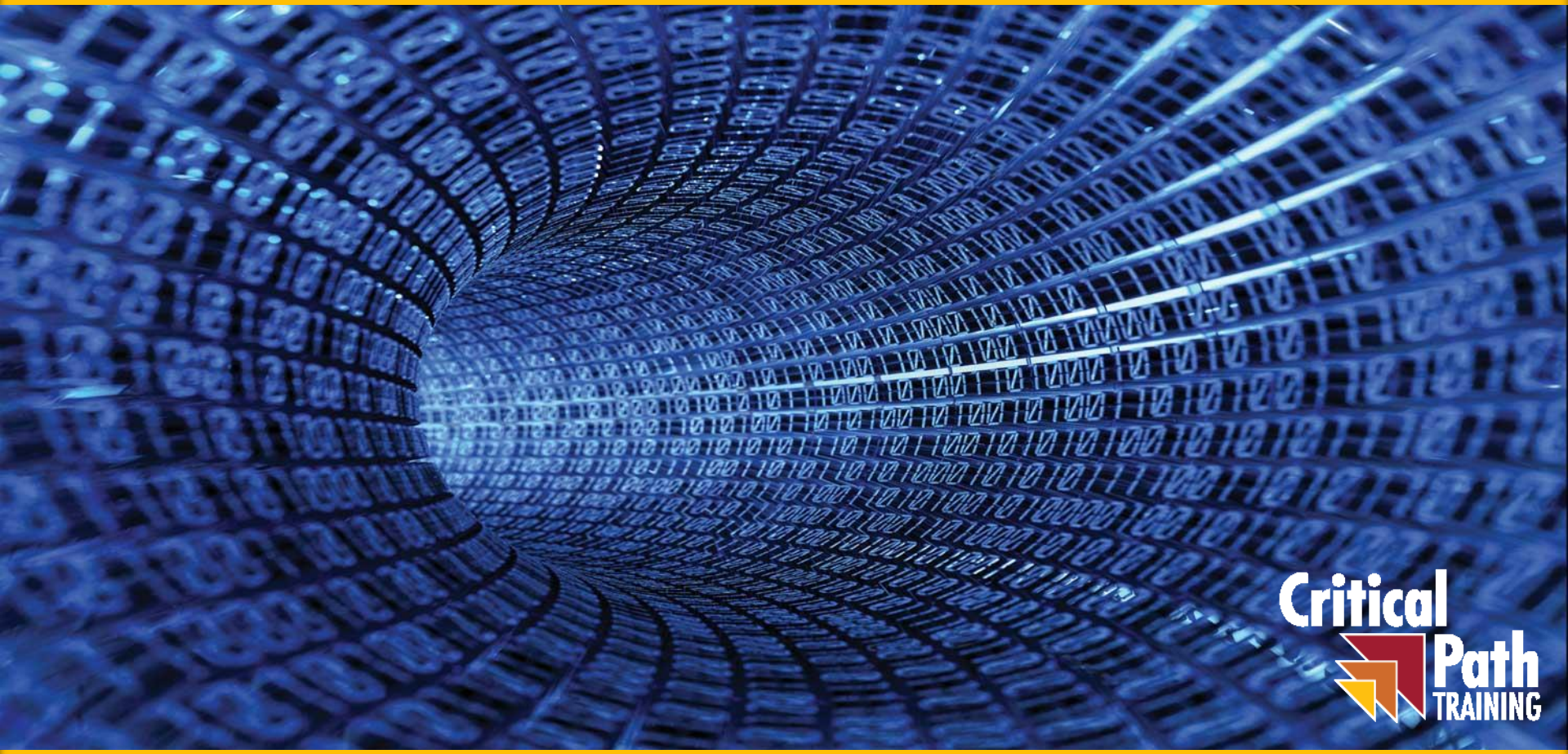
Programming with TypeScript and the D3 Library

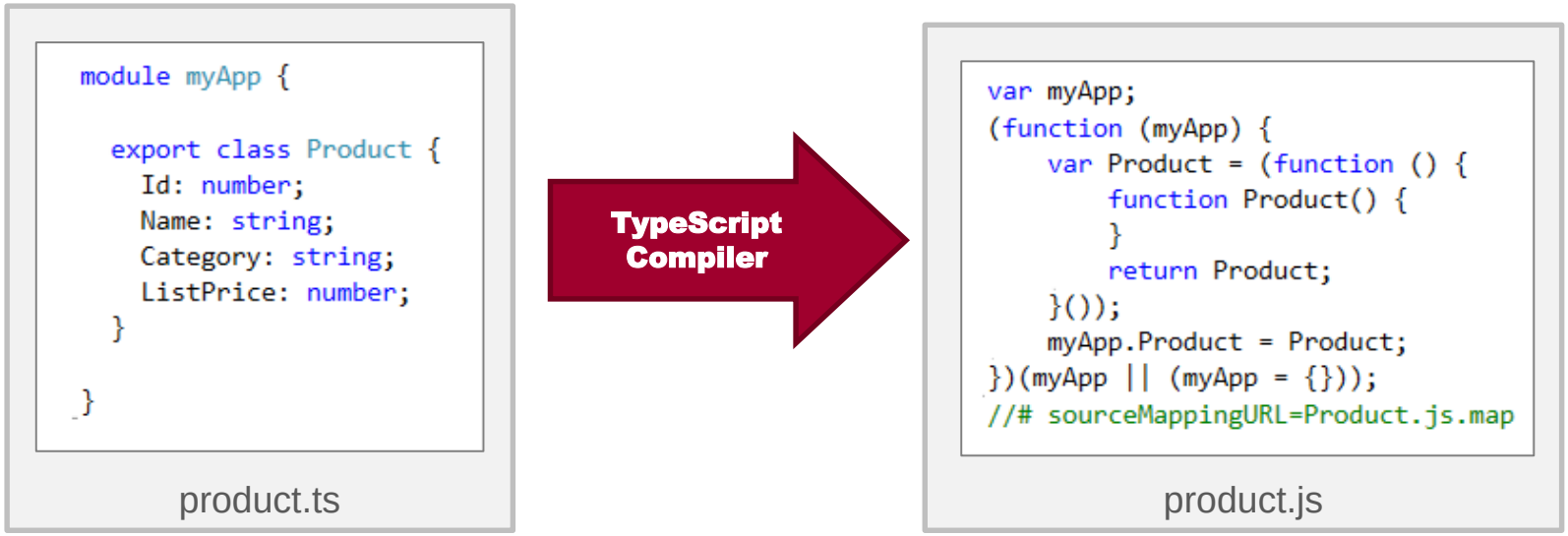


Agenda

- TypeScript Language Primer
- Getting Started with D3 and SVG Graphics
- Creating Data-driven Visuals
- Enhancing Visuals with Scales and Axes
- Event Handling and Transitions
- Using D3 Layouts

What is TypeScript?

