Shape Maker

Your Tasks (Mark these off as you go)

- ☐ Define key vocabulary
- ☐ Write a function to create a list
- ☐ Write code to implement a function
- ☐ Write a function to create a rectangle
- ☐ Write a function to create a triangle
- ☐ Write a function to create a circle
- ☐ Receive credit for this lab guide

☐ Define key vocabulary

Function (as it applies to JavaScript)

Abstraction (as it applies to programming)

Function parameter

Function return

Program

☐ Write a function to create a list

Let's suppose you wanted to write a program that enabled a user to create a shopping list. To begin creating your list you would need to do the following,

- Prompt the user for a list item
- Capture the item the user provided and assign it to a variable

Recall that we can a prompt user for input and assign the input to a variable using the prompt command,

```
var item = prompt("add an item");
```

We also learned that we could append variables using the plus (+) sign. The following code illustrates how we could append the user's input to a list. In the code below, we created an empty variable called list. Then we took the item that the user provided and appended it to this variable.

```
var list = "";
list = list + item;
```

Creating a shopping list of five items can be achieved by replicating the code above,

Code	Output
``var list = "";
var item = prompt("add an item");
list = list + item + ",";
var item = prompt("add an item");
list = list + item + ",";
var item = prompt("add an item");
list = list + item + ",";
var item = prompt("add an item");
list = list + item + ",";
var item = prompt("add an item");
list = list + item;
console.log(list);
``	eggs,bread,milk,lettuce,tomatoes App.js:12 >

That is a lot of redundant code to achieve such a simple task. And, what if we wanted separate the items with a space instead of a comma? We would have to change the code in five places. The rule of three states that,

> *you are allowed to copy and paste the code once, but that when the same code is replicated three times, it should be extracted into a new procedure*

A procedure in JavaScript is also referred to a function. A function is a reusable block of code that groups together a sequence of statements to perform a specific task.

In the space below, write a function called *makeList*. *makeList* should prompt the user for an item and add the item to a variable called *list*.

Write a function called *displayList*. *displayList* should create an html element on the screen, then display the contents of the list inside this element. The output to the html page should look as follows,

eggs ,coffee ,lettuce ,tomatoes ,bread ,

☐ Write code to implement a function

A function declaration binds a function identifier to a block of code. However, a function declaration does not ask the code inside the function body to run, it just declares the existence of the function. The code inside a function body runs, or executes, only when the function is called. To call a function in your code, you type the function name followed by parentheses. The code below for example would execute code associated with the function *makeList*.

```
makeList();
```

Write code to implement the *makeList* function you wrote above 5 times. After the fifth time, call *displayList* to display the contents of your list on the screen.

☐ Write a function to create a rectangle

The code below creates a rectangle and displays it on a screen.

Code
| ```
var rectangle = document.createElement("div");
var width = 200;
var height = 100;
var xPos = 10;
var yPos = 10;
rectangle.style.width = width + "px";
rectangle.style.height = height + "px";
rectangle.style.left = xPos + "px";
rectangle.style.top = yPos + "px";
rectangle.style.border = "thick solid #0000FF";
rectangle.style.position = "absolute";
document.body.append(rectangle);
``` |

Output

That is a lot of code to create such a simple element! Moreover, what if we wanted to create many rectangles of different dimensions and place them on different locations of the screen? Our code would get lengthy and difficult to maintain in a hurry!
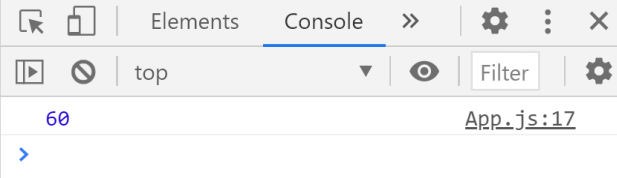
While the code above could certainly be reduced with a function, we can also add customization to our function using parameters. Parameters allow functions to accept input(s) and perform a task using the input(s). We use parameters as placeholders for information that will be passed to the function when it is called.

