**Git**

Used to back up and share code, and to document the development process.

***Getting files from Git***

* Fork files to your repository (git website), record URL
* Clone to desktop

git clone URL

* Use and update files
* Send files back to Git
  + Stage for committing (creates a list of files needing to be updated)

git add --all *adds all files in folder to the list*

git status *shows you what has been changed*

* + Commit to save snapshot of files

git commit –m “message” *saves a snapshot to local computer, stores with the message entered*

* + Push to the cloud

git push origin master *sends commit to the repository in the cloud, prompts for git username and password*

* + Pull request to merge with original document (for collaborative projects)

**Python**

Set up local http server. From the command line, navigate to the appropriate folder and then type:

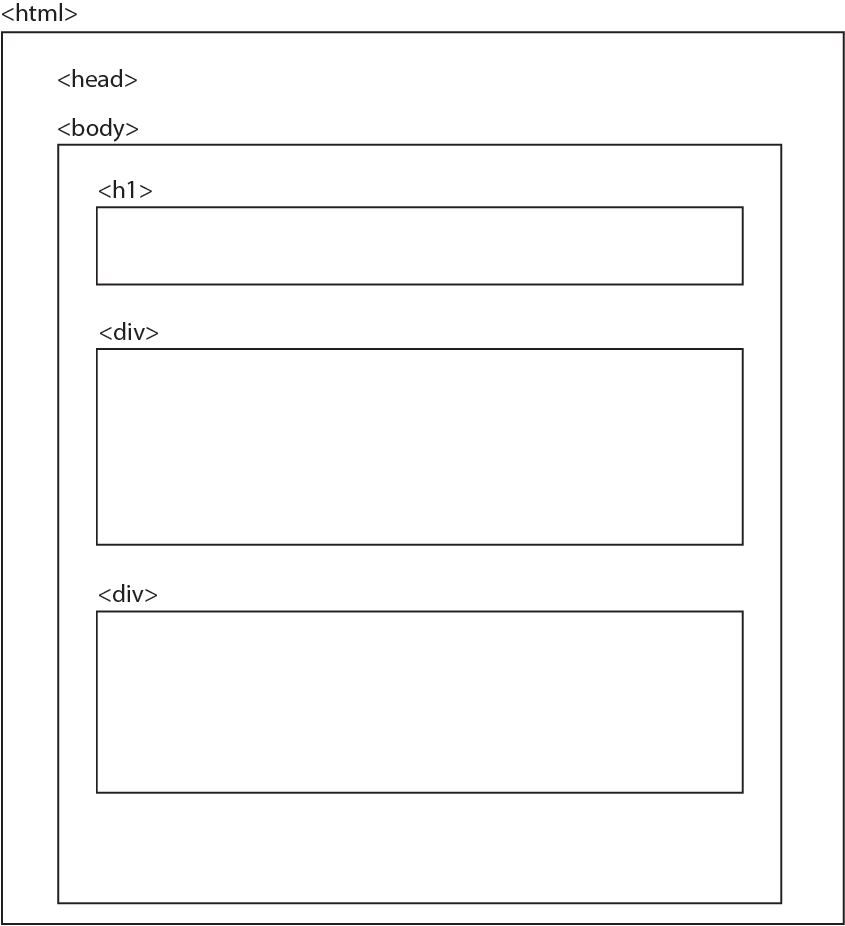
python –m http.server

Ctrl + C *to break communication at the end of the session*

Use address localhost:8000 in the browser to view locally served webpage. (May need to replace 8000 with the number printed in the command line after the server initiates.)