- A programming language which compiles into plain JavaScript
- A superset of JavaScript that adds a strongly-typed dimension
- It can be compiled into ECMAScript3, ECMAScript3 or ECMAScript 6
- It runs in any browser, in any host and on any OS



Type Annotation

- TypeScript allows you to annotate types
  - Provides basis for strongly-typed programming
  - Type annotations used by compiler for type checking
  - Type annotations are erased at the end of compile time

```
// define strongly-typed function
var myFunction = function (param1: number): string {
    return "You passed " + param1;
};

// define strongly-typed variables
var myNumber: number = 2017;
var myMessage: string = myFunction(myNumber);
var myContent: JQuery = $("<p>").text(myMessage);
var contentBox: JQuery = $("#content-box");

contentBox.empty().append(myContent);
```

Assignment with let versus var

- `var` does not recognize nor honor scope
- `let` will recognize and honor scope

```
var x:number = 2016;
let y: number = 2016;

{
  var x:number = 2017;
  let y:number = 2017;
}

let message = "x=" + x + " and " + "y=" + y;
```

x=2017 and y=2016

Parameter Arrays

```
// define function with a parameter array using (...) syntax
function createOrderedList(...names: string[]): string{
  let html: string = "<ol>";
  for (let index: number = 0; index < names.length; index++) {
    html += "<li>" + names[index] + "</li>"
  }
  return html += "</ol>";
};

// create a string array
let stooges: string[] = ["Moe", "Curly", "Larry"];

// call function with a parameter array
let stoogesList: string = createOrderedList(...stooges);
```

Arrow Function Syntax

- TypeScript supports arrow function syntax
  - Concise syntax to define anonymous functions
  - Can be used to retain this pointer in classes

```
// create anonymous function using function arrow sytax
let myFunction = () => {
  console.log("Hello world");
};

// use function arrow sytax with typed parameters
let myOtherFunction = (param1: number, param2: string) : string => {
  return param1 + " - " + param2;
};

// create function to assign to DOM event
window.onresize = (event: Event) => {
  let window: Window = event.target as Window;
  console.log("Window width: " + window.outerWidth);
  console.log("Window height: " + window.outerHeight);
};
```

Classes

- TypeScript supports defining classes
  - Class defines type for object
  - Export keyword makes class created across files
  - Class can be passed as factory function
  - Default accessibility is public

```
export class Product {
  Id: number;
  Name: string;
  Category: string;
  ListPrice: number;
}
```

```
// create new Product instance
let product1: Product = new Product();
product1.Id = 1;
product1.Name = "Batman Action Figure";
product1.Category = "Action Figure";
product1.ListPrice = 14.95;
```

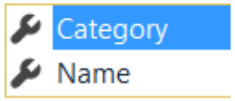
Class Constructors

- Constructor parameters become fields in class

```
export class Product {

  constructor(private Id: number, public Name: string, public Category: string, private ListPrice: number) {
    // no need to do anything here
  }

  MyPublicMethod() {
    // access to private fields
    let id: number = this.Id
    let price: number = this.ListPrice
  }

}
```

- Client-side code calls constructor using new operator

```
// create new Product instance
let product1: Product = new Product(1, "Batman Action Figure", "Action Figure", 14.95);

// access public properties
let product1Name: string = product1.Name;
let product1Category: string = product1.
```



Interfaces

- Interface defines a programming contract
  - Classes can implement interfaces

```
export interface IProductDataService {
  GetAllProducts(): Product[];
  GetProduct(id: number): Product;
  AddProduct(product: Product): void;
  DeleteProduct(id: number): void;
  UpdateProduct(product: Product): void;
}
```

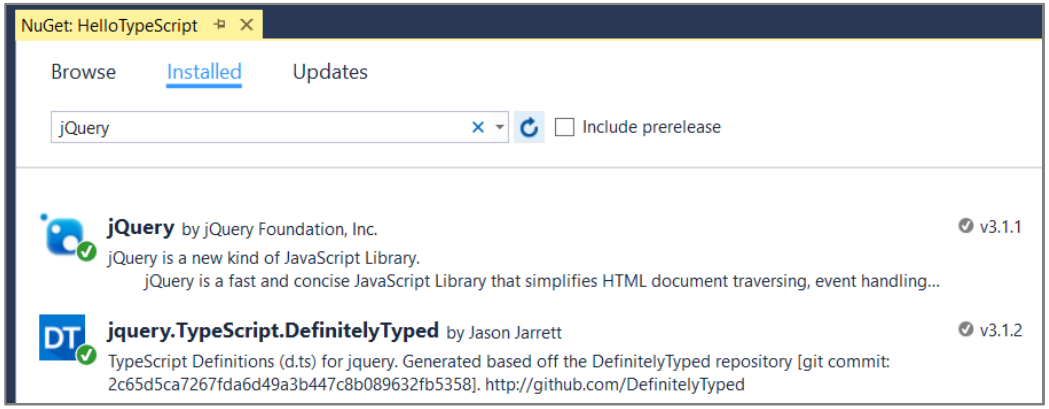
```
export class MyProductDataService implements IProductDataService {

  private products: Product[] =...;

  GetAllProducts(): Product[]...;

  GetProduct(id: number): Product...;

  AddProduct(product: Product): void...;

  DeleteProduct(id: number): void...;

  UpdateProduct(product: Product): void...;
}
```

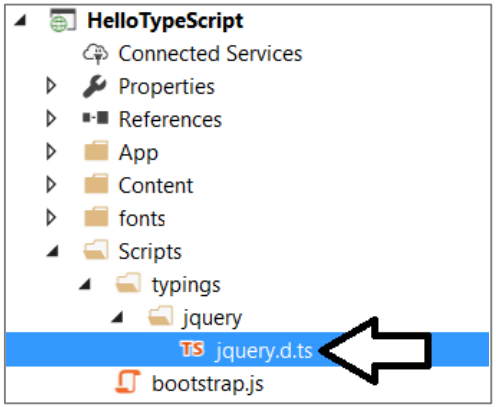
  - Client code can be decoupled from concrete classes

```
// program against variables based on interface type
let productService: IProductDataService = new MyProductDataService();

// clioent code is decoupled from underlying data access class implementations
let products: Product[] = productService.GetAllProducts();
let product1: Product = productService.GetProduct(1);
```

TypeScript Definition Files (d.ts)

- What are TypeScript definition files
  - Typed definitions for 3rd party JavaScript libraries
  - DefinitelyTyped provides great community resource
  - Typed definition files have a **`d.ts`** extension






```
// define strongly-typed variables
var myNumber: number = 2017;
var myMessage: string = myFunction(myNumber);
var myContent: JQuery = $("<p>").text(myMessage);
var contentBox: JQuery = $("#content-box");
```

Interface-based Design

- Interfaces define programming contracts

```
export interface IViewPort {
    width: number;
    height: number;
}
```

```
export interface ICustomVisual {
    name: string;
    load(container: HTMLElement): void;
    update(viewport: IViewPort): void;
}
```

