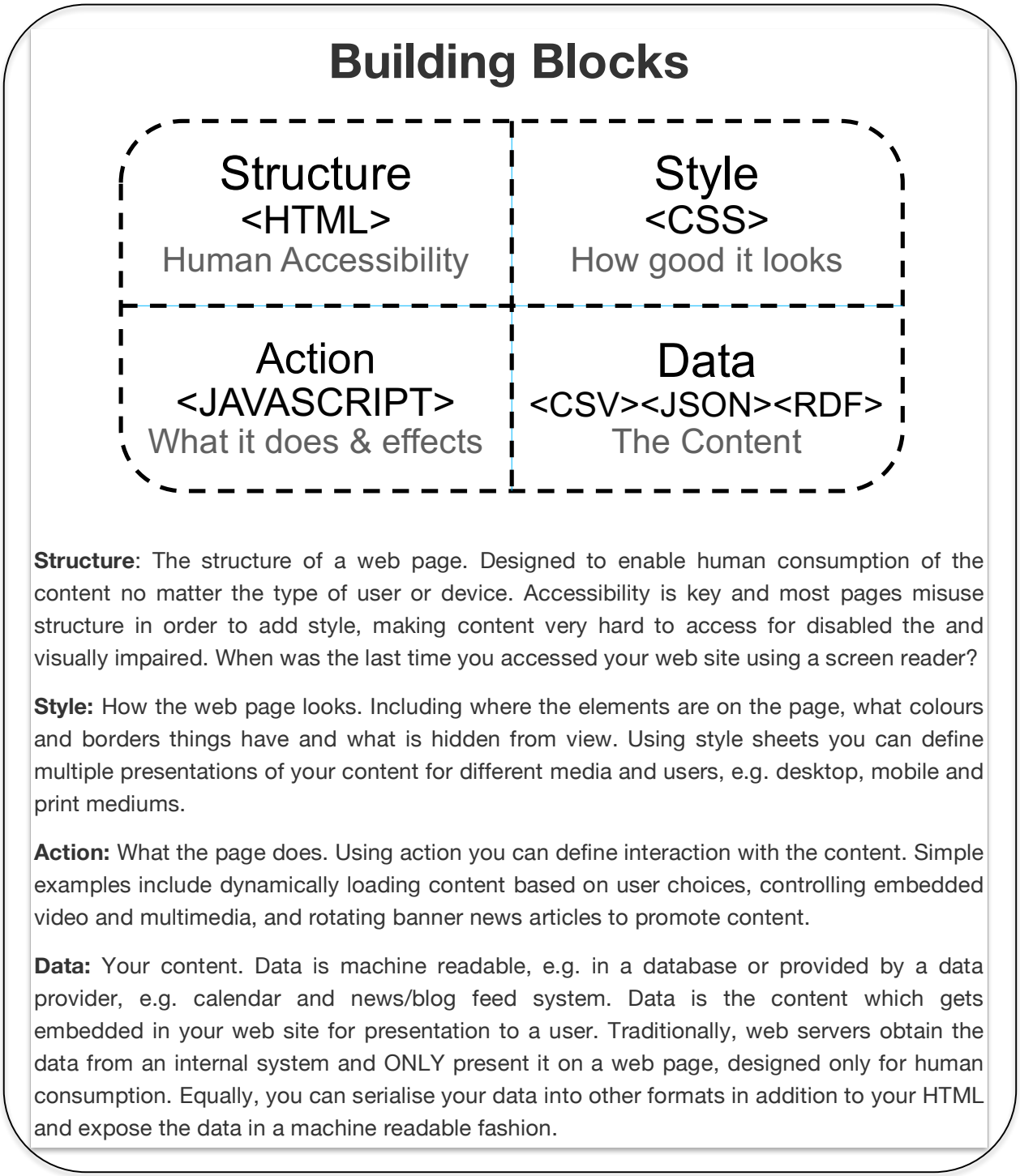
Building interactive dashboards

In this exercise we shall be using codepen.io to build an interface to explore complex data formats.

codepen.io provides a safe environment to experiment with building interactive web pages using modern HTML5 components outlined below. This exercise focusses on the **Action** block to load data and generate an advanced crossfilter visualisation.



Structure: The structure of a web page. Designed to enable human consumption of the content no matter the type of user or device. Accessibility is key and most pages misuse structure in order to add style, making content very hard to access for disabled the and visually impaired. When was the last time you accessed your web site using a screen reader?

