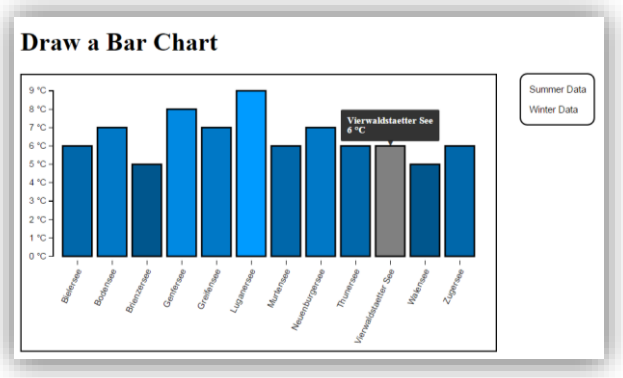
Tutorial D3.js: Bar Chart



Ein Bar Chart ist ein bekanntes Diagramm um Häufigkeiten, Merkmale von Kategorien, etc. darzustellen. Natürlich kann ein Bar Chart in vielen verschiedenen Programmen einfach erstellt werden (z.B. Microsoft Excel). Die Programmierung eines Bar Charts mit D3.js ist etwas schwieriger zu lernen, bietet aber zwei wesentliche Vorteile:

- Freier Zugang: Das Diagramm wird direkt im Browser gezeichnet und kann in jede beliebige Webseite eingebunden werden. Um es anzuzeigen sind keine zusätzlichen Programme nötig.
- Interaktivität: Der Bar Chart kann mit interaktiven Elementen beliebig erweitert werden: Tooltips, Hover-Effekte, onclick-Events, dynamisches Laden von neuen Daten mit Übergängen, usw.

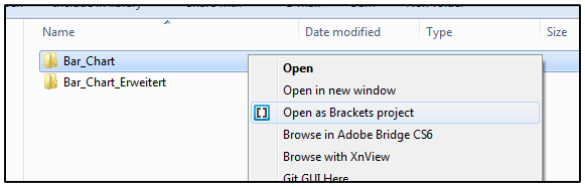
Wir werden unser Tutorial mit einem Überblick über die verwendeten Tools beginnen, danach folgt eine Beschreibung des verwendeten Datenformats. Im dritten Teil wird Schritt für Schritt die Anpassung eines einfachen Beispiels angeschaut und zum Schluss können dem Bar Chart beliebige Erweiterungen hinzugefügt werden.

Verwendete Tools

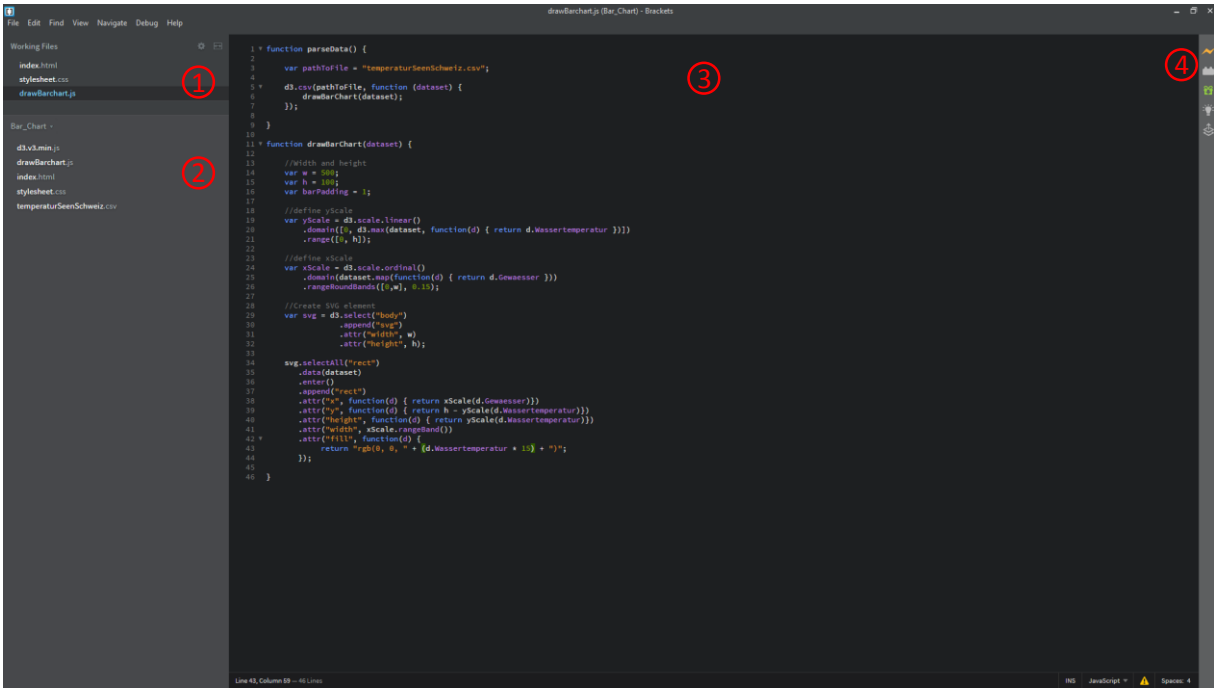
Brackets

Brackets ist ein Open Source Texteditor, der sich für Frontenddesign (CSS und Javascript) eignet und intuitiv bedienbar ist. Er kann unter folgendem Link heruntergeladen werden: http://brackets.io/

Um ein neues Projekt mit Brackets zu beginnen, wird am besten eine Ordnerstruktur mit Projektnamen und bereits vorhandenen Files mit der rechten Maustaste („Open as a brackets project“) geöffnet. Auch ein leerer Ordner genügt.