- Application design can use interfaces instead of concrete classes

```
module myApp {

    var leftNavCollapsed: boolean = true;
    var loadedVisual: ICustomVisual;

    var visuals: ICustomVisual[] = [
        new Viz01(), new Viz02(), new Viz03(), new Viz04()
    ];

    function LoadVisual(visual: ICustomVisual)...

    $(() =>...);
```

Sample Custom Visual using jQuery

```
export class Viz01 implements ICustomVisual {

    public name: string = "Visual 1: Hello jQuery";
    private container: JQuery;
    private message: JQuery;

    load(container: HTMLElement)...

    public update(viewport: IViewPort)...

}
```

```
load(container: HTMLElement) {

    this.container = $(container);

    this.message = $("<div>")
        .text("Hello jQuery")
        .css({
            "display": "table-cell",
            "text-align": "center",
            "vertical-align": "middle",
            "text-wrap": "none",
            "background-color": "yellow"
        });

    this.container.append(this.message);
}
```

```
public update(viewport: IViewPort) {

    let paddingX: number = 2;
    let paddingY: number = 2;
    let fontSizeMultiplierX: number = viewport.width * 0.15;
    let fontSizeMultiplierY: number = viewport.height * 0.4;
    let fontSizeMultiplier: number = Math.min(...[fontSizeMultiplierX,
                                                  fontSizeMultiplierY]);

    this.message.css({
        "width": viewport.width - paddingX,
        "height": viewport.height - paddingY,
        "font-size": fontSizeMultiplier
    });

}
```

Agenda

- ✓ TypeScript Language Primer
- ➢ Getting Started with D3 and SVG Graphics
- Creating Data-driven Visuals
- Enhancing Visuals with Scales and Axes
- Event Handling and Transitions
- Using D3 Layouts

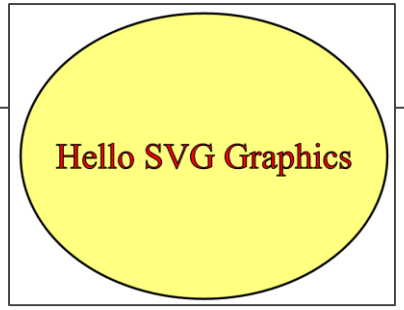
The D3 Library

- What does the D3 library do?
  - Loading data into the browser's memory
  - Binding data to create new set of SVG elements
  - Adding and removing SVG elements as needed
  - Transforming SVG elements by setting properties
  - Transitioning SVG elements in response to user actions

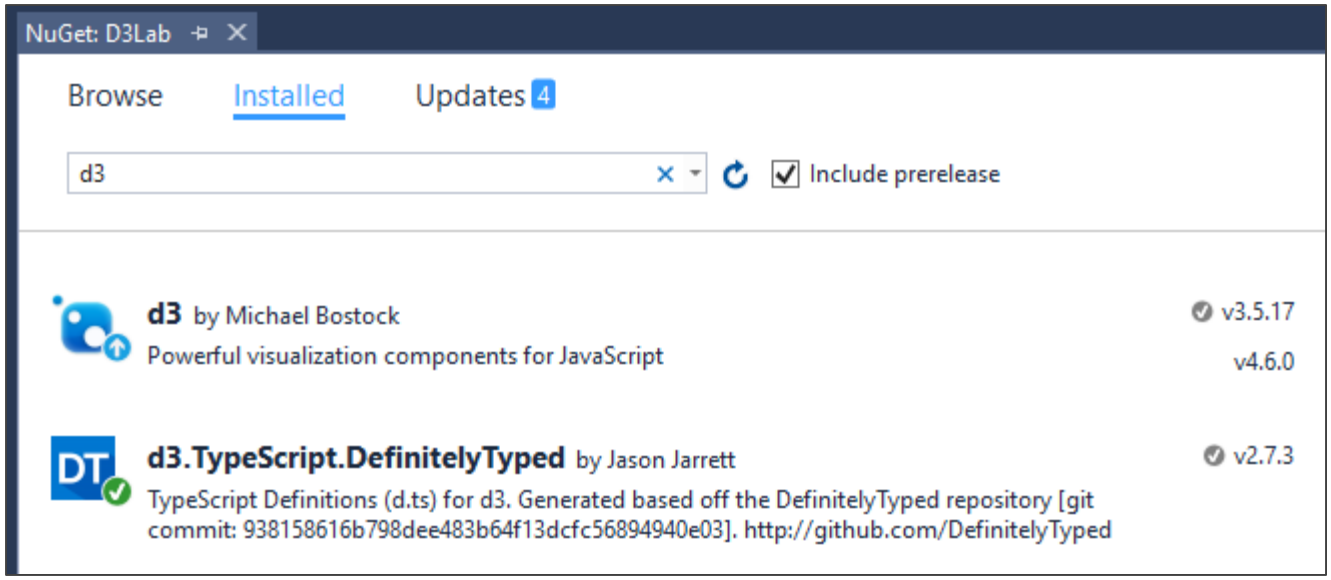
SVG Graphics

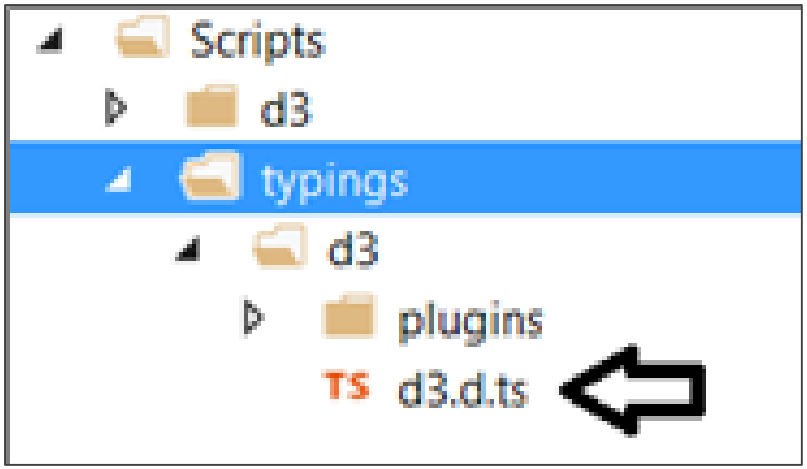
- SVG = Scalable Vector Graphics
  - Specialized type of HTML element
  - More reliable and consistent than other HTML elements

```
<svg width="700" height="550">

  <ellipse cx="350" cy="275" rx="330" ry="256"
           style="fill: rgba(255, 255, 0, 0.498039);
         stroke: rgb(0, 0, 0); stroke-width: 4;">
  </ellipse>

  <text x="350" y="275" width="660" height="512" font-size="132"
        text-anchor="middle" dominant-baseline="central"
        style="fill: rgb(255, 0, 0); stroke: rgb(0, 0, 0); stroke-width: 2;">
    Hello SVG Graphics
  </text>

</svg>
```



