MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA
UNIVERSITY OF NEW ZEALAND

# HTML5 Canvas Tutorial:
## *Solar System Model Simulator*

# The Tutorial Challenge: Modelling the Solar System

The tutorial challenge on the HTML5 canvas is for you to draw a moving model of the solar system.

For the sake of time, youâ€™ll show only the earth, sun, and moon.

In this challenge, the implementation will take advantage of two important features of the HTML5 canvas:

1. Paths; and
2. Transformations.

For the theory behind this tutorial challenge, see the Appendix section of these set of slides.

# Step 1: Create Solar System Model Web Page

Create a new web page, **solar.html**, using the the following basic markup:

```
<!DOCTYPE html>
<html lang="en">
        <head>
                <meta charset="utf-8" />
                <title>Chapter 23 - Solar System</title>
                <script src="Solar.js" defer></script>
        </head>
        <body>
        </body>
</html>
```

# Step 2: Create the Canvas

Add the canvas element inside the <body> element.

```
<canvas id="solarSystem" width="450" height="400">
        Not supported
</canvas>
```

# Step 3: Create the JavaScript File - solar.js

Add a new **solar.js** script using the following code:

```
"use strict";

// Get the canvas context
let solarCanvas = document.getElementById("solarSystem");
let solarContext = solarCanvas.getContext("2d");
```

This code gets the **<canvas>** element and then obtains the **2d** drawing context, just like in the previous examples in the lecture slides.

# Step 4: Add the animateSS() Function

Add the **animateSS()** function to the **solar.js** file using the following code:

```
function animateSS() {
        let ss = document.getElementById('solarSystem')
        let ssContext = ss.getContext('2d');

        // Clear the canvas and draw the background
        ssContext.clearRect(0, 0, 450, 400);
        ssContext.fillStyle = "#2F1D92";
        ssContext.fillRect(0, 0, 450, 400);
        ssContext.save();

        // Draw the sun
        ssContext.translate(220, 200);
        ssContext.fillStyle = "yellow";
        ssContext.beginPath();
        ssContext.arc(0, 0, 15, 0, Math.PI * 2, true);
        ssContext.fill();

        // Draw the earth orbit
        ssContext.strokeStyle = "black";
        ssContext.beginPath();
        ssContext.arc(0, 0, 150, 0, Math.PI * 2);
        ssContext.stroke();
```

```
        ssContext.restore()
}
```

# Step 4: Explanation of the animateSS() Function

- The **animateSS()** function is what does the real work.
- It clears the entire area and then fills it with dark blue.
- The rest of the code relies on *transformations,* so it first saves the drawing context and then restores it when finished.
- The **animateSS()** function uses the **translate()** function to move the origin to the approximate midpoint of the canvas.
- The sun and the earth orbits are drawn using the **arc()** function.
- Notice the center point for both is **(0, 0)** since the contextâ€™s origin is now in the middle of the canvas.
- Also, notice the start angle is **0** and the end angle is specified as **Math.PI * 2**.

- In radians, this is a full circle or **360Â°**.
- The arc for the sun is filled in, and the orbit is not.

# Step 5 and 6: Calling animateSS()

**Step 5:**

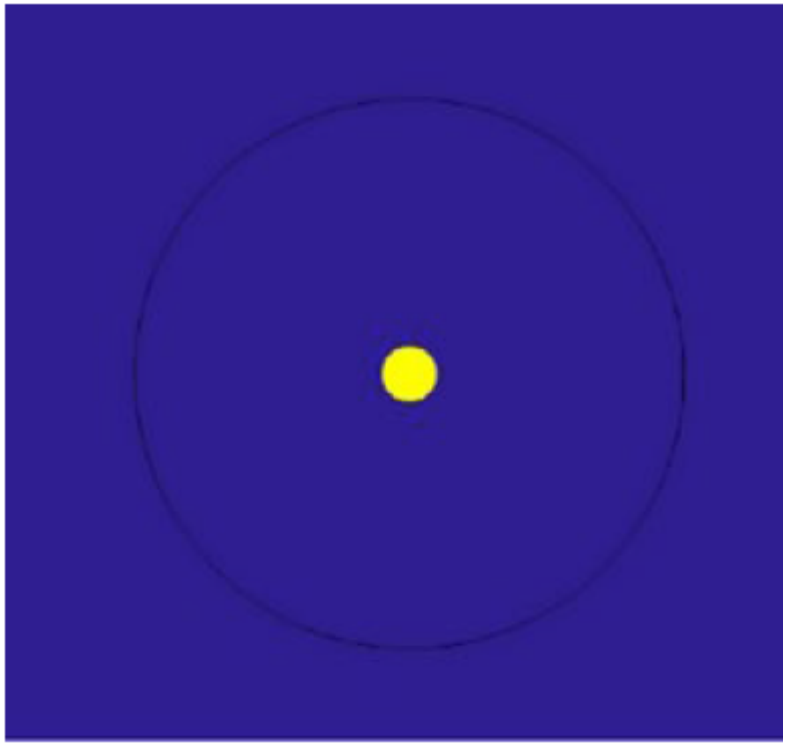
Call the **setInterval()** function to call the **animateSS()** function every **100** milliseconds.

```
setInterval(animateSS, 100);
```