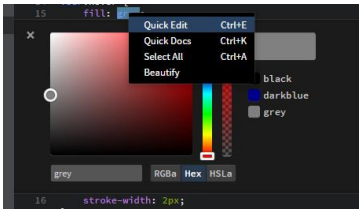
Folgende Ansicht bietet sich nach dem Öffnen des Programms:



1. **Working Files:** Momentan geöffnete Dokumente werden hier angezeigt.
2. **Projektdokumente:** Alle Dokumente des aktuellen Projektes werden hier aufgelistet. Auch Unterordner können enthalten sein. Mit Doppelklick werden die Dokumente den Working Files hinzugefügt.
3. **Dokumentinhalt:** Inhalte des momentan ausgewählten Dokuments werden hier angezeigt.
4. **Live-Preview**: Mit einem Klick auf das Blitzsymbol wird eine Live Vorschau in Google Chrome geöffnet (Wird geschlossen bei Aktivierung der Entwicklertools in Google Chrome).

Zwei nützliche Tools von Brackets:

- **Live Preview:** Bei Aktivierung der Live Vorschau von Brackets öffnet sich ein neues Google Chrome Fenster, in welchem der aktuelle Zustand des html-Dokuments angezeigt wird. Um die Live Vorschau zu aktivieren, muss das momentan geöffnete Dokument vom Typ html sein. Danach werden Änderungen in CSS und HTML Live im neu geöffneten Fenster übernommen.
- **Color Picker:** In einem CSS Dokument kann eine Farbe mit Rechtsklick oder Ctrl+E im Quick Edit Modus geöffnet und bearbeitet werden.



Google Chrome

Google Chrome ist ein Webbrowser von Google. Der grösste Teil davon wird als Open Source Paket zur Verfügung gestellt. Er eignet sich gut zum Programmieren und wird unter anderem von Brackets verwendet.

Das drücken von F12 in einem offenen Fenster von Chrome öffnet die **Entwicklertools**:

- **Elements:** Hier werden strukturierte Informationen über die momentane Seite dargestellt. Auf der rechten Seite befinden sich die Style-Regeln. Diese bewirken, dass das Element so dargestellt wird, wie auf dem Bildschirm ersichtlich. Mit Rechtsklick auf ein beliebiges Objekt auf dem Bildschirm und „Inspect“ können einzelne Elemente genauer untersucht werden. Alternativ kann zu jedem beliebigen Element in der DOM-Hierarchie links navigiert werden.

- **Console:** Allfällige Fehler oder mittels console.log() geschriebene Befehle im Code werden hier ausgegeben. Links stehen jeweils die Meldung über den Fehler oder der ausgegebene Wert und rechts das File und die Zeilennummer, die den Fehler oder die Ausgabe ausgelöst hat.

Verwendetes Datenformat

In unserem Beispiel verwenden wir das CSV Dateiformat.

```
Gewaesser,Wassertemperatur
Bielersee,6
Bodensee,7
Brienzersee,5
Genfersee,8
Greifensee,7
Luganersee,9
Murtensee,6
Neuenburgersee,7
Thunersee,6
Vierwaldstaetter See,6
Walensee,5
Zugersee,6
```

Beispiel einer CSV Datei

CSV steht für „Comma-separated values“. Das Format wird oft für einfache Tabellen verwendet. In MS Excel kann jede Tabelle als CSV exportiert werden („Speichern unter“ => „Dateityp auswählen), dabei muss aber beachtet werden, dass Excel auf die Regions- und Spracheinstellungen des Computers zugreift, um die Art des Listenseparators zu bestimmen. Ist der Computer auf Deutsch eingestellt, wird standardmässig ein Semikolon verwendet. Dies kann in den Regions- und Spracheinstellungen geändert werden. Allenfalls bieten andere Programme wie LibreOffice bessere Möglichkeiten für die Arbeit mit CSV Dateien.

Für die Entwicklung eines Bar Chart mit d3.js ist es am einfachsten, wenn als Listentrennzeichen ein Komma verwendet wird.

In der ersten Zeile einer CSV Datei befinden sich die Spaltenüberschriften. Ab Zeile Zwei sind die Wertepaare aufgelistet, wobei jedes Komma einen Wert vom nächsten trennt. Sind Kommas in den einzelnen Werten vorhanden, werden diese mit Anführungszeichen maskiert. In unserem Beispiel wäre das erste Tupel das Gewässer Bielersee mit einer Wassertemperatur von 6 Grad.