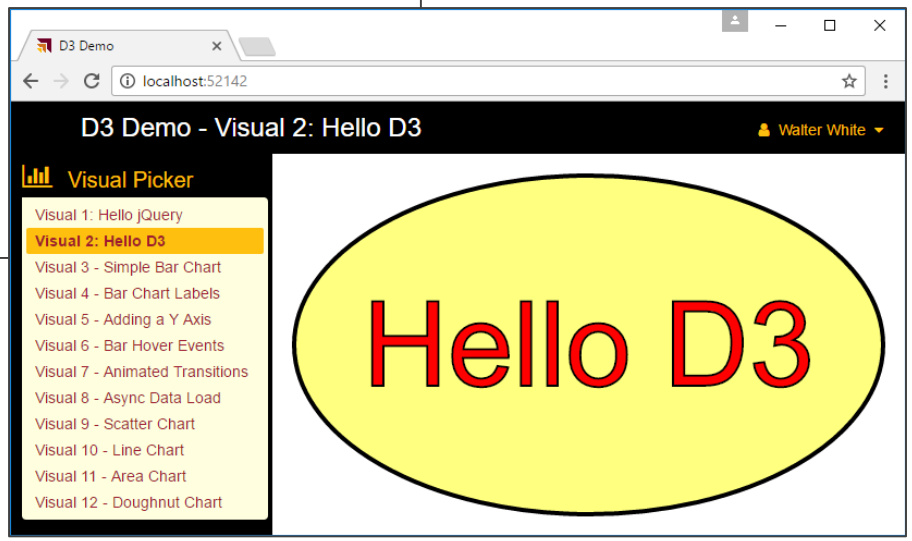
Adding d3 and d3 typings files





Designing a D3 Custom Visual

```
export class Viz02 implements ICustomVisual {

    name = "Visual 2: Hello D3";

    private svgRoot: d3.Selection<SVGElementInstance>;
    private ellipse: d3.Selection<SVGElementInstance>;
    private text: d3.Selection<SVGElementInstance>;
    private padding: number = 20;

    load(container: HTMLElement)...

    update(viewport: IViewPort)...

}
```



Implementing load

```
load(container: HTMLElement) {

  this.svgRoot = d3.select(container).append("svg");

  this.ellipse = this.svgRoot.append("ellipse")
    .style("fill", "rgba(255, 255, 0, 0.5)")
    .style("stroke", "rgba(0, 0, 0, 1.0)")
    .style("stroke-width", "4");

  this.text = this.svgRoot.append("text")
    .text("Hello D3")
    .attr("text-anchor", "middle")
    .attr("dominant-baseline", "central")
    .style("fill", "rgba(255, 0, 0, 1.0)")
    .style("stroke", "rgba(0, 0, 0, 1.0)")
    .style("stroke-width", "2");

}
```

Implementing update

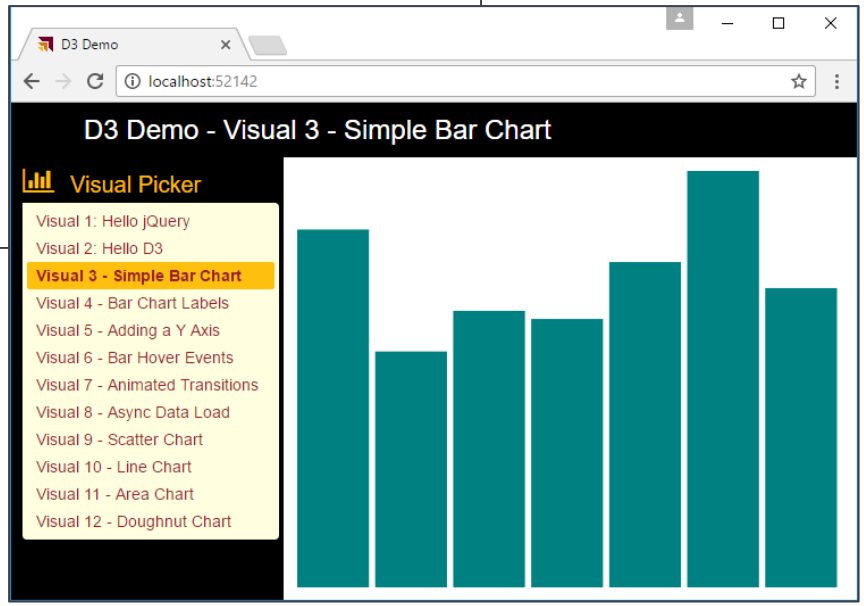
```
update(viewport: IViewPort) {

  this.svgRoot
    .attr("width", viewport.width)
    .attr("height", viewport.height);

  var plot = {
    xOffset: this.padding,
    yOffset: this.padding,
    width: viewport.width - (this.padding * 2),
    height: viewport.height - (this.padding * 2),
  };

  this.ellipse
    .attr("cx", plot.xOffset + (plot.width * 0.5))
    .attr("cy", plot.yOffset + (plot.height * 0.5))
    .attr("rx", (plot.width * 0.5))
    .attr("ry", (plot.height * 0.5))

  var fontSizeForWidth: number = plot.width * .20;
  var fontSizeForHeight: number = plot.height * .35;
  var fontSize: number = d3.min([fontSizeForWidth, fontSizeForHeight]);

  this.text
    .attr("x", plot.xOffset + (plot.width / 2))
    .attr("y", plot.yOffset + (plot.height / 2))
    .attr("width", plot.width)
    .attr("height", plot.height)
    .attr("font-size", fontSize);
}
```

Agenda

- ✓ TypeScript Language Primer
- ✓ Getting Started with D3 and SVG Graphics
- ➢ Creating Data-driven Visuals
- Enhancing Visuals with Scales and Axes
- Event Handling and Transitions
- Using D3 Layouts

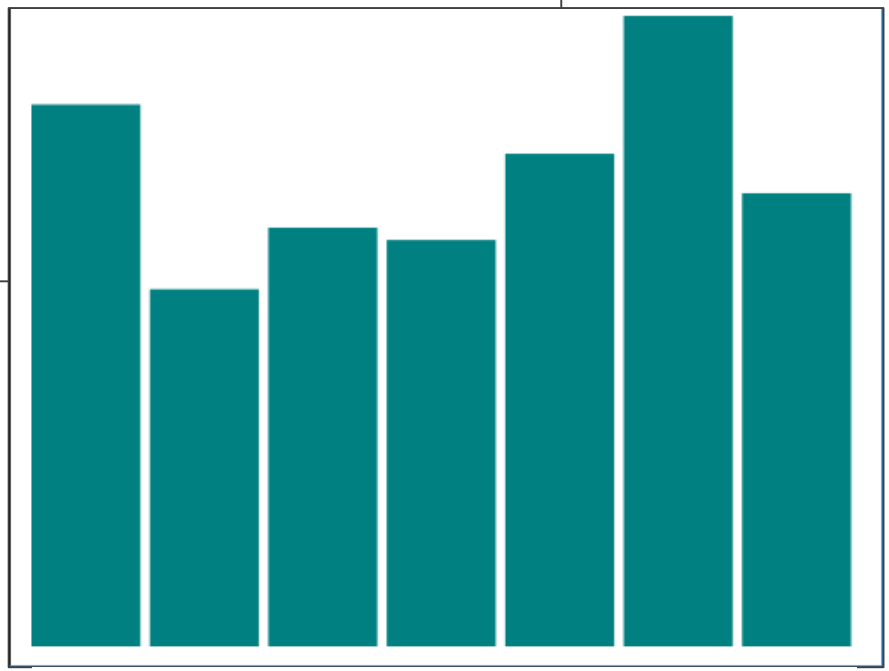
Designing a D3 Visual with a Bar Chart

```
export class Viz03 implements ICustomVisual {

    name = "Visual 3 - Simple Bar Chart";

    private dataset = [440, 290, 340, 330, 400, 512, 368];

    private svgRoot: d3.Selection<SVGElementInstance>;
    private bars: d3.Selection<number>;
    private padding: number = 12;

    load(container: HTMLElement)...

    update(viewport: IViewPort)...

}
```



