Javascript and D3
Javascript is ...

• NOT Java

• THE MOST popular language on GitHub... by far
Javascript is...

- a way to add interactive content to web pages
- a scripting language with an interpreter (does not compile to an intermediate language!)
- the language of the browser
Disclaimer

• I’m going to assume a basic working knowledge of
  – HTML
  – CSS
• If you need a refresher...

Web Architecture

Client (browser)

Server

Server Side Code (php, servlets, etc.)

database

url

server requests

query

data

html + JS

html + JS
• JS enables your browser, the client, to interact with the user without making a “round-trip” to the server

• JS gives you, the developer, access to the HTML Document Object Model (DOM) which structures the page
  – Catch and respond to user interface events
  – Insert elements into the page
  – Change or remove elements
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8>
<title>JS HelloWorld!</title>
<script src="helloWorld.js"></script>
</head>
<body>
   Body of the HTML Document Goes here!
</body>
</html>
Semicolons

• JavaScript uses a C-like syntax, which requires the use of semicolons to delimit statements.

• JavaScript attempts to do auto-insertion and make them optional (argh!!!)
  — Put them in yourself after each statement!

• In JavaScript, a linefeed can be whitespace, or it can act as a semicolon. This replaces one ambiguity with another.

```javascript
// Incorrect: return {
//   status: true
//};

// Correct:
return {
  status: true
};
```
Types

- **Single Number type**
- **Strings**
  - No Characters
- **booleans, null, undefined**

- Numbers and Strings have methods

```javascript
alert("2.5.toPrecision(10) = " + 2.5.toPrecision(10));
alert("'myString'.toUpperCase() = " +
    "myString".toUpperCase());
```
Equality

= (assignment)

== (equality)  [will coerce one argument into other]

=== (strict equality)  [same type, same value]
Variables and Scope

• When defined inside a function a `var` is private

• if you forget the `var`, your variable is global, *no matter where you define it*

• blocks `{}` do not create new scope
  – define all variables at the top of functions

• More on Scope and Variables to come later!
Control

• if, then, else
• switch, case
• loops
  – while
  – do while
  – for
• try, return, break
Objects and Arrays

• Object Literals
  – Convenience for creating new objects
  – Defined as associative arrays (maps, dictionaries)
  – key, value pairs
  – ALWAYS passed by reference, never copied

```javascript
var olit = { "name": "ron", "age": 25};
alert("olit name = " + olit["name"]); //or olit.name
```
Array Literals

- Convenience for specifying new arrays
- Arrays are objects
- Can contain a mixture of types
- Can delete elements however this leaves gaps
- Use `splice` to delete without gaps

```javascript
var alit = [1, 2, 3, 10];
delte alit[2];
alit.splice(2,1);
// alit is now [1, 2, 10]
```
Functions...

• Are objects
  – therefore, a collection of name-value pairs
  – linked to prototype: Function.prototype

• Contain two additional properties
  – context (Important Later!)
  – code to implement the behavior
`var add = function fname (a, b) {
  return a + b;
};`
Infovis Reference Model

--Card, Mackinlay and Schneiderman 1999
Enter D3...

- Javascript Library
- Facilitates
  - Binding data to the DOM
  - Creating visual Marks in the DOM through SVG
  - Setting attributes of those marks as functions of the bound data
  - Updating data values and therefore attributes
Basic Approach

• Bind Data to the DOM
• Tell the DOM what to do when data values change
• Tell the DOM what to do when new values arrive
• Tell the DOM what to do when values go away
• style the DOM elements
• specify interaction callbacks
• `.Select` or `.SelectAll` (domElement)
• Returns a `selection`
  – An array of elements
  – Augmented with a bunch of useful methods
Selections

- subclass of JS Arrays
- an array of groups and each group is an array of DOM elements

```javascript
var selection = d3.select("body");
```
d3.selectAll("h2");
`d3.selectAll("tr").selectAll("td");`

**selectAll**
every element in the old selection becomes a group in the new selection
Select vs. SelectAll

- **select** preserves existing groupings
- **selectAll** flattens any hierarchy

For a thorough treatment:
http://bost.ocks.org/mike/nest/
Data

• Is a property of individual elements, NOT of selections__data__
• Is stored in the DOM, not in the selection
• Is persistent...unlike selections (transient)
• three ways to bind data:
  • Joined to groups of elements: selection.data
  • Assigned to indiv element: selection.ddatum
  • Inherited from parents: append, insert, select
```javascript
document.body.__data__ = 42;

d3.select("body").datum(42);

d3.select("body").datum(42).append("h1");
```
• Data in D3 is any array of values
  – array of numbers
  – array of objects
  – array of arrays
  – etc.

