Contents

1 Introduction

On a basic web page, we can construct and page element called a "canvas".
It needs a bit of CSS to see what we have done as it begins as an empty transparent rectangle on the page

```
<!DOCTYPE html>
<html lang="en">
<head> <meta charset="UTF-8"> <title>JavaScript</title>
    <style>
        body{ background-color: #aaaaaa; }
        canvas{
            background-color: #ffffff;
            width:800px;
            height:600px;}
    </style></head>
<body>
<canvas id='myCanvas'></canvas>
</body>
</html>
```

Listing 1: Basic Canvas

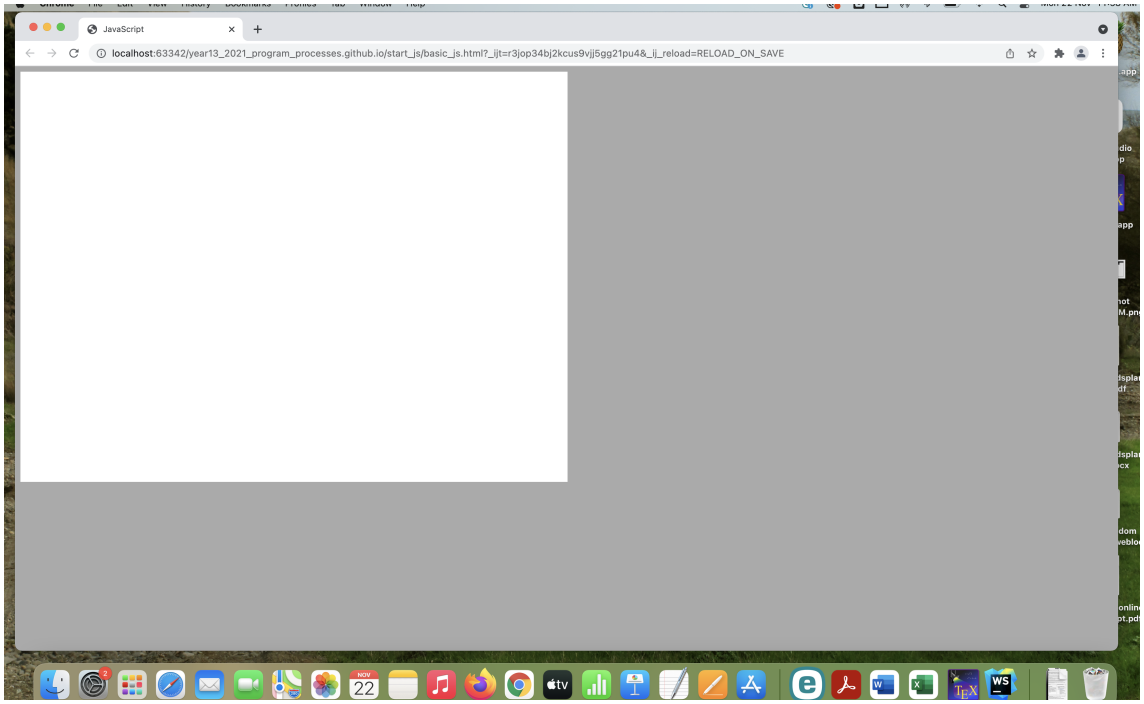


Figure 1: Canvas in browser

The canvas environment allows us to draw shapes and also run animations, so with a bit of work, we are able to create a small web applications such as games or other more interactively rich experiences. Our aim is to learn about this environment and to create a small drawing/painting program.

2 Getting started

To be able to do thing to the canvas we use JavaScript. This is a programming language and in general has many similarities to Python but has some differences in syntax.
JavaScript can be written in a web page within the < script > tag.
It can also be loaded from an external file (which we will do for most of the unit). We will have to learn quite a few things about JavaScript as we go through this unit, but regarding drawing shapes, it will always follow the same basic set of steps.

- begin a path (i.e the shape)
- define the path with its associated parameters (x,y , width, height, ..)
- define the fill, stroke and line width (do not need all)
- tell the canvas to fill and/or stroke the path

```
<!DOCTYPE html>
<html lang="en">
<head> <meta charset="UTF-8"> <title>JavaScript</title>
    <style>
        body{ background-color: #aaaaaa; }
        canvas{
            background-color: #ffffff;
            width:800px;
            height:600px;}
    </style></head>
<body>
<canvas id='myCanvas'></canvas>
<script>
    // get the canvas element using the id name
    canvas = document.querySelector('#myCanvas');
    // define a 2d context and associate it with the variable ctx
    // all canvas commands will require the ctx.
    let ctx = canvas.getContext('2d');
    // specify height and width of canvas (which should be identical to the CSS
    let width = 800;
    let height = 600;
    canvas.width = width;
    canvas.height = height;
```

```
24
25      // draw some things
26      // rectangle
27      // start the path then define as a rectangle with parameters
28      ctx.beginPath();
29      ctx.rect(10,10,100,100);
30      // set the context
31      ctx.fillStyle='rgb(0,153,204)';
32      ctx.strokeStyle='rgb(0,0,0)';
33      ctx.lineWidth=10;
34      // actually fill and stroke
35      ctx.stroke();
36      ctx.fill();
37      // the following follow exactly the same pattern
38      // circle
39      ctx.beginPath();
40      ctx.arc(200,60, 50, 0, 2*Math.PI);
41      ctx.fillStyle='rgb(255,204,51)';
42      ctx.strokeStyle='rgb(51,51,255)';
43      ctx.lineWidth=10;
44      ctx.stroke();
45      ctx.fill();
46      // line
47      ctx.beginPath();
48      // set start point of the line
49      ctx.moveTo(0, 200);
50      // set next point of the line
51      ctx.lineTo(750,200);
52      ctx.strokeStyle="rgb(255,0,0)";
53      ctx.lineWidth=1;
54      ctx.stroke();
55
56      // rectangle with a gradient fill
57
58      ctx.beginPath()
59      ctx.rect(10,350, 200,200);
60      let my_gradient=ctx.createLinearGradient(10,350,10,550);
61      my_gradient.addColorStop(0,"rgb(255,102,102)");
62      my_gradient.addColorStop(0.5,"rgb(255,255,153)");
63      my_gradient.addColorStop(1,"rgb(0,153,204)");
64      ctx.fillStyle=my_gradient;
65      ctx.fill();
66      // note that the stroke picks up the previous context
67      ctx.stroke();
68
69      // quadratic curves (bezier)
70      ctx.strokeStyle="rgb(255,0,0)";
```

```
    ctx.beginPath();
    ctx.moveTo(300,400);
    ctx.lineWidth=10;
    ctx.quadraticCurveTo(500, 550, 700, 400);
    ctx.lineCap = "round";
    ctx.stroke();


    // add text, set the context then fill the text
    ctx.fillStyle="rgb(0,0,255)";
    // shorthand css to set basic options
    let myFont= "bold 30px monospace";
    ctx.font=myFont;
    ctx.fillText("Hello World", 300,50);

    // images can be placed on the canvas but we need to know if they have loaded
    // there other ways of dealing with this
    let img = new Image();
    img.onload = function(e){
        let img_h = img.height;
        let img_w= img.width
        console.log(e);
        ctx.drawImage(img, 500,220, img_w/4, img_h/4);
    }
    img.src= "red_kangaroo.jpeg"
</script>
</body>
</html>
```