Implementing load

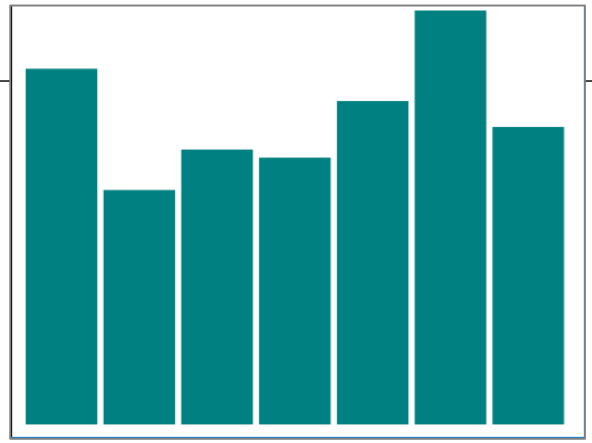
```
load(container: HTMLElement) {

  this.svgRoot = d3.select(container).append("svg");

  this.bars = this.svgRoot
    .selectAll("rect")
    .data(this.dataset)
    .enter()
    .append("rect");

}
```



[440, 290, 340, 330, 400, 512, 368]

Implementing update

```
update(viewport: IViewPort) {

  this.svgRoot
    .attr("width", viewport.width)
    .attr("height", viewport.height);

  var plot = {
    xOffset: this.padding,
    yOffset: this.padding,
    width: viewport.width - (this.padding * 2),
    height: viewport.height - (this.padding * 2),
  };

  var datasetSize = this.dataset.length;
  var xScaleFactor = plot.width / datasetSize;
  var yScaleFactor = plot.height / d3.max(this.dataset);
  var barWidth = (plot.width / datasetSize) * 0.92;

  this.bars
    .attr("x", (d, i) => { return plot.xOffset + (i * (xScaleFactor)); })
    .attr("y", (d, i) => { return plot.yOffset + plot.height - (Number(d) * yScaleFactor); })
    .attr("width", (d, i) => { return barWidth; })
    .attr("height", (d, i) => {  return (Number(d) * yScaleFactor); })
    .attr("fill", "teal");
}
```



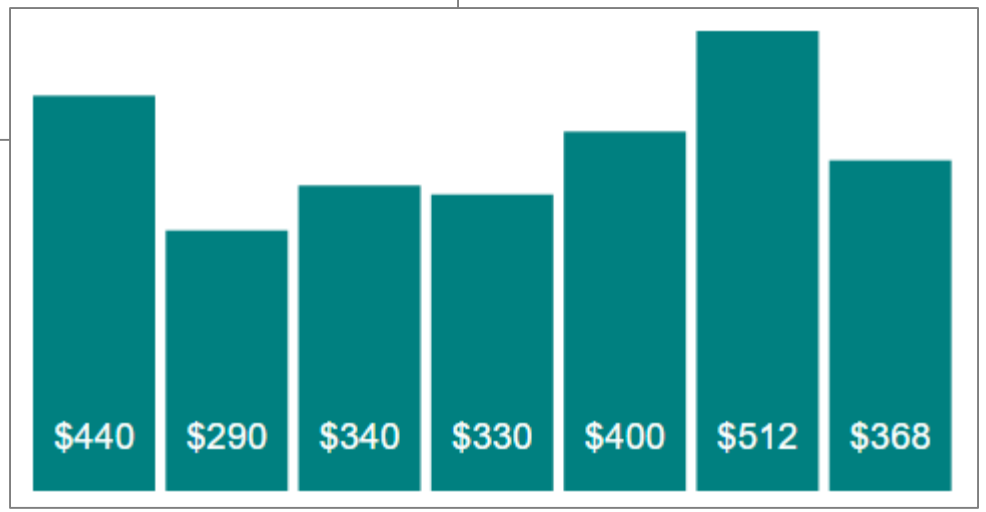
[440, 290, 340, 330, 400, 512, 368]