**Step 6:**

Display the page in the browser.

*So far the drawing is not very interesting; itâ€™s a sun with an orbit drawn around it, as shown in the diagram to the right.*



# Step 7: The Earth and its orbit

- Now youâ€™ll draw the earth and animate it around the orbit.
- Normally the earth will revolve around the sun once every 365.24 days, but weâ€™ll speed this up a bit and complete the trip in 60 seconds.
- To determine where to put the earth each time the canvas is redrawn, you must calculate the number of seconds.
- The amount of rotation per second is calculated as: **Math.PI *2 / 60**.
- Multiply this value by the number of seconds to determine the angle where the earth should be.

# Step 7: Animating the Earth's Orbit

Add the following code appearing between the comments, "START ADDING HERE" and "STOP ADDING HERE":

```
// Draw the earth orbit
ssContext.strokeStyle = "black";
ssContext.beginPath();
ssContext.arc(0, 0, 150, 0, Math.PI * 2);
ssContext.stroke();

// START ADDING FROM HERE

// Compute the current time in seconds (use the milliseconds
// to allow for fractional parts).
var now = new Date();
var seconds = ((now.getSeconds() * 1000) + now.getMilliseconds()) / 1000;

//-----------------------------------------
// Earth
//-----------------------------------------
// Rotate the context once every 60 seconds
var anglePerSecond = ((Math.PI * 2) / 60);
ssContext.rotate(anglePerSecond * seconds);
ssContext.translate(150, 0);
```

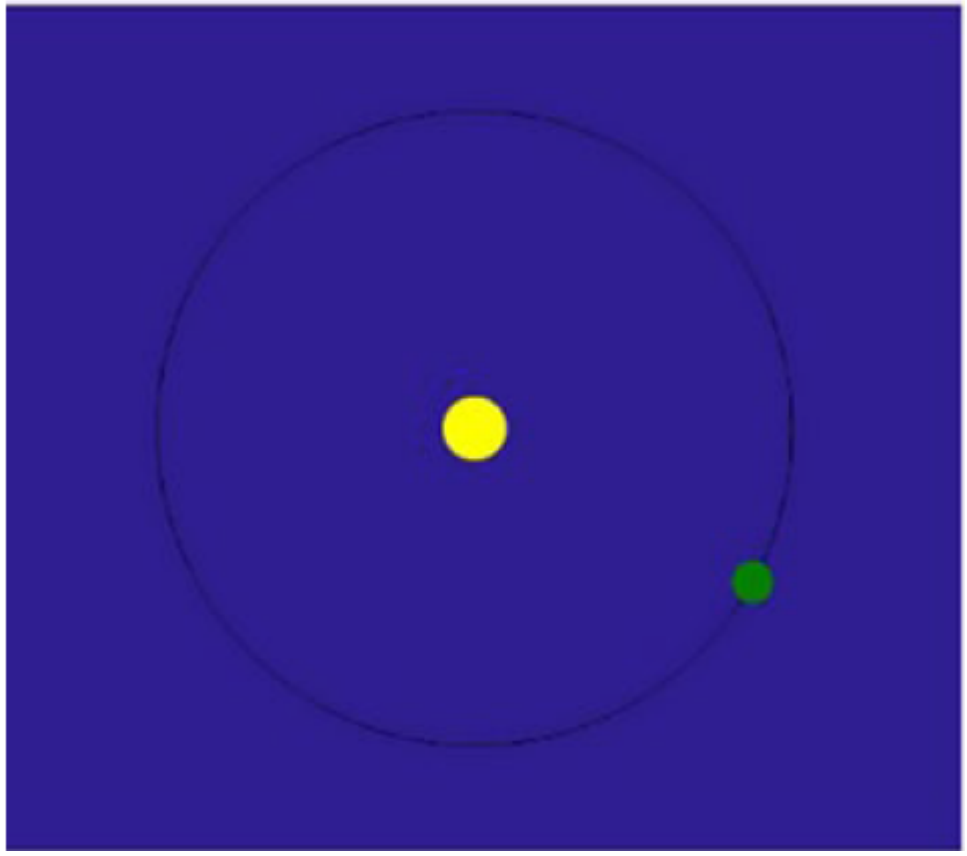
```
// Draw the earth
ssContext.fillStyle = "green";
ssContext.beginPath();
ssContext.arc(0, 0, 10, 0, Math.PI * 2, true);
ssContext.fill();

//STOP ADDING HERE

ssContext.restore()
```

