FUNCTIONS

A function is a block of code that performs a SPECIFIC TASK. We have already made use of some built in functions – alert, prompt, Math.floor and others. A function takes the following form:

```
function  functionName(parameters) {
        code to be executed
}
```

The functionName is a descriptive name of the task performed by the function. The rules for naming a variable apply to a function as well (can't start with a number, no special characters, no keywords). Data is given, or passed, to a function through **PARAMETERS**. A variable or value passed to a function is called an **ARGUMENT.**
The parameters of a function are listed in the parenthesis after the functionName and are the values the function NEEDS to run. As an example, consider the heading for the alert function. It would look like this:

```
function alert(message) {
        //code to make a box pop up with message
}
```

message is a parameter of alert - alert needs to know what you want to put in the popup, so you pass it into alert in the parenthesis

When we invoke (call) the alert function we use the function name and include the argument(s) to match the parameter(s) it needs.

```
alert("Hey what is up?");
```

- *alert* is the function name
- "Hey what is up?" is the argument passed into the function; It gets passed into the variable **parameter** *message*

```
Math.floor(10 * Math.random() + 5);
```

In this line of code, it calls two *functions – floor* and *random*, they are coming from the class (where they are stored) called Math

Functions can also be user-defined. Functions simplify a program by breaking code into smaller, more manageable blocks. A function reduces redundancy (copy and paste) by performing a task that may need to be executed many times throughout the program.
A function must be called from another procedure in order to execute.

Consider the following:

```
//display a response for the user
function coinToss(){

  var n = Math.floor(2*Math.random() +1);
  if (n==1) {
    alert("heads");
    }
  else {
    alert("tails");
  }
}

//call coinToss()
coinToss();
```

```
//BETTER – send a response back (use return)
and the program can use it how it needs it
function coinToss(){

  var n = Math.floor(2*Math.random() +1);
  if (n==1) {
    return "heads";
    }
  else {
    return "tails"
  }
}
document.write("Coin flip is ... "+ coinToss());
alert("Coin flip is ... "+ coinToss());
```

Example: Create a function that will return the Discriminant (D=b*b - 4*a*c)

```
// What does the Discriminant need?  An a, b, and c value.  What does it send back? A number.

function discriminant(a,b,c){
  return b*b-4*a*c;
}
//use the function
document.write("D = " + discriminant(1,5,6));
document.write("D = " + discriminant(prompt("Enter a value"), prompt("Enter b value"), prompt("Enter c value")));
```

Consider our guessing game. We needed to repeatedly check to see if the user guessed correctly. We could include a function that checks the guess and returns the result. It is often better to send back (return) a response to the procedure that called the function than to output it directly in the function. That way you can do what you want with the result of the function – you can calculate values and use them when you want to. Use the return statement to send back a result.

```
function checkNumber (guess, num) {
  if (num==guess){
    return "Correct";
  }
  else if (num>guess){
    return "Too Low";
  }
  else {
    return "Too High";
  }
  }//end of function
```

```
//outside of the function, our main script CALLS checkNumber
var randomNumber = Math.floor(10* Math.random() + 1);
var userGuess = prompt("Guess a number between 1 and 10");

alert(checkNumber(userGuess,randomNumber));
```

When a function is called the ARGUMENTS must match the PARAMETERS. We are trying to make functions INDEPENDENT from the rest of the program. It should not rely on any information EXCEPT WHAT IT IS PASSED IN THE PARAMETERS. Generic functions can be easily reused in other programs, making coding more efficient.

The following points are important to keep in mind when working with functions that have parameters:

- The order in which arguments are passed is important because the order of the arguments corresponds to the order of the parameters. Therefore, the first argument in the function corresponds to the first parameter and so on. checkNumber(randomNumber, userGuess) is not the same as checkNumber(userGuess, randomNumber)
- Arguments can be in the form of constants, variables, values or expressions. Could also use
  var answer = checkNumber(prompt("Enter a guess"), randomNumber);
  var answer = checkNumber(2,7);

KNOWLEDGE REVIEW:

Given the function heading, indicate the parameters, if any. Write an appropriate call to the function

```
function letterGrade(score) {//returns a letter grade…}
```

```
function bigNumber(num1, num2) {//returns the larger number…}
```

```
function pickaNum(low, high) {//returns a number in the range of low to high…}
```

```
function sayHello() {//shows an alert with Hello…}
```

PRACTICE EXERCISES:

Create a new folder inside your JavaScript folder entitled Functions (save as f1.html, f2.html, f3.html)

1. Write a function that will return the area of a circle given the radius as a parameter.
2. Write a function that will return a random secret word. Include at least 4 choices.
3. Write a function that will generate a random math question.

