Part 1: Module coupling and cohesion

For this part, I picked the music visualization example program that we used in the ITP2 module. This program is built based on the principles of OOP and consists of the main program module, six additional modules, and two library files (a big collection of modules). Each original module defined by a constructor function that creates an object:

- "constrolsAndInputs" - the module that handles UI menus and controls
- "playbackButton" - the module that handles the playback button
- "visualisations" - the intermediary module that aggregates information from different visualization modules
- "needles", "spectrum", "wavepattern" - three modules that define different visualization patterns that drawn on the screen

This program is based on the p5 and p5sound js libraries and uses object constructors, functions, basic objects, arrays, variables, and methods that work on them.

For the first example of the module cohesion, we look at the "wavePattern" module. This is a small, simple, and clean module that consists of two elements: name property, which is mostly used as an identifier, and a draw method.

```
1  //draw the waveform to the screen
2  function WavePattern(){
3      //vis name
4      this.name = "wavepattern";
5
6      //draw the wave form to the screen
7  ›   this.draw = function(){ ···
27     };
28 }
29
```

This method handles the visualization of the sound data (variable wave, line 15) in the predetermined logic. Because this module tackles only one single task, it can be used as an example of "functional cohesion", one of the best ones.

```
6      //draw the wave form to the screen
7      this.draw = function(){
8          push();
9          noFill();
10         stroke(255, 0, 0);
11         strokeWeight(2);
12
13         beginShape();
14         //calculate the waveform from the fft.
15         var wave = fourier.waveform();
16         for (var i = 0; i < wave.length; i++){
17             //for each element of the waveform map it to screen
18             //coordinates and make a new vertex at the point.
19             var x = map(i, 0, wave.length, 0, width);
20             var y = map(wave[i], -1, 1, 0, height);
21
22             vertex(x, y);
23         }
24
25         endShape();
26         pop();
27     };
```

For the second example of the module cohesion, we inspect the "controlsAndInputs" module. It consists of a boolean variable "menuDisplayed", playback button object as an instance of the different module, methods "mousePressed" and "keyPressed" that handle user controls, and methods "menu" and "draw" visualize interface elements.

```
1  //Constructor function to handle the onscreen menu, keyboard and mouse
2  //controls
3  function ControlsAndInput(){
4
5      this.menuDisplayed = false;
6
7      //playback button displayed in the top left of the screen
8      this.playbackButton = new PlaybackButton();
9
10     //make the window fullscreen or revert to windowed
11 ›   this.mousePressed = function(){···
21     };
22
23     //responds to keyboard presses
24     //@param keycode the ascii code of the keypressed
25 ›   this.keyPressed = function(keycode){···
35     };
36
37     //draws the playback button and potentially the menu
38 ›   this.draw = function(){···
55     };
56
57 ›   this.menu = function(){···
64     };
65 }
```

All these different parts do different things and are brought together by the basic idea - to handle user controls. This general idea includes two distinct aspects: visualization and control. But even methods that are related to the same aspect ("mousePressed" and "keyPressed") do different things, despite all their similarities.

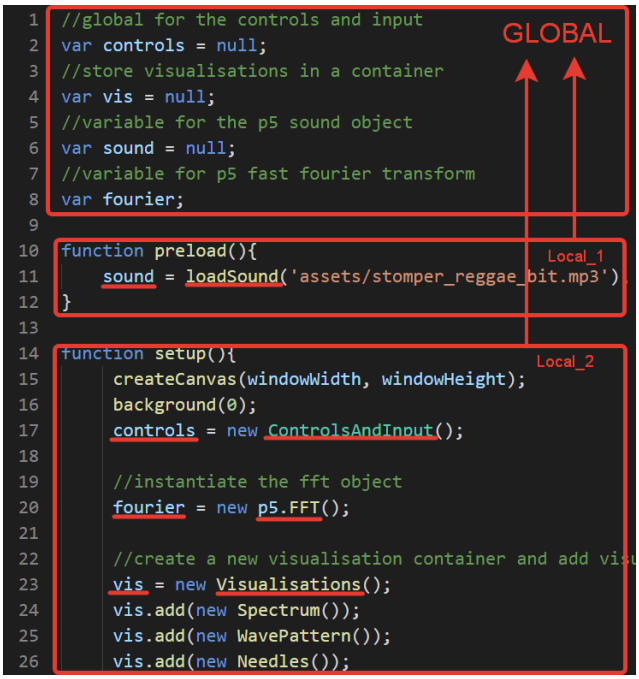
```
10     //make the window fullscreen or revert to windowed
11     this.mousePressed = function(){
12
13         //check if the playback button has been clicked
14         //if not make the visualisation fullscreen
15         if(!this.playbackButton.hitCheck())
16         {
17             let fs = fullscreen();
18             fullscreen(!fs);
19         }
20
21     };
22
23     //responds to keyboard presses
24     //@param keycode the ascii code of the keypressed
25     this.keyPressed = function(keycode){
26         console.log(keycode);
27         if(keycode == 32){
28             this.menuDisplayed = !this.menuDisplayed;
29         }
30
31         if(keycode > 48 && keycode < 58){
32             var visNumber = keycode - 49;
33             vis.selectVisual(vis.visuals[visNumber].name);
34         }
35     };
```

Because of this, I reason that as a whole, this module has "logical cohesion" - one of the less desirable ones.

My first example of the module coupling is the main program module "sketch". It combines all additional modules as well as modules from libraries "p5" and "p5sound" in one environment, so as a whole it can be a great example of the "common environment coupling".

```
//global for the controls and input
var controls = null;
//store visualisations in a container
var vis = null;
//variable for the p5 sound object
var sound = null;
//variable for p5 fast fourier transform
var fourier;

function preload(){ ···
}

function setup(){ ···
}

function draw(){ ···
}

function mouseClicked(){ ···
}

function keyPressed(){ ···
}

//when the window has been resized. Resize canvas to fit ···
function windowResized(){ ···
}
```

The big environment itself divided into several smaller ones ("preload", "setup", "draw", etc.) with encapsulated local scopes. All local scopes get access to the number of global variables. But the bad thing is - these global variables themselves are dependent on smaller modules, so we get a pretty tangled relation, which is not desirable.



As a more preferable type of module coupling, we can use a small instance inside the "sketch" program - setup function.

```
14  function setup(){
15      createCanvas(windowWidth, windowHeight);
16      background(0);
17      controls = new ControlsAndInput();
18
19      //instantiate the fft object
20      fourier = new p5.FFT();
21
22      //create a new visualisation container and add visualisations
23      vis = new Visualisations();
24      vis.add(new Spectrum());
25      vis.add(new WavePattern());
26      vis.add(new Needles());
27
28  }
```

On lines 15 and 17 we can see an example of "data coupling", both "createCanvas" and "background" functions (modules from the "p5" library) take arguments that they later process and use. It should be noted, that this data doesn't affect the control flow of these modules, which is associated with less favorable "control coupling".