Anpassung eines einfachen Beispiels

Im Internet gibt es viele, teils sehr ausführliche Tutorials zur Erstellung eines Bar Charts mit d3.js. Trotzdem wird nachfolgend die Programmierung eines einfachen Bar Chart Schritt für Schritt beschrieben. Dabei wird oft auf die Kapitel des online verfügbaren Buches „Interactive Data Visualization for the Web“ von O’reilly verwiesen, welche die jeweiligen Themen sehr ausführlich und gut beschreiben.

Ziel ist es, das nachfolgende Beispiel nachzubauen und zu verstehen.

Grundgerüst

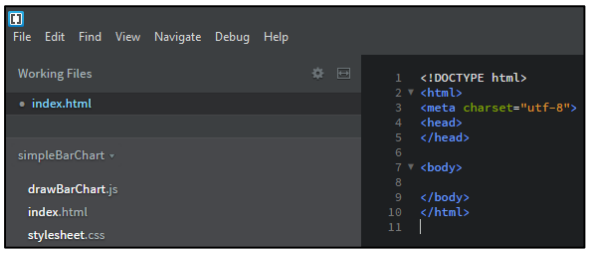
Als Erstes erstellen wir einen Ordner für unser Projekt. Wir benennen ihn zum Beispiel mit „simpleBarChart“. Darin sollten folgende (noch leeren) Dokumente erstellt werden

- index.html
- drawBarChart.js
- stylesheet.css

Mit Rechtsklick im ausgewählten Ordner kann ein neues Textdokument erstellt werden, bei welchem der Name und die Dateiendung entsprechend angepasst werden kann.

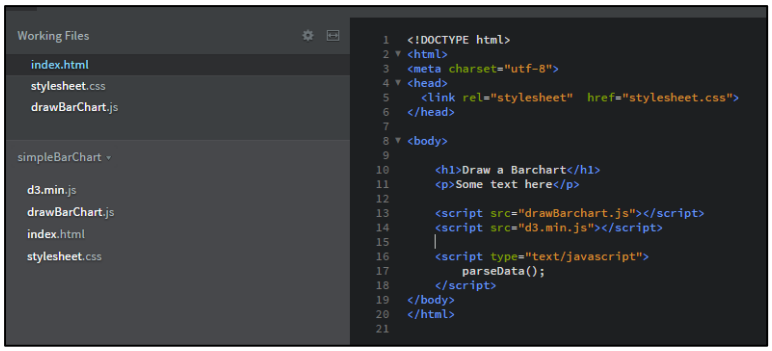
HTML Dokument

Nun öffnen wir den Ordner „simpleBarChart“ mit Rechtsklick „Open as Brackets project“. Wir öffnen das index.html File mit Doppelklick und schreiben das HTML Grundgerüst:

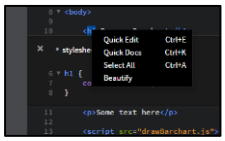


Danach binden wir unsere zuvor erstellten Dokumente, drawBarChart.js und stylesheet.css, ein. So erreichen wir, dass die CSS Design-Regeln, welche wir in unserem Stylesheet definieren und die Funktionen des JavaScript Codes, auf unser html Dokument angewendet werden.
Da wir unseren Bar Chart mit D3.js erstellen, müssen wir noch ein weiteres Dokument einbinden. Dazu navigieren wir auf d3js.org. Wir finden dort einerseits Informationen zur D3.js JavaScript Bibliothek und andererseits den d3.zip Ordner. Wir laden diesen herunter und extrahieren das d3.min.js Dokument in unseren Ordner „simpleBarChart“. Um das File im Brackets Editor zu sehen, müssen wir auf der linken Seite des Editors, mit Rechtsklick, die Option „Refresh File Tree“ ausführen. Wir binden das Dokument wie ein normales JavaScript Dokument ein. Nun sind wir bereit, die D3.js Library zu benutzen.
Zum Schluss fügen wir noch eine Überschrift und falls gewünscht einen Text (Paragraph) ein.

Unser HTML Dokument sollte nun folgendermassen aussehen:



Jetzt können wir zum ersten Mal die Live Vorschau (Blitzzeichen rechts oben) verwenden. Wir werden bis auf die Überschrift und den Paragraph eine leere Seite antreffen. Wird der Bildschirm getrennt und im Code z.B. die Überschrift angepasst, können wir beobachten, wie sich simultan dazu die Live Vorschau im Browser verändert.



Eine weitere schöne Funktion von Brackets ist, dass wir mit Rechtsklick auf ein HTML Element => „Quick Edit“ direkt dessen Styleregeln bearbeiten können. Wir setzen für h1 z.B. color: darkblue; und sehen das Resultat in der Livevorschau.

Weitere Informationen:

- Zu HTML: http://www.w3schools.com/html/
- Zu CSS: http://www.w3schools.com/css/
- Zu D3.js: http://d3js.org/
  http://chimera.labs.oreilly.com/books/1230000000345/ch02.html#_origins_and_context
- Zu Brackets: http://brackets.io/

Parse CSV Data

Zuerst wählen wir die Daten welche wir visualisieren möchten aus. Unter http://tutorial-bar-chart.opendata.iwi.unibe.ch/ => Links sind einige Datenbeispiele vorhanden. Auch eigene können gewählt werden. Wir kopieren das CSV File in den Projektordner.