DATA INPUT WITH FORMS

```
<body>
<script>
function recArea() {
    var l = document.getElementById("myLength").value;
    var w = document.getElementById("myWidth").value;
    document.getElementById("areaParagraph").innerHTML = l*w;
    }
</script>
<form>
Calculate the Area of a Rectangle: <br><br>
  Length<br>
  <input type="text" id = "myLength" value=20>
  <br>
  Width<br>
  <input type="text" id = "myWidth" value=4>
  <br><br>
 <button type="button" onclick="recArea()">Calculate Area</button>
    <br>
</form>
<br>
The area is:
<p id="areaParagraph"></p>

</body>
```

Get the value from the input with id "myLength"

Find in the HTML "areaParagraph" and display the area

When the button is clicked it will call the function recArea()

HTML paragraph with id

PRACTICE EXERCISES (save as f4.html, f5.html):

4. Add to the example and create a buttons to calculate the perimeter of the rectangle and any other 2 calculations.

5. Take a look at other form elements: https://www.w3schools.com/html/html_form_elements.asp
   Create a new form that will use input boxes, radio buttons, select element (pull down list) and buttons. Allow the user to enter information and when the button is clicked, output some result. Include a minimum of 10 elements in your form. Items could include name (text), age (text or range), birthday (drop down menu for month, day, year), grade (drop down or text), gender (radio buttons) any other 3 questions/items

Strings

Recall that a String is a variable-type that stores a sequence of characters. Javascript includes several built-in functions for working with strings. The following is a list of some string functions available in JS. Your task is to investigate the functions and experiment with each function in order to understand it. Create a file demonstrating the use of each function.

In Javascript, every character in a String can be referred to with an *index*, with the first character having an index of 0. **For example**, in the String "ICS20", the character 'C' would have an index of 1.

In addition, all String functions ***return*** a new string and **do not** modify the original variable. Formally, we then say that Strings are ***immutable***, meaning they cannot be changed directly through a function.

A complete list of available String functions can be found at www.w3schools.com.

substring(start, end) or substring(start)

⇒ returns the string starting from the index *start* and ending one index before *end*. If *end* is not included, the substring will include all characters in the String from *start*.

```
Ex:     var title = "I love this class";
        var part = title.substring(2,6);        //part has the value 'love'
```

Also, notice that to find the substring, we had to call the function using the String title. This will be used with almost all String functions.

indexOf(search, index) or indexOf(search)

⇒ returns the starting position of the *first occurrence* of the substring *search*. If the substring is not found, -1 is returned. The parameter *index* is the position to start searching. If *index* is not included, searching automatically starts at index 0.

```
Ex:     var word = "brighten";
        var search = "right";
        var result = word.indexOf(search);    //result has the value 1
```

lastIndexOf(search)

⇒ returns the starting position of the *last occurrence* of the substring *search*. Works similarly to indexOf() above.

replace(find, replacement)

⇒ returns a new String with the first occurrence of *find* replaced by *replacement*.

```
Ex:     var word = "Microsoft";
        var replacement = "HA";
        var result = word.replace("o", replacement);        //result has the value 'MicrHAsoft'
```

toLowerCase() or toUpperCase()

⇒ returns the string argument as an all lowercase or uppercase string.

```
Ex:     var original = "brighten";
        var nowUpper = original.toUpperCase();
```

length

⇒ the length property returns the number of characters in a string.

includes(search)

⇒ returns true if the given String includes the substring search. Returns false otherwise.

startsWith(search) or endsWith(search)

⇒ returns true if the given String begins are ends with the substring *search*. Returns false otherwise.

charAt(index)

⇒ returns the character located at a certain index of a String.

charCodeAt(index)

⇒ returns the ASCII code corresponding to a character at a certain *index*.