Listing 2: Basic Shapes

This just gives a "general idea" and more complete references:
https://www.w3schools.com/html/html5_canvas.asp
https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

3 JavaScript in an external file

It will become quite cumbersome having the JavaScript in the same file as the HTML page.
So it is better to have it as a separate file that is linked in.
This may make some of the code easier to re-use, as well.

```
<canvas id='myCanvas'></canvas>
<script type="text/javascript" src="basic.js"></script>
```

Listing 3: Loading javascript

4 Improving the set up

It is also helpful to add to the code that connects to that initialises the canvas.
The code below can be discussed further, but it allows us the manage the size and basic styling of the canvas from within the JavaScript.

```
canvas = document.querySelector('#myCanvas');
let ctx = canvas.getContext('2d');
// define width and height
let width = 1000;
let height = 500;
// define scale of 1. This may be changed later to improve resolution
let scale = 2;
// set the canvas width and height
canvas.width = width*scale;
canvas.height = height*scale;
// scale the canvas
ctx.scale(scale,scale);
// get the canvas element
// style it here so it will be consistent
let my_c = document.getElementById('myCanvas');
my_c.style.backgroundColor = "rgb(100,100,100)"
my_c.style.width = width+"px";
my_c.style.height = height+"px";
my_c.style.border = "6px solid rgba(200,200,200,0.5)";
my_c.style.display = "block";
my_c.style.margin = "auto";
document.body.style.backgroundColor = "rgb(190,190,190)";
```

Listing 4: Initial JS

5 Functions

We can immediately see that even drawing a small number of shapes starts tp build up code, so we want to look at ways of reducing this and having as much as possible available for "re-use". Let's look at designing a rectangle function and see what we can do with it.

Picture of design

```
/**
 * Draw a rectangle
 *
 * @param {number} x corner x
 * @param {number} y corner y
 * @param {number} w width
 * @param {number} h height
 * @param {string} fillColour rgb string
 * @param {string} strokeColour rgb string.
 * @param {number} strokeWidth x coordinate of second point.
 * @return {null}
 */
function drawRect(x,y,w,h, fillColour, strokeColour, strokeWidth){
    ctx.fillStyle = fillColour;
    ctx.strokeStyle = strokeColour;
    ctx.lineWidth = strokeWidth;
    ctx.beginPath()
    ctx.rect(x,y,w,h)
    ctx.fill();
    ctx.stroke();
```

```
}

drawRect(20,20,200,130, "rgb(232,109,135)", "rgb(236,198,76)", 2);
drawRect(20,170,200,130, "rgba(234,225,144,0.83)", "rgb(80,0,80)", 4);
drawRect(20,320,200,130, "rgba(239,89,217,0.83)", "rgb(46,169,239)", 4);
drawRect(240,20,700,430, "rgba(17,96,239,0.83)", "rgb(46,169,239)", 4);

//loop
for(let i = 0; i<26; i++){
    drawRect(260+25*i,40,20,20, "rgba(170,193,238,0.83)", "rgb(46,169,239)", 1);
}
// double loop for a grid
for(let i=0; i<5 ; i++){
    for(let j = 0; j<5; j++){
        drawRect(260+25*i,100+25*j,20,20, "rgba(170,193,238,0.83)", "rgb
   (46,169,239)", 1);
    }
}
```

Listing 5: Python example

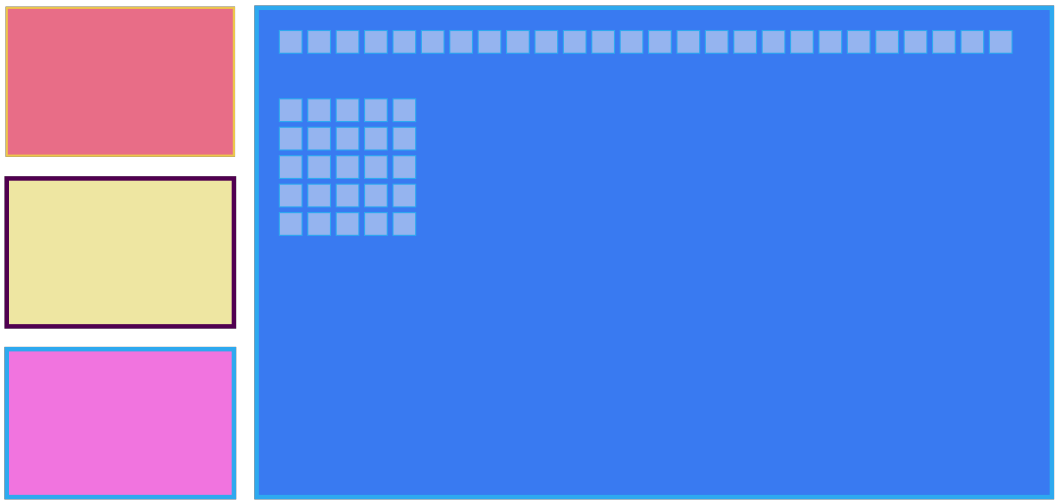


Figure 2: Canvas in browser

Design Functions for:

