Name _____ Period _____ Role (Circle one) Programmer/Driver

Name _____ Period _____ Role (Circle one) Programmer/Driver

Creating Functions

Your Tasks (Mark these off as you go)

- ☐ Explain the purpose of a function
- ☐ Declare a function
- ☐ Call a function
- ☐ Have Ms. Pluska check off the above tasks
- ☐ Pass a parameter to a function
- ☐ Use a *return* statement in a function
- ☐ Have Ms. Pluska check off the above tasks
- ☐ Brainstorm a program
- ☐ Receive credit for the group portion of this lab

☐ Explain the purpose of a function

In this lab you will learn how to group and name your own procedures (or "functions"). Grouping and naming commands for easy and repeated use is a form of abstraction. Abstractions enable the programmer to reduce complexity by removing details and generalizing functionality.

Consider the problem of calculating the area of a rectangle. When first learning how to calculate the area of a rectangle, there's a sequence of steps to calculate the correct answer:

1. Measure the width of the rectangle.
2. Measure the height of the rectangle.
3. Multiply the width and height of the rectangle.

With practice, you can calculate the area of the rectangle without being instructed with these three steps every time.

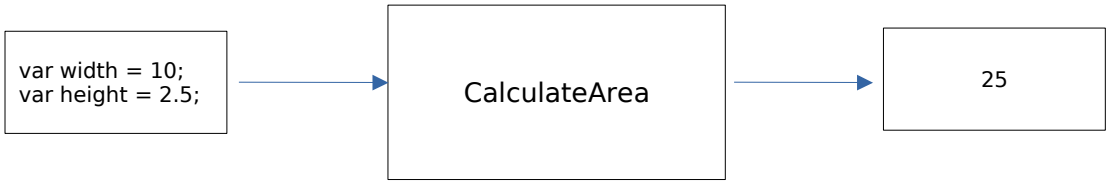
We can also calculate the area of one rectangle with the following code:

```
var width = 10;
var height = 6; var
area = width * height;
console.log(area); // Output: 60
```

Now, Imagine being asked to calculate the area of three different rectangles. To do so would require us to repeat the above code three times.

In programming, we often use code to perform a specific task multiple times. Instead of rewriting the same code, we can group a block of code together and associate it with one task, then we can reuse that block of code whenever we need to perform the task again. We achieve this by creating a ***function***. A function is a reusable block of code that groups together a sequence of statements to perform a specific task.

It is possible to implement a function without understandng the the underlying complexity. For example, once the calculateArea function is defined, it should be possible to calculate the area of many rectangles without understanding how the calculation takes place. This process is illustrated below. Moreover, because functions help us manage the complexity of our code they are also referred to as ***abstractions***.



□ Declare a function

In JavaScript, there are many ways to create a function. One way to create a function is by using a function declaration. Just like how a variable declaration binds a value to a variable name, a function declaration binds a function to a name, or an identifier. Take a look at the anatomy of a function declaration below:

```
function greetWorld(){
     console.log(“Hello World!”);
}
```

Each portion of the above function is described below,

- function – the keyword used to create a the function.
- `GreetWorld()` - the name of the function, or its identifier, followed by parentheses.
- `console.log(“Hello World!”);` - the function body, or the block of statements required to perform a specific task, enclosed in the function’s curly brackets, { }.

(a) Using a function declaration, create a function called getReminder() that prints a reminder to the console. In the function body of getReminder(), log the following reminder to the console: 'Water the plants.' (b) Using a function declaration, create a function called greetInSpanish(). In the function body add a console.log() that prints: 'Buenas Tardes.'

□ Call a function

As we saw above, 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.

```
greetWorld();
```

The function call above executes the function body, or all of the statements between the curly braces in the function declaration below.

```
function greetWorld(){
     console.log(“Hello World!”);
}
```

We can call the same function as many times as needed.

Imagine that you manage an online store. When a customer places an order, you send them a thank you note. Let's create a function to complete this task: (a) Define a function called sayThanks() as a function declaration. (b) In the function body of sayThanks(), add code such that the function writes the following thank you message to the console when called: 'Thank you for your purchase!' (c) Call sayThanks() to view the thank you message in the console. (d) Functions can be called as many times as you need them. Imagine that three customers placed an order and you wanted to send each of them a thank you message. Update your code to call sayThanks() three times.