# Step 7: Explaining the code

- This code uses the **rotate()** function to rotate the drawing context the appropriate angle.
- Since the arc for the earth orbit is 150px, this code then uses the **translate()** function to move the context 150 pixels to the right so the earth can be drawn at the adjusted **(0,0)** coordinate.
- Notice that is combining two separate **transforms**, one to **rotate** based the earth position in its orbit and one to **translate** the appropriate distance from the sun.
- The earth is then drawn using a filled arc with a center point of **(0,0)**, the new origin of the context.

# Step 8: View the Solar System Model

Save your changes and refresh the browser.
Now you should see the earth make its way around the sun, as shown in the diagram below.



# Modelling the Moon

- Now youâ€™ll show the moon revolving around the earth, which will demonstrate the real power of using transformations.
- The specific position of the moon is based on two moving objects.
- While itâ€™s certainly possible to compute this using some complex formulas (scientists have been doing this for centuries) with transformations, you donâ€™t have to.
- The drawing context was rotated the appropriate angle based on current time (number of seconds).
- It was then translated by the radius of the orbit, so the earth is now at the origin of the context.

- It doesnâ€™t really matter where the earth is; you can simply draw the moon relative to the current origin.

# Step 9: Drawing and Animating the Moon

- You will now draw the moon just like you drew the earth.
- Instead of the origin being at the sun and rotating the earth around the sun, the origin is on the earth, and youâ€™ll rotate the moon around the earth.
- The moon will rotate around the earth approximately once each month; in other words, it will complete about 12 revolutions for

```
// Draw the earth
ssContext.fillStyle = "green";
ssContext.beginPath();
ssContext.arc(0, -0, 10, 0, Math.PI * 2, true);
ssContext.fill();

//START ADDING FROM HERE

//-------------------------------------------
// Moon
//-------------------------------------------
// Rotate the context 12 times for every earth re
anglePerSecond = 12 * ((Math.PI * 2) / 60);
ssContext.rotate(anglePerSecond * seconds);
ssContext.translate(0, 35);

// draw the moon
ssContext.fillStyle = "white";
ssContext.beginPath();
ssContext.arc(0, 0, 5, 0, Math.PI * 2, true);
ssContext.fill();

//STOP ADDING HERE
```

