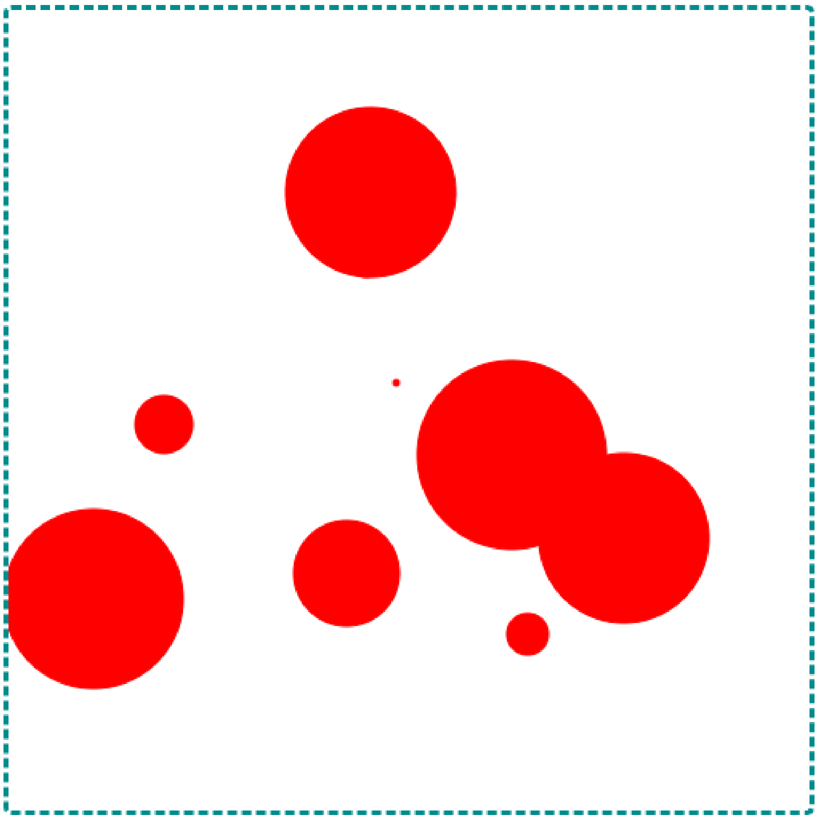
Teddywaddy Code Club

Activity 4g

Graphics Coding



Graphics Coding

This set of activities will introduce some 2D graphics coding using JavaScript, within a web page. It would be best to complete activity4f before this activity.

HTML includes an element called the Canvas; this is where all the graphics are displayed. The Canvas has an associated library that provides many useful graphics functions. So, programming graphics in JavaScript mainly involves calling Canvas functions. The Canvas can be just a part of a web page, but often it will be expanded to take up the whole window.

Open a new folder in VS Code.

Create a new HTML file called canvas01.html.

Enter the following code (remember to use the ! abbreviation to get the basic template first).

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Canvas01</title>
    <link rel="stylesheet" href="canvas01.css">
</head>
<body>
    <canvas id="myCanvas" width="500" height="500"></canvas>

    <script src="canvas01.js" type="text/javascript"></script>
</body>
</html>
```

Notice that in this case the canvas has been restricted to 500 pixels square.

Create a new CSS file called canvas01.css

Enter the following code.

```css
canvas {
    border-style: dashed;
    border-color: darkcyan;
    border-radius: 5px;
}
```

The canvas will have borders around it to highlight its extent.

The CSS file is very simple and could have been entered into the <head> section of the html, but it is desirable to get into the habit of using a separate file.

The first graphics to be drawn will be a circle. The Canvas function for this is called *arc*. The definition for this function is:

arc(x, y, radius, startAngle, endAngle)

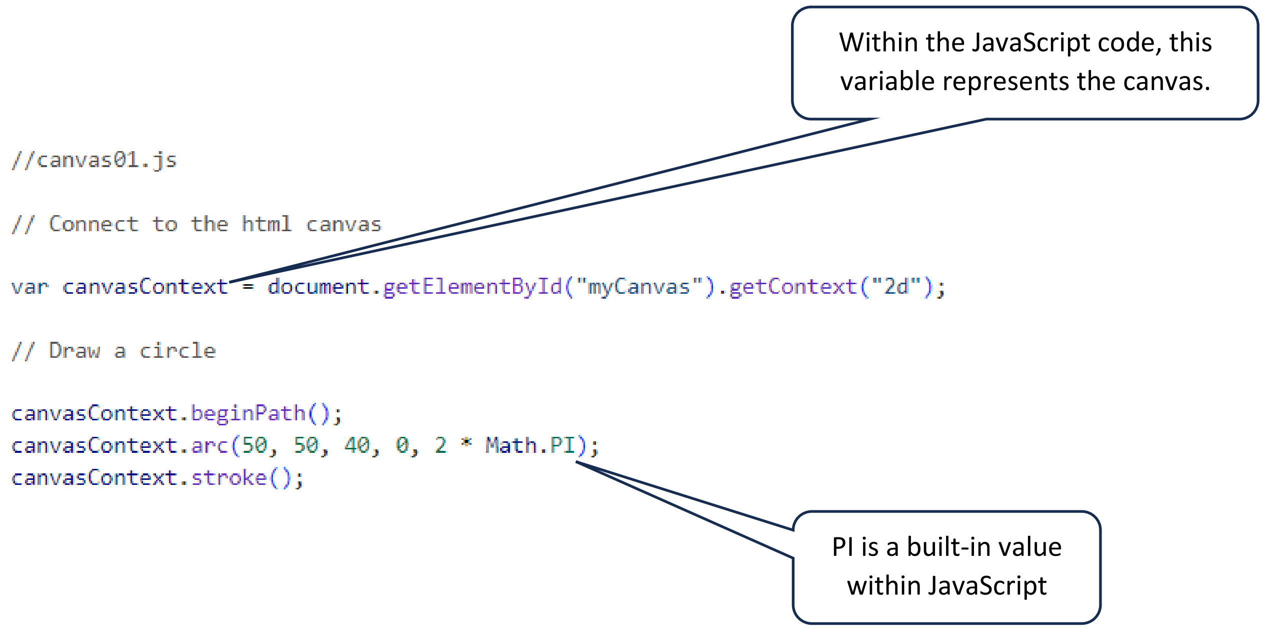
To use this function a variable that represents the canvas must be established and then the arc function is called using the dot connector. For example,

canvasVariable.arc(220, 160, 50, 0, 6.28);

Using 0 to 6.28 should give a full circle (6.28 is 2 x PI).

Of course there are other drawing functions as well, such as rectangles, lines and curves.

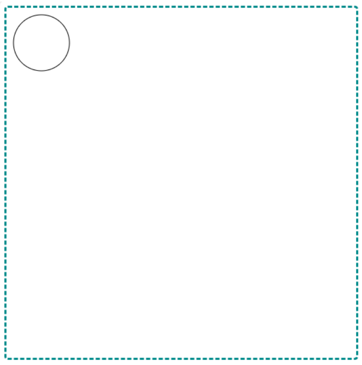
Create a new file called canvas01.js. Enter the following code.