Wir öffnen nun das JavaScript Dokument drawBarChart.js. Dort definieren wir die Funktion parseData();

Damit die Funktion parseData() ausgeführt wird, müssen wir noch einmal unser HTML Dokument anpassen:

Wir definieren mit <script type“text/javascript“> einen Bereich, welchen den Browser als JavaScript erkennt. In diesem Bereich können wir JavaScript Code schreiben. Wir rufen die Funktion parseData() auf. Dies funktioniert, da wir das drawBarChart.js Dokument bereits in das HTML Dokument eingebunden haben.
JavaScript verändert normalerweise die Struktur und den Inhalt der Seite, welche mit dem HTML

Dokument erstmals definiert wird (DOM). Um sicher zu gehen, dass alles geladen wurde, was JavaScript später verändern will, wird die Funktion erst zum Schluss des HTML Dokuments aufgerufen (Browser lädt von oben nach unten).

Das Ziel der Funktion parseData() ist es, ein CSV einzulesen und in einem Array abzuspeichern. Wir finden die Funktion d3.csv() welche genau dies tut.

In einem ersten Schritt definieren wir die Variable pathToCSVFile, welche den Pfad zu dem gewünschten CSV Dokument speichert. Diesen Pfad übergeben wir als erstes Argument der d3.csv() Funktion. Das zweite Argument nennt sich callback function. Die callback function wird aufgerufen, sobald d3.csv() das CSV Dokument fertig in unseren Array „dataset“ geladen hat.

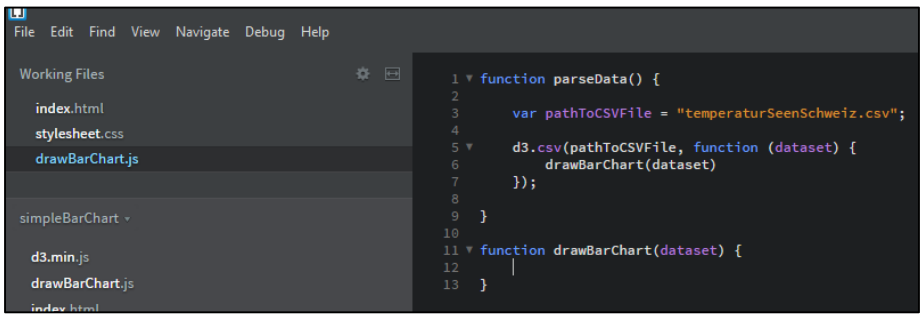
Wir geben die Variable dataset via console.log() Funktion in der Konsole von Google Chrome aus. Wir erkennen einen Array von JavaScript Objekten:

Weiterführende Informationen:

- Relative Pfade: http://www.peterkropff.de/site/html/pfade.htm
- D3.csv():http://chimera.labs.oreilly.com/books/1230000000345/ch05.html#_data
- Javascript callback functions: http://www.impressivewebs.com/callback-functions-javascript/
- Console.log() und Output: http://www.w3schools.com/js/js_output.asp
- DOM : https://wiki.selfhtml.org/wiki/JavaScript/DOM
- Javascript: http://www.w3schools.com/js/

SVG Element definieren

Nun kommt der eigentliche Teil unserer Arbeit. Das Darstellen der Daten als Bar Chart.
Zuerst definieren wir eine neue Funktion drawBarChart() und übergeben dieser als Argument die Variable dataset.



Wir definieren ein SVG Element. SVG steht für Scalable Vector Graphics und ist ein HTML5 Kontainer, welcher diverse Methoden bereitstellt um Pfade, Boxen, Kreise, Text, etc. zu zeichnen. In diesem SVG Element wird sich später unser Bar Chart befinden.

```
function drawBarChart(dataset) {

    //Width and height of SVG Element
    var w = 500;
    var h = 100;

    //Create SVG element
    var svg = d3.select("body")
                .append("svg")
                .attr("width", w)
                .attr("height", h)
                .attr("id", "svgElement");

}
```

Mit // können wir unseren Code kommentieren. Kommentare werden vom Browser ignoriert und dienen nur dazu, den menschlichen Lesern des Codes bei der Orientierung und beim Verständnis zu helfen. Um einen Kommentar über mehrere Zeilen zu schreiben wird /* *Kommentar* */ verwendet.

Die Variable w definiert später die Breite des SVG Elements in Pixel. Analog definiert Variable h die Höhe. Variablen welchen einen fixen Wert zugewiesen werden (hardcoded), sollten übersichtlich dargestellt werden, damit sie später einfach angepasst werden können.

Um das SVG Element zu zeichnen, verwenden wir wiederum eine Funktion von D3.js. D3 erlaubt eine Technik, die sich Chain Syntax nennt. Diese ermöglicht es, verschiedene Funktionen in nur einer Linie Code zu schreiben. Es entstehen Ketten von Methoden, die jeweils mit einem Punkt verknüpft werden (Siehe Bild oben). Jede Funktion sollte auf eine eigene Linie geschrieben werden, um die Übersicht zu wahren.