**HTML**

Use tags to build site structure. Elements that end up inside other elements will have their tags nested inside those of the outer element (the h1 and div tags will show up inside of the body tag, and the body and head tags will show up inside of the html tag)



***General tag format:***

<tag> things that describe tag </tag>

<tag/> *use a self-terminating version when the tag does not need descriptors*

To begin, declare a document of type html

<! doctype HTML>

You can add comments to explain what the code does

<!—comment --> *nothing inside the comment will be run as code*

***Common tags:***

<html> all of the tags that define the page structure go in here </html>

<head> links to files and settings needed to run the page go in here </head>

Things that you might include in <head>:

<meta charset = “utf-8”> *sets the kind of characters used on the page*

<title> title goes here </title> *creates the title text that shows up at the top of the tab or browser window*

<link href=’URL’ rel=”relation to your webpage”> *links to a file stored online – href means hypertext reference. Rel defines the relationship of the reference to your webpage - often “stylesheet.” For font definitions, also include type=”text/css”*

<link href=”style.css” rel=”stylesheet”/> *loads css stylesheet called style.css, which is stored in the parent directory for the webpage, and sets it as the stylesheet for the webpage*

Links we used this semester:

<link href='<https://fonts.googleapis.com/css?family=Droid+Serif:400,400italic>' rel='stylesheet' type='text/css'>

<link href='../bootstrap.min.css' rel="stylesheet"/>

<link href="style.css" rel="stylesheet"/>

<link href='<https://fonts.googleapis.com/css?family=Droid+Serif:400,400italic>' rel='stylesheet' type='text/css'>

 <link href="../bower\_components/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet"/>

<body> contains all of the page elements *title, text blocks, lists, etc* </body>

Things that you might include in <body>:

Header – contains text, pictures, or information about the page that is separate from the main content. Elements are nested inside header using tags that depend on the kind of information and the style/formatting.

<header>

<h1> text that shows up at the top of the page </h1> *sets the text style*

</header>

Divs – separate content of the page into sections. Divs are also often nested. Each div can also be given a class, which allows you to access them separately. Class names cannot have spaces. Each item can have more than one class, separated by a space (see lists, below).

<div class=”container”> </div>

<div class=”lists”> </div>

Headings – text, displayed with different styles defined for each heading level. Can have multiple levels, and styles are set in the CSS document.

<h1>text goes here</h1>

Lists – create lists with special formatting (different linespacing, bullets, etc.). Can also assign classes to lists, and individual list items can be given an id, which allows you to call an individual item rather than a whole class of items.

<ul class=”optional-class-definition secondary-class”> *creates an “unordered list” with no number ordering*

<li class=”class” id=”id”> list item text goes here </li>

Spans allow you to change the formatting of a specific word or section of text without changing anything around it. If you wanted to turn one word blue, you could use a span, and set the style for that span class to blue in the CSS document. Using the example above,

<li class=”class” id=”id”> list item with one word in <span class=”blue”>blue</span> </li> *note that the word blue inside the span is the one that will be highlighted, using the class “blue” to set the style in the CSS document*

Embedding scripts in your webpage allows you to run Javascript functions in your website. This is necessary for creating any interactive elements. The code for those elements will be written in a Javascript file called script.js, which is loaded into the HTML document using a script tag.

<script src=”filename”></script> *filename here can be either a file stored in the parent directory, or a webpage*

Scripts we used this semester:

<script src="script/script.js"></script>

<script src="<http://d3js.org/d3.v3.min.js>" charset="utf-8"></script>

<script src="script/lib/queue/queue.min.js"></script>

<script src="script/lib/jquery-1.11.3.min.js"></script>

<script src="<https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.6/d3.min.js>" charset="utf-8"></script>

Hyperlinks are used to create a clickable link to another document or webpage (this is what most people think of as a “link” in the internet).

<a href=”destination URL”>link text that the reader sees</a>

<a href="filename.html">text the reader sees</a> *is a link to a page on the same website.*

Paragraphs are used to set paragraph formatting

<p>text goes here</p>

Breaks insert spaces in the document

<br>

Insert an image from a file stored in the site parent directory

<img src=”image.png” alt=”not sure what this does” width = “200”, height = “200”>

An SVG element creates a drawing canvas for making shapes – more on this in the Javascript section. Pixels are the default unit for svg elements, but you can explicitly call em, pt, in, cm, mm. The origin is located at the upper left corner of the screen.