- Circle
- Line
- Triangle
- Square
- A Grid

- Text Box

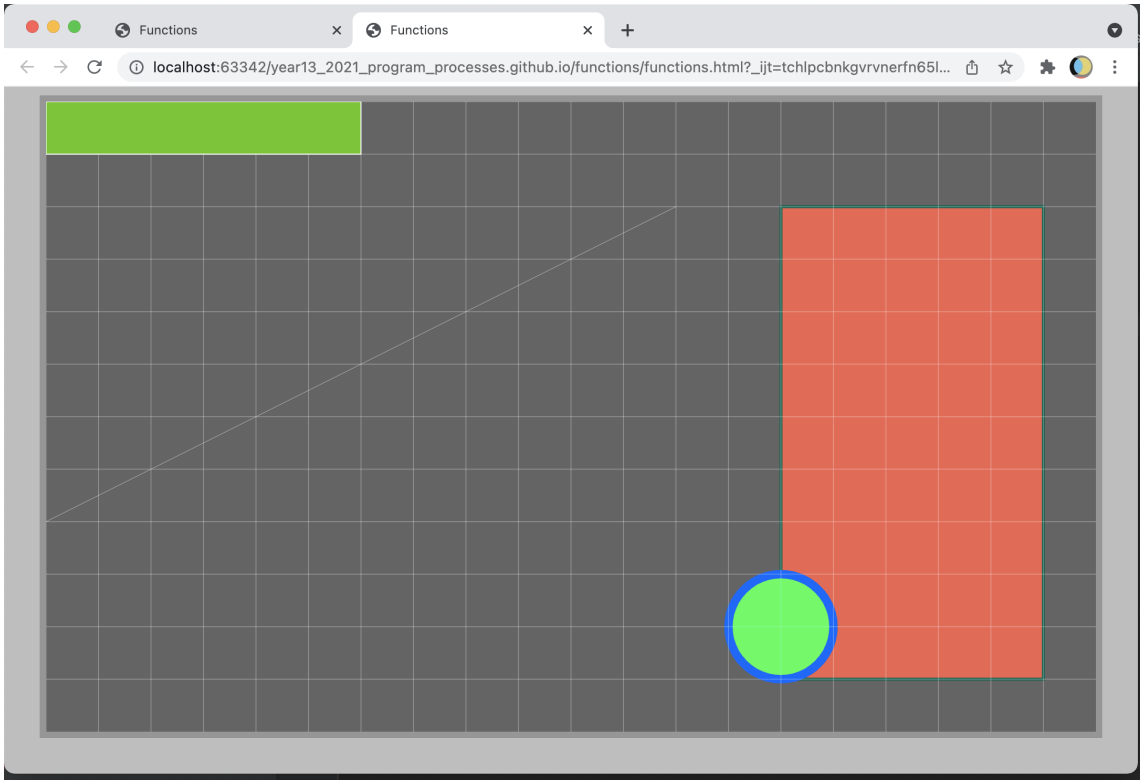


Figure 3: Canvas in browser

6 Making a grid

7 Update Context Function

We want to reduce code repetition as much as much as possible.
It might be an idea to have a small function to manage the fill and stroke commands.// This 'update context' function can sit near the top of the code and then be called from each of the shape drawing functions.

```
function updateContext(fillColour, strokeColour, strokeWidth){
    ctx.fillStyle = fillColour;
    ctx.strokeStyle = strokeColour;
    ctx.lineWidth = strokeWidth;
    if(fillColour){
        ctx.fill();
    }
    if(strokeColour|| strokeWidth){
        ctx.stroke();
    }
}
/**
 * Draw a rectangle
 *
 * @param {number} x corner x
 * @param {number} y corner y
 * @param {number} w width
 * @param {number} h height
 * @param {string} fillColour rgb string
 * @param {string} strokeColour rgb string.
 * @param {number} strokeWidth
 * @return {null}
 */
function drawRectangle(x,y,w,h, fillColour, strokeColour, strokeWidth){
    ctx.beginPath();
    ctx.rect(x,y,w,h);
    updateContext(fillColour, strokeColour, strokeWidth)
}
```

Listing 6: Loading javascript

8 Colour management

Rather than writing the rgb strings all of the time, it might be helpful to start a little colour management data structure.

The structure below is actually a simple JavaScript object. But at this stage we can treat much like a Python dictionary.

It has a keyword and an associated value.

```
// two dimensional array of colours
const col= [
[ // opaque
// black (0)                grey (1)                white (2)
"rgba(0,0,0,1)" , "rgba(150,150,150,1)", "rgba(255,255,255,1)" ,
// pink  (3)            purple (4)        deep blue (5)
"rgb(243,92,155,1)", "rgb(153,19,206,1)", "rgb(16,16,162,1)",
// pale blue (6)            yellow   (7)          bright yellow (7)
"rgba(135,211,243,1)", "rgba(246,244,193,1)", "rgba(250,250,0,1)"
],
[ // semi-transparent
// black (0)                grey (1)                white (2)
"rgba(0,0,0,0.5)" , "rgba(150,150,150,0.5)", "rgba(255,255,255,0.5)" ,
// pink  (3)            purple (4)        deep blue (5)
"rgb(243,92,155,0.5)", "rgb(153,19,206,0.5)", "rgb(16,16,162,0.5)",
// pale blue (6)            yellow   (7)          bright yellow (7)
"rgba(135,211,243,1,0.5)", "rgba(246,244,193,1,0.5)", "rgba(250,250,0,0.5)"
]
    ]
```

We can see its use in the function calls below

```
 * Draw a rectangle
 *
 * @param {number} x corner x
 * @param {number} y corner y
```

9 Organising an init file

We probably want to use the functions in various projects (subject to modifications).
We also have the set up code.
It might be a good idea to assemble this into a separate "initialisation" file (aka init.js). We could then have a separate file that does the "doing" of the program. We can load the files separately into the html document.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Functions Collection</title>
</head>
<body>
<canvas id='myCanvas'></canvas>
<script type="text/javascript" src="init.js"> </script>
<script type="text/javascript" src="interface.js"> </script>
</body>
</html>
```

Listing 7: Loading javascript

```
drawRectangle(20,20,200,130, col[0][3], undefined, undefined);
drawRectangle(20,170,200,130, undefined, col[1][6], 9);
drawRectangle(20,320,200,130, col[0][3], col[0][6], 4);
drawRectangle(240,20,700,430,undefined,col[0][6], 4);

drawCircle(590, 235, 100, col[0][7], col[1][0], 10)

drawLine(240,450, 940,20, col[0][2], 1);
drawLine(240,20, 940,450, col[0][2], 1);
let x= 50
let space = 300
drawTextBox(x, 480, 260,"Button One", col[0][2],col[0][5]);
drawTextBox(x+space, 480, 260,"Button Two", col[0][2],col[0][5]);
drawTextBox(x+2*space, 480, 260,"Button Three", col[0][2],col[0][5]);
```

Listing 8: Loading javascript

