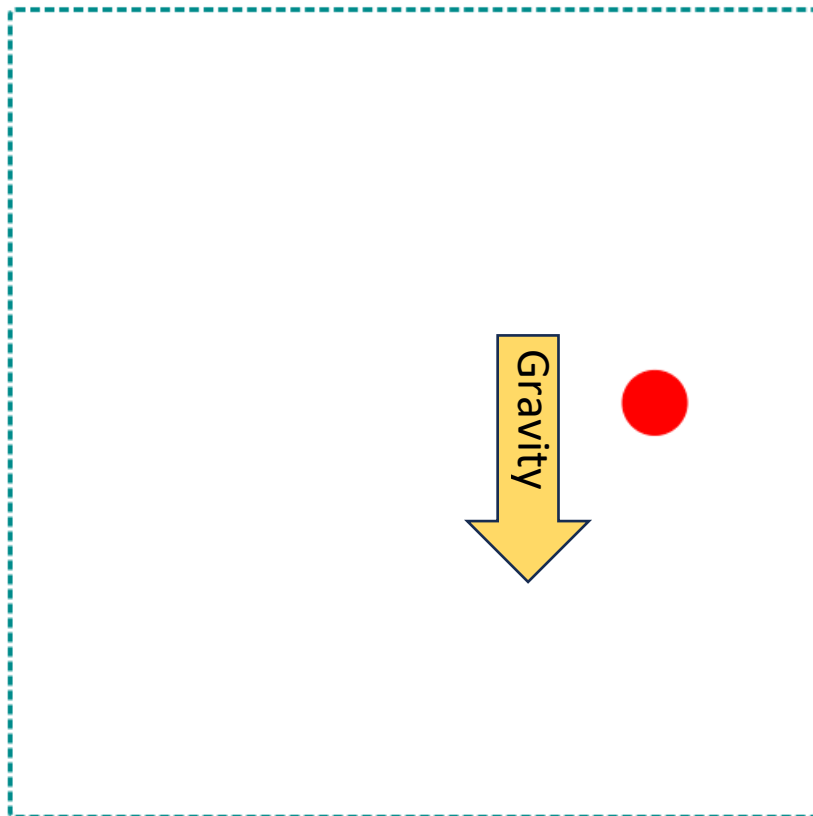


# Teddywaddy Code Club

## Activity 4h

### Beginning Animation



# Graphics Coding

This activity will introduce the concept of animation for a simple ball falling under gravity.

To begin this exercise,

- create a new html file called canvas02.html

- create a new css file called canvas02.css

- create a new JavaScript files called canvas02.js

## The HTML file.

```
<!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>Canvas02</title>
  <link rel="stylesheet" href="canvas02.css">
</head>
<body>
  <canvas id="myCanvas" width="500" height="500"></canvas>

  <script src="canvas02.js" type="text/javascript"></script>
</body>
</html>
```

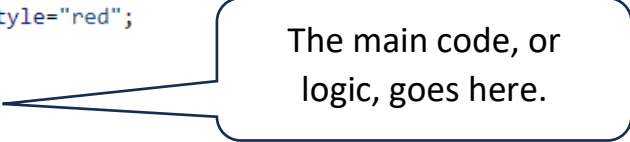
## The CSS file.

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

The initial JavaScript file will be similar to activity4g.

## The JavaScript file.

```
//canvas02.js  
  
// Connect to the html canvas  
  
var canvasContext = document.getElementById("myCanvas").getContext("2d");  
  
// Set initial colours  
  
canvasContext.strokeStyle="red";  
canvasContext.fillStyle="red";
```



The main code, or logic, goes here.

```
function drawCircle(x,y,r) {  
    // Draw a circle  
  
    canvasContext.beginPath();  
    canvasContext.arc(x, y, r, 0, 2 * Math.PI);  
    canvasContext.fill();  
    canvasContext.stroke();  
}
```

This code doesn't do anything yet but has a function for drawing a filled circle (or 2D ball) on the canvas. The overall objective is to make the ball appear to be falling according to gravity.

In graphics, animation is achieved primarily by redrawing the scene very quickly, with some components of the scene moved slightly between redraws. This is just like a movie, where each frame is a still image, but they are changed fast enough to look like motion.

In code, this generally involves using a loop to repetitively clear and redraw the scene with some minor changes in between.

Most computers can now do this repetition a minimum of 60 times per second, which is fast enough to look like animation.

One consequence of this it that as the scene becomes more complicated and takes longer to draw, there is a limit to how much can be drawn before the next cycle. Some of this is overcome by the use of specialist graphics hardware.

## The high-level design of what is required.

Broadly the code must do this.

Set some values for,

- the initial position of the ball
- the initial speed of the ball
- the force of gravity

Then use a loop to,

- clear the scene
- draw the ball
- update the current position of the ball

Also, there will be some way to represents time (typically this is just a variable). Quite often animations are run at real-time, so loop counters are actually using seconds (or milliseconds) to count with.