<svg width=”50” height =”50”>

SVG elements can be drawn and styled using HTML, but we will usually do this from within the Javascript and/or CSS files. Any SVG element can also be assigned a class, which can be used for CSS styling. There are no layers in SVG, so whatever element is called first, will be drawn first. Transparency can be achieved with RGB-A colors, or with the opacity attribute. The use of RGB-A can cause strange effects with color mixing, so it is often better to use opacity. You can also mix transparency and opacity to multiply their effects.

<circle cx=”25” cy=”25” r=”22” fill=”blue” stroke=”gray” stroke-width=”1px”/> *cx and cy are the coordinates of the circle center, by default*

<rect x=”0” y=”0” width=”500” height=”50” /> *x and y give the coordinates of the upper left corner of the rectangle*

<ellipse cx=”250” cy=”25” rx=”100” ry=”25” />

<line x1=”0” y1=”0” x2=”500” y2=”50” stroke=”black”/>

<text x=”250” y=”25” font-family=”serif” font-size=”25”>text to display goes here</text> *x is the left edge of the text box, y is the text baseline, by default. Text will inherit CSS styling, unless otherwise specified.*

A drawing canvas is an HTML element that can only be drawn to using a script.

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;"></canvas>

***HTML styling***

Most of the time, it is better to set the style for HMTL objects in the CSS file. However, it is also possible to set the style in the HTML code. In many cases, this simply involves setting the style attribute using the CSS code enclosed in quotation marks.

<body style=”background-color:lightgray”>

<h1 style=”color:purple”>

<p title=”test” style=”color:green”>

<p style=”font-family:Times”>

**CSS**

Uses HTML element tags, classes, and ids to set the style of objects that show up on the webpage. This includes setting all spacing between items, text style and size, and the background colors for the page.

Comments

/\* your comment here \*/

***General CSS format:***

In CSS, you make a selection of HTML elements using a selector, and then you assign values for different properties associated with that element. Property names are pre-defined, and values must be given in the correct unit and type (must be a string if the property expects a string, or a number if it expects a number)

selector {

property-name: property-value; (the colon and semicolon are important here!)

}

Selectors call HTML objects based on tag, class, and id set in the HTML file, and select all elements with that particular tag, class, or id.

by tag: tagname { put style definition here }

by class: .class { put style definition here }

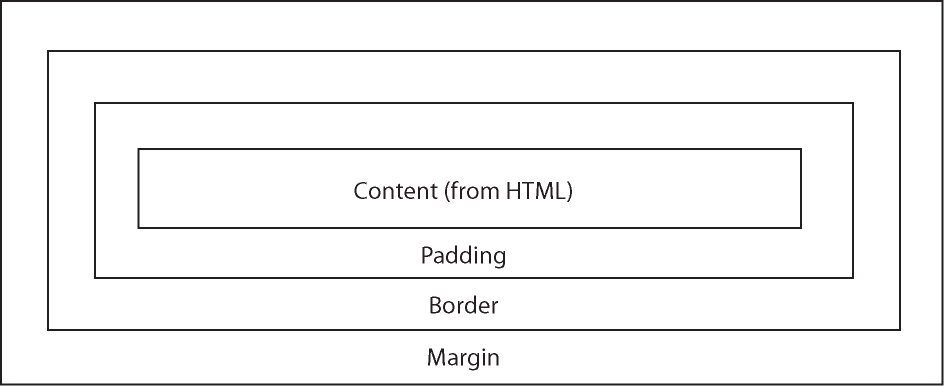
by id: #id { put style definition here }

by class nested inside of a particular tag: h1.class { put style definition here } grabs objects of class “class” inside the h1 tag.

.lists>.list restricts inheritance to one level down – this is useful when a style is defined as a percentage and might be applied several times. Since the percentage is based on the containing box size, an element contained in a nested div might be scaled up or down more than desired. This command says that you apply the formatting to any class higher in the DOM than the .list class, but not to anything below it, so that items further down the hierarchy will not be scaled repeatedly.

***CSS properties that you might want to change:***

Object size and spacing

All CSS objects are boxes that contain the content to be displayed, some amount of padding or white space around the content, a border that may or may not be displayed, and a margin that defines more whitespace between the border and other things.