```
canvas = document.querySelector('#myCanvas');
let ctx = canvas.getContext('2d');
// define width and height
let width = 1000;
let height = 600;
// define scale of 1. This may be changed later to improve resolution
```

```
let scale = 2;
// set the canvas width and height
canvas.width = width*scale;
canvas.height = height*scale;
// scale the canvas
ctx.scale(scale,scale);
// get the canvas element
// style it here so it will be consistent
let my_c = document.getElementById('myCanvas');
my_c.style.backgroundColor = "rgb(100,100,100)"
my_c.style.width = width+"px";
my_c.style.height = height+"px";
my_c.style.border = "6px solid rgba(200,200,200,0.5)";
my_c.style.display = "block";
my_c.style.margin = "auto";
document.body.style.backgroundColor = "rgb(190,190,190)";


// two dimensional array of colours
const col= [
    [ // opaque
// black (0)               grey (1)               white (2)
        "rgba(0,0,0,1)" , "rgba(150,150,150,1)", "rgba(255,255,255,1)" ,
// pink  (3)           purple (4)       deep blue (5)
        "rgb(243,92,155,1)", "rgb(153,19,206,1)", "rgb(16,16,162,1)",
// pale blue (6)           yellow   (7)         bright yellow (7)
        "rgba(135,211,243,1)", "rgba(246,244,193,1)", "rgba(250,250,0,1)"
    ],
    [ // semi-transparent
// black (0)               grey (1)               white (2)
        "rgba(0,0,0,0.5)" , "rgba(150,150,150,0.5)", "rgba(255,255,255,0.5)" ,
// pink  (3)           purple (4)       deep blue (5)
        "rgb(243,92,155,0.5)", "rgb(153,19,206,0.5)", "rgb(16,16,162,0.5)",
// pale blue (6)           yellow   (7)         bright yellow (7)
        "rgba(135,211,243,0.5)", "rgba(246,244,193,0.5)", "rgba(250,250,0,0.5)"
    ]
]
/**
 * Fill and or Stroke the Current Path
 *
 * @param {string} fillColour rgb string
 * @param {string} strokeColour rgb string.
 * @param {number} strokeWidth
 * @return {null}
 */
function updateContext(fillColour, strokeColour, strokeWidth){
    ctx.fillStyle = fillColour;
```

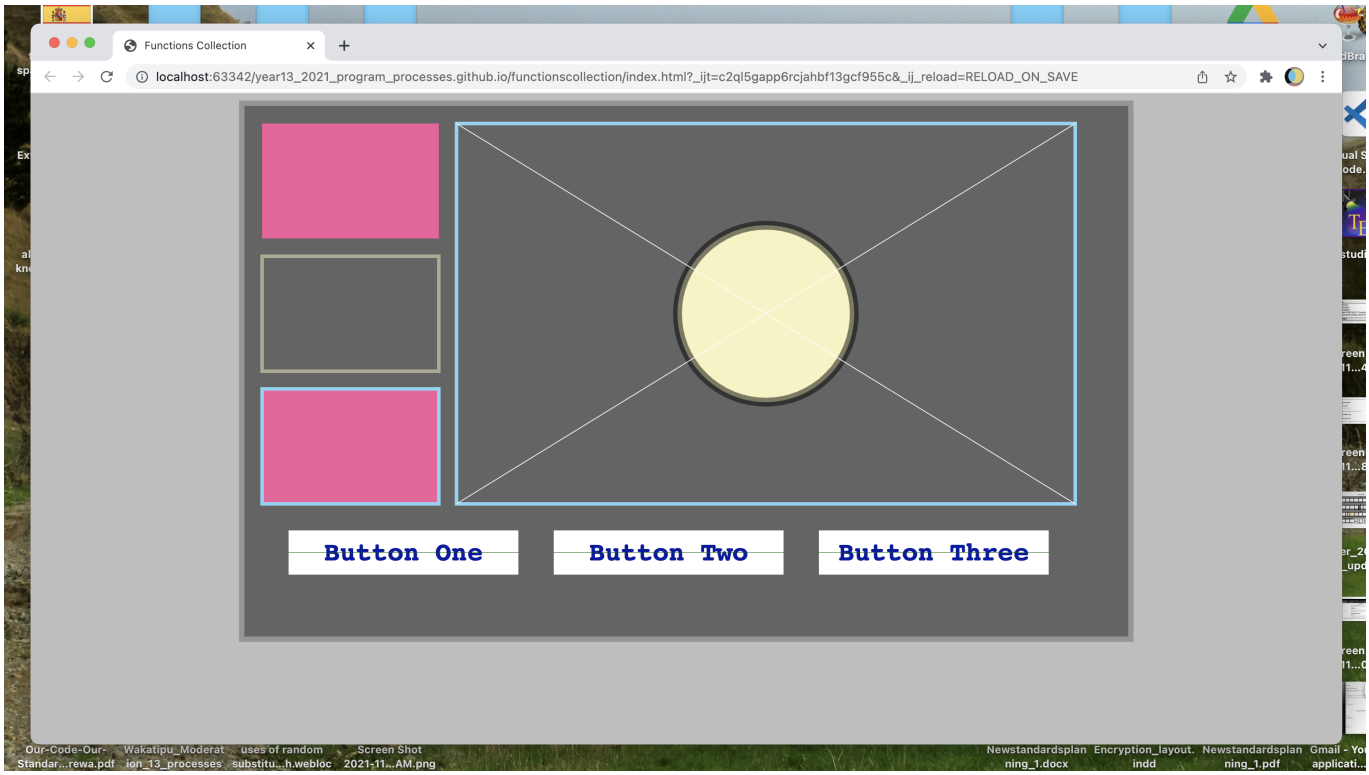
```
    ctx.strokeStyle = strokeColour;
    ctx.lineWidth = strokeWidth;
    if(fillColour){
        ctx.fill();
    }
    if(strokeColour|| strokeWidth){
        ctx.stroke();
    }
}
/**
 * Draw a rectangle
 *
 * @param {number} x corner x
 * @param {number} y corner y
 * @param {number} w width
 * @param {number} h height
 * @param {string} fillColour rgb string
 * @param {string} strokeColour rgb string.
 * @param {number} strokeWidth
 * @return {null}
 */
function drawRectangle(x,y,w,h, fillColour, strokeColour, strokeWidth){
    ctx.beginPath();
    ctx.rect(x,y,w,h);
    updateContext(fillColour, strokeColour, strokeWidth)
}
/**
 * Draw a circle
 *
 * @param {number} x centre x
 * @param {number} y centre y
 * @param {number} r radius
 * @param {string} fillColour rgb string
 * @param {string} strokeColour rgb string.
 * @param {number} strokeWidth
 * @return {null}
 */
function drawCircle(x,y,r,  fillColour, strokeColour, strokeWidth){
    ctx.beginPath();
    ctx.arc(x,y,r,0,2*Math.PI);
    updateContext(fillColour, strokeColour, strokeWidth)
}
/**
 * Draw a line
 *
    * @param {number} x_1 start x
    * @param {number} y_1 start y
```