Schauen wir nacheinander die einzelnen Methoden der Definition des SVG Elementes an:

- **d3**: Referenziert das D3 Objekt, so dass wir dessen Methoden aufrufen können.
- **select(„body“):** Der select() Methode von D3.js wird ein CSS Selektor übergeben. Select() gibt das erste passende Element zurück, während selectAll() alle Elemente referenziert.
- **append(„svg“):** Append() platziert jedes gewünschte Element (in unserem Fall SVG) in dem dem Element, welches zuvor mit select() gewählt wurde. Falls sich in diesem Element bereits andere Elemente befinden, wird das neue am Ende platziert.
- **attr(„width“, w):** Mit der attr() Methode können Attribute dem gewählten Element hinzugefügt werden, welche Eigenschaften des Elements definieren. Wichtige Attribute sind „id“, „class“, „width“, „heigth“. In unserem Fall wird die Breite des SVG Elements mit der Variable w definiert.
- **attr(„height“, h):** Analog wird die Höhe definiert
- **attr(„id“, „svgElement“):** Dem SVG Element wird eine Id hinzugefügt, welche zum Beispiel von CSS verwendet werden kann. Wir erstellen im Stylesheet eine neue Regel:

  ```
  #svgElement {
      border: 2px solid black;
  }
  ```

  Der Rahmen des SVG Elements sollte nun schwarz und 2px breit erscheinen.

Weiterführende Informationen:

- SVG: http://www.w3schools.com/html/html5_svg.asp
- Chaining methods: http://chimera.labs.oreilly.com/books/1230000000345/ch05.html#_chaining_methods
- CSS Selektoren: http://www.w3schools.com/css/css_syntax.asp

Bar Chart zeichen

Der fertige Code für den simplen Bar Chart sieht folgendermassen aus:

```
function drawBarChart(dataset) {

    //Width and height of SVG Element
    var w = 500;
    var h = 100;
    var barPadding = 1;

    //define yScale
    var yScale = d3.scale.linear()
        .domain([0, d3.max(dataset, function(d) { return d.Wassertemperatur })])
        .range([0, h]);

    //define xScale
    var xScale = d3.scale.ordinal()
        .domain(dataset.map(function(d) { return d.Gewaesser }))
        .rangeRoundBands([0,w], 0.15);

    //Create SVG element
    var svg = d3.select("body")
        .append("svg")
        .attr("width", w)
        .attr("height", h)
        .attr("id", "svgElement");

    svg.selectAll("rect")
       .data(dataset)
       .enter()
       .append("rect")
       .attr("x", function(d) { return xScale(d.Gewaesser)})
       .attr("y", function(d) { return h - yScale(d.Wassertemperatur)})
       .attr("height", function(d) { return yScale(d.Wassertemperatur)})
       .attr("width", xScale.rangeBand())
       .attr("fill", function(d) {
            return "rgb(0, 0, " + (d.Wassertemperatur * 15) + ")";
       });

}
```

Gehen wir Schritt für Schritt durch, was sich verändert hat:

Wir haben zwei verschiedene Skalen definiert, die yScale und die xScale. Falls nachfolgende Erklärung nicht genügt werden auf dieser Seite die Skalen sehr genau beschrieben:

- http://chimera.labs.oreilly.com/books/1230000000345/ch07.html

y-Skala

Wir nehmen die Methodenketten wieder auseinander um die einzelnen Funktionen besser zu verstehen:

- **var yScale = d3.scale.linear()**: Für die y-Skala (yScale) definieren wir eine Funktion des D3 Objekts. Wir verwenden eine lineare Skala, welche wir immer dann benutzen können, wenn die Werte mindestens Intervallskaliert sind. Das bedeutet, dass sich die Abstände zwischen den verschiedenen Merkmalsausprägungen exakt bestimmen lassen.
  Wir weisen die Funktion einer Variablen zu, damit wir sie später einfach verwenden können.

- **.domain():** Die Domäne (Definitionsbereich) der Skala bestimmt den Wertebereich, welche sich durch die Skala verarbeiten lassen. In unserem Beispiel sehen wir uns die Temperatur von Seen im Winter an. Da Wasser in einem See (normalerweise) keine Temperatur unter 0 Grad annehmen kann, nehmen wir den Wert 0 als Untergrenze. Als Obergrenze der Werte, die durch die Funktion verarbeitet werden können, wäre es sinnvoll, den Maximalwert unseres Datensatzes zu bestimmen. Wenn der Datensatz verändert wird (z.B. Temperatur der Seen im Sommer) muss sich dieser Wert dynamisch anpassen können. Dies erreichen wir, indem wir eine Funktion des D3 Objektes verwenden und schreiben:
  d3.max(dataset, function(d) {return d.Wassertemperatur})