The content is determined by the HTML document, but the other attributes are set in the CSS. Each attribute is expecting a certain kind of data, in a specific order. Do not insert spaces between a number and its unit!

width:200px; *sets the width of the HTML element*

height:200px;

height:100%; *sets the height of the HTML element to be 100% of the parent element*

padding: 20px;

border: 2px solid black; *number in pixels, line style, line color*

margin:20px; *assumes a uniform 20 pixel margin all around*

margin: 0; *assumes default unit of px*

It is also important to tell an HTML element how it should relate to other elements on the page. Can use either absolute pixel values or percentages (better for responsive design). Percentages will be calculated based on the size of the containing box, not the overall screen size.

position: absolute; *defines a fixed position relative to the rest of the screen – useful for tooltips. Width has to be set manually*

top:100px; *define the distances for absolute placement*

left:100px;

bottom:100px;

right:100px;

position: relative; *stacks elements from top to bottom in the screen, unless otherwise specified*

float: left; *tells the element that it should float toward the top of the screen, and stack from left to right. This does allow elements to hide behind non-floating elements*

clear:both; *tells floating elements to pay attention to non-floating elements*

margin-bottom:10px;

margin:top right;

margin:top right bottom left;

padding:top right bottom left;

***Setting color***

There are several ways or representing color. The simplest way is to use the html color names, a set of built-in colors that will be recognized by your browser (<http://www.w3schools.com/html/html_colornames.asp>). Alternatively, you can use the rgb(100,100,100), rgba(100,100,100,.2), hsl(0,100%,97%), or hsv(0,15%,25%) formats, or hexadecimal codes (#F0F8FF) to represent color values.

background:black;

fill:yellow;

fill:none;

stroke:black;

fill-opacity:.05;

***Line styling***

stroke-width:1px;

stroke-dasharray:3px 3px; *creates a dashed line with dashes of 3px and spaces of 3 px*

***Font***

font-size:12pt;

font-size:100%; *sets font size as a percent of the default size*

font-family: Helvetica, Arial, sans-serif; *sets preferred font family, an alternate class in case the preferred one is not available, and the style of type. If the font used is not a system font, use quotes for the name*

font-size:16px; *could also use 1.5em*

***Formatting lists***

list-style:none; *removes bullets from an unordered list*

***Axis styling***

.axis .domain{fill:none;}

.axis .tick line{

stroke:rgb(180,180,180);

Stroke-width:1px;

Stroke:black; }

.axis .tick text{

font-size: 10px;

fill:rgb(180,180,180);}

**Javascript/D3**

***Comments***

// or /\* … \*/

***Script output***

console.log(‘message’) *outputs a string to the developer console*

console.log(variable) *outputs the value of a variable to the console*

(The developer console in Chrome can be accessed using Ctrl+Shift+J, or More Tools, Developer Tools in the menu)

***Loading files and libraries***

d3.csv(‘/folder/filename.csv’, parsefunction, callbackfunction); *loads a .csv file from a folder in the parent directory, passes it to a parsefunction, and waits for a callback.*

queue()

.defer(d3.json, “datafolder/filename.json”) ); *Also works for geojson*

.d3.csv(‘/folder/filename.csv’, parsefunction)

.await(function(err, array1, array2){ do this when ready });

***Variables***

var varname = ; *declares a variable, gives it the name “varname,” sets it equal to   
whatever value is placed in the blank.*

= []; *empty array*

= [‘a’,’b’,’c’,’d’,’e’]; *array populated with string values a-e.*

= {}; *empty object*

= {a: 10, b:20, c:30, d:40, e:50}; *object with properties a-e, each containing a numeric*

*value*

= {t:20, r:20, b:20,l:20}; *populates an object with four named parameters that can be*

*accessed using dot notation (var.t = 20, etc.)*

= 12; *numerical value*

= function(); *runs the function when the variable name is called*

= function(var1,var2*); runs the function and passes it values for var1 and var2 (stored in*

