Part 1: Module coupling and cohesion

Chosen program's purpose

The program I've chosen is from Introduction to Programming II module, the case study is Data Visualization.

The basic idea for the program is to have a menu on the left for the user to click and navigate. Each section has a different visualization based on a certain data. The first 5 in the gallery are part of the template given, while we are free to modify to our abilities and likings, I mostly work on the two new visualizations, which use the same dataset but present itself in completely different ways.

Coupling

Content Coupling → p5 library

The definition of **content coupling** according to the lectures is *some or all contents of one module are included in the content of another module*.

In this case, the former module is the p5 library, and the latter is my program, the entire program. Some of the contents of the p5 module are included in the my program, including the main `p5.min.js` library for all the visualization as well as the `DOM` library for the menu mentioned above.

```
 9     <!-- Libraries -->
10     <script src="lib/p5.min.js"></script>
11     <script src="lib/p5.dom.min.js"></script>
```

Data/Input-Output Coupling → helper-function.js line 31-33

The definition of **data/input-output coupling** according to the lectures is *the output from one module serves as input to another module*.

This part of the code is part of the helper function come within the template we're given. In this case, the output from `row.getNum(i)` servers as input to `rowData.push()`.

```
23  function sliceRowNumbers (row, start=0, end) {
24    let rowData = [];
25
26    if (!end) {
27      // Parse all values until the end of the row.
28      end = row.arr.length;
29    }
30
31    for (i = start; i < end; i++) {
32      rowData.push(row.getNum(i));
33    }
34
35    return rowData;
36  }
```

Cohesion

Communicational Cohesion → Gallery

The definition of **communicational cohesion** according to the lectures is *the tasks performed by a software module use the same input data or contribute to producing the same output data*.

In this case, the “gallery”, which is the aforementioned menu, is taking the same type of data — classes in the same type — throughout. This is used to add each visualization to the menu so that the user can interact with it, and actually sees it on screen.

```
12    // Create a new gallery object.
13    gallery = new Gallery();
14
15    // Add the visualisation objects here.
16    gallery.addVisual(new TechDiversityRace());
17    gallery.addVisual(new TechDiversityGender());
18    gallery.addVisual(new PayGapByJob2017());
19    gallery.addVisual(new PayGapTimeSeries());
20    gallery.addVisual(new ClimateChange());
21    gallery.addVisual(new ParentViewOfSchoolBar());
22    gallery.addVisual(new ParentViewOfSchoolLine());
```

Logical Cohesion → Sketch.js

The definition of **logical cohesion** from lectures is *the tasks performed by a software module perform logically similar functions*.

In this case, the classes inside `sketch.js` perform logically similar functions, they all are classes and have similar properties and methods

Here’s the list of classes presented in the `sketch.js`, every class has a `this.name`, `this.id`, `this.loaded`, `this.preload()`, `this.setup()`, `this.draw()`, `this.destroy()`.

sketch.js

```
// Global variable to store the gallery object. The gallery object is
// a container for all the visualisations.
var gallery;

function setup() {
  // Create a canvas to fill the content div from index.html.
  canvasContainer = select('#app');
  var c = createCanvas(1400, 1000);
  c.parent('app');

  // Create a new gallery object.
  gallery = new Gallery();

  // Add the visualisation objects here.
  gallery.addVisual(new TechDiversityRace());
  gallery.addVisual(new TechDiversityGender());
  gallery.addVisual(new PayGapByJob2017());
  gallery.addVisual(new PayGapTimeSeries());
  gallery.addVisual(new ClimateChange());
  gallery.addVisual(new ParentViewOfSchoolBar());
  gallery.addVisual(new ParentViewOfSchoolLine());
}

function draw() {
  background(255);
  if (gallery.selectedVisual != null) {
    gallery.selectedVisual.draw();
  }
}
```

climate-change.js

```
function ClimateChange() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Climate Change';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'climate-change';

  // Names for each axis.
  this.xAxisLabel = 'year';
  this.yAxisLabel = '°C';

  var marginSize = 35;

  // Layout object to store all common plot layout parameters and
  // methods.
  this.layout = {···
  };

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
  this.preload = function() {···
  };

  this.setup = function() {···
  };

  this.destroy = function() {···
  };

  this.draw = function() {···
  };
```

