<!-- internal script -->
<script type="text/javascript">
    let foo = function(){return 5;}
</script>

<!-- external script -->
<script src="foo.js"></script>
```

BUDGET COLLECTOR

JavaScript Basics: Scopes + Variables



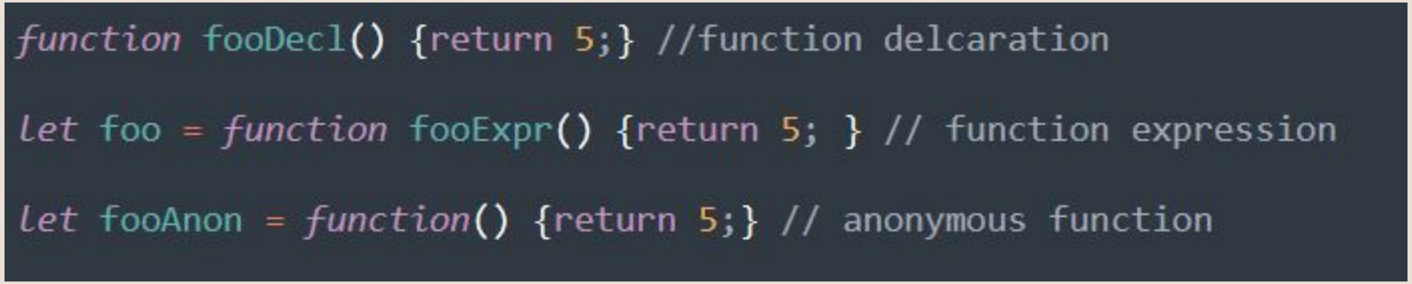
```
var x = "David"
x = 5.0 // loosely typed

let z = "Amir"
z = 18 // ERROR, z is a string
```

- Scopes:
  - Global
  - Local
    - Function
    - Block
- Variable Declaration:
  - Variables created with **‘var’** can have **function scope** and are **loosely typed**
  - Variables created with **‘let’ and ‘const’** have **all local scopes** and are **strictly typed**
    - were introduced in 2015
  - Variables created outside a function or block have global scope.

BUDGET COLLECTOR

JavaScript Basics: Functions

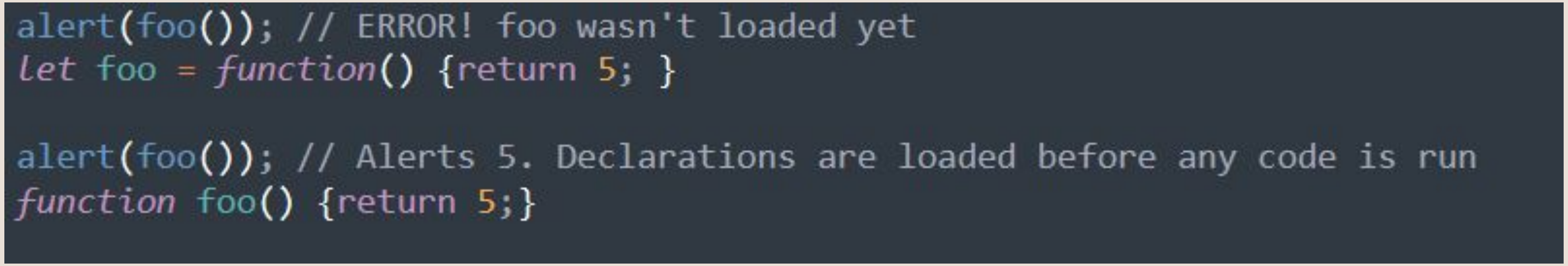
- Functions are first-class objects
  - Created, destroyed, passed to a function or returned as a value
- Three ways to define a function
  - Function declaration
  - Function expression
  - Anonymous function