```javascript
function calculateArea(width, height){

    console.log(width*height);

}
```

In the example above, *calculateArea*, computes the area of a rectangle, based on two inputs, *width* and *height*. The parameters are specified between the parenthesis as *width* and *height*, and inside the function body, they act just like regular variables.

When calling a function that has parameters, we specify the values in the parentheses that follow the function name. The values that are passed to the function when it is called are called arguments. Arguments can be passed to the function as values or variables.

Now, let's consider the flow of data as we call the *calculateArea* function above,

Code	Output

`calculateArea(10, 6);`
 	
`function calculateArea(width, height)`
`{`
 	
`    console.log(width*height);`
 	
`}`	

In the function call above, the number 10 is passed as the width and 6 is passed as height. Notice that the order in which arguments are passed and assigned follows the order that the parameters are declared.

Consider the code block below which creates a rectangle of a fixed size at a fixed location on the screen,

```
var rectangle = document.createElement("div");
var width = 200;
var height = 100;
var xPos = 10;
var yPos = 10;
rectangle.style.width = width + "px";
rectangle.style.height = height + "px";
rectangle.style.left = xPos + "px";
rectangle.style.top = yPos + "px";
rectangle.style.border = "thick solid #0000FF";
rectangle.style.position = "absolute";
document.body.append(rectangle);
```
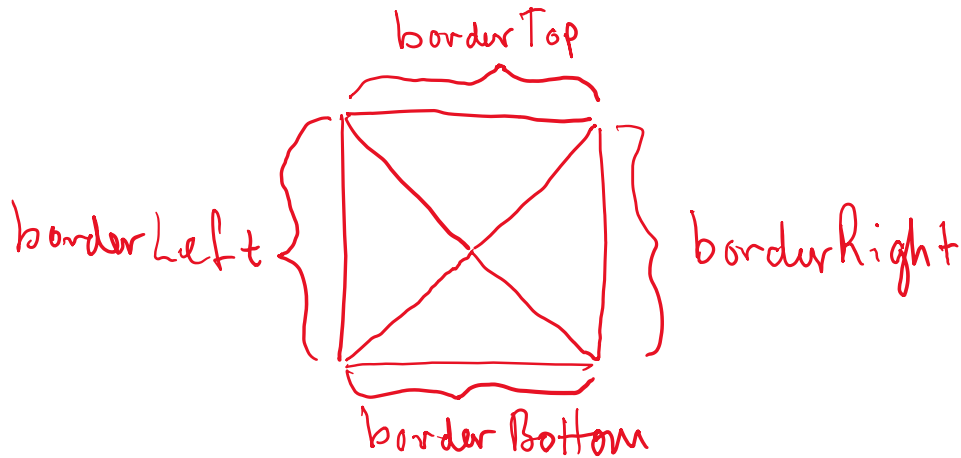
Write a function called drawRectangle that accepts four parameters (*w, h, x, y*). The four parameters represent the *width, height, xPos*, and *yPos* of the rectangle. Use these parameters to set the values of *width, height, xPos,* and *yPos* inside the body of the function.

Now that we have written our rectangle function we can create many rectangles by simply calling our function. Because we have added parameters to represent the *width, height, xPos*, and *yPos*, we can create rectangles of different dimensions and different sizes on the screen.
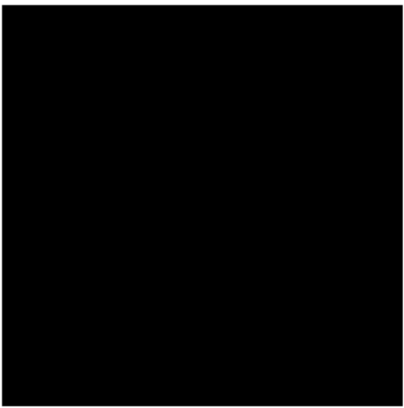
Write code that could be used to call the *drawRectangle* function you wrote above. Create at least three different rectangles with different dimensions and locations on the screen.
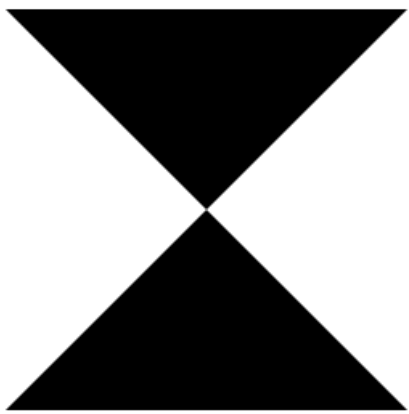
☐ Write a function to create a triangle

Recall that when you create a *div* element we can specify the width of the border. So, imagine a *div* element with a width and height of zero, but with a really thick border.

The code below would create such an element,

Code	Output
```javascript\nvar triangle = document.createElement("div");\nvar width = "100";\nvar xPos = "200";\nvar yPos = "200";\ntriangle.style.borderWidth = width + "px";\ntriangle.style.borderStyle = "solid";\ntriangle.style.borderColor = "black";\ntriangle.style.position = "absolute";\ntriangle.style.left = xPos + "px";\ntriangle.style.top = yPos + "px";\ndocument.body.append(triangle);\n```	

Now let's consider what happens when we set the transparency of different sides. In the code below we have created two triangles by setting the left and right borders to transparent and the top and bottom borders to *null*. Although, we didn't need to set the top and bottom borders explicitly to *null*, this trick will come in handy for creating different triangles using a function.