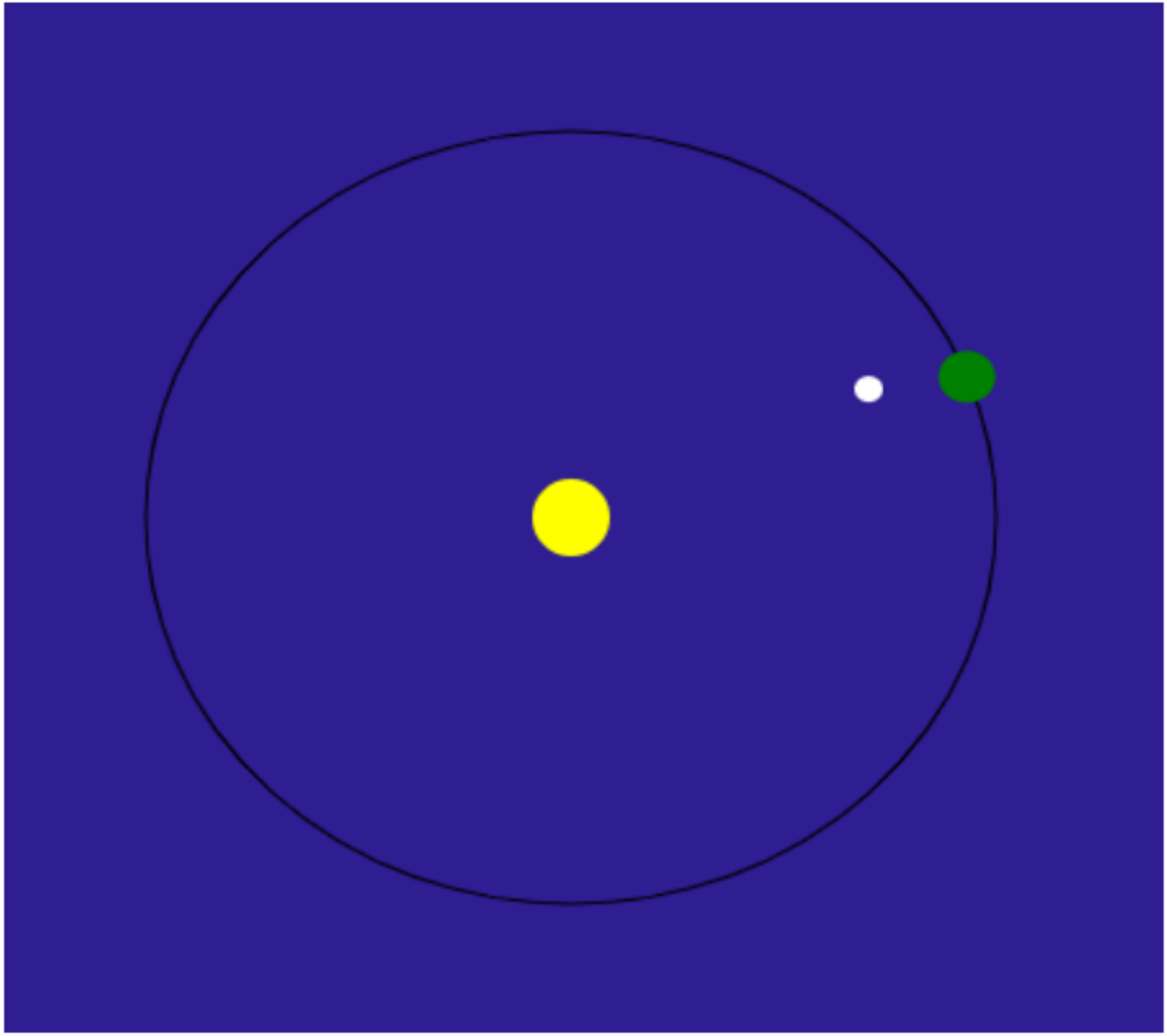
each earth orbit.

```
ssContext.restore()
```

- So, youâ€™ll need to rotate 12 times faster. The anglePerSecond is now computed as **12 * ((Math.PI * 2) / 60)**.
- Add the following code shown to the right between the comments: "START ADDING FROM HERE" and "STOP ADDING HERE".

## Step 10: View the final solar model

Save your changes and refresh the browser. You should now see the moon rotating around the earth, and the earth round the sun. The diagram below shows a snapshort of this model. Upload the final solution to your website.



# The Appendices:

Based on materials from:

**Chapter 23 of the book:**

Mark J Collins (2017),
, Apress. ISBN: 978-1-4842-2463-2

**Web content from developer.mozilla.org:**

, Accessed: 2019-09-18:23:37

# Appendix 1: Using Paths

- The two simple or primitive shapes that the HTML5 canvas supports are: (1) the **rectangle;** and (2) the **path** (list of points connected by lines)
- For all other shapes you must define a path.
- The basic approach to defining paths in canvas is similar to SVG.
- You use a move command to set the starting point and then some combination of line and curve commands to draw a shape.

(Mark J Collins, 2017)

# Appendix 2: Functions for Drawing Rectangles

The three functions for drawing rectangles are:

**fillRect(x, y, width, height):**

Draws a filled rectangle.

**strokeRect(x, y, width, height):**

Draws a rectangular outline.

**clearRect(x, y, width, height):**

Clears the specified rectangular area, making it fully transparent.

(Drawing shapes with canvas, developer.mozilla.org. Accessed: 2019-09-18:23:37)

# Appendix 3: The beginPath() command

- In the HTML5 canvas, drawing shapes always start with a **beginPath()** command.
- After calling the desired drawing commands (see next slide), the path is completed by calling either **stroke()** to draw an outline of the shape or **fill()** to fill in the shape.
- The shape is not actually drawn to the canvas until either **stroke()** or **fill()** is called.
- If you call **beginPath()** again, before completing the current shape (with a call to **stroke()** or **fill()**), the canvas will ignore the previous uncompleted commands.
- The same **strokeStyle** and **fillStyle** properties that you used with rectangles

also define the color of the path.