```
function fooDecl() {return 5;} //function delcaration

let foo = function fooExpr() {return 5; } // function expression

let fooAnon = function() {return 5;} // anonymous function
```

JavaScript Basics: Functions cont.

- Function declarations are loaded before any code is run
- Function expression load when the interpreter reaches the line

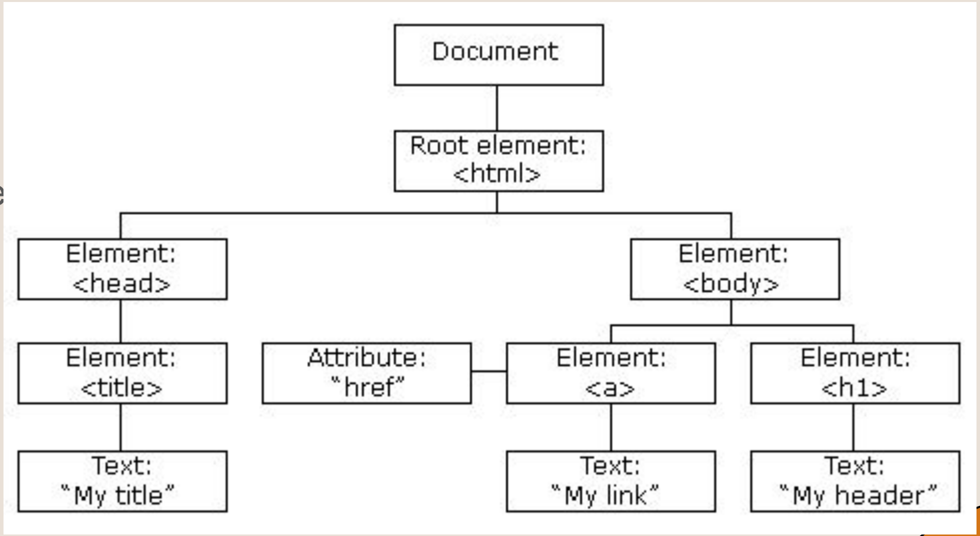


```
alert(foo()); // ERROR! foo wasn't loaded yet
let foo = function() {return 5; }

