APPENDIX E
Quick Reference

Here is a list of the most commonly used D3 methods covered in this book, plus a brief summary of its use, and one example for each. (Methods that require a bit more explanation—such as line and area generators, geographic projections, layouts, and scale-specific methods—have been omitted.)

Selections

`d3.select()`

Returns a reference to the first element found:

```
// Selects an SVG element and stores a reference to it in 'svg'
var svg = d3.select("svg");
```

`d3.selectAll()`

Returns references to all found elements:

```
// Selects all circle elements and stores references to them in 'circles'
var circles = d3.selectAll("circle");
```

`selection``.append()`

Takes a selection, creates a new element inside of it, then returns a reference to the newly created element:

```
// Creates a new circle inside of the 'svg' selection established earlier
d3.select("svg").append("circle");

// This would accomplish the same thing…
svg.append("circle");

// …but it's often useful to store a reference to the new element
var newCircle = svg.append("circle");
```

selection.remove()
: Takes a selection, removes it from the DOM, and returns a reference to the deleted selection:

```
// Removes the first rect element
d3.select("rect").remove();
```

selection.text()
: Takes a selection, sets its text content, and returns a reference to the acted-upon selection:

```
// Sets the text content of #tooltip to "15%"
d3.select("#tooltip").text("15%");
```

selection.attr()
: Takes a selection, sets an attribute value, and returns a reference to the acted-upon selection:

```
// Assigns a radius value of 10 to all circle elements
d3.selectAll("circle").attr("r", 10);
```

selection.style()
: Takes a selection, sets an inline CSS style, and returns a reference to the acted-upon selection:

```
// Assigns a CSS fill of "teal" to all circle elements
d3.selectAll("circle").style("fill", "teal");
```

selection.classed()
: Takes a selection, adds or removes a class, and returns a reference to the acted-upon selection; `true` adds the specified class, `false` removes it:

```
// Adds a class of "highlight" to the first circle element
d3.select("circle").classed("highlight", true);

// Removes the class of "active" from all circle elements
d3.selectAll("circle").classed("active", false);
```

selection.each()
: Takes a selection, and runs an arbitrary function once for each element in the selection, with the `this` context set to the element being acted upon:

```
d3.selectAll("circle")
  .each(zoomAndEnhance);
  // Assumes a function named 'zoomAndEnhance' already defined
```

Data

selection.data()
: Takes a selection, calculates the difference between the number of elements and number of data values, and binds the array of data values to any existing elements (or not-yet-existing placeholder elements):

```
d3.selectAll("circle")
    .data(dataset)  // Binds data to all circles (or placeholders)
    .enter()
    .append("circle");
```

selection.datum()
: Takes a selection, and binds a single data value to a single element (or not-yet-existing placeholder element):

```
svg.append("path")
    .datum(dataset)
    .attr("d", line);
```

selection.enter()
: Takes a selection and returns a subselection of "new" placeholder elements:

```
d3.selectAll("circle")
  .data(dataset)
  .enter()  // Returns the placeholders for circles to-be-created
  .append("circle");  // Creates a circle for each placeholder
```

selection.merge()
: Takes a selection and merges it with another specified selection, returning a newly merged selection:

```
bars.enter()  // Get the enter subselection
    .append("rect")
    …  // Set attributes for new elements…
    .merge(bars)  // Merge enter subselection with existing bars selection
    …  // Set attributes for all elements…
```

selection.exit()
: Takes a selection and returns a subselection of "exiting" elements:

```
bars.exit()     // Get the exit subselection
    .transition()
    …  // Set attributes for exiting elements, e.g. dial down opacity…
```

selection.remove()
: Takes a selection and removes associated elements from the DOM:

```
bars.exit()     // Get the exit subselection
    .remove();  // Delete exiting elements immediately
```

`function(d) { … }`
: Use anonymous functions to access data values bound to elements via `d`:

```
d3.selectAll("rect")
  .attr("height", function(d) {
      return d.value;  // Set each rect's height to 'value'
  });
```

`function(d, i) { … }`
: Include `i` to get the index value of each element in the selection:

```
d3.selectAll("rect")
  .attr("x", function(d, i) {
      return i * 10;  // Move each successive rect more to the right
  });
```

`selection``.filter()`
: Takes a selection and returns a new (sub)selection:

```
d3.selectAll("circle")
  .filter(function(d) {
       return d > 15;  // If 'true', element is included
  })
  .style("color", "red");
```

`d3.csv()`
: Loads an external CSV file, parses the contents into JSON, then hands off the results to a callback function:

```
d3.csv("food.csv", function(data) {
    console.log(data);
});
```

`d3.json()`
: Loads an external JSON file, parses the contents into JSON, then hands off the results to a callback function:

```
d3.json("waterfallVelocities.json", function(json) {
    console.log(json);
});
```

`d3.request()`
: Loads an arbitrary external file, then hands off the results to a callback function:

```
d3.request("interesting_data.txt")
  .get(function(response) {
      // Do something with the response string
});
```

Transitions

selection.transition()

Takes a selection, and initiates a new transition, so values specified after this point will be interpolated over time (rather than set immediately):

```
d3.selectAll("circle")
  .attr("cx", 0)     // Initial value for 'cx' is set
  .transition()      // Transition is initiated
  .attr("cx", 100);  // 'cx' will be interpolated to 100
```

transition.delay()

Takes a transition, and sets the delay, in milliseconds:

```
d3.selectAll("circle")
  .attr("cx", 0)
  .transition()
  .delay(1000)  // Wait 1 second before starting
  .attr("cx", 100);
```

transition.duration()