```
    * @param {number} x_2 end x
    * @param {number} y_2 end y
    * @param {string} strokeColour rgb string.
    * @param {number} strokeWidth
    * @return {null}
 */
function drawLine(x_1,y_1, x_2, y_2, strokeColour,strokeWidth){
    ctx.beginPath();
    ctx.moveTo(x_1,y_1);
    ctx.lineTo(x_2,y_2);
    ctx.lineCap = "round";
    updateContext(undefined, strokeColour, strokeWidth);
}
/**
 * Draw text
    *
    * @param {number} x top corner x
    * @param {number} y top corner y
    * @param {string} txt
    * @param {string} fillColour rgb string.
    * @param {string} font css shorthand font style
    * @return {null}
 */
function drawText(txt ,x,y,fillColour, font = "bold 30px monospace" ) {
    ctx.font = font;
    ctx.fillStyle = fillColour;
    ctx.fillText(txt, x,y);
}
/**
 * Draw text box
    *
    * @param {number} x top corner x
    * @param {number} y top corner y
    * @param {number} w width
    * @param {string} txt
    * @param {string} backColour rgb string.
    * @param {string} fillColour rgb string.
    * @param {string} font css shorthand font style
    * @return {null}
 */
function drawTextBox(x,y,w,txt,backColour, fillColour, font = "bold 30px
   monospace"){
    let h = 50;
    drawRectangle(x,y,w,h, backColour, undefined, undefined)

    drawLine(x,y+h/2,x+w,y+h/2, col[0][4],1);
    ctx.textAlign = "center";
```

```
147     ctx.textBaseline = "middle";
148     drawText(txt, x+w/2,y+h/2, fillColour);
149 }
```

Listing 9: Loading javascript



10 Rotations

11 Objects

We are intending to introduce animation into our canvas environment.
To do so we will need to create objects.
At present, it will appear to serve no new purpose, so please be patient. Just about everything we make will be object based from now on.
Diagram

```
// init file with canvas set up
// colours
// functions
//...
```

Listing 10: Init file (not shown)

```
/**
 * Filled Ball
 * @param {number} x top corner of bounding box
 * @param {number} y top corner of bounding box
 * @param {string} fill fill colour
 * @param {string} stroke stroke colour
 * @param {number} strokeWidth  width of outline
 */
class Ball{
    constructor(x,y,r,fill, stroke, strokeWidth){
        this.x = x;
        this.y = y;
        this.r=r;
        this.fill=fill;
        this.stroke = stroke;
        this.strokeWidth = strokeWidth;
    }
    update(){
        this.draw()
    }
    draw(){
        this.drawCircle(this.x, this.y, this.r, this.fill, this.stroke, this.
   strokeWidth)
    }
}
Ball.prototype.drawCircle = drawCircle
Ball.prototype.updateContext = updateContext

/**
 * Filled Rectangle
 * @param {number} x top corner of bounding box
```

```
 * @param {number} y top corner of bounding box
 * @param {number} w width
 * @param {number} h height
 * @param {string} fill fill colour
 * @param {string} stroke stroke colour
 * @param {number} strokeWidth  width of outline
 */
class Rectangle{
    constructor(x,y,w,h,fill, stroke, strokeWidth){
        this.x = x;
        this.y = y;
        this.w=w;
        this.h=h;
        this.fill=fill;
        this.stroke = stroke;
        this.strokeWidth = strokeWidth;
    }
    update(){
        this.draw()
    }
    draw(){
        this.drawRectangle(this.x, this.y, this.w, this.h, this.fill, this.stroke
    , this.strokeWidth)
    }
}
Rectangle.prototype.drawRectangle = drawRectangle
Rectangle.prototype.updateContext = updateContext
/**
 * Filled TextBox
 * @param {number} x top corner of bounding box
 * @param {number} y top corner of bounding box
 * @param {number} w width
 * @param {string} txt text
 * @param {string} fill fill colour
 * @param {string} txtColour colour of text
 */
class TextBox{
    constructor(x,y,width,txt, fillColour, txtColour) {
        this.x = x;
        this.y = y;
        this.w = width;
        this.txt = txt;
        this.fillColour = fillColour;
        this.txtColour = txtColour;
    }
    update(txt){
        this.txt = txt
```

```
77          this.draw()
78      }
79
80      draw(){
81          this.drawTextBox(this.x, this.y,this.w,this.txt, this.fillColour, this.
    txtColour)
82
83      }
84  }
85  TextBox.prototype.drawTextBox = drawTextBox;
86
87
88  /**
89   * Grid - square grid
90   * @param {number} w width of canvas
91   * @param {number} h height of canvas
92   * @param {number} intervalWidth height of canvas
93   * @param {string} strokeColour stroke colour
94   * @param {number} strokeWidth  width of outline
95   */
96  class Grid{
97      constructor(w,h,intervalWidth, strokeColour, strokeWidth){
98          this.w = w;
99          this.h = h;
100         this.intervalWidth=intervalWidth;
101         this.strokeColour = strokeColour;
102         this.strokeWidth = strokeWidth;
103     }
104     update(){
105         this.draw()
106     }
107     draw(){
108
109         for(let i = -this.w ; i <= this.w ; i+= this.intervalWidth){
110
111             this.drawLine(i,-this.h, i,this.h, this.strokeColour, this.
    strokeWidth);
112         }
113         for(let j = -this.h ; j <= this.h ; j+= this.intervalWidth){
114             this.drawLine(-this.w,j, this.w,j, this.strokeColour, this.
    strokeWidth);
115         }
116         this.drawCircle(0,0,20,undefined,col[0][4], 5);
117         this.drawLine(-this.w,0, this.w, 0, col[0][4], 4);
118         this.drawLine(0, -this.h, 0, this.h, col[0][4], 4);
119     }
120 }
```