alert(foo()); // Alerts 5. Declarations are loaded before any code is run
function foo() {return 5;}
```

BUDGET COLLECTOR

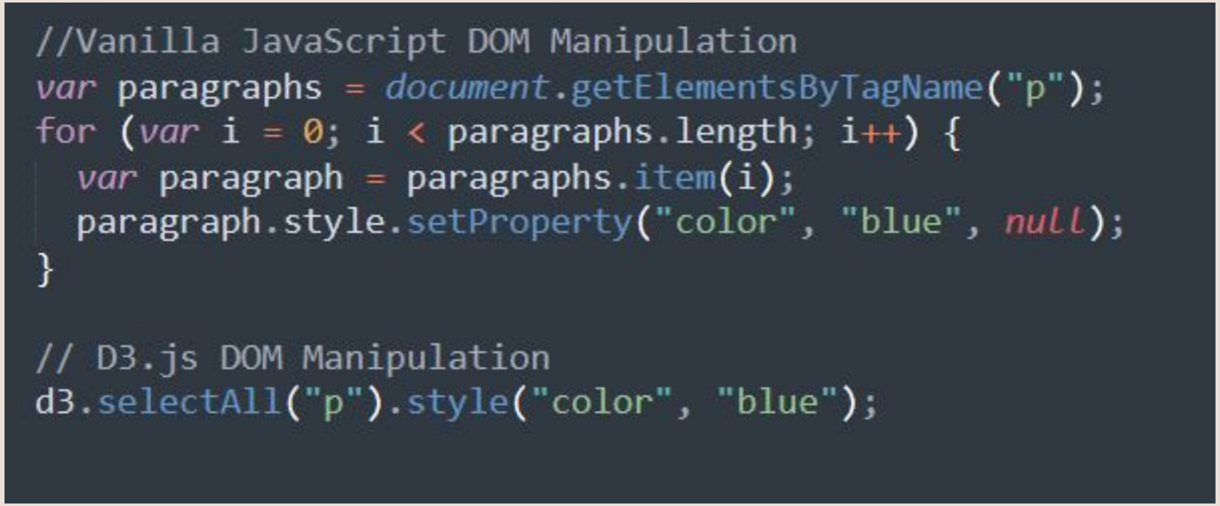
DOM Manipulation

- DOM stands for Document Object Model (aka document tree)
  - Made from our markup languages
- DOM manipulation entails:
  - Adding, deleting, or modifying the node on document tree
- JavaScript is common way of manipulating the DOM



DOM Manipulation w/ D3

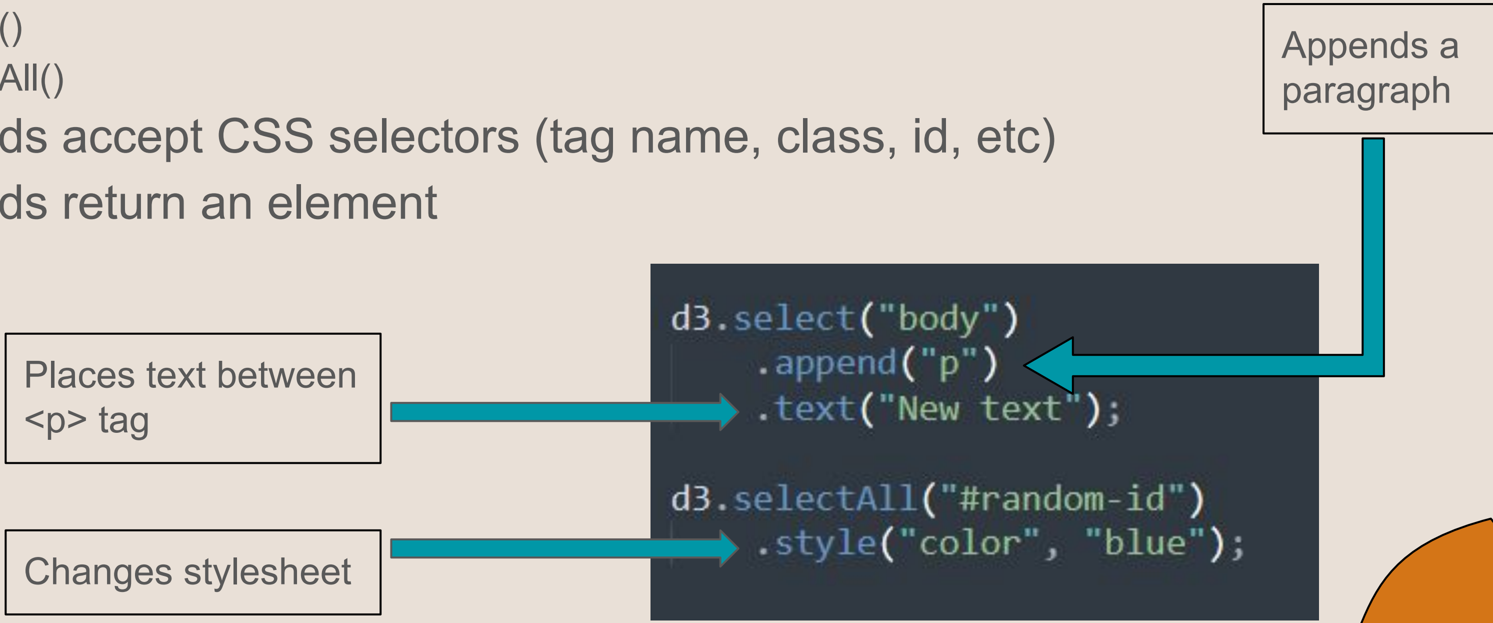
- D3.js
  - a JavaScript library for manipulating documents based on data.
  - combines visualization components to a data-driven approaches



```
//Vanilla JavaScript DOM Manipulation
var paragraphs = document.getElementsByTagName("p");
for (var i = 0; i < paragraphs.length; i++) {
  var paragraph = paragraphs.item(i);
  paragraph.style.setProperty("color", "blue", null);
}