Here's a code block that could be used to set some initial values.

```
var ballRadius = 20;

// The starting location and speed of the ball

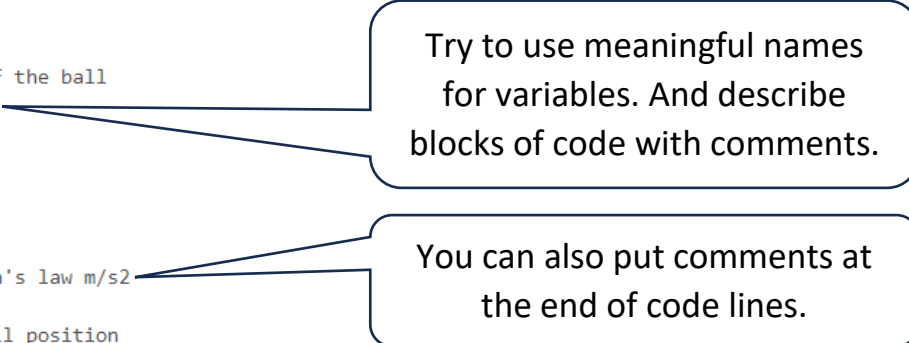
var initialPositionX = 400;
var initialPositionY = 30;
var initialVelocityX = 0;
var initialVelocityY = 0;

var gravity = 9.8;           // Newton's law m/s2

// Variables to store the current ball position

var currentPositionX;
var currentPositionY;

var timeIncrement = 100;    // How much time to move the animation forward each cycle (in mS)
var currentTime = 0;       // This variable will store the elapsed or current time (where the animation is up to)
```



Before putting in the loop, examine the complete code below - just to draw one ball but using all these variables.

```

//canvas02.js

var ballRadius = 20;

// The starting location and speed of the ball

var initialPositionX = 400;
var initialPositionY = 30;
var initialVelocityX = 0;
var initialVelocityY = 0;

var gravity = 9.8;          // Newton's law m/s2

// Variables to store the current ball position

var currentPositionX;
var currentPositionY;

var timeIncrement = 100;   // How much time to move the animation forward each cycle (in mS)
var currentTime = 0;       // This variable will store the elapsed or current time (where the animation is up to)

// Connect to the html canvas

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

// Set the initial canvas colours

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

// Establish the current position to start off

currentPositionX = initialPositionY;
currentPositionY = initialPositionY;

// Animation

// Move and draw the ball

canvasContext.clearRect(0,0,500,500);
drawCircle(currentPositionX,currentPositionY,ballRadius);

function drawCircle(x,y,r) {

    // Draw a circle

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

}

```

This is the code that needs to become an animation loop. The current position needs to change each time the canvas is cleared and redrawn.

Going back to the design, the animation loop needs to,

- clear the screen
- draw the ball
- update the ball position

Updating the ball position requires two steps,

- update the time
- calculate the new position from the time (Newton's law)

This can be accomplished using the following function.

```

function moveBall() {
    // Clear the screen
    canvasContext.clearRect(0,0,500,500);

    // Draw the ball at its current position
    drawCircle(currentPositionX,currentPositionY,ballRadius);

    // Update the elapsed time
    currentTime = currentTime + timeIncrement/1000;

    // Update both the x and y coordinate positions for the new time
    currentPositionX = initialPositionX + initialVelocityX*currentTime; // S = vt
    currentPositionY = initialPositionY + initialVelocityY*currentTime + 0.5*gravity*currentTime*currentTime; // S = vt + 1/2 at2
}

```

Now the animation code could become just that shown below.

```

// Animation
// Move and draw the ball
moveBall()

```

This still isn't a loop, so the moveBall() function could be placed into a for loop.

```

// Animation
// Move and draw the ball
for (let i=0;i<50;i++) {
    moveBall()
}

```

However, the ball doesn't appear to move although it is drawn in a different position. The problem is that the for loop goes so fast that the ball is at the final position without being able to see it move.

What is required is a loop that can repeat the moveBall() function at say 60 times per second. Ideally though it would be good to be able to slow it down even more when required, for testing and further code development.

For this, JavaScript has the setInterval() function.

The setInterval() function is defined like this.

**setInterval(function, delay)**

- function is whatever function you want called repeatedly - like moveBall()
- delay is the time between calls, in mS

The way `setInterval()` works is a subtle yet very important aspect of JavaScript. JavaScript is an asynchronous language such that many blocks of code can be happening at the same time. This won't impact the code in this activity, but does change some aspects of coding in JavaScript.

To get the animation happening now is relatively simple.

```
// Animation  
  
// Move and draw the ball  
  
var intervalTimer = setInterval(moveBall,timeIncrement);
```

This variable could also be declared with the other global variables.

The ball will now fall according to the equations of motion. The ball does continue on past the canvas. To stop the ball at the canvas boundary, use an if test to stop the timer.

```
// Update both the x and y coordinate positions for the new time  
  
currentPositionX = initialPositionX + initialVelocityX*currentTime; // S = vt  
currentPositionY = initialPositionY + initialVelocityY*currentTime + 0.5*gravity*currentTime*currentTime; // S = vt + 1/2 at2  
  
if(currentPositionY > 500) clearInterval(intervalTimer);
```

Add this line of code at the end of the `moveBall()` function to stop the animation.

If the ball is given some horizontal velocity as well, then the ball might hit the right-hand side of the canvas before the bottom, so the if test would need to test both possibilities.

```
if(currentPositionY > 500 || currentPositionX>500) clearInterval(intervalTimer);
```

- A much smaller number for the `timeIncrement` variable will give a smoother animation.
- Try adding some initial x velocity.
- Try changing the starting position.
- What code would need to be changed to make the ball bounce?
- The dimensions of the canvas shouldn't be "hard-coded" as numbers (500), they should be variables and should also be requested using a function call.

```
// Get the size of the canvas as set in the HTML or CSS  
  
canvasWidth = canvasContext.canvas.width;  
canvasHeight = canvasContext.canvas.height;
```

Try updating the code with these variables and making the canvas bigger.