```
Grid.prototype.drawCircle = drawCircle
Grid.prototype.drawLine = drawLine
Grid.prototype.updateContext = updateContext
```

Listing 11: objectSet file

```
<canvas id='myCanvas'></canvas>
<script type="text/javascript" src="init.js"> </script>
<script type="text/javascript" src="objectSet.js"> </script>
<script>



    let G= new Grid(width, height, 100, col[0][2], 0.5);
    let R = new Rectangle(-50,-25,100,50, col[0][6], undefined, undefined)

    G.update();
    ctx.save();
    ctx.translate(500,300);
    ctx.rotate(60*Math.PI/180);
    //G.update();
    R.update();
    ctx.restore()

    ballSet = []
    let r = 20
    let x = 100
    let y = 400
    let space = 5*r
    for(let i= 0; i< col[0].length; i++){
        let temp = new Ball(x+ space*i, y, r , col[0][i], col[0][5], 2)
        ballSet.push(temp)

    }
    console.log(ballSet)
    for(let j=0; j<ballSet.length; j++){
        ballSet[j].update()
    }

    let T = new TextBox(100,100,150, "", col[0][1], col[0][2])
    T.update("5")
```

Listing 12: Implementing (using HTML file)

12 Animation Frame

Set up code

```
// init file with canvas set up
// colours
// functions
//...
canvas = document.querySelector('#myCanvas');
let ctx = canvas.getContext('2d');
// define width and height
let width = 1000;
let height = 600;
// define scale of 1. This may be changed later to improve resolution
let scale = 2;
// set the canvas width and height
canvas.width = width*scale;
canvas.height = height*scale;
// scale the canvas
ctx.scale(scale,scale);
// get the canvas element
// style it here so it will be consistent
let my_c = document.getElementById('myCanvas');
my_c.style.backgroundColor = "rgb(100,100,100)"
my_c.style.width = width+"px";
my_c.style.height = height+"px";
my_c.style.border = "6px solid rgba(200,200,200,0.5)";
my_c.style.display = "block";
my_c.style.margin = "auto";
document.body.style.backgroundColor = "rgb(190,190,190)";


// two dimensional array of colours
const col= [
    [ // opaque
// black (0)               grey (1)               white (2)
        "rgba(0,0,0,1)" , "rgba(150,150,150,1)", "rgba(255,255,255,1)" ,
// pink  (3)           purple (4)       deep blue (5)
        "rgb(243,92,155,1)", "rgb(153,19,206,1)", "rgb(16,16,162,1)",
// pale blue (6)           yellow   (7)         bright yellow (7)
        "rgba(135,211,243,1)", "rgba(246,244,193,1)", "rgba(250,250,0,1)"
    ],
    [ // semi-transparent
// black (0)               grey (1)               white (2)
        "rgba(0,0,0,0.5)" , "rgba(150,150,150,0.5)", "rgba(255,255,255,0.5)" ,
// pink  (3)           purple (4)       deep blue (5)
        "rgb(243,92,155,0.5)", "rgb(153,19,206,0.5)", "rgb(16,16,162,0.5)",
```

```
// pale blue (6)            yellow   (7)          bright yellow (7)
        "rgba(135,211,243,0.5)", "rgba(246,244,193,0.5)", "rgba(250,250,0,0.5)"
    ]
]
```

Listing 13: init

```
/**
 * Grid - square grid
 * @param {number} w width of canvas
 * @param {number} h height of canvas
 * @param {number} intervalWidth distance each grid unit
 * @param {string} strokeColour stroke colour
 * @param {number} strokeWidth  width of outline
 */
class Grid{
    constructor(w,h,intervalWidth, strokeColour, strokeWidth){
        this.w = w;
        this.h = h;
        this.intervalWidth=intervalWidth;
        this.strokeColour = strokeColour;
        this.strokeWidth = strokeWidth;
    }
    update(){
        this.draw()
    }
    draw(){
        for(let i = -this.w ; i <= this.w ; i+= this.intervalWidth){
            this.drawLine(i,-this.h, i,this.h, this.strokeColour, this.
    strokeWidth);
        }

        for(let j = -this.h ; j <= this.h ; j+= this.intervalWidth){
            this.drawLine(-this.w,j, this.w,j, this.strokeColour, this.
    strokeWidth);
        }

    }

    drawLine(x_1,y_1, x_2, y_2, strokeColour,strokeWidth){
        ctx.beginPath();
        ctx.moveTo(x_1,y_1);
        ctx.lineTo(x_2,y_2);
        ctx.lineCap = "round";
        ctx.strokeStyle = strokeColour;
        ctx.lineWidth = strokeWidth;
        ctx .stroke()
    }
```

```
}

/**
 * Filled TextBox
 * @param {number} x top corner of bounding box
 * @param {number} y top corner of bounding box
 * @param {number} w width
 * @param {string} txt text
 * @param {string} fill fill colour
 * @param {string} txtColour colour of text
 */
class TextBox{
    constructor(x,y,width, fillColour, txtColour) {
        this.x = x;
        this.y = y;
        this.w = width;
        // fixed height
        this.h = 50;
        // text managed through update
        this.txt = "Placeholder";
        console.log(this.txt)
        this.fillColour = fillColour;
        this.txtColour = txtColour;
    }
    update(txt ="Placeholder"){
        this.txt = txt
        this.draw()
    }