(Mark J Collins, 2017)

# Appendix 4: Drawing Commands for the HTML5 Canvas

The actual drawing commands are as follows:

**moveTo(x, y):**

Moves the pen to the coordinates specified by x and y.

**lineTo(x,y):**

Draws a line from the current drawing position to the position specified by x and y.

**arc(x,y,radius,startAngle,endAngle,antiClockwise):**

Draws an arc which is centered at (x, y) position with radius r starting at startAngle and ending at endAngle going in the given direction indicated by anticlockwise (defaulting to clockwise).

**arcTo($x_1$, $y_1$, $x_2$, $y_2$, radius):**

Draws an arc with the given control points and radius, connected to the previous point by a straight line.

**bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y):**

Draws a cubic BÃ©zier curve from the current pen position to the end point specified by x and y, using the control points specified by (cp1x, cp1y) and (cp2x, cp2y).

**quadraticCurveTo(cp1x, cp1y, x, y):**

Draws a cubic BÃ©zier curve from the current pen position to the end point specified by x and y, using the control points specified by (cp1x, cp1y) and (cp2x, cp2y).

**closePath():**

This performs a **lineTo()** command from the current position to the starting position to close in the shape.
If you use the **fill()** command, the **closePath()** function is automatically

called if youâ€™re not currently at the starting position.

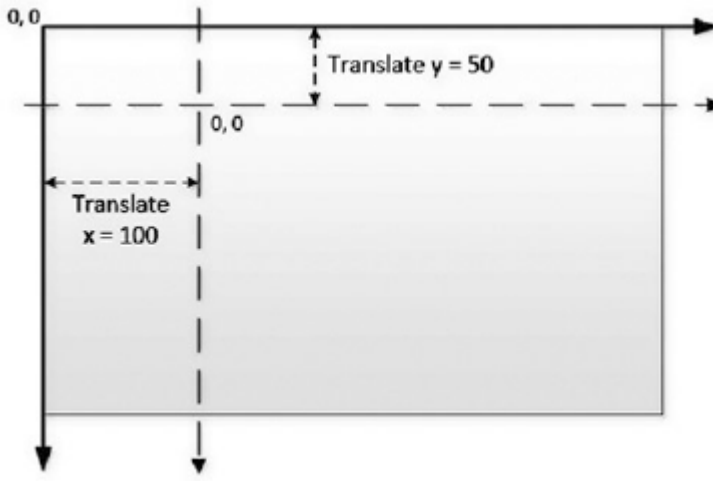
(Drawing shapes with canvas, developer.mozilla.org. Accessed: 2019-09-18:23:37)

# Appendix 5: Using Transformations

- Transformations have no effect on what has already been drawn on the canvas;
- They modify the grid system that will be used to draw subsequent shapes;
- Three types of transformations are useful in this course. These are as follows:

    1. *Translating;*
    2. *Rotating; and*
    3. *Scaling.*

- A canvas element uses a grid system where the




The Translate Transformation (Mark J Collins, 2017)

origin is at the top-left corner of the canvas;

- Thus, a point at (100, 50) will be 100 pixels to the **right** and 50 pixels **down** from that corner as shown in the diagram to the right of this slide.
- Transformations simply adjust the grid system, e.g., the following command will shift the origin 100 pixels to the right and 50 pixels down:

```
context.translate (100, 50);
```

-

(Mark J Collins, 2017)