*the names listed in the function definition)*

= d3.select(handles); *stores a selection of DOM elements using the d3 selection tool* (selectAll also works here). The “handles” parameter is a class, id, or other HTML identifier: ‘.canvas’ selects all items of class canvas. The variable name can then be used to access the selection multiple times, or to pass it from one function to another. The d3 selection handles and their respective HTML elements selected are shown below:

#var = <any id = “var”>

var = <var>

.var <class = “var”>

[var = bar] = <any var = “bar”>

var bar = <var> <bar> </var>

var.bar = <var class = “bar”>

var#bar = <var id = “bar”>

“var, bar” = all <var>, <bar>

= d3.select(this).attr(‘id’);

Usually used with on-click functions. Takes the result of a selection (‘this’ is whatever was previously selected), and re-sets an attribute

= document.getElementbyId(‘plot’).clientWidth-margin.l-margin.r

This example is similar to using d3 select, but (I believe) this is a native Javascript command. It

gets the ‘plot’ element in the document using its HTML id, and returns the value of clientWidth

minus the values stored in the .r and .l parameters of a margin object (margin.r and margin.l,

respectively). Similar syntax can be used for clientHeight and margin.t and margin.b.

Can declare multiple variables at once: var var1, var2, var3, var4;

typeof var/value *identifies variable type (+var forces var to adopt a numerical type)*

***Arrays and objects***

arrayname.push(item); *append an item onto the end of an array*

arrayname.shift(); *removes front item from an array*

arrayname.unshift(); *adds a new item to the front of an array*

arrayname.slice(beginindex, endindex) *returns a subset of an array*

arrayname.forEach(function(element,index) { do something });

arrayname.sort(function(a,b){ return b-a}) *sort an array in descending order*

arrayname.indexOf(value) *finds the first instance of a value in an array, returns its index*

.lastIndexOf(value)

.concat(newarray) *concatenates 2 arrays*

arrayname.length *returns the length of an array*

***Functions***

var functionname = function(inputvariables) { do this };

***Dot notation***

symbols[index].type *accesses the type property for the array element stored in the symbols*

*variable that matches the index given*

d[‘GDP per capita, PPP (constant 2011 international $)’] *accesses the attribute with the name GDP per capita, PPP (constant 2011 international $) inside the variable d.*

***DOM object manipulation***

.append(‘svg’) *other options: ‘g’ , ‘circle’, ‘line’, etc.*

.attr(‘attributename’, valuetoassign) *‘class’, shape definition parameters are common*

*attributes*

.attr(‘transform’, ‘translate(number, number)’)

.attr(‘stroke’, ‘blue’)

.attr(‘stroke-weight’, 3)

.style(

.call(axisX) *calls the axisX generator function to draw an axis inside the selected DOM element (usually plot, svg, canvas, or g)*

***Numbers***

Scientific notation 1e8 = 1x108

Basic math functions (+, -, /, =) work as expected, - is also used to write negative numbers

% returns the remainder of a division

***Strings***

Can use either single or double quotes for a string (but they have to match)

ln inserts a linebreak (“enter”) in a string

\t=tab \ to insert a tab space

+ concatenates strings: ‘this’ + ‘that’ gives ‘thisthat’ - all spaces have to be added manually.

***Math functions***

Math.random() *returns a random value between 0 and 1*

Math.floor(value) *returns the integer below a value*

Math.ceil(value)

Math.round(value) *rounds to the nearest integer*

Math.max *returns max value*

Math.sqrt *returns square root*

Math.PI *returns pi*

Math.cos(), .sin(), .tan(), .acos(), .asin(), .atan() *defaults to radians(?)*

d3.min(array) *returns the minimum value in an array*

d3.max(array, function (e) { return e.value}); *returns the maximum value of a particular attribute in an array of objects that have a parameter called .value (look at the .value parameters for all objects in the array, return the maximum one). Could also return the index of the object that contains the maximum value, using function(e,i), where i is the index for the current element in the array.*