• Useful to think of data as a mirror of selections
The **Key** to Joins

To join data to elements, we must pair the data keys to the selections keys

```javascript
var numbers = [4, 5, 18, 23, 42];

d3.selectAll("div").data(numbers);
```
Indexing with Functions

```
var letters = [
    {name: "A", frequency: .08167},
    {name: "B", frequency: .01492},
    {name: "C", frequency: .02780},
    {name: "D", frequency: .04253},
    {name: "E", frequency: .12702}
];

function name(d) {
    return d.name;
}
```
d3.selectAll("div").data(letters, name);
Data Joins

When joining Elements to Data by key, there are three possible outcomes

1. The selection has less elements than the data [enter]
2. The selection size is equal to the data size [update]
3. The selection size has more elements than the data [exit]
Data Joins

1) selection.enter()
2) selection.data()
3) selection.exit()
Suppose we have a bar chart with a bar (e.g. div) for each of the first 5 letters of the alphabet (ABCD) but want to transition to a bar chart of your favorite vowels (YEAOI)

```javascript
var div = d3.selectAll("div").data(vowels, name);
```

This selection (update) accounts for only two of the elements already there!
Exit & Enter

```javascript
div.exit();
```

```javascript
div.enter();
```

Enter() is a special selection of ‘placeholders’
Enough Already....how do we draw stuff
Setting Attributes of Primitives

• Use selections to set attributes for DOM elements
• Typically as functions of data

```javascript
selection.style("height", function(d) {
  var barHeight = d * 5;
  return barHeight + "px";
});
```
Drawing Bars with Divs

• divDemo.html
SVG Basics

Divs are limiting as ‘marks’

Typically, we use SVG elements

SVG Element: width, height in pixels

0,0

width

height

100, 35

300, 125
SVG Primitives = Marks!

- rect
- circle
- ellipse
- line
- text
- ...and more

All are added to the DOM as elements and can be ‘selected’ and bound to data!
var dataset = [ 5, 10, 15, 20, 25 ];

var circles = svg.selectAll("circle")
  .data(dataset)
  .enter()
  .append("circle")
  .attr("cx", function(d, i) {
    return (i * 50) + 25;
  })
  .attr("cy", h/2)
  .attr("r", function(d) {
    return d;
  });

What do you expect to see?
Let’s Build a Bar Chart
function svgBars () {

    //define some data
    var w = 500;
    var h = 100;

    var myData = [ 10, 20, 5, 32, 16];

    //create an append an svg element to put your vis in
    //create the update selection for ‘rects’ and bind data
    //create enter set and append new ‘rects’
    // Set the style/attribute values for your rects (assume scale of 4 for height)
    // add the mouseover event and handler
    // add text labels
//Create a selection
var text = svg.selectAll("text")
    .data(data);

//Update
text.attr("class", "update");

//Enter
text.enter().append("text")
    .attr("class", "enter")
    .attr("x", function(d, i) { return i * 32; })
    .attr("dy", ".35em");

//Update + Enter
text.text(function(d) { return d; });

// EXIT
text.exit().remove();

1 to 1 mapping of data to elements using index of data value to match index of element.
Data Joins: Key on Index

//Create a selection
var text = svg.selectAll("text")
  .data(data);

//Update
text.attr("class", "update");

//Enter
text.enter().append("text")
  .attr("class", "enter")
  .attr("x", function(d, i) { return i * 32; })
  .attr("dy", ".35em");
//Update + Enter
text.text(function(d) { return d; });

// EXIT
text.exit().remove();

Since new data values essentially overwrite the values of existing update elements, we need to set values for both update and enter selections.

The positions of all update elements are already correct since they are determined only by the number of elements in the selection. The position of new elements in enter(), however, must be set to append them to the end of the list.

See the Tutorial on General Update Pattern for more details.
http://bl.ocks.org/mbostock/3808218
<table>
<thead>
<tr>
<th>Exists</th>
<th>Data Arrives</th>
<th>enter</th>
<th>enter + update</th>
<th>exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c</td>
<td>a b c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a b c</td>
<td>b h i j k</td>
<td>j k</td>
<td>abc[bhi]</td>
<td></td>
</tr>
<tr>
<td>b h i j k</td>
<td>a b</td>
<td></td>
<td>b h [a b]</td>
<td>i j k</td>
</tr>
<tr>
<td>a b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Joins: Key with Value Function

Use the character value as key to index into selection. So, first datum no longer maps to first element in selection. If it’s a ‘c’, for example, it maps to the selection element at index ‘c’.

```javascript
var text = svg.selectAll("text")
  .data(data, function(d) { return d; });

text.attr("class", "update");

text.enter().append("text")
  .attr("class", "enter")
  .attr("dy", ".35em")
  .text(function(d) { return d; });

text.attr("x", function(d, i) { return i * 32; })

text.exit().remove();
```

Before, the position was set on enter() only. Now, since the position selection depends on how many alphabet values are represented in the selection (update and enter), we need to recalculate the position for all update and enter elements.

only set text upon enter. Values for particular indices never change. ie. the selection for ‘b’ is always at index 1 and always has value ‘b’ so just set it once upon enter.
### Example: Key with Value Function