// D3.js DOM Manipulation
d3.selectAll("p").style("color", "blue");
```

D3.js: Selections

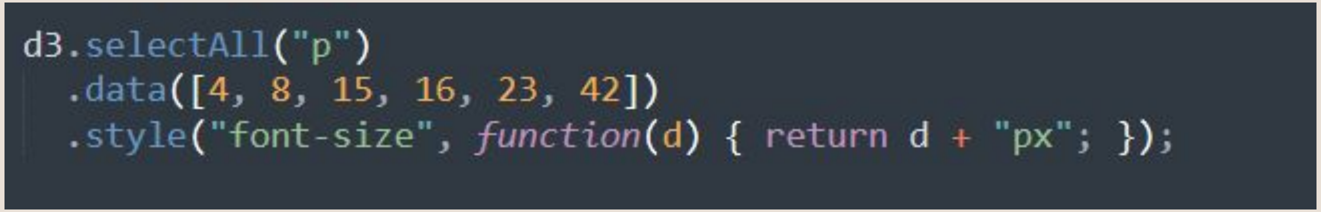
- D3.js operates on arbitrary sets of nodes called *selections*.
  - d3.select()
  - d3.selectAll()
- Both methods accept CSS selectors (tag name, class, id, etc)
- Both methods return an element



```
d3.select("body")
    .append("p")
    .text("New text");

d3.selectAll("#random-id")
    .style("color", "blue");
```

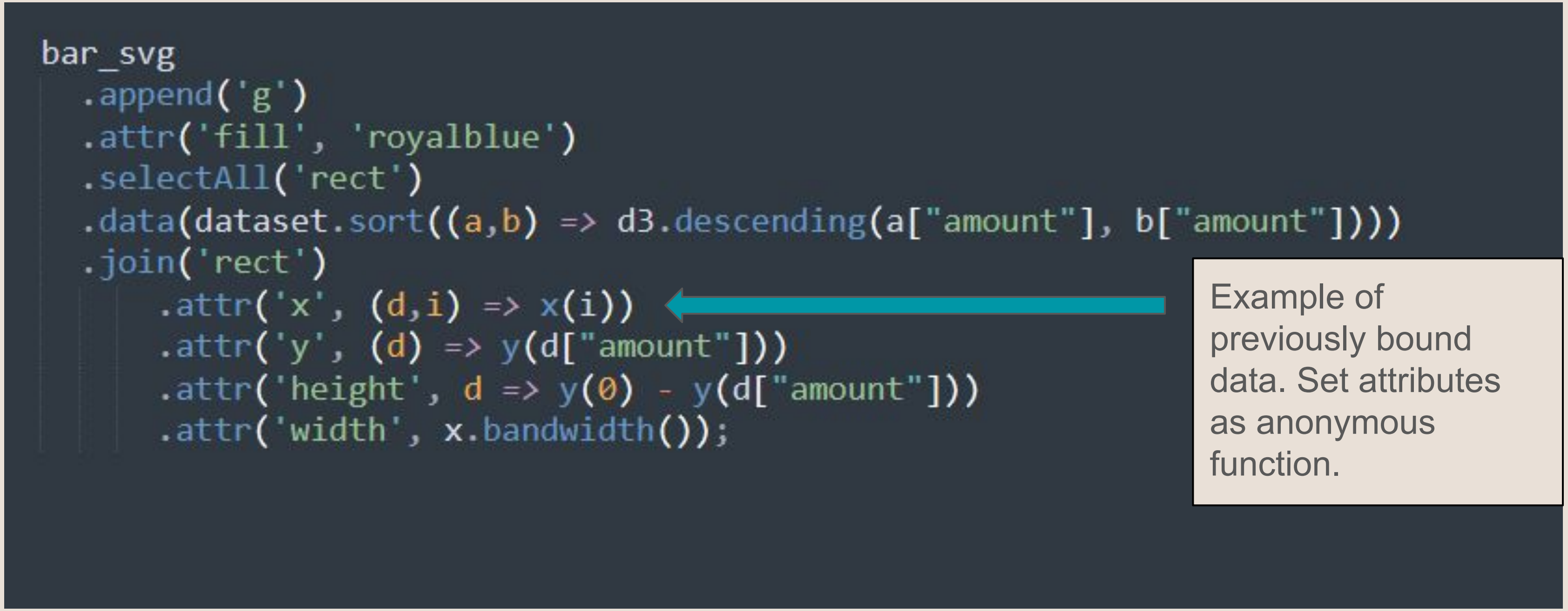
D3.js: Bound Data + Attributes

- With the d3.data() method, data can be bound to a selection
- Once bound to the selection, you can omit the data operator.
    - D3.js will retrieve the previously-bound data
- By default, data is bound sequentially where element *i* is bound to data *i* and so forth.