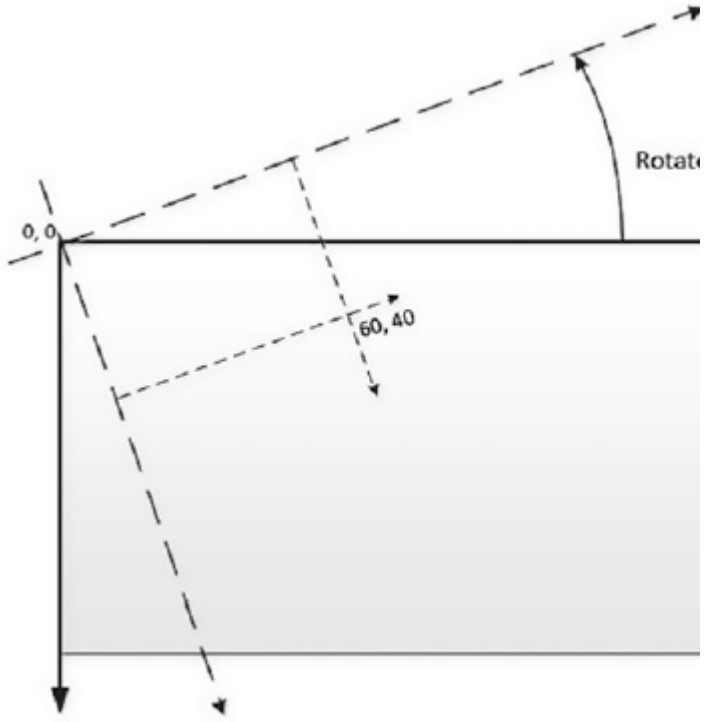
# Appendix 6: The Rotate Transformation

The rotate transformation doesnâ€™t move the origin; instead, it rotates the x- and y-axes by the specified amount.

A *positive* amount is used for a *clockwise rotation*, and a *negative* amount is used to *rotate counterclockwise*.

The diagram to the right on this slide demonstrates how a rotate transformation works.

**Note:** The rotation angle is indicated as 30° since that is what most people are familiar with. However, the **rotate()** command expects the value in **radians**. If your geometry is a little rusty,




The Rotate Transformation (Mark J Collins, 2017)

a full circle is 360° or 2𝜋 radians. In JavaScript, you can use the **Math.PI** property to get the value of 𝜋 (Pi). For example, 30° is 1/12 of a full circle, so you can write this as **(Math.PI * 2/12)**. In general, **radians** are calculated as **degrees * (Math.PI/180)**.

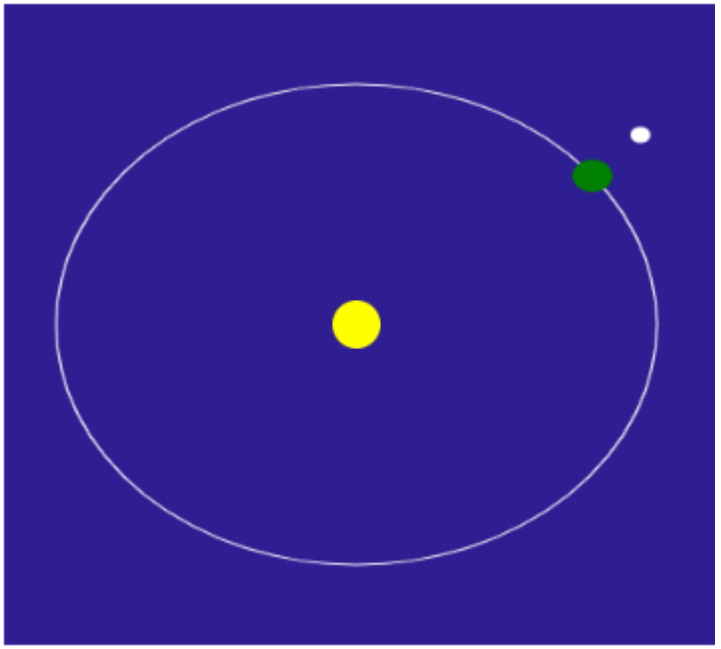
(Mark J Collins, 2017)

# Appendix 7: The Scaling Transformation

- The earthâ€™s orbit is not actually a perfect circle. This attribute is known as **eccentricity**.
- (If youâ€™re curious about **orbital eccentricity**, check out the article in

Wikipedia).

- To model this in your drawing, youâ€™ll **stretch** the orbit, *making it a little bit wider than it is tall.*To do this, youâ€™ll use **scaling**.
- The **scale(scaleX,scaleY)** function performs the third type of transformation, the **scaling transformatrion**.
- This function takes two parameters, scaleX and scaleY, that specify the scaling factors along the x- and y-axes. A scale factor of 1 is the normal scale.
- A factor less than 1 (one) will compress the drawing, and a factor greater than 1 (one) will stretch it.



Adding scaling to Earth orbit (Mark J Collins, 2017)

- While the imperfection in the earthâ€™s orbit is extremely slight, youâ€™ll exaggerate it here and use a scale factor of 1.25 for the x-axis.
- Add the following code shown in bold just before the earth orbit is drawn in the file, solar.js:

```
// Draw the earth orbit
ssContext.scale(1.25, 1);
ssContext.strokeStyle = "white";
```

- Refresh the browser; the page should look like the picture to the right on this slide if you change the strokeStyle of the orbit from black to white.