Adding Labels to the Bar Chart

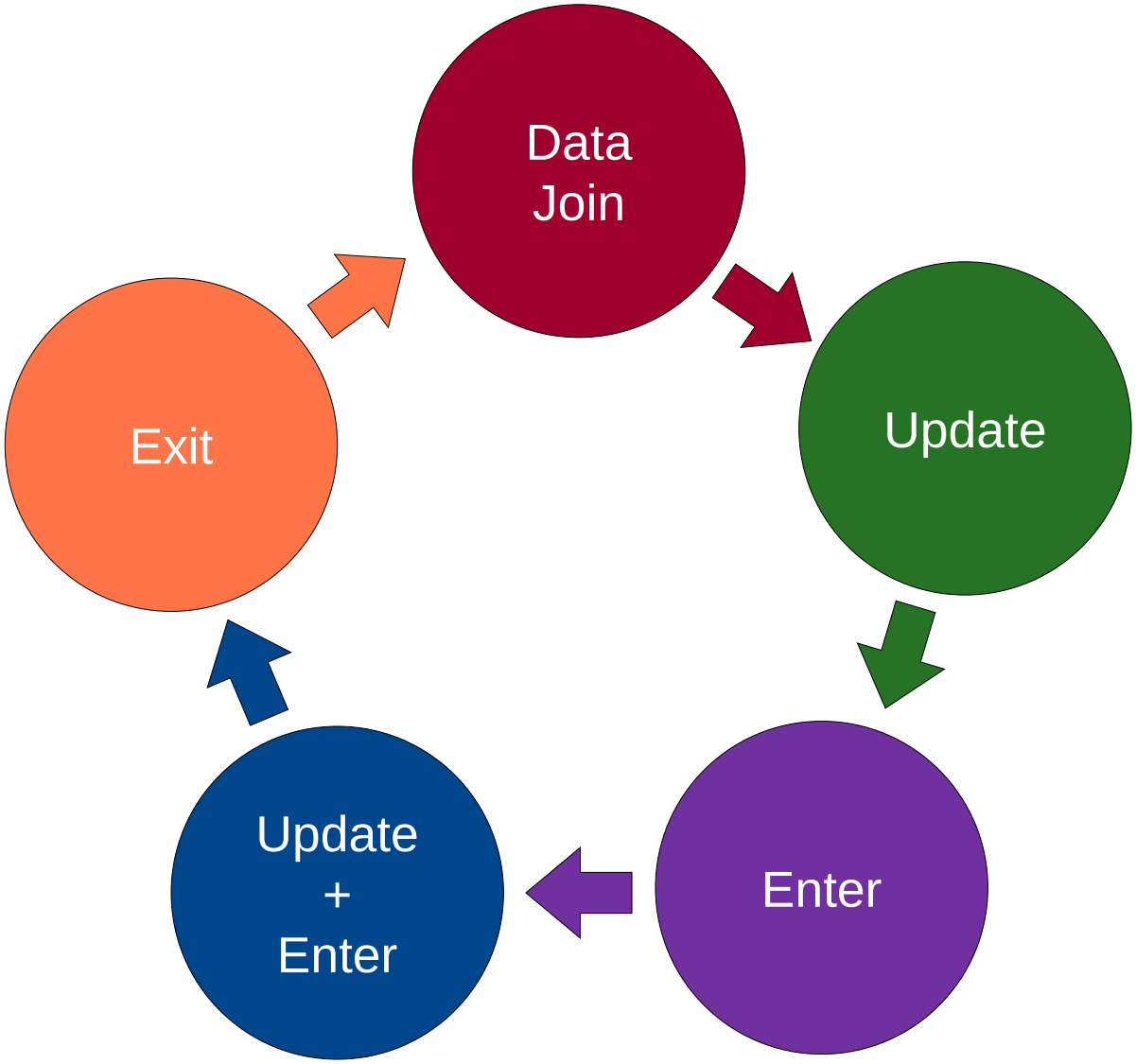
```
export class Viz04 implements ICustomVisual {

    name = "Visual 4 - Bar Chart Labels";

    private dataset = [440, 290, 340, 330, 400, 512, 368];

    private svgRoot: d3.Selection<SVGElementInstance>;
    private bars: d3.Selection<number>;
    private labels: d3.Selection<number>;
    private padding: number = 12;

    load(container: HTMLElement)...

    update(viewport: IViewPort)...
}
```

Implementing load

```
load(container: HTMLElement) {

  this.svgRoot = d3.select(container).append("svg");

  this.bars = this.svgRoot
    .selectAll("rect")
    .data(this.dataset)
    .enter()
    .append("rect");

  this.labels = d3.select("svg").selectAll("text")
    .data(this.dataset)
    .enter()
    .append("text");
}
```



D3 Update Pattern



D3 Update Pattern Details

- D3 Data Operator provides 3 virtual selections
  - Three section include Update, Enter and Exit

```
// update => d3.selection.data(...);
// enter  => d3.selection.data(...).enter();
// exit   => d3.selection.data(...).exit();
```

  - Update selection contains all of the existing DOM elements that had their data attributes updated
  - Enter selection contains placeholder elements for data not yet bound to DOM elements
  - Exit selection contains all of the existing DOM elements which did not have their data attributes updated

Generating Labels in update

```
var yTextOffset = (d3.min(this.dataset) * yScaleFactor) * 0.2;
var textSize = (barWidth * 0.3) + "px";

this.labels.text((d, i) => { return "$" + d; })
  .attr("x", (d, i) => { return plot.xOffset + (i * (xScaleFactor)) + (barWidth / 2); })
  .attr("y", (d, i) => { return plot.yOffset + plot.height - yTextOffset; })
  .attr("fill", "white")
  .attr("font-size", textSize)
  .attr("text-anchor", "middle")
  .attr("alignment-baseline", "middle");
```

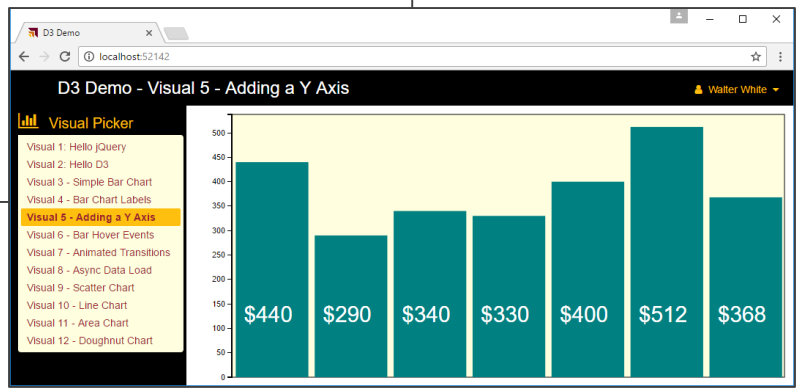
Agenda

- ✓ TypeScript Language Primer
- ✓ Getting Started with D3 and SVG Graphics
- ✓ Creating Data-driven Visuals
- ➢ Enhancing Visuals with Scales and Axes
- Event Handling and Transitions
- Using D3 Layouts

Adding a Scale and an Axis

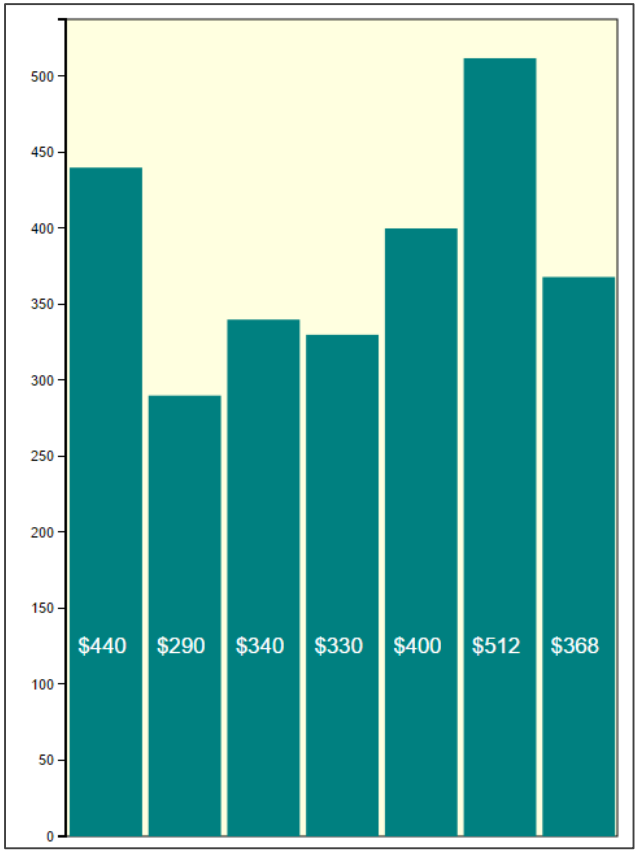
- **Scale** is a function that maps input domain to output range
- **Axis** is a function used to generate the HTML elements of visual axis

```
export class Viz05 implements ICustomVisual {

  name = "Visual 5 - Adding a Y Axis";

  private dataset = [440, 290, 340, 330, 400, 512, 368];

  private svgRoot: d3.Selection<SVGElementInstance>;
  private plotArea: d3.Selection<SVGElementInstance>;
  private axisGroup: d3.Selection<SVGElementInstance>;
  private bars: d3.Selection<number>;
  private labels: d3.Selection<number>;
  private padding: number = 12;
  private xAxisOffset: number = 50;
  private yScale: d3.scale.Linear<number, number>;
  private yAxis: d3.svg.Axis;

  load(container: HTMLElement)...

  update(viewport: IViewPort)...

}
```