tech-diversity-race.js

```
function TechDiversityRace() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Tech Diversity: Race';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'tech-diversity-race';

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
  this.preload = function() {···
  };

  this.setup = function() {···
  };

  this.destroy = function() {···
  };

  // Create a new pie chart object.
  this.pie = new PieChart(width / 2, height / 2, width * 0.4);

  this.draw = function() {···
  };
}
```

tech-diversity-gender.js

```
function TechDiversityGender() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Tech Diversity: Gender';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'tech-diversity-gender';

  // Layout object to store all common plot layout parameters and
  // methods.
  this.layout = {···
  };

  // Middle of the plot: for 50% line.
  this.midX = (this.layout.plotWidth() / 2) + this.layout.leftMargin;

  // Default visualisation colours.
  this.femaleColour = color(255, 0 ,0);
  this.maleColour = color(0, 255, 0);

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
  this.preload = function() {···
  };

  this.setup = function() {···
  };

  this.destroy = function() {
  };

  this.draw = function() {···
  };

  this.drawCategoryLabels = function() {···
  };
```

pay-gap-1997-2017.js

```
function PayGapTimeSeries() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Pay gap: 1997-2017';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'pay-gap-timeseries';

  // Title to display above the plot.
  this.title = 'Gender Pay Gap: Average difference between male and female pay.';

    // Names for each axis.
  this.xAxisLabel = 'year';
  this.yAxisLabel = '%';

  var marginSize = 35;

  // Layout object to store all common plot layout parameters and
  // methods.
  this.layout = {···
  };

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
  this.preload = function() {···
  };

  this.setup = function() {···
  };

  this.destroy = function() {
  };

  this.draw = function() {···
  };
```

pay-gap-by-job-2017.js

```
function PayGapByJob2017() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Pay gap by job: 2017';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'pay-gap-by-job-2017';

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Graph properties.
  this.pad = 20;
  this.dotSizeMin = 15;
  this.dotSizeMax = 40;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
  this.preload = function() {···
  };

  this.setup = function() {
  };

  this.destroy = function() {
  };

  this.draw = function() {···
  };

  this.addAxes = function () {···
  };
}
```

parentViewOfSchool(bar).js

```
function ParentViewOfSchoolBar() {
  this.name = "Parent's View of School (Bar)";

  this.id = "parent-view-of-school-bar";

  this.layout = { // where the axis are drawn···
  };

  this.legendColor = ["#FFC0CB", "#F0E68C", "#90EE90", "#AFEEEE", "#F4A460",
  "#FFD700",];
  this.barColor = ["#9370DB", "#52CDAA", "#FFB900", "#F52832", "#807E7E"];

  let xlabel = [];
  let q = [];
  let currentQ;
  let ans = [];

  this.loaded = false;

  this.preload = function() {···
  };

  this.setup = function() {···
  };

  this.destroy = function() {···
  };

  this.draw = function() {···
  };

  this.drawBar = function(q){···
  };

  this.legendItem = function(){···
  };

}
```

parentViewOfSchool(line).js

```
function ParentViewOfSchoolLine(){
    this.name = "Parent's View of School (Line)";

    this.id = 'parent-view-of-school-line';

    this.layout = { // where the axis are drawn···
    };

    this.legendColor = ["#FFC0CB", "#F0E68C", "#90EE90", "#AFEEEE", "#F4A460",
    "#FFD700",];
    this.barColor = ["#9370DB", "#52CDAA", "#FFB900", "#F52832", "#807E7E"];

    let xlabel = [];
    let q = [];
    let currentQ;
    let ans = [];

    this.loaded = false;

    this.preload = function() {···
    };

    this.setup = function() {···
    };

    this.destroy = function() {···
    };

    this.draw = function() {···
    };

    this.drawLine = function(q){···
    };

    this.legendItem = function(){···
    };

    this.lines = function(ans){···
    };
}
```