***Logical operators***

== *is equal to*

!= *is not equal to*

< *is less than*

> *is greater than*

<= , >= *less than or equal to, greater than or equal to*

&& AND

|| OR

! NOT

Order of operations: || has lowest precedence, then &&, then <>, etc.

condition ? expr1: expr2 *Conditional operator. If the condition is true, then the function*

*returns expr1. Otherwise, it returns expr2.*

To override automatic type conversions, it may be necessary to use === and !==

***Loops***

for (i=0; i<number; i++){

do this thing that depends on i *“return” exits the loop. “return value” returns a value to the calling function or workspace*

}

if (something == 0) {

do this

}

else if (condition) { do this }

***Binding data***

var storage = canvas.selectAll(‘.class1’) *select everything of class “class1” in the canvas DOM element, and save the selection in the storage variable*

.data(dataarray) *bind the selection to the data stored in the dataarray*

.enter() *create an empty DOM element for each element in the dataarray*

.append… *add the desired kind of DOM element to each placeholder*

.exit() *select DOM elements that no longer have an array element*

.transition(time) *over time period specified (in ms, after the function call)*

.delay(time) *after a time specified in ms (from the function call)*

.remove() *remove the selected elements from the DOM*

.update() *update elements whose values have changed.*

.insert(‘line’,’.node’) *used once, inserts lines before nodes in a force layout*

*(Wk11, Ex1)*

***Scale function***

var scaleX = d3.scale.log().domain([minGdpPerCap, maxGdpPerCap]).range([0,width])

Stores a scale function in the variable scaleX for future use. The function is a logarithmic distribution, and scales values within the limits of the domain array (here given as variables that store the max and min GDP values) to match the size of the range array, which usually corresponds to the length of an axis or other DOM element. This example will scale the values between min and max GDP to take up the entire width of the screen.

.linear().domain([value, value]).range([value, value]) *linear scale*

.sqrt().domain([value, value]).range([value, value]) *square root scale*

*(for scaling circle radii, areas)*

***Axis function***

d3.svg.axis() *creates an axis within the svg object.*

.orient(‘bottom’) *defines the orientation of the tick marks relative to the axis (other*

*options: top, left, right)*

.tickSize(-height) *sets the length of the tick marks*

.tickValues([1e+4, 1e+5, 1e+6]) *specifies where the tick marks are shown*

.scale(scaleX) *scales the axis using the scale saved in the variable scaleX.*

***On click***

Defines a function to be run when the user clicks on an item in a defined selection.

d3.selectAll(‘buttonclass1 class2’).on(‘click’, function() { do something})

Here, if the user clicks on anything with class buttonclass1 or class2, it will run the user-defined function defined in the d3 on click function.

***Projection function***

var projection = d3.geo.mercator() *Alternate projections: albersUSA*

.center(coordinatearray)

.translate([x,y])

.scale(value);

Call the function using a map path generator:

var pathGenerator = d3.geo.path().projection(projection);

pathGenerator.centroid(dataarray); c*alculates the geographical center of objects in*

*the geojson data array; returns an array of values*.

***Force layout***

var force = d3.layout.force()

.size([width, height])

.charge(value) *attractive(+)/repulsive(-) “force” (default -30)*

.linkDistance(value) *sets the equilibrium value for the link length, in pixels*

.gravity(value) *values from 0-1, centered in window unless otherwise defined*

.friction(value) *values from 0-1, higher value means nodes respond faster to*

*forces*

force.nodes(data) *sets the nodes for the force layout*

.links(data) *sets the links for the force layout (connect the nodes)*

.on(‘tick’, onForceTick) *sets up a counter, runs onForceTick one per set period*

*of time*

.start(); *starts the force layout*

force.alpha(0) freezes minimization, force.stop stops the computation, force.start, force.resume, force.alpha(1) resets minimization parameter manually, acts like force.start

function onForceTick(e) {

program desired behavior here *see assignment 6 and in-class week 11, Ex 1 for examples*

}

***Lookup table***

var lookup = d3.map();

lookup.set(data.prop1, data.prop2)

prop1 and prop2 are the names of the columns in the original data array that you want to link with a 1:1 relationship in the lookup table. (If this is set inside a function, then you can use d.prop1 instead.) Can also link an id to an object using {prop1:prop1data, prop2:prop2data}

var test = (lookupTable.get(d.prop1)).prop2 *returns the prop2 value of the object returned by looking up d.prop1 in the lookupTable*.