□ Have Ms. Pluska check off the above tasks

Before you continue have Ms. Pluska check off the above tasks

Do not continue until you have Ms. Pluska's (or her designated TA's) signature _____

□ Pass a parameter to a function

So far, the functions we've created execute a task without an input. However, some functions can take inputs and use the inputs to perform a task. When declaring a function, we can specify its 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.

```
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. width and height act as placeholders for values that will be multiplied together.

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.

```
calculateArea(10, 6);
```

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.

```
var rectWidth = 10;
var rectHeight = 6;

calculateArea(10, 6);
```

The variables rectWidth and rectHeight are initialized with the values for the height and width of a rectangle before being used in the function call.

By using parameters, calculateArea() can be reused to compute the area of any rectangle! Functions are a powerful tool in computer programming so let's practice creating and calling functions with parameters.

(a) Add a parameter called name to the function declaration for sayThanks().
(b) With name as a parameter, it can be used as a variable in the function body of sayThanks(). Using name and string concatenation, change the thank you message into the following:

```
'Thank you for your purchase '+ name + '! We appreciate your business.'
```

(c) A customer named Cole just purchased something from your online store. Call sayThanks() and pass 'Cole' as an argument to send Cole a personalized thank you message.

□ Use a *return* statement in a function

When a function is called, the computer will run through the function's code and evaluate the result of calling the function. By default that resulting value is undefined.

```
function rectangleArea(width, height){
     var area = width * height;
}

console.log(rectangleArea(5, 7); //prints undefined
```

In the code example, we defined our function to calculate the area of a width and height parameter. Then rectangleArea() was invoked with the arguments 5 and 7. But when we went to print the results we got undefined. Did we write our function wrong? No! In fact, the function worked fine, and the computer did calculate the area as 35, but we didn't capture it. So how can we do that? With the keyword return!

```
function rectangleArea(width, height){
     var area = width * height;
     return area;
}

console.log(rectangleArea(5, 7); //prints 35
```

To store information from a function call, we use a ***return*** statement. To create a return statement, we use the ***return*** keyword followed by the value that we wish to return. Like we saw above, if the value is omitted, undefined is returned instead.

When a return statement is used in a function body, the execution of the function is stopped and the code that follows it will not be executed.

The return keyword is powerful because it allows functions to produce an output. We can then save the output to a variable for later use. Consider the example below,

```
function rectangleArea(width, height){
     var area = width * height;
     return area;
}

var myArea = rectanglearea(10, 5);

console.log(myArea);//prints 50
```

Image we want to prompt a user for series of five numbers that we then want to add together. We could use a functions to help us do this!
(b) Declare two variables: num and sum. Initialize sum to zero.
(a) Write a function called getNumber that prompts a user for a number, assigns the value to num, then returns the value of num as a number.
(c) Write another function called addNumbers that accepts a parameter, which represents the number the user provided, then adds it to the variable sum (sum = sum + n)
(d) Implement your functions above to prompt the user for five numbers, and then add them together.
(e) Print the final result to the consol

□ Have Ms. Pluska check off the above tasks

Before you continue have Ms. Pluska check off the above tasks

Do not continue until you have Ms. Pluska's (or her designated TA's) signature

_____

□ Brainstorm a program

On a separate sheet of paper, brainstorm a program that allows a user to create a shopping list. Begin your program by declaring a variable called *list* and another variable called *item*.

Write a function called getItem that prompts the user for an item and returns the value. Assign the result of getItem to the variable item.

Write another function called addItem that accepts a parameter. The parameter is the item the user provides above. Each time addItem is called, the item should be added to the list. You should separate the items on the list using some kind of delimter (spaces, commas, new line ("\n"), etc).

Your program should prompt the user for at least five items.

A final function called displayList, should alert the user of their final shopping list. For example,

```
alert("here is your list " + "\n" + list);
```

Compare your ideas with your partner, then obtain a large piece of butcher paper and a marker. Write your final code on this paper.

□ Receive Credit for the group portion of this lab

- Indicate the names of all group members.
- Make sure both you and your partner have completed the above tasks
- Have Ms. Pluska check off the group tasks
- Submit your lab to the needs to be graded folder to receive credit for the group portion of this lab.
- Do not submit your lab until you have Ms. Pluska's (or her designated TA's) signature

_____