Ex: var word = " super" ;
var character = word.charCodeAt(0) ; //character has the value 115

115 is the decimal value of the character 's' in the ASCII table. Check for a link to the ASCII table on the website.

String.fromCharCode(integer)

⇒ returns the character corresponding to the *integer* between (0-255) using the ASCII table.

Ex: var result = String.fromCharCode(115) //result has the value 's'

Notice that use the variable name String when calling this function, as opposed to the name of a existing variable.

String Comparisons (>, <, ==, !=, >=, <=)

⇒ You are able to use comparison operators to compare strings. Javascript compares the ASCII codes of the starting character in each String.

⇒ String comparisons can be tricky because comparisons using operators like <, >, == are case sensitive since they are using the ASCII code to compare them. Ex: the comparison "Apple" == "apple" is false because the uppercase "A" and lowercase "a" are represented by two different ASCII codes.

THE FOR LOOP

The **for** statement is a looping structure that executes a set of statements a fixed number of times. In computer science, this is known as *iteration*. The **for** statement executes until a counter reaches an ending value. The **for** statement takes the following form:

```
for (counter = 1; counter <= 10; counter++)
{
        Statements you want to loop
}
```

There are 3 main parts of a **for** loop:

1. **Initialization** – the counter variable used to count the loop is set to an initial value. This can be set with a value or a variable.

2. **Boolean Expression** – controls the loop by testing the counter variable. If the expression is *true*, the loop runs again, if the expression is *false*, the loops stops running.

3. **Modifier** – this statement changes the counter variable after the statements in the loop finish. `counter++` has the same effect as `counter = counter + 1`.

Example 1: In this case, 10 alerts in all will be displayed showing the numbers 1 to 10.

```
for (counter = 1; counter <= 10; counter++)
{
      alert(counter);
}
```

Example 2: In this case, 5 alerts in all will be displayed showing even numbers counting down from 10 to 2.

```
for (i = 10; i >= 2; i = i - 2)
{
      alert(i);
}
```

Before moving on, experiment with different starting values, boolean expressions and modifier values. Try different loops that count up (increment) or down (decrement), as well as, a variety of operations (+,-,*,/).

INFINITE LOOPS

The condition of a loop is used to determine when the loop should stop executing. A for loop continues until its condition is false. What happens, though, if the condition never becomes false? The result is an infinite loop - one which continues forever.
A logic error can lead to an infinite loop. For example, the following statements create an infinite loop. Can you see why?

```
for (i = 10; i >= 2; i++)
{
      alert(i);
}
```

ACCUMULATOR PATTERN

Sometimes when we are iterating through a sequence of numbers, it can be useful for us to count something. The accumulator pattern allows us to keep a 'running total' as a loop runs. To do this, we must initialize a variable outside of the loop that will store the sum. As the loop runs, we will increment that sum as needed. After the loop finishes, it will store the result of our counting.
Here is an example:

```
var sum = 0;
var input = 0;
alert("Please enter 10 numbers, and I will tell you the sum!");

for (i = 1; i <= 10; i++)
{
      input = Number(prompt("Enter number " + i);
      sum = sum + input;
}

alert("Your sum is " + sum);
```

PRACTICE EXERCISES

1. **FOR COUNTER:** Use a for loop to output a range numbers determined from what the user enters.
    a. Every Number. If the user enters: 1 and 10, you would output 1,2,3,4 ..10.
    b. Even Numbers Only. If the user enters 1 and 10, you would output 2, 4, 6, …10.
    c. From highest to lowest. If the user enters 1 and 10, you would output 10, 9, 8, …1

2. **FOR SUM:** Write a for loop that will sum the numbers given a user entered start and stop value. Example: the user enters 5 and 9. You will add them =>5 + 6 + 7 + 8 + 9=35.

3. **FOR NUM WORDS:** Use a for loop to count the number of WORDS in a sentence.