```
d3.selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .style("font-size", function(d) { return d + "px"; });
```

D3.js: Bound Data + Attributes cont.



```
bar_svg
  .append('g')
  .attr('fill', 'royalblue')
  .selectAll('rect')
  .data(dataset.sort((a,b) => d3.descending(a["amount"], b["amount"])))
  .join('rect')
      .attr('x', (d,i) => x(i))
      .attr('y', (d) => y(d["amount"]))
      .attr('height', d => y(0) - y(d["amount"]))
      .attr('width', x.bandwidth());
```

Example of previously bound data. Set attributes as anonymous function.

BUDGET COLLECTOR

D3.js: Update, Enter and Exit

Update

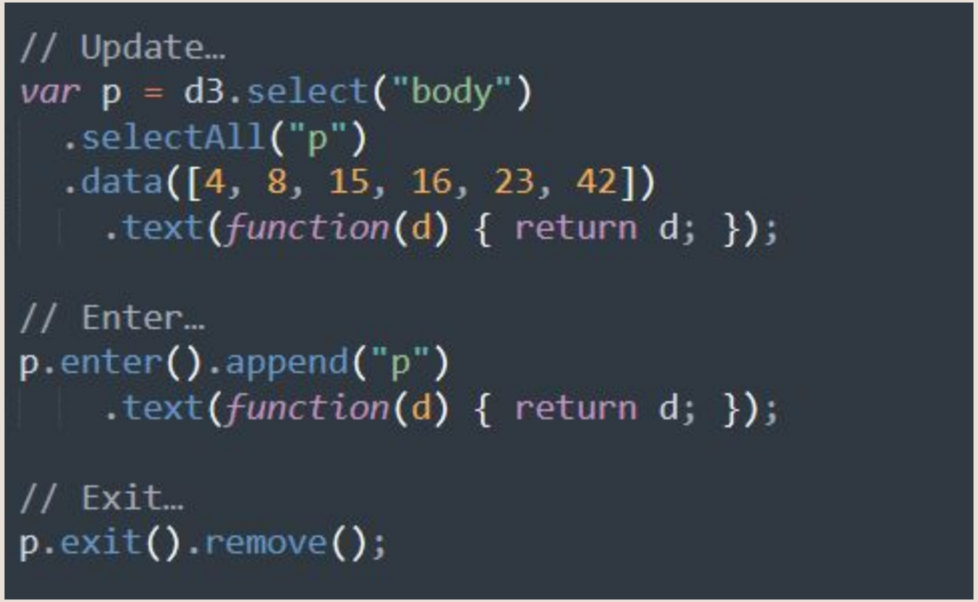
- Updates existing nodes, bound to the data

Enter

- A placeholder for missing nodes

Exit

- Removes the remaining nodes



```
// Update…
var p = d3.select("body")
  .selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
    .text(function(d) { return d; });

// Enter…
p.enter().append("p")
    .text(function(d) { return d; });