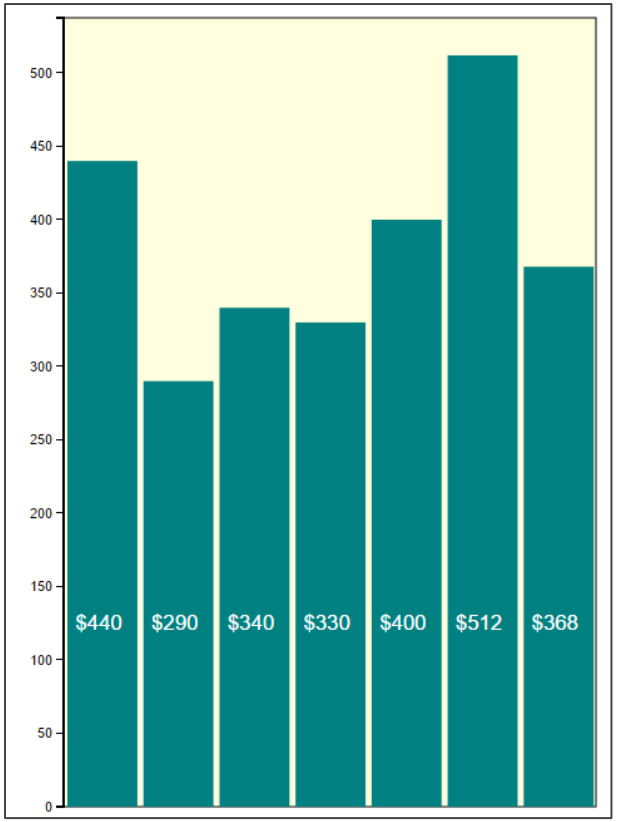
Implementing load

```
load(container: HTMLElement) {

  this.svgRoot = d3.select(container).append("svg");

  this.plotArea = this.svgRoot.append("rect")
    .attr("fill", "lightyellow")
    .attr("stroke", "black")
    .attr("stroke-width", 1);

  this.bars = this.svgRoot.append("g")
    .selectAll("rect")
    .data(this.dataset)
    .enter()
    .append("rect");

  this.labels = this.svgRoot
    .selectAll("text")
    .data(this.dataset)
    .enter()
    .append("text");

  this.axisGroup = this.svgRoot.append("g");
  this.yScale = d3.scale.linear();
  this.yAxis = d3.svg.axis();

}
```



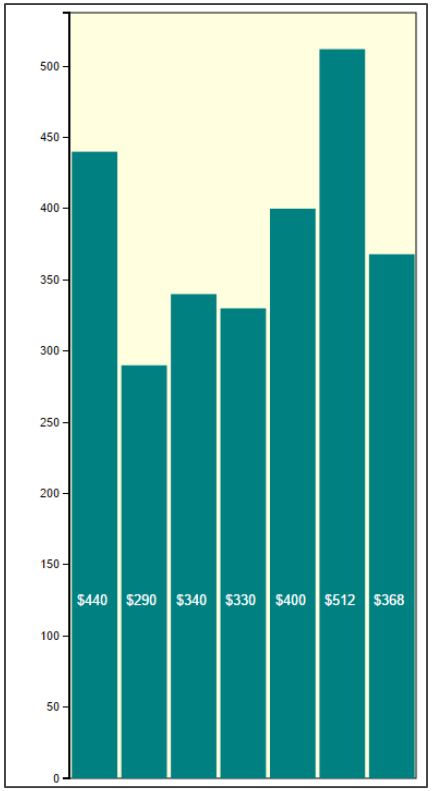
Implementing update (part 1)

```
update(viewport: IViewPort) {

  this.svgRoot
    .attr("width", viewport.width)
    .attr("height", viewport.height);

  var plot = {
    xOffset: this.padding + this.xAxisOffset,
    yOffset: this.padding,
    width: viewport.width - this.xAxisOffset - (this.padding * 2),
    height: viewport.height - (this.padding * 2),
  };

  var yDomainStart: number = d3.max(this.dataset) * 1.05;
  var yDomainStop: number = 0;
  var yRangeStart: number = 0;
  var yRangeStop: number = plot.height;

  this.yScale
    .domain([yDomainStart, yDomainStop])
    .range([yRangeStart, yRangeStop]);

  var datasetSize = this.dataset.length;
  var xScaleFactor = plot.width / datasetSize;
  var barXStart = (plot.width / datasetSize) * 0.05
  var barWidth = (plot.width / datasetSize) * 0.92;
  var yScaleFactor = plot.height / d3.max(this.dataset);

  // to be continued...
```



Implementing update (part 2)

```
// to be continued...

this.plotArea
  .attr("x", plot.xOffset)
  .attr("y", plot.yOffset)
  .attr("width", plot.width)
  .attr("height", plot.height);

this.bars
  .attr("x", (d, i) => { return plot.xOffset + barXStart + (i * (xScaleFactor)); })
  .attr("y", (d, i) => { return plot.yOffset + this.yScale(Number(d)); })
  .attr("width", (d, i) => { return barWidth; })
  .attr("height", (d, i) => { return (plot.height - this.yScale(Number(d))); })
  .attr("fill", "teal");

var yTextOffset = this.yScale(d3.min(this.dataset)) * 0.5;
var textSize = (barWidth * 0.3) + "px";

this.labels
  .text((d, i) => { return "$" + d; })
  .attr("x", (d, i) => { return plot.xOffset + (i * (xScaleFactor)) + (barWidth / 2); })
  .attr("y", (d, i) => { return plot.yOffset + plot.height - yTextOffset; })
  .attr("fill", "white")
  .attr("font-size", textSize)
  .attr("text-anchor", "middle")
  .attr("alignment-baseline", "middle");

this.yAxis.scale(this.yScale).orient('left').ticks(10);
var transform = "translate(" + (this.padding + this.xAxisOffset) + "," + this.padding + ")";
this.axisGroup.attr("class", "axis").call(this.yAxis).attr({ 'transform': transform });
}
```