<table>
<thead>
<tr>
<th>Exists</th>
<th>Data Arrives</th>
<th>enter</th>
<th>enter + update</th>
<th>exit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a b c</td>
<td>a b c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a b c</td>
<td>b h i j k</td>
<td>hijk</td>
<td>b [hijk]</td>
<td>ac</td>
</tr>
<tr>
<td>b h i j k</td>
<td>a b</td>
<td>a</td>
<td>b [a]</td>
<td>hijk</td>
</tr>
<tr>
<td>ab</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:
- Key
- Value
- Func1on
- Exists
- Data Arrives
- enter
- enter + update
- exit
• Functions that map from an input domain to an output range
• Specify dataspace values instead of pixel space (i.e. converts data vals to pixel vals)

```javascript
.attr("y", function(d) {
    return h - (d * 4) + 14;
});
```
var dataset = [ 5, 10, 13, 19, 21, 25, 22, 18, 15, 13, 11, 12, 15, 20, 18, 17, 16, 18, 23, 25 ];

• Want to create bars for this data on an svg canvas that is 50px tall
  • Domain: 0..25
  • Range: 0..50

var yscale = d3.scale.linear().domain([0,25]).range([0,h]);
Scales

```javascript
var yscale = d3.scale.linear()
    .domain([0, d3.max(dataset, function(d { return d; }))])
    .range([h, 0]);
```

• What if data was an array of points... how would you set the *range*?

```javascript
dataset = [ [23, 12], [14, 35], [2, 55] ];
d3.max(dataset, function(d) {return d[1]; })
```
Scale Variations

- Quantitative: linear, log, pow, etc..
- Ordinal

```javascript
var data = [ "A", "B", "C" ];

var x = d3.scale.ordinal().rangeRoundBands([0, width], .1);
x.domain(data);

///Use it to create an axis
var xAxis = d3.svg.axis().scale(x).orient("bottom");

svg.append("g")
  .attr("class", "x axis")
  .call(xAxis);
```
Axes

• A visual representation of a scale
  
  ```javascript
  var yAxis = d3.svg.axis()
      .scale(yScale)
      .orient("left");
  ```

• Now, attach it to the chart
  
  ```javascript
  svg.append("g").call(yAxis)
  ```
Interaction

- selection.on(event type, listener)
  - event type
    - "mouseover"
    - "mouseout"
    - "click", etc.
  - listener
    - user defined function
    - has access to d, i
function chart(width, height) {
    // generate chart here,
    // using `width` and `height`
}

• User must store and manage width, height, etc. separate from the chart!
• Will have to pass them in whenever an update is needed
function chart(config) {
    // generate chart here,
    // using `config.width` and `config.height`
}

• Better...but must still manage config object separate from the chart
• Really want to ‘bind’ the config object to the chart so don’t have to manage it!
A closure is a function that has access to the context (or outer scope) in which it was created.

The pattern is as follows:

- Create the context you want to attach to the function.
- Call a ‘constructor’ that returns the function.
- You now have a handle to the function and it’s context!
See closureDemo.html
Transitions

• Like selections, but operators animate smoothly over time

```javascript
  text.attr("class", "update")
    .transition()
    .duration(750)
    .attr("x", function(d, i) { return i * 32; });
```
• Use judiciously....they are attention getters
• Don’t accidentally create a masking effect
• Don’t animate over many states (data values...) they are hard to follow and compare. (unless Hans Rosling is talking over your vis!)
Transitions – entering elements

- start above in green, slightly transparent
- move down into place

```javascript
  text.enter().append("text")
    .attr("class", "enter")
    .attr("dy", ".35em")
    .attr("y", -60)
    .attr("x", function(d, i) { return i * 32; })
    .style("fill-opacity", 1e-6)
    .text(function(d) { return d; })
  .transition()
    .duration(750)
    .attr("y", 0)
    .style("fill-opacity", 1);
```
Here, we transform the group by half the height, essentially making the halfway mark the zero y value.

```javascript
var svg = d3.select("body").append("svg")
               .attr("width", width)
               .attr("height", height)
               .append("g")
               .attr("transform", "translate(32," + (height / 2) + ")");
```
Transitions – Exiting Elements

- start at zero y in black
- turn red, move down, fade out

```javascript
text.exit()
  .attr("class", "exit")
  .transition()
  .duration(750)
  .attr("y", 60)
  .style("fill-opacity", 1e-6)
  .remove();
```
Transitions – Update Elements

- keep current y value
- update x value (slide over!)

```javascript
  text.attr("class", "update")
    .transition()
    .duration(750)
    .attr("x", function(d, i) { return i * 32; });
```
Useful Tutorials/Reading

- Thinking with Joins: http://bost.ocks.org/mike/join/
- General Update Pattern I: http://bl.ocks.org/mbostock/3808218
- General Update Pattern II (key functions): http://bl.ocks.org/mbostock/3808221
- General Update Pattern III (Transitions): http://bl.ocks.org/mbostock/3808234
- Towards Reusable Charts http://bost.ocks.org/mike/chart/
References

• All captured figures/images for selections, joins, etc.

http://bost.ocks.org/mike/