Als erstes Argument übergeben wir der Funktion unseren Datensatz. Das zweite Argument nennt sich anonyme Funktion. Sie sorgt dafür, dass sich D3 nur die relevanten Daten ansieht und daraus das Maximum bestimmt. In unserem Fall besitzt jedes Datenobjekt sowohl eine Eigenschaft „Beschreibung" wie auch „Wassertemperatur" (z.B. Gewaesser: Bielersee, Wassertemperatur: 6). Wir wollen aber, dass unsere Funktion nur das Maximum der Wassertemperatur herausfindet. Dies erreichen wir mit **function(d) { return d.Wassertemperatur }**
Wir fügen die beiden Werte in einen Array ein und übergeben diese unserer Funktion als Argument:
.domain([0, d3.max(dataset, function (d) {return d.Wassertemperatur})])
(Genaueres unter weitere Informationen)

- **.range():** Die y-Skala wird in unserem Beispiel verwendet, um die Höhe der Rechtecke für unseren Bar Chart zu bestimmen. Jedes Mal, wenn wir die Funktion yScale aufrufen und ihr einen Wert (Wassertemperatur) übergeben, muss die Funktion einen Wert zurückgeben (Standardmässig in Anzahl Pixel). Damit die Funktion weiss, wie hoch sie die Bars absolut darstellen darf, braucht sie einen Bereich: range().
  Dieser definiert, welches Minimum und welches Maximum an Höhe die Bars haben dürfen. Wir übergeben die minimale Höhe (z.B. 0) und die maximale Höhe (z.B. die Höhe des SVG Elements) in einem Array als Argument und schreiben:
  range([0, h])

x-Skala

- **var yScale = d3.scale.ordinal()**: Die Skala auf der x-Achse ist nicht vom Typ Intervallskala. Die Gewässer können zwar in eine Reihenfolge gebracht werden (z.B. Alphabethisch), die Abstände dazwischen sind aber nicht messbar. Wir verwenden darum eine Ordinalskala. Das Ziel der Funktion ist es, die Bars so zu platzieren, dass sie nebeneinander stehen.

- **domain():** Die Ordinalskala benötigt einen eindimensionalen Array als Input. Beispielsweise [Bielersee, Bodensee, Brienzersee]. In unserer Variablen dataset haben wir aber einen Array von Javascript Objekten. Wir schreiben diesen in einen simplen Array um mit
  dataset.map(function(d) { return d.Gewaesser })
  Fügen wir dies als Argument in unsere domain() Funktion ein erhalten wir:
  domain(dataset.map(function(d) { return d.Gewaesser }))

- **rangeRoundBands():** rangeBands ist eine Funktion von D3.js, die uns eine Menge Arbeit abnimmt. Sie dividiert die Breite des Diagramms durch die Anzahl Bars und platziert sie danach automatisch richtig. Als erstes Argument übergeben wir wie bei der y-Skala einen Array mit der Anzahl Pixel im Minimum (0) und im Maximum (w).
  rangeBands([0, w])
  Bei rangeBands() ist es möglich, ein zweites Argument hinzuzufügen, welches den Abstand zwischen den einzelnen Bars bestimmt. Wir setzen eine neue Variable barPadding = 0.15 und fügen diese als zweites Argument hinzu.
  rangeBands([0 , w], barPadding)
  Nun schreiben wir noch **rangeRoundBands()** anstatt **rangeBands()**, was bewirkt, dass die Anzeige immer auf den nächstliegenden Pixel gerundet wird.

Weitere Informationen:

- Intervallskalen: https://de.wikipedia.org/wiki/Intervallskala
- Skalen in D3: http://chimera.labs.oreilly.com/books/1230000000345/ch07.html
- D3 max() und min(): http://chimera.labs.oreilly.com/books/1230000000345/ch07.html#_d3_min_and_d3_max

Momentan sieht unser Code folgendermassen aus:

```
function drawBarChart(dataset) {

    //Width and height of SVG Element
    var w = 500;
    var h = 100;
    var barPadding = 0.15;

    //define yScale
    var yScale = d3.scale.linear()
        .domain([0, d3.max(dataset, function(d) { return d.Wassertemperatur })])
        .range([0, h]);

    //define xScale
    var xScale = d3.scale.ordinal()
        .domain(dataset.map(function(d) { return d.Gewaesser }))
        .rangeRoundBands([0,w], barPadding);

    //Create SVG element
    var svg = d3.select("body")
        .append("svg")
        .attr("width", w)
        .attr("height", h)
        .attr("id", "svgElement");

}
```

Grundlegender Codeaufbau zum Zeichnen eines Diagramms mit D3

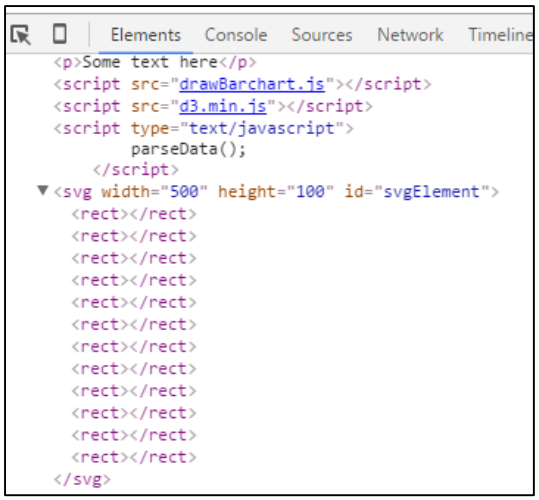
Wenn wir uns die Live-Vorschau im Browser ansehen, sehen wir immer noch kein Diagramm. Die Bars werden erst im nächsten Schritt gezeichnet:

```
    svg.selectAll("rect")
       .data(dataset)
       .enter()
       .append("rect");
```