// Exit…
p.exit().remove();
```

BUDGET COLLECTOR

Web App Frameworks

What?

- A software framework that is designed to support development and deployment on the web.

Why?

- Structures the code in an understandable — and hopefully efficient — manner
- Allows for the coder to be feature focused instead of configuration focused.

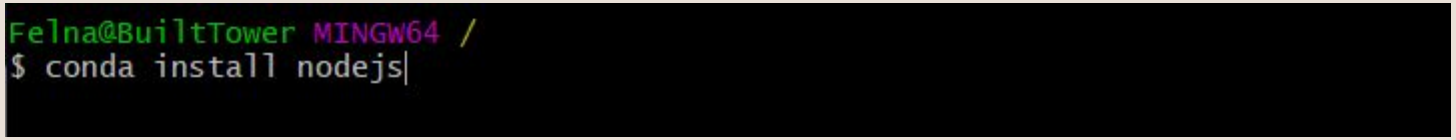
Angular

- Web framework, built with TypeScript (relative of JavaScript)
- Cross-platform and component based
- Has a tons of libraries with a tons of features

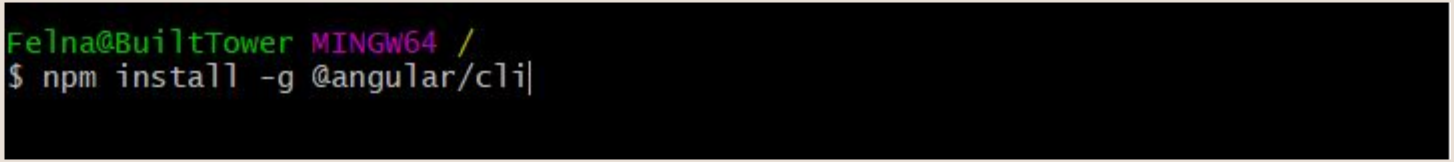
BUDGET COLLECTOR

Angular: Requirements

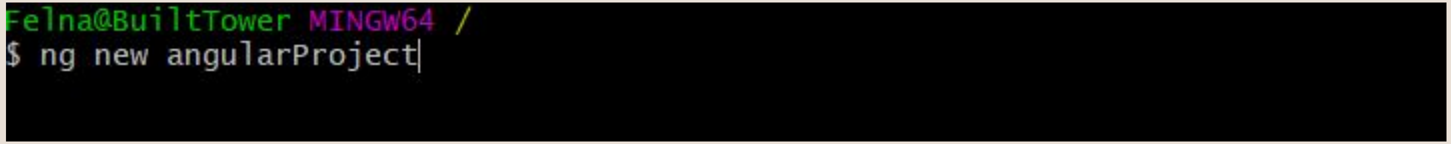
- Node.js: javascript runtime environment
- NPM: package manager for javascript
- Angular Cli: command line interface tool to create angular projects



```
Felna@BuiltTower MINGW64 /
$ conda install nodejs
```



```
Felna@BuiltTower MINGW64 /
$ npm install -g @angular/cli
```



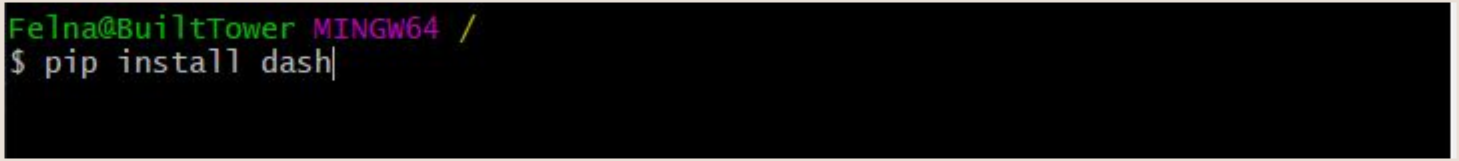
```
Felna@BuiltTower MINGW64 /
$ ng new angularProject
```

Dash Plotly

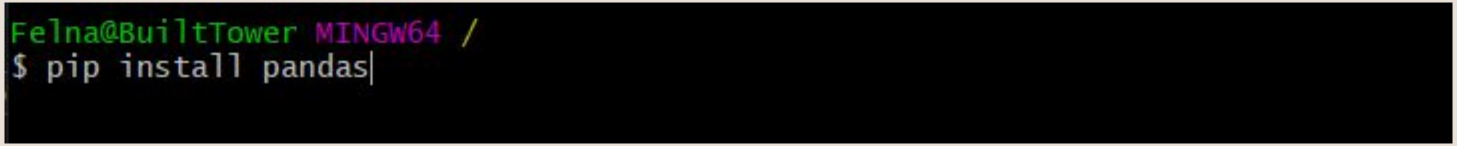
- Low code web framework built for Python, R and Julia
- Cross platform and component based
- Focused on creating analytic driven dashboards
  - due to this, it has native analytical features

Dash Plotly: Requirements

- Python: coding language
- Pandas: data analysis manipulation tool
- Dash: library for creating dash projects