    draw(){
        ctx.beginPath();
        ctx.rect(this.x,this.y,this.w,this.h);
        ctx.fillStyle= this.fillColour;
        ctx.fill();
        ctx.font = "20px monospace";
        ctx.textAlign = "center";
        ctx.textBaseline = "middle";
        ctx.fillStyle = this.txtColour;
        ctx.fillText(this.txt, this.x+this.w/2, this.y+this.h/2);
    }
}
```

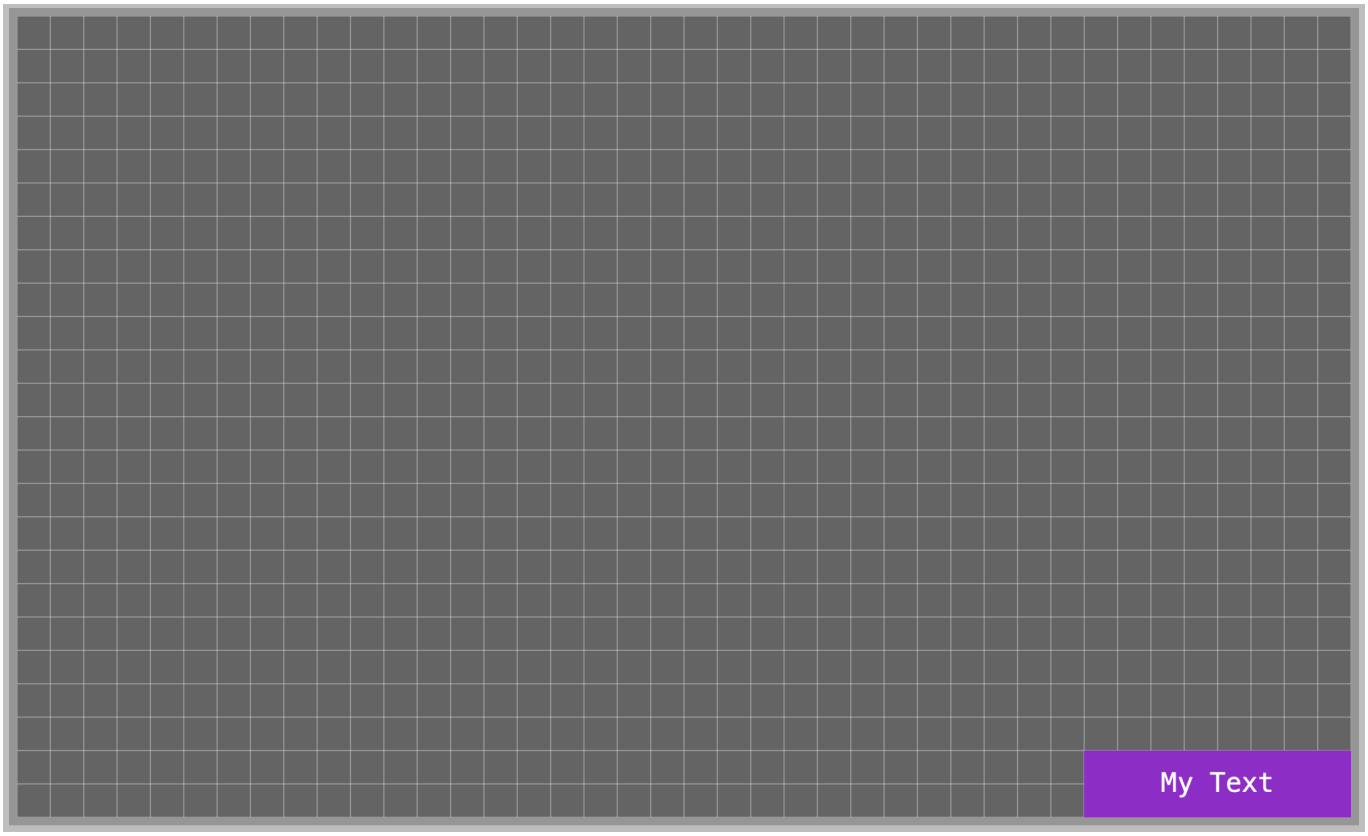
Listing 14: objects

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Objects Start</title>
</head>
<body>
<canvas id='myCanvas'></canvas>
<script type="text/javascript" src="init.js"> </script>
<script type="text/javascript" src="objects.js"> </script>
<script>
  let G = new Grid(width, height, 25, col[0][2], 0.3)
  let T = new TextBox(800,550,200, col[0][4], col[0][2])
  G.update();
  // note that the text is set through the update function call
  T.update("My Text");

</script>
</body>
</html>
```

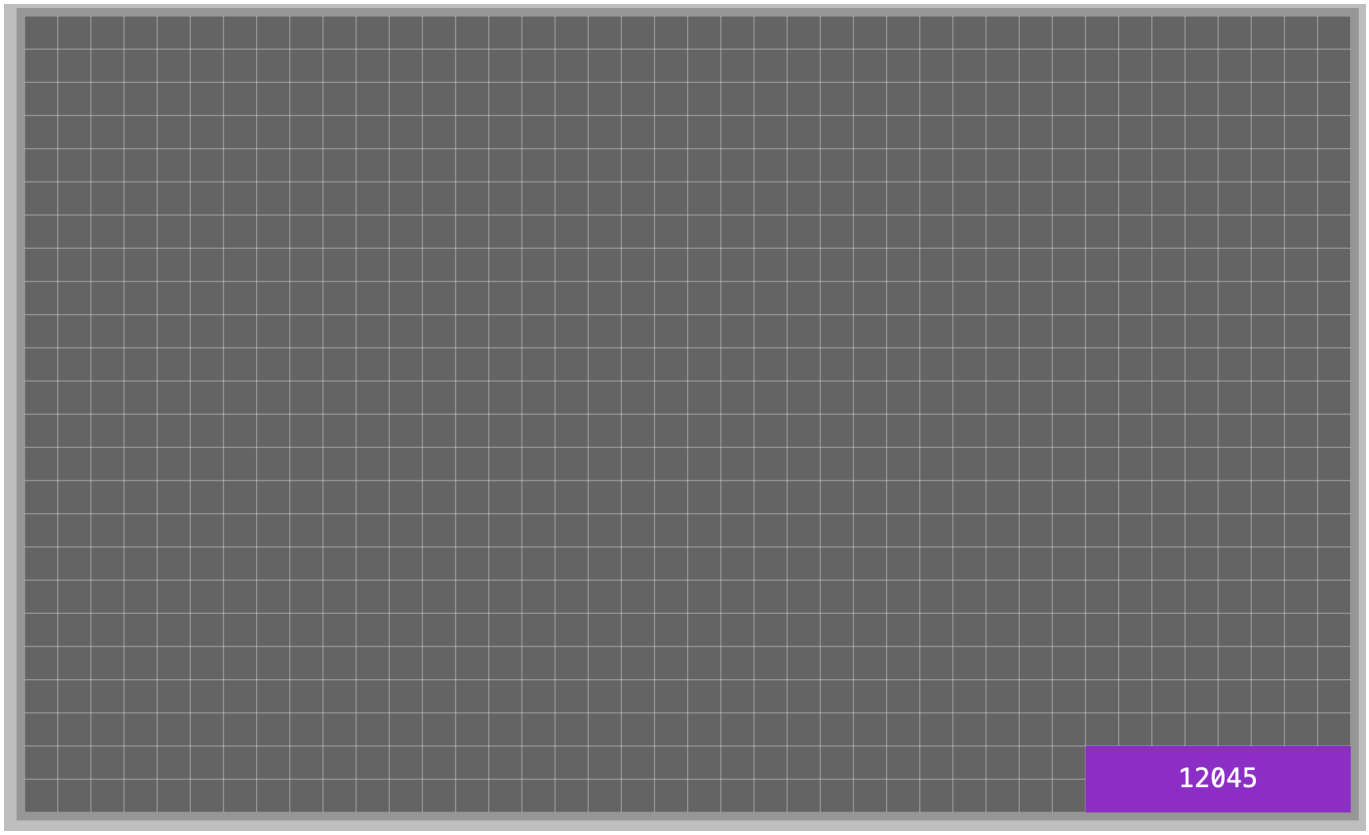
Listing 15: index




```
<script>
  let G = new Grid(width, height, 25, col[0][2], 0.3)
  let T = new TextBox(800,550,200, col[0][4], col[0][2])
  // create an animation function
  function animate(t){
```