Style: How the web page looks. Including where the elements are on the page, what colours and borders things have and what is hidden from view. Using style sheets you can define multiple presentations of your content for different media and users, e.g. desktop, mobile and print mediums.

Action: What the page does. Using action you can define interaction with the content. Simple examples include dynamically loading content based on user choices, controlling embedded video and multimedia, and rotating banner news articles to promote content.

Data: Your content. Data is machine readable, e.g. in a database or provided by a data provider, e.g. calendar and news/blog feed system. Data is the content which gets embedded in your web site for presentation to a user. Traditionally, web servers obtain the data from an internal system and ONLY present it on a web page, designed only for human consumption. Equally, you can serialise your data into other formats in addition to your HTML and expose the data in a machine readable fashion.

Exercise

The idea of this exercise is to extend beyond simple HTML5 pages and look at how Javascript can be used to generate interactive graphics from raw data

In order to complete this exercise we are going to use the codepen.io tool.

codepen.io

codepen (shown below) is an online sandbox which allows you freely to experiment with the building blocks of the web. It has three main panels that allow you to define structure, style and action related to your web page. The fourth panel (at the bottom) shows the result of your editing; this panel automatically updates as you enter your code.

In this exercise we shall be loading data dynamically and the aim is to produce a live, automatically updating, dashboard view of data on the web.

Starting point

In order to speed up this exercise a basic template has been created.

http://codepen.io/davetaz/pen/BZwred

This template has already defined the basic HTML elements as well as core JavaScript functions that will draw a basic graph of some data.

Load this template and fork it into you own account for editing.

Note that the **D3**, **bootstrap**, **crossfilter** and **dc** libraries have already been configured and loaded. These libraries make our job of making an interactive visualising much easier.

1. The framework

Before we begin, lets look at the content of the HTML and JavaScript blocks.

```
<div id="riskType-chart"></div>
<div id="bedroom-chart"></div>
<div id="riskLevel-chart"></div>
<div id="map"></div>
<table id="dc-data-table"
class="table table-hover"></table>
```

The HTML block defines 5 elements that are going to make up the parts of our dashboard. In total we will end up with 3 charts, a map and a data table.

```
// Define some variables to control our three charts.
var riskLevelChart = dc.pieChart("#riskLevel-chart");
//var riskTypeChart = dc.rowChart("#riskType-chart");
//var categoryChart = dc.rowChart("#bedroom-chart");
//var dataTable = dc.dataTable("#dc-data-table");
//var map = dc.geoChoroplethChart('#map');

// Load data from the web into a variable called data.
d3.csv(
  "https://raw.githubusercontent.com/theodi/training-web/gh-
pages/ods/course/en/exercises/Rapex-Publication.csv",
  function(data) {
    // Create the cross filter from the data
    var cf = crossfilter(data);
    var all = cf.groupAll();

    // ALL YOUR CODE GOES HERE

    // Render the dashboard
    dc.renderAll();
  });
```

The JavaScript block (shown above) provide a template to get you started. The comments (lines that start "//" explain what is going on. Note that you can see where where the data is loaded from an external source in CSV format.

The most important part of this code is to load the data. In this exercise we are going to be looking at producing an interactive dashboard for the rapex alerts data.

http://ec.europa.eu/consumers/consumers_safety/safety_products/rapex/alerts/

To help start this exercise Report 25 has been downloaded, cleaned and saved as a CSV at

https://raw.githubusercontent.com/theodi/training-web/gh-pages/ods/course/en/exercises/Rapex-Publication.csv

It is advisable to download this data for your own reference.

In order to complete this exercise ensure that the variables at the top of this block are uncommented so they become usable at the appropriate points. All other code added in this exercise goes in the function block, marked "`ALL YOUR CODE GOES HERE`". It is important that your code goes in the right place which is after the cross filter has been set up and before you render the output to the screen.

3. Add a risk type pie chart

As we are using a dataset about product recalls, we are going to add a number of charts that pull out key data to instantly draw the eye to key data. The first one we are going to add details risk level which includes “Serious risk”, something we should draw attention to!

Adding graphics involves two main steps:

1) Define the data.
2) Define the chart and load the data into it.

