D3 PROJECT
GRAPH OF FLU TRENDS IN CANADIAN PROVINCES

Used dataset: http://www.google.org/flutrends
Project was tested in *Mozilla Firefox* - the behaviour in other browsers is not fine-tuned :-)

Map of Canada

It is necessary to find suitable map in svg format, in this project we used the following one (after small adjustments):
https://upload.wikimedia.org/wikipedia/commons/3/38/Canada_blank_map.svg
Insert all **g** elements from the map file to the **svg** panel in the file *index.html*.

Import of data

Data were originally downloaded in TXT format. We then saved the file as CSV file. After inserting the following code into the file myJScode.js and running the project in web browser, we can check the format of our data in the console – it is an array of objects, where object represents a record of values in given time.

```
d3.csv("data.csv").then(function(d) {
  console.log(d);
});
```

The header of the CSV file was used as **Property names** for data objects. The function d3.csv requires the loaded csv file to have the header defined.

All the values are strings, but we don't want that for numerical values. For the correct data interpretation, we therefore must convert their type. A simple way to do this is the use of **unary operator plus**. For iteration over all the elements of the array we use *forEach* function. Using dot notation, we can access each Property. If the Property has spaces in the name, it is necessary to use bracket notation.

```
d3.csv("data.csv").then(function(d) {
  data.forEach(function(d){
    d.Canada = +d.Canada;
    d["New Brunswick"] = +d["New Brunswick"];

  })
  console.log(d);
});
```

For this adjustment, we can also use **parameters of the function d3.csv** which sets the **scheme of the Properties** of the dataset. We save the names of the Properties as *codes of the provinces.*

```
d3.csv("data.csv", function(d) {
  return {
    date : d.Date,
    canada : +d.Canada,
    AB : +d.Alberta,
    BC : +d["British Columbia"],
    MB : +d.Manitoba,
    NB : +d["New Brunswick"],
    NL : +d["Newfoundland and Labrador"],
    NS : +d["Nova Scotia"],
    ON : +d.Ontario,
    SK : +d.Saskatchewan,
    QC : +d.Quebec
  };
}).then( function(d){
  console.log(d[0]);
});
```

D3.csv is asynchronous method so anything following this method will be executed before the loading of the data. Processing of the data must therefore be executed directly within the block we have so far used for the console output. This block will be executed exactly at the moment when the data are loaded. To be able to process the data comfortably, we will save the reference to the data in a variable. We will also create the function, which will serve the whole data visualization. The whole code will look as follows:

```
var data;

d3.csv("data.csv", function(d) {
  return {
    date : d.Date,
    canada : +d.Canada,
    AB : +d.Alberta,
    BC : +d["British Columbia"],
    MB : +d.Manitoba,
    NB : +d["New Brunswick"],
    NL : +d["Newfoundland and Labrador"],
    NS : +d["Nova Scotia"],
    ON : +d.Ontario,
    SK : +d.Saskatchewan,
    QC : +d.Quebec
  };
}).then(function(d){
 data = d;
 // data visualization
  visualization();
});

function visualization() {}
```

Subtasks of visualization

1) highlight selected province in the map;
2) after clicking on the province, a graph should appear – bar chart of all the values corresponding to the selected province in time;
3) in the bar chart, there is a vertical line denoting the selected week – by clicking in the graph the selection should change and all the provinces in the map should be also coloured by their value in the selected week;

4) after clicking in the graph the selected values should be also displayed in a „status bar“ (textual information about conde of the selected province, selected time point and the number of infected people in the selected province at given time point – if you want something extra, add also the full name of the province ☺);
5) display legend.

Create initial variables

To be able to co comfortably work with the svg panel with the map, we will save the reference to the panel in a variable canvas. In the same way we will create other variables – the number of data records and the currently selected province.

```
var dataEntriesCount = data.length; //number of records
var chosenState = null;

//variable with reference to map (for selection based on class use dot
notation)
var canvas = d3.select(".map");
```

Clicking on the province and its highlighting

To handle the clicking on arbitrary province, we will add an event handler to each group g using the method on. The province, on which the user clicks will be saved to variable **chosenState**. Next, we will need a function, which will execute the change of style of selected province.

```
function borderIt(obj){
      //remove border of previously selected state (for selection based on
      id use # notation)
      if(chosenState != null){
             d3.select("#"+chosenState).style("stroke-width", 0);
      }
      //add border to the selected state
      d3.select(obj).style("stroke", "yellow").style("stroke-width", 4);
}


canvas.selectAll("g").on("click", function(){
      // highlight the selected province
      borderIt(this);
      // find the attribute ID of the selected element
      chosenState = d3.select(this).attr('id');
      console.log(chosenState);
});
```