Code	Output
| ```javascript
var triangle = document.createElement("div");
var width = "100";
var xPos = "200";
var yPos = "200";
triangle.style.borderWidth = width + "px";
triangle.style.borderStyle = "solid";
triangle.style.borderColor = "black";
triangle.style.borderLeftColor = "transparent";
triangle.style.borderRightColor = "transparent";
triangle.style.borderTopColor = null;
triangle.style.borderBottomColor = null;
triangle.style.position = "absolute";
triangle.style.left = xPos + "px";
triangle.style.top = yPos + "px";
document.body.append(triangle);
``` | |

Now, if we set the top or bottom border to transparent we will have a single triangle,

Code	Output
| ```javascript
var triangle = document.createElement("div");
var width = "100";
var xPos = "200";
var yPos = "200";
triangle.style.borderWidth = width + "px";
triangle.style.borderStyle = "solid";
triangle.style.borderColor = "black";
triangle.style.borderLeftColor = "transparent";
triangle.style.borderRightColor = "transparent";
triangle.style.borderTopColor = "transparent";
triangle.style.borderBottomColor = null;
triangle.style.position = "absolute";
triangle.style.left = xPos + "px";
triangle.style.top = yPos + "px";
document.body.append(triangle);
``` | |

Consider the code block below which creates a rectangle of a fixed size at a fixed location on the screen,

```javascript
var triangle = document.createElement("div");
var width = "100";
var xPos = "200";
var yPos = "200";
triangle.style.borderWidth = width + "px";
triangle.style.borderStyle = "solid";
triangle.style.borderColor = "black";
triangle.style.borderLeftColor = "transparent";
triangle.style.borderRightColor = "transparent";
triangle.style.borderTopColor = "transparent";
triangle.style.borderBottomColor = null;
triangle.style.position = "absolute";
triangle.style.left = xPos + "px";
triangle.style.top = yPos + "px";
document.body.append(triangle);
```

Write a function called drawTriangle that accepts three parameters (*w, x, y*). The three parameters represent the *width, xPos*, and *yPos* of the triangle. Use these parameters to set the values of *width, xPos,* and *yPos* inside the body of the function.

Now let's consider a function that allows us to draw different triangles. The following calls to *drawTriangle* creates the corresponding triangles,

```
drawTriangle(100, 200, 200, null, "transparent", "transparent", "transparent");
```



```
drawTriangle(100, 200, 200, "transparent", null, "transparent", "transparent");
```



```
drawTriangle(100, 200, 200, "transparent", "transparent", null, "transparent");
```



```
drawTriangle(100, 200, 200, "transparent", "transparent", "transparent", null);
```
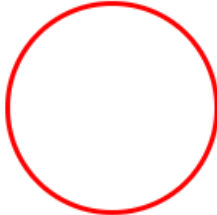

Inside the body of your drawTriangle function, create four new variables to represent the leftColor, rightColor, bottomColor, and topColor.

Now add four additional parameters to the function. Use these parameters to set the values of *leftColor, rightColor, bottomColor*, and *topColor* inside the body of the function.

☐ Write a function to create a circle

A circle can be created from a *div* element by modifying the *borderRadius* property. The code below for example creates a red circle with a diameter of 100 pixels,

Code	Output
``var diameter = 100;
var circle = document.createElement("div");
circle.style.border = "medium solid red";
circle.style.width = diameter + "px";
circle.style.height = diameter + "px";
circle.style.borderRadius = "50%";
document.body.append(circle);```	

Consider the code block below which creates a circle of a fixed diameter at a fixed location on the screen,

```
var diameter = 100;
var xPos = 10;
var yPos = 10;
var circle = document.createElement("div");
circle.style.border = "medium solid red";
circle.style.width = diameter + "px";
circle.style.height = diameter + "px";
circle.style.borderRadius = "50%";
circle.style.position = "absolute";
circle.style.left = xPos + "px";
circle.style.top = yPos + "px";
document.body.append(circle);
```

Write a function called drawCircle that accepts three parameters (*d, x, y*). The three parameters represent the *diameter, xPos*, and *yPos* of the circle. Use these parameters to set the values of *diameter, xPos,* and *yPos* inside the body of the function.

□ Receive Credit for this lab guide

Submit this portion of the lab to Pluska to receive credit for the lab guide.