Add the following to the MIDDLE of your code block to define the data required for the pie chart.

```
var riskLevel = cf.dimension(function(d) {
      return d["Risk level"];
    });

var riskLevelGroup = riskLevel.group();
```

Here an `cf.dimension` is a row in the table. This row is loaded into a variable called `d`. We select the column called “Risk level” from this row and return it.

Add the following below the data definition to create a pie chart for the data. Don’t forget to uncomment the riskTypeChart chart at the top of the JavaScript block.

```
  riskLevelChart
     .width(180)
     .height(180)
     .radius(80)
     .dimension(riskLevel)
     .group(riskLevelGroup)
     .ordinalColors(["red", "orange"])
     .label(function(d) {
       var label = d.key;
       return label;
     });
```

Feel free to change the height, width, radius and colours of the pie chart to suit your own preferences.

Once done you should have rendered a pie chart on your page that shows the different drisk levels.

This is a simple visualisation, but not a great cross filter as we only have one chart. So let’s plot another.

4. Adding a second chart

In this section we are going to add a second chart to our cross filter. In particular we are going to add a row chart looking at the risk type.

Add the following below your pie chart code to load the data.

```
var riskType = cf.dimension(function(d) {
      return d["Risk type"];
});
var riskTypeGroup = riskType.group();
```

Add the following below your data load to add the chart to your cross filter. Remember to uncomment the riskTypeChart chart at the top in order to display it!

```
riskTypeChart
    .width(480)
    .height(400)
    .group(riskTypeGroup)
    .dimension(riskType)
    .label(function(d) {
      return d.key;
    })
    // Title sets the row text
    .title(function(d) {
      return d.value;
    })
    .elasticX(false);
```

With this chart rendered you now have your first cross filter. To use it select an option from any chart to filter all other charts.

With the line chart you can also change axis properties. Try changing the elasticX property to false and see what the difference is when you select a value from the pie chart.

5. Adding a third chart (less guidance)

The last chart to add is the category chart. Again this is going to be a row chart.

Follow the steps in (4) to create this chart, remember to change the names to variables appropriately and load the right column from the source data.

6. Adding a map

The crossfilter library allows you to add a map, allowing you to filter the data by clicking on countries. We are going to add a map that details the country of origin of our products.

First, select the data from the crossfilter in the normal way.

```
var country = cf.dimension(function(d) {
      return d["Country of origin"];
});
var countryGroup = country.group();
```

Before we can use this data (as per the previous steps) we first need to define the map we want to display. To do this we need to load in a geographic data file detailing the map features. In this case we need a world map of countries. We are going to a use a low resolution 100m outline map of the world.

To load this add the following code below your countryGroup.

```
d3.json("https://raw.githubusercontent.com/theodi/training-
web/gh-pages/ods/course/en/exercises/countries.geo.json",
function (mapJson) {

     // USE MAP DATA HERE

     dc.renderAll();
});
```

Note also that you need to move the render line inside this code for the map to render properly.

Finally, where we have left a comment we need to then define our map similarly to the other charts.

```
map
    .width(1280)
    .height(600)
    .dimension(country)
    .group(countryGroup)
    .projection(d3.geo.mercator()
        .translate([600,400])
        .scale(200)
     )
     .colors(d3.scale.quantize().range(["#0061B5", "#C4E4FF"]))
     .colorDomain([0, 200])
     .colorCalculator(function (d) { return d ? map.colors()(d) :
'#ccc'; })
     .title(function (d) {
      if (!d.value) {
            d.value = 0;
      }
      return d.key + ": " + Math.round((d.value / all.value()) * 100) +
"% of products";
      })
      .overlayGeoJson(mapJson.features, "name", function (d) {
          return d.properties.name;
      });
```

6. View tabular data

As you are filtering you can also view the raw data in a table. We do not need to select the data as we can use any dimension to access the current filtered data. In this case we are going to use the riskLevel dimension.

Add the following code to where the rest of the charts are made in order to draw the table.

```
dataTable
      .dimension(riskLevel)
      .group(function(d) {
        return d.target;
      })
      .size(10)
      .columns([
        "Alert number",
        "Brand",
        "Product",
        "Name",
        "Description"
      ]);
```

Now you can filter the data based upon many conditions and also see the raw data in a table. Feel free to change the columns and number of rows that display in the table.

The completed example is available from the following URL:

http://codepen.io/davetaz/pen/RgLQXz

Documentation and examples for the crossfilter library can be found at

https://github.com/dc-js/dc.js/blob/master/web/docs/api-1.6.0.md