```
Felna@BuiltTower MINGW64 /
$ pip install dash
```



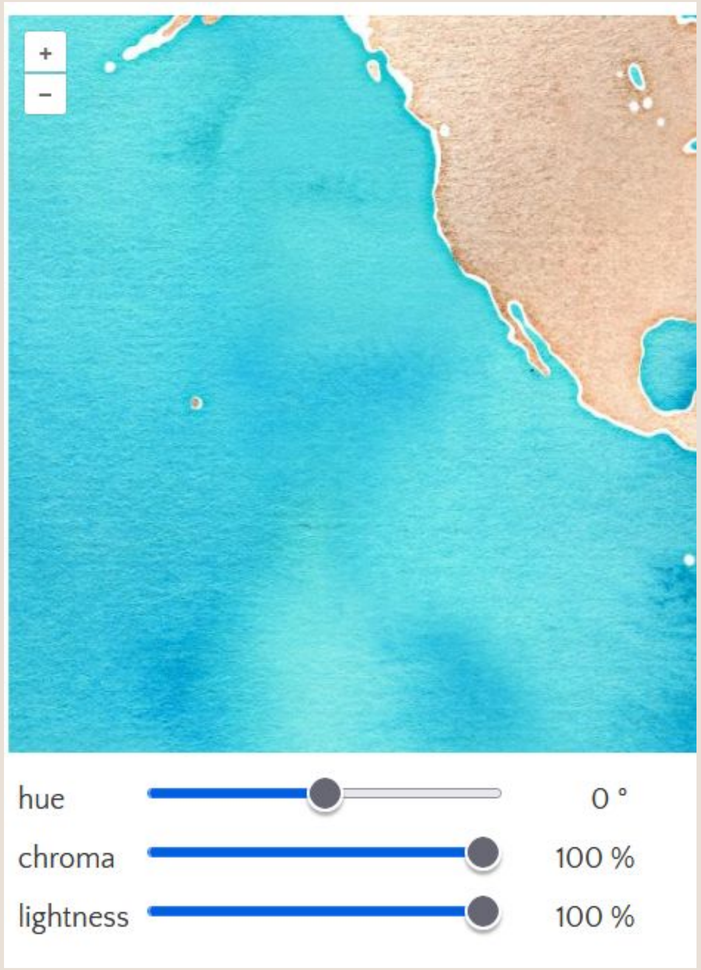
```
Felna@BuiltTower MINGW64 /
$ pip install pandas
```

Topography Tools

- OpenLayers
- Google Geochart
- Plotly (has native regional maps)

OpenLayers

- Built in **Javascript**, renders elements with SVG
- Makes it easy to use dynamic maps
- Displays
  - map tiles,
  - vector data
  - markers



hue 0 °
chroma 100 %
lightness 100 %

GeoChart

- Built in **Javascript**, renders elements with SVG
- Makes it easy to use dynamic maps
- Can
  - Color regions
  - Set markers
  - Label with texts