***Shapes***

.append(‘circle’)

.attr(‘class’, ‘circle’) *optional, but setting a class helps to identify objects later*

.attr(‘cx’, value) *sets center x position*

.attr(‘cy’, value)

.attr(‘r’, value) *sets circle radius*

.style(‘fill’, colorname) *optional style settings (can also be done in the css,*

*using class)*

.style(‘stroke’, colorname)

.append(‘rect’)

.attr(

.append(‘line’)

.attr(‘class’, ‘line’)

.attr(‘x1’, value) *sets beginning point x position*

.attr(‘y1’, value)

.attr(‘x2’, value) *sets endpoint x position*

//.attr(‘x2’, function (d){return scaleX(d.value);}) ) *uses an accessor function to set attribute value. The variable d is the data bound to the selection, and it must have a property called .value. The function returns the scaled value, and uses it to set the attribute for the line accordingly*.

.attr(‘y2’, value)

.style(‘fill’, colorname)

.style(‘stroke’, colorname)

.append(‘text’)

.text(‘string’)

.attr(‘class’, ‘label’)

.attr(‘text-anchor’, ‘middle’) *sets positioning coordinate to the center of the*

*textbox, rather than lower left edge)*

.attr(‘font-size’, ‘12px’)

.style(‘fill’,rgb(value, value, value));

.append(‘path’)  *This appends a map to the selected canvas*

.datum(dataarray) *Use data if you want each item in the map to be appended as a separate path, datum if you want a single map object made up of many parts.*

.attr(‘class’, ‘boundary’)

.attr(‘d’, pathGeneratorFunction)

.style(‘stroke-width’, ‘2px’);

***Accessing values stored in data structures***

Simple array:

var array = [‘a’,’b’,’c’,’d’,’e’]; *creates a simple array populated with*

*string values a-e.*

array[0] = a

array[4] = d

Simple object:

var object = {a: 10, b:20, c:30, d:40, e:50}; *object with properties a-e, each containing a numeric value*

object.a = 10

object.d = 40

object[0] = undefined *(can’t use array indices to access the contents of an object)*

Array of objects:

var arrayOfObjects = [

{a: 10, b:20, c:30, d:40, e:50},

{a: 60, b:70, c:80, d:90, e:100},

{a: 110, b:120, c:130, d:140, e:150}

];

*returns an array of 3 objects, each containing a set of properties with numerical values*

arrayOfObjects.a= undefined (have to specify which element of the array you want the

property “a” for. Can use the syntax below and write a “for” loop, use

a .forEach, or create a key function to access more directly)

arrayOfObjects[1].a = 60

arrayOfObjects[0] = {a: 10, b:20, c:30, d:40, e:50};

Object of arrays:

var objectOfArrays = {a: [10,20,30,40,50], b:[60,70,80,90,100], c:[110,120,130,140,150]};

*returns an object containing three arrays that contain 5 values, stored in properties a,b*

*and c.*

objectOfArrays.a = [10, 20, 30, 40, 50] *returns the whole array stored in .a*

objectOfArrays.c[3] = 140 *returns just the 4th element of the array stored in .c*

Mixed array containing both variables and objects:

var mixedArray = [value1, value2, {prop1:valueprop1}]; *creates an array that contains two values and one object, which has one attribute (prop1) that is assigned a value (valueprop1). The values do not have to be of the same type: can have numerical mixed with string values. The value inputs can be numerical (replace value1 with 10) or variable quantities (declare and define value1 elsewhere in the program).*

mixedArray[0] = value1

mixedArray[2] = {prop1:valueprop1} *returns the object stored in array position [2]*

mixedArray[2].prop1 = valueprop1 *returns the prop1 attribute of the object in [2]*