Takes a transition, and sets the duration, in milliseconds:

```
d3.selectAll("circle")
  .attr("cx", 0)
  .transition()
  .duration(2000)  // Transition will occur over 2 seconds
  .attr("cx", 100);
```

transition.ease()

Takes a transition, and sets the easing to be used:

```
d3.selectAll("circle")
  .attr("cx", 0)
  .transition()
  .ease(d3.easeLinear)  // Transition will be linear
  .attr("cx", 100);
```

transition.on()

Takes a transition, and binds a function to be executed at either the `"start"` or `"end"`:

```
d3.selectAll("circle")
  .attr("cx", 0)
  .transition()
  .attr("cx", 100)
  .on("end", function() {  // <-- Executes after transition
      console.log("All done!")
  });
```

Scales

`d3.scaleLinear()`
: Creates a new linear scale function:

```
var xScale = d3.scaleLinear();
```

`scaleLinear``.domain()`
: Sets a linear scale's input domain:

```
xScale.domain([ 0, 2000 ]);
```

`scaleLinear``.range()`
: Sets a linear scale's output range:

```
xScale.range([ 0, width ]);
```

`scaleLinear``.rangeRound()`
: Sets a linear scale's output range, and has all values output by the scale rounded to the nearest whole number:

```
xScale.rangeRound([ 0, width ]);
```

`scaleLinear``.nice()`
: Expands a linear scale's domain to the nearest round values:

```
xScale.nice();
```

`scaleLinear``.clamp()`
: Forces any values output by this scale to be constrained (rounded to be) within the specified range:

```
xScale.clamp(true);
```

Other scale types—such as `scaleSqrt`, `scalePow`, and `scaleOrdinal`—may share similar methods or have unique methods of their own. Double-check the documentation for each type of scale.

`d3.min()`
: Returns the smallest value in an array:

```
d3.min([ 10, 20, 70, 35 ]);  // Returns 10
```

`d3.max()`
: Returns the largest value in an array:

```
d3.max([ 10, 20, 70, 35 ]);  // Returns 70
```

Axes

`d3.axisTop`, `d3.axisRight`, `d3.axisBottom`, *and* `d3.axisLeft`

Creates a new axis generator function, with the specified orientation:

```
var xAxis = d3.svg.axisBottom();
```

`axis``.scale()`

Takes an axis, and specifies the scale to be used:

```
xAxis.scale(xScale);
```

`axis``.ticks()`

Takes an axis, and specifies a target number of ticks to be used:

```
xAxis.ticks(5);
```

`axis``.tickValues()`

Takes an axis, and specifies the values to be labeled with ticks:

```
xAxis.tickValues([0, 100, 250, 600]);
```

`selection``.call()`

Takes a selection, and calls an arbitrary method to act upon the selection; commonly used to generate an axis:

```
// Calls xAxis(), generating axis elements inside 'g'
svg.append("g").call(xAxis);
```

Interactivity

`selection``.on()`

Takes a selection, and binds an event listener:

```
// Binds click functionality to #button
d3.select("#button")
  .on("click", function() { … });
```

`d3.select(this)`

Within an anonymous function, `this` refers to "the element being acted upon":

```
d3.selectAll("rect")
  .on("mouseover", function() {
      // The 'this' below refers to the rect underneath the mouse
      d3.select(this).classed("highlight", true);
  });
```

Numbers, Dates, and Times

`d3.range()`

Generates an array of sequential numbers:

```
d3.range(5);
//Returns [0, 1, 2, 3, 4]
```

`d3.format`

Creates a new number formatter, for converting numbers to strings:

```
var formatAsPercentage = d3.format(".1%");
formatAsPercentage(1.2);
//Returns "120.0%"
```

See the API reference for number formatting values.

`d3.timeParse`

Creates a new time parser, for converting strings to Date objects:

```
var parseTime = d3.timeParse("%m/%d/%y");
parseTime("02/20/17");
//Could return: Mon Feb 20 2017 00:00:00 GMT-0800 (PST)
```

`d3.timeFormat`

Creates a new time formatter, for converting Date objects to strings:

```
var formatTime = d3.timeFormat("%b %e");
formatTime(new Date);
//Returns today's date, e.g.: "Apr 28"
```

See the API reference for time formatting values.

Other Useful JavaScript

`parseInt()`

Converts a string (typically) to an integer:

```
parseInt("123")  // Returns 123
```

`parseFloat()`
: Converts a string (typically) to a floating-point (decimal) number:

```
parseFloat("456.789")  // Returns 456.789
```

`+`
: The unary plus operator attempts to convert what follows it into a number (like shorthand for `parseInt()` or `parseFloat()`):

```
+"123"      // Returns 123
+"456.789"  // Returns 456.789
```

`Math.random()`
: Returns a random value between 0.0 (inclusive) and 1.0 (exclusive):

```
Math.random() * 100  // Could return 61.87844036612…
```

`Math.round()`
: Rounds a value to the nearest integer (or, in the case of 0.5, to the nearest greater integer value):

```
Math.round(1.012)  // Returns 1
Math.round(1.5)    // Returns 2
Math.round(-1.5)   // Returns -1
```

`Math.ceil()`
: Rounds a value up to the nearest integer:

```
Math.ceil(23.011231444)  // Returns 24
```

`Math.floor()`
: Rounds a value down to the nearest integer:

```
Math.floor(61.87844036612)  // Returns 61
```

`array``.push()`
: Appends a new value to an existing array:

```
var numbers = [ 2, 3, 4, 5 ];
numbers.push(6);  // Now numbers is [ 2, 3, 4, 5, 6 ]
```

`array``.shift()`
: Removes the first value from an existing array, and returns that value:

```
var animals = [ "dog", "cat", "bird" ];
animals.shift();  //Returns "dog"
//Now animals is [ "cat", "bird" ]
```