THE WHILE AND DO-WHILE LOOP

The **while** statement is a looping structure that executes a set of statements while a *boolean expression* is true. If the expression is false, the loop is exited. The *boolean expression* is checked before the statements in the loop execute. The **while** statement takes the following form:

```
while (boolean expression)
{
        Statements you want to loop
}
```

The **do-while** statement is a almost identical to the while statement, except that the *boolean expression* is checked *after* the statements in the loop execute. Also, notice that a semi-colon is included after the while statement in a **do-while** loop. The **do-while** statement takes the following form:

```
do
{
        Statements you want to loop
}
while (boolean expression);
```

Example 1: In this case, 10 alerts in all will be displayed showing the numbers 1 to 10.

```
var counter = 1;

while(counter <= 10)
{
      alert(counter);
      counter++;
}
```

Notice that the counter variable needs to be declared BEFORE the loop is created. This is because any variable you create INSIDE of the loop cannot be checked in the while condition. The following code would NOT work because counter was declared INSIDE of the loop.

```
while(counter <= 10) //does not work!
{
      var counter = 1;
      alert(counter);
      counter++;
}
```

Example 2: Here is an example of a do-while loop that will accomplish the same task as Example 1.

```
var counter = 1;

do
{
      alert(counter);
      counter++;
}
while(counter <= 10);
```

INFINITE LOOPS

Infinite loops can also be created in while loops as well. The following loops would be infinite. Can you see why in each case?

```
var counter = 1;

while(counter <= 10)
{
      alert(counter);
}
```

```
var counter = 11;

while(counter >= 10)
{
      alert(counter);
      counter++;
}
```

SENTINEL PATTERN

The sentinel pattern in computer science is used to determine when a loop will terminate (stop iterating). The sentinel value is a variable that is used to accomplish this. A certain value (*true/false*) will indicate the loop should continue to run. When the sentinel value changes, the loop will terminate. The sentinel value changes when the loop has accomplished the particular task it is being used for.

```
var finished = false; //this variable will store the sentinel value
var sum = 0;
alert("This program will calculate the sum of the given numbers.  Enter 0 to terminate.");

while (!finished) //this line literally reads as "while NOT finished"
{
      input = Number(prompt("Please enter a number"));
      if (input == 0)
        finished = true; //this changes the sentinel value in order to exit the loop
      else
        sum = sum + input;
}

alert("Your sum is " + sum);
```

TRACING A LOOP

When using loops, a useful skill is to be able to trace a loop as it runs. This means keep track of values relevant to the loop as they change to help understand how the loop is working. Here is an example:

Loop	**Trace Steps to show each Calculation**	

var num1, num2, answer;
num1 = 11;
num2 = 5;
while (num1 > num2)
{
answer = num1 – num2;
alert(answer);
num1 = num1 - 3;
}
	1. num1 = **11**
	2. num2 = **5**
	3. num1 > num2 = 11 > 5 is true
	4. answer = 11 – 5 = **6**
	5. num1 = 11 – 3 = **8**
	6. num1 > num2 = 8 > 5 is true
	7. answer = 8 – 5 = **3**
	8. num1 = 8 – 3 = **5**
	9. num1 > num2 = 5 > 5 is false
	10. loop ends
		Notice that the tracing the code involves recording the execution of each line and keeping track of the variable values.
		
		This skill is useful because it allows you to manually check your code to understand how it is working, or to find a bug if one exists.

ADDITIONAL INFORMATION

For additional information about for loops, feel free to check out https://www.w3schools.com/js/js_loop_while.asp.

CHECK FOR KNOWLEDGE

1. Why would you use a for loop over a while loop? Give an example.

2. Trace the following pieces of code and describe:
   a) What is the value of num1, num2 and answer when the loop begins?
   b) What are the values of answer on the first, second and last pass of the loop?
   c) What is the value of num1 when the loop stops – if it does stop?

var num1, num2, answer;
num1 = 1;
num2 = 10;
while (num1 < num2)
{
answer = num2 – num1;
alert(answer);
num1 = num1 + 2;
}	

var num1, num2, answer;
num1 = 0;
num2 = 20;
do
{
answer = num2 – num1;
alert (answer);
num1 = num1 + num1;
}
while (num1 < num2);	

var num1, num2, answer;
num1 = 10;
num2 = -10;
while (num1 < num2)
{
answer = num2 – num1;
alert (answer);
num1 = num1 + num1;
}	

var num1, num2, answer;
num1 = -10;
num2 = 10;
while (num1 < num2)
{
answer = num2 – num1;
alert (answer);
num1 = num1 + num1;
}	