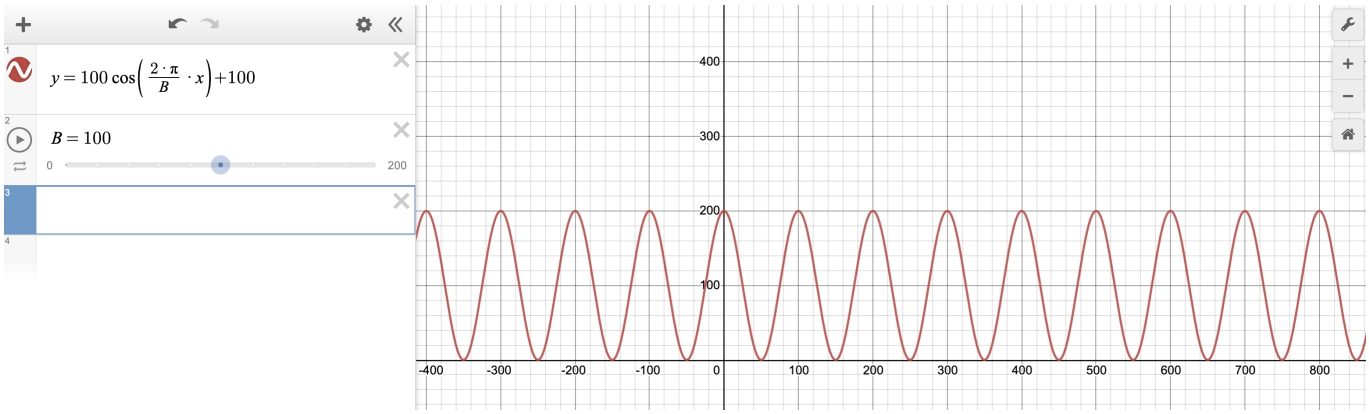
```
    ctx.clearRect(0,0, width, height);
    G.update();
    let timer = Math.round(t)
    T.update(timer);
    // the call below is a request to the browser
    // the function is called again (about 50 times a second)
    window.requestAnimationFrame(animate)
  }
  // start off call to get it going
  animate()
</script>
```

Listing 16: index



13 Animation

Making a ball go up and down: The cos function goes up and down.
It repeats itself every 100 'ticks'.



14 Event Handling

15 Basic Draggable Point

16 Better Draggable Point

```
console.log("point js file has been called");
class Point{
// class Point x,y,r, stroke, fill, over, canvas
constructor(x,y,r, stroke, fill, over){
    //basic position, size and colours
    this.x = x;
    this.y = y;
    this.r = r;
    this.stroke = stroke;
    this.fill = fill;
    this.over = over;
    //set true if mouse inside point circle
    this.inBounds = false;
    //cointinually registered mouse position
    this.xMouse = 0;
    this.yMouse = 0;
    //listeners
    canvas.addEventListener('mousedown', this.mDown.bind(this));
    canvas.addEventListener('mousemove', this.mMove.bind(this));
    canvas.addEventListener('mouseup', this.mUp.bind(this));
}
mDown(e){
```

```
23      // if the mouse is pressed (goes down) and the mouse is inside the point
        circle,
24      // set the this object as taken
25      if(this.inBounds){
26          Point.taken = this;
27      }
28  }
29  mMove(e){
30      // event registered every time the mouse moves
31      // object variables updated with current mouse position
32      this.xMouse = e.offsetX;
33      this.yMouse = e.offsetY;
34      //update boundary boolean
35      this.inBounds = this.boundsCheck(this.xMouse, this.yMouse, this.x, this.y,
        this.r);
36  }
37  mUp(e){
38      //when mouse goes up set taken point as nothing
39      //hence deselect this point
40      Point.taken = "";
41  }
42  /**
43   * called from animation loop
44   */
45  update(){
46  // make x,y coordinates of the point the same as the mouse position
47  // if the point has been taken
48      if(Point.taken == this){
49          this.x=this.xMouse;
50          this.y=this.yMouse;
51      }
52      this.draw();
53  }
54  draw(){
55      // change fill state if mouse is over or the point is selected
56      if(this.inBounds || Point.taken == this){
57      ctx.fillStyle= this.over;
58      }else{
59          ctx.fillStyle= this.fill;
60      }
61      ctx.strokeStyle = this.stroke;
62      ctx.lineWidth = 2;
63      ctx.beginPath()
64      ctx.arc(this.x,this.y, this.r, 0, 2*Math.PI);
65      ctx.fill();
66      ctx.stroke();
67  }
```

```
/**
 * Pythagoras distance check
 * @param  x,y,positions of mouse and of point circle and radius of point circle
    (number)
 * @return boolean
 */
boundsCheck(x_1, y_1, x_2, y_2, r){
        var d = Math.sqrt( Math.pow(x_2 - x_1, 2) + Math.pow(y_2 - y_1, 2) );
        if(d<r){
            return true;
        }else{
            return false;
        }
}
/**
 * Make x, y coordinates of point available outside of object
 * @return number
 */
getX(){
    return this.x;
}
getY(){
    return this.y;
}
}
// static variable available to all Point objects
// the same for all Point objects
// means only one Point can be selected and moveable
Point.taken="";
```

Listing 17: Python example

17 Buttons