D3.js bindet jedes Datenobjekt an ein Element (in unserem Beispiel „rect“). Wir bestimmen dann Attribute wie Höhe, Breite, Position und Farbe jedes Elements mithilfe von Werten des gebundenen Datenobjekts. Der grundlegende Codeaufbau dafür ist folgender:

- **svg.selectAll(„rect“):** Mit svg.selectAll() wählen wir alle bestehenden Rechtecke im SVG Element aus. Dieser Schritt ist nötig, obwohl in unserem Beispiel bisher noch keine Rechtecke vorhanden sind. Die Variable svg wurde bereits früher beim Zeichnen des SVG Elements definiert und referenziert dieses.
- **.data(dataset):** Mit dieser Methode wählen wir die Daten aus und binden sie an die Elemente, die wir definierten werden. In unserem Beispiel wird die Variable dataset gewählt, welche unseren Datensatz bereithält.
- **.enter():** Enter gibt ein Platzhalter zurück, welcher auf das neue Element referenziert.
- **.append(„rect“)**: Fügt ein Rechteck für jeden Datensatz dem DOM (Struktur des HTML Dokuments) hinzu.

Wenn wir nun die Entwicklertools von Google Chrome öffnen, sehen wir zahlreiche leere „rect“ Elemente, welche dem SVG Element angefügt wurden. Die Anzahl entspricht unserer Anzahl Zeilen im CSV Dokument ohne die Überschriftenzeile. Denn jede Zeile wurde zum Datenobjekt und für jedes Datenobjekt ein neues „rect“ Element erstellt.



Genauere Informationen zum Binden der Daten an Elemente, generieren von Elementen und den grundlegenden Codeaufbau von D3.js:
http://chimera.labs.oreilly.com/books/1230000000345/ch05.html

Platzieren und definieren von Höhe und Breite der „rect“ Elemente

Um die leeren „rect“ Elemente in Bars zu verwandeln, fügen wir folgende Codezeilen hinzu:

```
36      svg.selectAll("rect")
37         .data(dataset)
38         .enter()
39         .append("rect")
40         .attr("x", function(d) { return xScale(d.Gewaesser)})
41         .attr("y", function(d) { return h - yScale(d.Wassertemperatur)})
42         .attr("height", function(d) { return yScale(d.Wassertemperatur)})
43         .attr("width", xScale.rangeBand())
44         .attr("fill", function(d) {
45              return "rgb(0, 0, " + (d.Wassertemperatur * 15) + ")";
46         });
```

SVG Elemente haben Attribute, welche mit CSS Eigenschaften vergleichbar sind. Die wichtigsten davon sind:

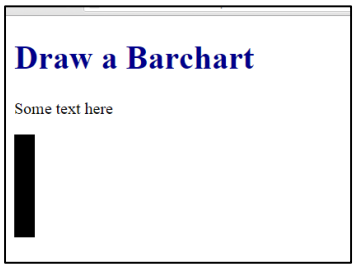
- x = bestimmt Position des Elements auf der x-Achse im SVG Element. Pixel 0 der x-Achse befindet sich im SVG Element am linken Rand.
- y = Position auf y-Achse. Am oberen Rand des SVG Elements befindet sich Pixel 0 der y-Achse.
- width = Breite des Elements
- height = Höhe des Elements
- fill = Farbe des Elements

Um nun die einzelnen Säulen des Bar Charts zu zeichnen werden die Attribute von D3.js bei jedem einzelnen „rect“ Element, mithilfe der Datenobjekte, die daran gebunden wurden, verändert. Wir schreiben Methoden in die Methodenkette (vgl. Bild oben):

- **attr(„height“, function(d) { return yScale(d.Wassertemperatur)}):** Wir übergeben die Wassertemperatur jedes Datenelements der y-Skala, welche diese in die entsprechende Säulenhöhe umwandelt. Dazu benötigen wir eine anonyme Funktion, welche für jedes Datenobjekt einzeln die Wassertemperatur ausliest und der y-Skala übergibt. Entsprechend des Rückgabewertes der y-Skala, bestimmt die Funktion das Attribut „height“ für jedes „rect“.

- **Attr(„width“, xScale.rangeBand()):** Wir führen die Methode rangeBand() der zuvor definierten xScale durch, welche die Rect Elemente entsprechend platziert. Wir benötigen hier keine anonyme Funktion, da alle Elemente gleichzeitig der rangeBand() Methode übergeben werden können.