Access to data

To access the whole object, we will use:
`data[positionInArray]`

To access the value saved for the province, we will use:
`data[positionInArray][provinceCode]`

Value of Property for given object:
`Object[propertyName]`

Creating bar chart

Again, we will need couple of variables and we will also set up the svg panel for drawing the bar chart.

```
var t = 0; //time
var max = 0; //highest value

var ratio = 2.5;              // stretching ratio of the graph
var gh = 120;                 // height of the graph
var gw = ratio*dataEntriesCount; //width of the graph

// Setting up canvas of the graph
var graph = d3.select("body")
      .select(".graph")
      .attr("width",gw)
      .attr("height",gh);
```

Finding highest value in data

We will use the object access. We need to test, if we are not comparing the value with the date.

```
//Finding the highest value
data.forEach(function(object){
  for(var key in object) {
    if (key != "date") {
      if(object[key]>max){
        max = object[key];
      }
    }
  }
});
```

Binding the bar chart with the clicking on the province

For drawing the bar chart, we will create new function, which will handle the creation of the chart. For each rect in barchart we will also set the class, so we can then easily delete all rects, when selecting new province.

```
//create scale for computing bar height
var heightScale = d3.scaleLinear()
                        .domain([0, max])
                        .range([0,gh - 10]);

//Draw gaph
function graphIt(){
      graph.selectAll(".gData").remove();
      // Draw graph by selection
      for (var i = 0; i < dataEntriesCount; i++) {
            for(var key in data[i]) {
                  if (key == chosenState) {

                        var value = data[i][key];
                        var barHeight = heightScale(value);

                        graph.append("rect")
                              .attr("x", ratio*i)
                              .attr("y", gh-barHeight -10)
                              .attr("width", 2)
                              .attr("height", barHeight)
                              .attr("class", "gData")
                              .attr("i", i)
                              .attr("fill","white");
                  }
            }
      }
}
```

We then just add the command:
`graphIt();`
to the function, which handles the clicking on the province.

Handling the clicking on the bar chart

Similarly to clicking on the province, we add the event handler for clicking on bar chart using the on function. We save the position of the click to variable t. To gain the position, we use the method *d3.mouse(container)*, which returns the position of the click relative to the given container – in our case the whole bar chart. The coordinates are in the form of array with two elements.

```
graph.on('click', function () {
      t = parseInt(d3.mouse(this)[0]/ ratio);
});
```

Alternative way of accessing time coordinate (unlike previous method it will only work when clicking on the bars):

```
graph.selectAll(".gData").on('click', function () {
      t = d3.select(this).attr("i");
}
```

We will add the line object to the bar chart svg panel, so we can see which position in the graph we have selected.

```
graph.append("line")
            .attr("x1", t*ratio)
            .attr("y1", 0)
            .attr("x2", t*ratio)
            .attr("y2", gh)
            .attr("stroke-width", ratio)
            .attr("stroke", "silver");
```

The line will be drawn on the position of the click as an indicator of selected time.
We will change the line position in the function handling the clicking in the bar chart:

```
graph. on('click', function () {
      t = parseInt(d3.mouse(this)[0]/ ratio);
      graph.selectAll("line")
            .attr("x1", ratio*t + ratio/2)
            .attr("x2", ratio*t + ratio/2);

});
```

SVG gradient (for legend)

```
var gradient = canvas.append("defs")
    .append("linearGradient")
      .attr("id", "gradient")
      .attr("x1", "0%") //x,y coordinates define direction of gradient
      .attr("y1", "50%")
      .attr("x2", "100%")
      .attr("y2", "50%")
      .attr("spreadMethod", "pad"); //defines what happens outside 0-100%
range – the endpoind color is used (alternatives: repeat, reflect)

// define colors
  gradient.append("stop")
          .attr("offset", "0%") // position along the gradient vector
          .attr("stop-color", "black")
          .attr("stop-opacity", 1);
  gradient.append("stop")
          .attr("offset", "100%") // position along the gradient vector
          .attr("stop-color", "red")
          .attr("stop-opacity", 1);

  canvas.append("rect")
  .attr("width",200)
  .attr("height", 20)
  .attr("x", 10)
  .attr("y",460)
  .attr("stroke", "lightgray")
  .attr("stroke-width",2)
  .attr("fill", "url(#gradient)"); //use url with id notation to access it

…add labels indicating values
```

Colouring the province and barchart by the linear scale (black - red), adding the labels for the legend and status bar

Try alone

It would be also beneficial to add the scrolling with mouse wheel possibility to scroll over the time line, or buttons for finer, more continuous browsing of the neighbourhood of the selected time point. You can try this at home ☺