```
//canvas01.js

// Connect to the html canvas

var canvasContext = document.getElementById("myCanvas").getContext("2d");

// Draw a circle

canvasContext.beginPath();
canvasContext.arc(50, 50, 40, 0, 2 * Math.PI);
canvasContext.stroke();
```

Within the JavaScript code, this variable represents the canvas.
PI is a built-in value within JavaScript

Notice that drawing on the canvas involves mapping out a path before making the path visible with the stroke() function.



To change the colour of the circle, set the stroke colour by setting the strokeStyle property.

```
canvasContext.strokeStyle="red";
```

This is not a function. strokeStyle is a property.

To fill the circle with colour, use the fill() function and similarly to change the fill colour, set the fillStyle property.

```
//canvas01.js

// Connect to the html canvas

var canvasContext = document.getElementById("myCanvas").getContext("2d");

// Set initial colours

canvasContext.strokeStyle="red";
canvasContext.fillStyle="red";

// Draw a circle

canvasContext.beginPath();
canvasContext.arc(50, 50, 40, 0, 2 * Math.PI);
canvasContext.fill();
canvasContext.stroke();
```

For more flexibility it would be good to make the code that draws the circle into a function.

```
//canvas01.js

// Connect to the html canvas

var canvasContext = document.getElementById("myCanvas").getContext("2d");

// Set initial colours

canvasContext.strokeStyle="red";
canvasContext.fillStyle="red";

drawCircle();

function drawCircle() {

    // Draw a circle

    canvasContext.beginPath();
    canvasContext.arc(50, 50, 40, 0, 2 * Math.PI);
    canvasContext.fill();
    canvasContext.stroke();

}
```

Notice that a function doesn't execute unless *called*. If this line is left out nothing will be drawn.

While that would work for this code, it doesn't provide any extra flexibility because it always draws the circle at coordinates 50,50 with a radius of 40. Functions can receive values that can then be used within the function.

```
//canvas01.js

// Connect to the html canvas

var canvasContext = document.getElementById("myCanvas").getContext("2d");

// Set initial colours

canvasContext.strokeStyle="red";
canvasContext.fillStyle="red";

drawCircle(50,60,40);

function drawCircle(x,y,r) {

    // Draw a circle

    canvasContext.beginPath();
    canvasContext.arc(x, y, r, 0, 2 * Math.PI);
    canvasContext.fill();
    canvasContext.stroke();

}
```

The x,y and r variables only exist within the function. Their scope is limited to the function. But they receive values from where the function was called. So, in this case x receives 50, y receives 60 and r receives 40.

If more circles are required, this is now simple just add more calls to the drawCircle() function.

```
drawCircle(50,60,40);
drawCircle(350,260,40);
drawCircle(150,60,40);
```

Instead of putting numbers in the function call, use the random() function, which is part of the Math library. It returns a number between 0 and up to but not including 1. To get larger numbers, multiply by the largest value required. The floor() function can be used to round down to a whole number.

```
// Generate random values for x, y and r

const x = Math.floor(Math.random() * 400);
const y = Math.floor(Math.random() * 400);
const r = Math.floor(Math.random() * 70);

// Draw the random circle

drawCircle(x,y,r);
```

Finally, use the for loop to draw 10 random circles.

```
//canvas01.js

// Connect to the html canvas

var canvasContext = document.getElementById("myCanvas").getContext("2d");

// Set initial colours

canvasContext.strokeStyle="red";
canvasContext.fillStyle="red";

// Draw 10 random circles

for(let i=1;i<10;i++) {

    // Generate random values for x, y and r

    const x = Math.floor(Math.random() * 400);
    const y = Math.floor(Math.random() * 400);
    const r = Math.floor(Math.random() * 70);

    // Draw the random circle

    drawCircle(x,y,r);

}

function drawCircle(x,y,r) {

    // Draw a circle

    canvasContext.beginPath();
    canvasContext.arc(x, y, r, 0, 2 * Math.PI);
    canvasContext.fill();
    canvasContext.stroke();

}
```

There are many variations to explore with this activity.

- Change colours of the fill or border.
- Add some HTML for a heading.
- Centre the canvas.
- Change values so the circles are always completely within the canvas.