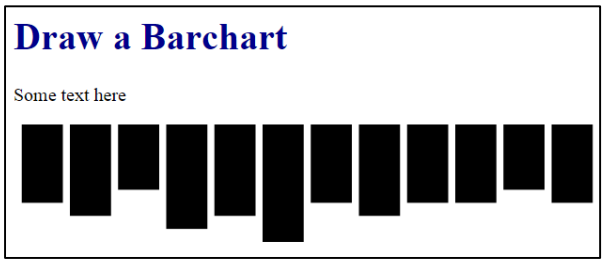
Sehen wir uns nun die Live Vorschau an:



Die Elemente haben die richtige Breite und Höhe, wurden aber noch nicht positioniert. Die Positionierung auf der x-Achse erreichen wir folgendermassen:

- **attr(„x“, function(d) {return xScale(d.Gewaesser)}):** Für die x Werte jedes einzelnen Bars benützen wir wiederum eine anonyme Funktion, die jedes Datenobjekt einzeln der Funktion xScale übergibt. Die Skala bestimmt die Anzahl Pixel zwischen dem linken Rand des SVG Elements und der Bars und gibt diese zurück. Der Rückgabewert wird als Attribut „x“ des bindenden „rect“ Elements abspeichert.

Wir sehen nun folgendes Bild:

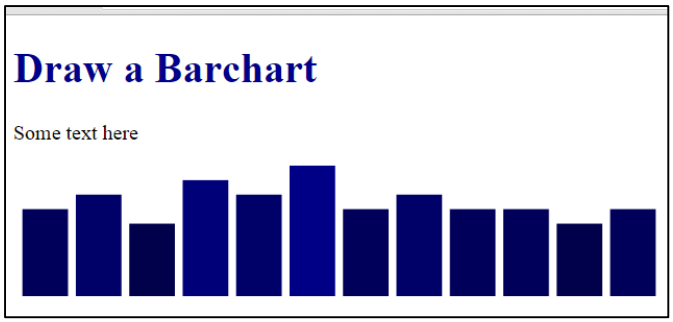


Um die Position auf der y-Achse zu bestimmen, ist es wichtig zu wissen, dass die y-Achse des SVG Elements oben beginnt und dort auch der Wert Null ist.

- **.attr(„y“, function(d) { return h - yScale(d.Wassertemperatur)}) :** Wir verschieben jedes Element um die Maximale Höhe (h) minus die Höhe des jeweiligen Bars nach unten. Jetzt erhalten wir ein korrektes Diagramm.

Nun können wir zum Beispiel die Farbe jedes einzelnen Bars entsprechend dessen Wassertemperatur anpassen:

- ```
  .attr("fill", function(d) {
      return "rgb(0, 0, " + (d.Wassertemperatur * 15) + ")";
  });
  ```
  Hier berechnen wir den blauen Wert der RGB Farbe des Elements mit einer einfachen Rechnung: Wir nehmen die Wassertemperatur und multiplizieren sie mal 15. Der Fantasie sind aber keine Grenzen gesetzt, wir können auch kompliziertere Berechnungen verwenden.
  ```



Fertig! Der erste Bar Chart wurde gezeichnet!

Weiterführende Informationen:

- SVG: http://www.w3schools.com/html/html5_svg.asp
- Attribute verändern mit d3.js: http://chimera.labs.oreilly.com/books/1230000000345/ch06.html
- RGB Farben: http://www.w3schools.com/cssref/css_colors.asp

Code Erweiterung

Folgende, nach zunehmender Schwierigkeit sortierte Möglichkeiten bieten sich nun, den Code weiter anzupassen:

- **Änderungen mit bereits Gelerntem**:
  Änderungen von Farbe der Säulen oder Hintergrund, ändern der Breite und Höhe des SVG Elements mit bereits bekannten Möglichkeiten.
  Neu positionieren des SVG Elements mit CSS und Anpassen des HTML Dokuments mit neuen Elementen.
  Hinzufügen von Hover Effekten mit CSS.

- **Hinzufügen von Event listeners** (Onclick, Onmouseover):
  http://chimera.labs.oreilly.com/books/1230000000345/ch09.html#_interaction_via_event_listeners

- **Hinzufügen von Tooltips mit D3.tip():**
  - Link zu d3.tip(): http://labratrevenge.com/d3-tip/javascripts/d3.tip.v0.6.3.js (Downloaden und einbinden wie d3.js)
  - Quick Usage: https://www.npmjs.com/package/d3-tip#quick-usage
  - Beispiel: http://bl.ocks.org/Caged/6476579

- **Hinzufügen von Achsen zum Bar Chart:**
  - Theorie dazu: http://bost.ocks.org/mike/bar/3/ (Ab Encoding Ordinal Data)
    http://chimera.labs.oreilly.com/books/1230000000345/ch08.html
  - Beispiele: http://bl.ocks.org/Caged/6476579
    http://bl.ocks.org/mbostock/3048450

- **Changing the Data:**
  - Theorie:
    http://chimera.labs.oreilly.com/books/1230000000345/ch09.html#_changing_the_data
  - Beispiel: http://bl.ocks.org/RandomEtc/cff3610e7dd47bef2d01
    http://bar-chart-extended.opendata.iwi.unibe.ch/

⇨ Es ist auch möglich, ein Beispiel wie http://bl.ocks.org/Caged/6476579 zu nehmen und mit eigenen Daten anzupassen.