GD 14
Drawing App, Part 2

In GD 13, we learned that creating a simple drawing program requires not having to erase the screen. Instead, we keep the current content and then add onto it.

Being able to specify a starting point, an end point, and then connecting them with a line allows us to create one part of a shape's outline. If we use the same process for all points, then we have created a shape. Being able to manually connect two points with a line also gives us the ability to create custom shapes.

In this lesson, we will learn how to create custom shapes by using the **M** command followed by the **L** command.

1. Custom Shapes 2. Path2D

---

1. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ================================================
* Keyboard Event & Event Handler
*
* ================================================ */
window.addEventListener( "keydown", key_decode );
```

Explanation

The code above set our website up to listen for and then react to the user pressing down on the keyboard. The event is "`keydown`". When the user presses down on any button of the keyboard, the website will react by calling the JS function **key_decode().**

2. Manual Connection of Shapes

We know that a shape is an array of points. The HTML 5 canvas has premade functions that help us connect the points together with a line to create a shape. We used `ctx.rect()` and `ctx.fill()` to create a square or rectangle.

There are situations where we want to create custom shapes. HTML 5 canvas allows us to do that by manually connecting the points. We use `ctx.Path2D( pathString )` to manually create custom shapes. The pathString is a variable that contains a mix of variables and custom quotations.

For example,

```
var pathString = "M 0 0 L 100 10 L 100 300 Z";
```

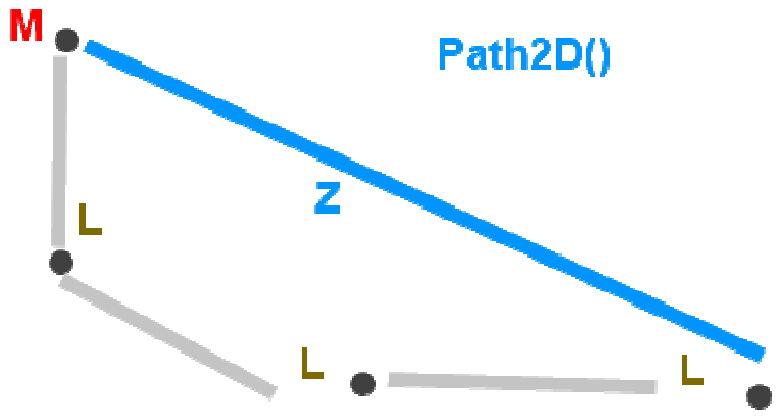
The example code translates into "**Move to** point (0,0) as the starting point. Next, **draw a line** from the starting point to (100,10). Next, **draw a line** from the previous point to the next point of (100,300). **Finally, connect the most recent point to the starting point**".

There are several important points to remember.

1. We use the **M** command to indicate the starting (x,y) point. The canvas will interpret the M command as "**Move to (x,y)." There should only be one M command per shape.**

2. Next, we use the **L** command to connect the starting point with the second point, the second point with the third point, the third point with the fourth point, and so on. **A shape can use many L commands.**

3. The **Z** command is a special command that tells the canvas to connect the most recent point with the starting point specified by the **M** command. **There should only be one Z command per shape.**



We can now use the M, L, and Z command with (x,y) pair to map out the points of our shape. The more points we have, the more detailed the shape but it does take longer to draw.

3. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ====================================
* Trapezoid Shape
*
* ==================================== */
class TrapezoidShape
{
	constructor()
	{
		this.xPos    = 0;
		this.yPos    = 0;
		this.sWidth  = 50;
		this.sHeight = 50;

		this.moveToString = " ";
		this.lineToString = " ";
		this.pathString   = " ";
	}

	update_state( posX, posY )
	{
		this.xPos = posX;
		this.yPos = posY;
		this.moveToString = "M " + this.xPos + " " + this.yPos + " ";
	this.lineToString = "L " + (this.xPos + this.sWidth) + " " + this.yPos + " L
" + (this.xPos + this.sWidth) + " " + (this.yPos + this.sHeight ) + " L " +
(this.xPos-50) + " " + (this.yPos + this.sHeight );

		this.pathString = this.moveToString + this.lineToString + " Z ";

	}

}
```

Explanation

There are many ways to create a custom path string. To make it easier for us to read and write code, I split the path string into different variables. This will make it easier for use to specify the starting point for the M command, the middle points for the L command, and then finally the Z command to finish the shape.

In the code above, the M, L, and Z commands are in between double quotations. If we don't put the commands in between double quotations, then HTML 5 canvas will think it is a variable.

However, the (x,y) pair can be on the outside of the double quotations. We are using variables to store the (x,y) pair and so it is ok to put the variables outside of the double quotations.

The last line of code combine them together using the + plus operator, which can mean two things based on the data we give it.

a. If we give it **numbers**, then the + plus operator **adds** the two numbers together as a math operation

b. If we give it **letters or words**, then the + plus operator **joins** them together to form a sentence.

The Javascript programming language will do this automatically for us and that is why the code above works.

CONTINUE TO THE NEXT PAGE

4. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

This is also JS Challenge 0

```
/* ========================================
* Square Shape
*
* ======================================== */
class SquareShape
{
	constructor()
	{
		this.xPos    = 0;
		this.yPos    = 0;
		this.sWidth  = 50;
		this.sHeight = 50;

		this.moveToString = " ";
		this.lineToString = " ";
		this.pathString   = " ";
	}

	update_state( posX, posY )
	{
		this.xPos = posX;
		this.yPos = posY;

		this.moveToString = "M " + this.xPos + " " + this.yPos + " ";

		this.lineToString = "L "  ;


		this.pathString = this.moveToString + this.lineToString + " Z ";

		}

	}
```

Explanation

Finish the code in red to create a square.

JS Challenge 1

1. Create the class definition for a pentagon shape.
2. Push it into the **`colorBoxArr`**

JS Challenge 2

1. Create the class definition for an arrow shape.
2. Push it into the `colorBoxArr`

JS Challenge 3

1. Create the class definition for a hexagon shape.
2. Push it into the `colorBoxArr`