Agenda

- ✓ TypeScript Language Primer
- ✓ Getting Started with D3 and SVG Graphics
- ✓ Creating Data-driven Visuals
- ✓ Enhancing Visuals with Scales and Axes
- ➢ Event Handling and Transitions
- Using D3 Layouts

Adding Mouse Event Handlers

```
this.bars
  .attr("x", (d, i) => { return plot.xOffset + barXStart + (i * (xScaleFactor)); })
  .attr("y", (d, i) => { return plot.yOffset + this.yScale(Number(d)); })
  .attr("width", (d, i) => { return barWidth; })
  .attr("height", (d, i) => { return (plot.height - this.yScale(Number(d))); })
  .attr("fill", "teal")
  .on("mouseover", function () { d3.select(this).attr("fill", "black") })
  .on("mouseout", function () { d3.select(this).attr("fill", "teal") });
```

Programming D3 Animation

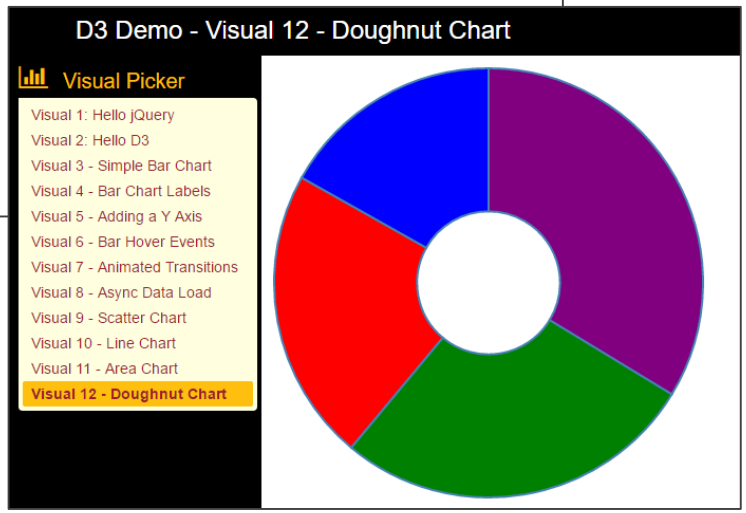
```
.on("mouseover", function () { d3.select(this).attr("fill", "black") })
.on("mouseout", function () { d3.select(this).attr("fill", "teal") })
.on("click", function (d, i) {
  // get reference to current bar
  var currentBar = d3.select(this);
  // determine current bar Y position and height
  var currentY: number = parseInt(currentBar.attr("y"));
  var currentHeight: number = parseInt(currentBar.attr("height"));
  // transition bar to height of zero
  currentBar.transition().duration(1000)
    .attr("y", currentY + (currentHeight))
    .attr("height", 0)
    .each("end", () => {
      // transition bar back to previoys height
      currentBar.transition().duration(500).delay(100)
        .attr("y", currentY)
        .attr("height", currentHeight);
    });
});
```

Agenda

- ✓ TypeScript Language Primer
- ✓ Getting Started with D3 and SVG Graphics
- ✓ Creating Data-driven Visuals
- ✓ Enhancing Visuals with Scales and Axes
- ✓ Event Handling and Transitions
- ➢ Using D3 Layouts

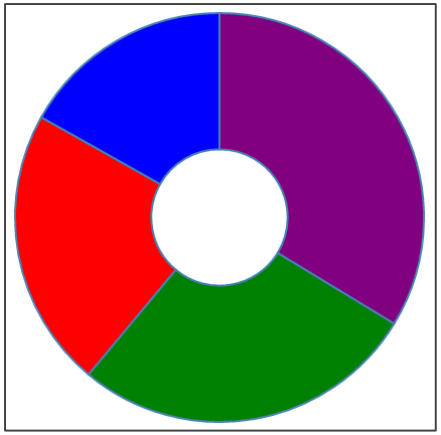
Using a D3 Layout

```
export class Viz12 implements ICustomVisual {

  name = "Visual 12 - Doughnut Chart";

  dataset = [21, 26, 16, 32];

  private svgRoot: d3.Selection<SVGElementInstance>;
  private plotArea: d3.Selection<SVGElementInstance>;

  private arc: d3.svg.Arc<d3.layout.pie.Arc<number>>;
  private pie: d3.layout.Pie<number>;
  private pieDataset: d3.layout.pie.Arc<number>[];
  private arcSelection: d3.Selection<d3.layout.pie.Arc<number>>;
  private arcSelectionPath: d3.Selection<d3.layout.pie.Arc<number>>;

  private padding: number = 12;

  load(container: HTMLElement)...

  update(viewport: IViewPort)...
}
```



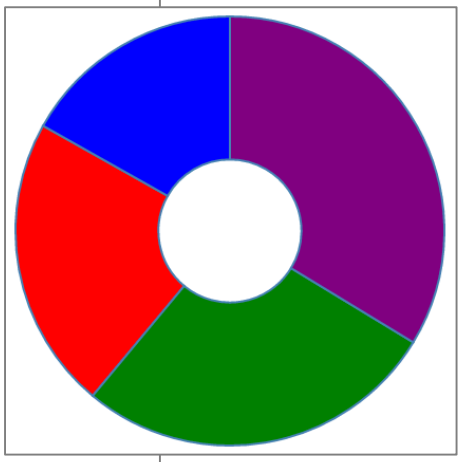
Implementing load

```
load(container: HTMLElement) {

  this.svgRoot = d3.select(container).append("svg");
  this.arc = d3.svg.arc<d3.layout.pie.Arc<number>>();
  this.pie = d3.layout.pie();

  this.arcSelection = this.svgRoot.selectAll("arc")
    .data(this.pie(this.dataset))
    .enter()
    .append("g")
    .attr("class", "arc");

  this.arcSelectionPath = this.arcSelection.append("path");

}
```



Implementing Update

```
update(viewport: IViewPort) {

  this.svgRoot
    .attr("width", viewport.width)
    .attr("height", viewport.height);

  var plot = {
    xOffset: this.padding,
    yOffset: this.padding,
    width: viewport.width - (this.padding * 2),
    height: viewport.height - (this.padding * 2),
  };

  var outerRadius = d3.min([plot.width / 2, plot.height / 2]);
  var innerRadius = outerRadius / 3;

  this.arc
    .innerRadius(innerRadius)
    .outerRadius(outerRadius);

  this.arcSelection
    .attr("transform", "translate(" + (outerRadius + this.padding) + ", " +
                                      (outerRadius + this.padding) + ")");

  var color = ["red", "green", "blue", "purple", "Yellow"];

  this.arcSelectionPath
    .attr("fill", (d, i) => { return color[i]; })
    .attr("d", this.arc);

}
```



Summary

- ✓ TypeScript Language Primer
- ✓ Getting Started with D3 and SVG Graphics
- ✓ Creating Data-driven Visuals
- ✓ Enhancing Visuals with Scales and Axes
- ✓ Event Handling and Transitions
- ✓ Using D3 Layouts