Objective

- Put into practice what you have learned about writing JavaScript, including functions, variables, arrays, operators, if statements, do ... while loops, and more.

Instructions

DO NOT START ON THE ASSIGNMENT PRIOR TO YOUR WORKSHOP

- Read the instructions and watch the assignment video prior to your workshop, but do not start on it until you are at the workshop for this week.

Assignment Overview

For this assignment, you will write a color guessing game.

This game is similar to the number guessing game exercise from earlier this week, but there are some significant differences. For example, that game involved the generation of one single random number, and no arrays were used in that game. This assessment task requires an array of colors e.g. ['aqua', 'black', 'cyan', . . .] and the target color which the player has to guess will be a randomly selected color from that array.

Another difference is that you will use a single HTML file with the JavaScript inside the **script** element, rather than an external JavaScript file. This is to make it easier for your instructor to grade your assignment. Best practice in the real world is still to use external JavaScript.

JavaScript

JavaScript was created by Netscape programmer Brendan Eich. Netscape was looking for a client-side scripting language.

- Netscape introduced JavaScript (first as “Mocha” then "Livescript"). Its official name is “ECMAScript”
- It was renamed JavaScript in 1995. (it was first used to validate forms… remember most users had 28.8 kbps modems)

It is not a 'true' Object Oriented language – it is object-based (prototype)

JavaScript is also ASCII based, so you can use almost any editor to write it.

JavaScript is not compiled; it is interpreted by the browser. (All modern browsers)

JavaScript Terms and Definitions: Variables, Arrays, Functions, Properties, Methods, Events, Data Types…

JavaScript is its own (object-based/*object-oriented*) **language**

It is not HTML (but it has 1 HTML Tag/element: <script></script>)

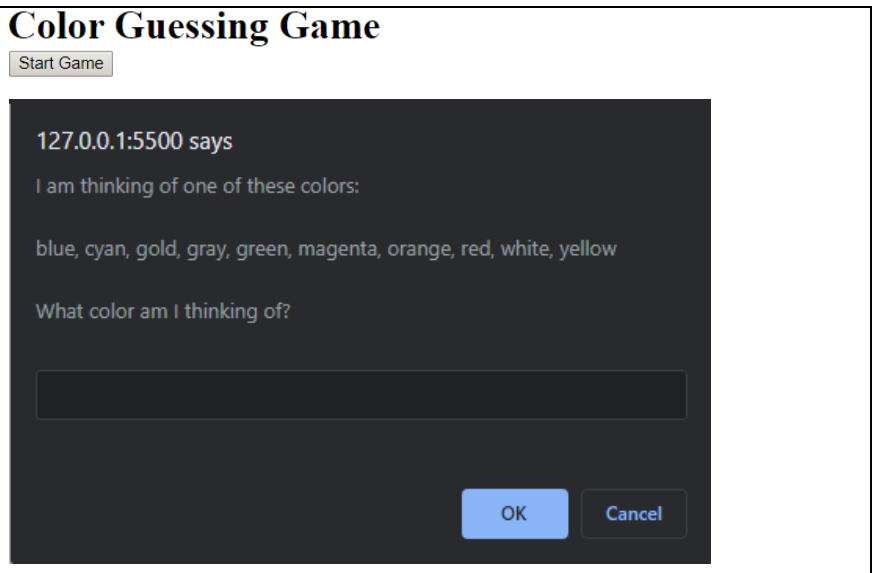
It is not Java (but it is similar in syntax to Java)

It is the same language used with **Node.js** for Server-side/middle-tier programming (next semester).

Here is a quick overview of the game

You will set up an HTML page with a **button** that, when clicked, will begin the game with a prompt that looks like the screenshot below.

```
<button type="button" onclick="runGame()">Start Game</button>
```



The appearance may be different depending on your browser and settings, but the text should be more or less as follows:

As you can see, the objective of the game is for the player to guess the color your program is thinking of. The player needs to enter their guess, such as *cyan*.

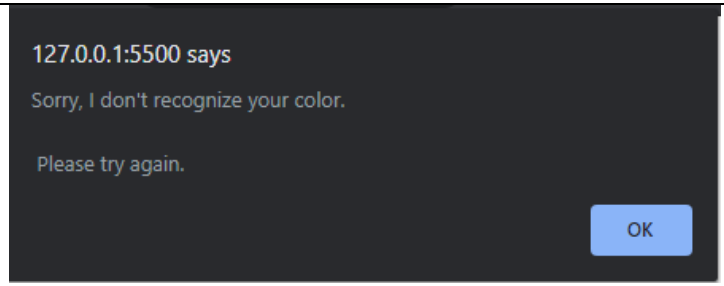
```
guess = prompt('I am thinking of one of these colors:\n\n' +
            COLORS_ARRAY.join(', ') + '\n\nWhat color am I thinking of?\n');
```

A response from the browser will then be shown, as follows:

Four possible reactions to the prompt…

1. **If the color** entered by the player is **not in the pre-set array** of colors used by the game, an appropriate message such as this should be shown:

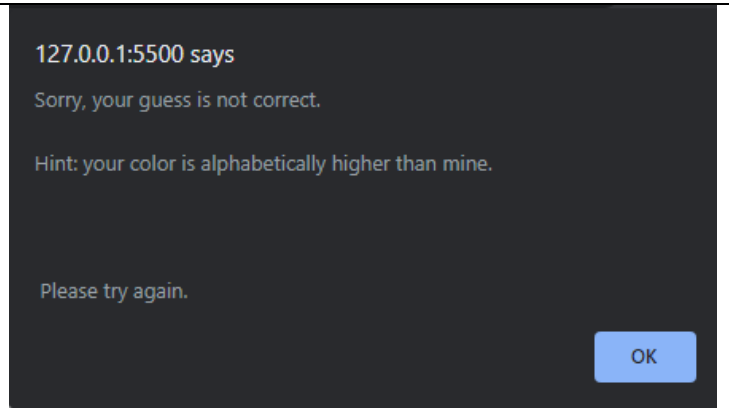
```
    alert('Sorry, I don\'t recognize your color. ' + tryMsg);
```



`tryMsg` is just a reusable js var string that says “Please Try again”

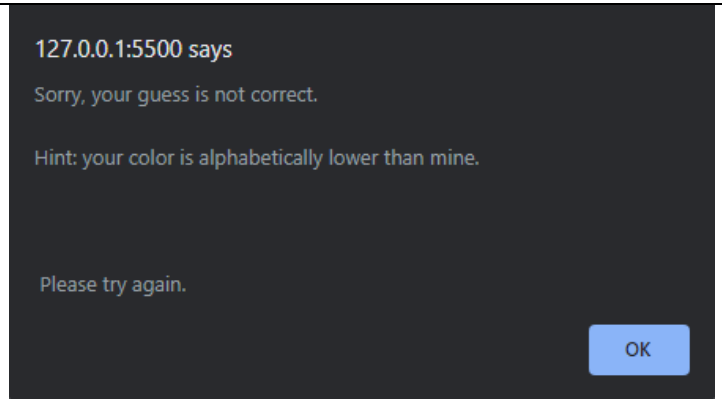
2. **If the color** entered by the player is in the pre-set array of colors, but the color entered by the player **is alphabetically higher** than the answer, a message such as this should be shown:

```
alert(sorryMsg + 'Hint: your color is alphabetically higher than mine.\n\n' + tryMsg);
```



3. **If the color** entered by the player is in the pre-set list of colors, but the color entered **is alphabetically lower** than the answer, a message such as this is shown:

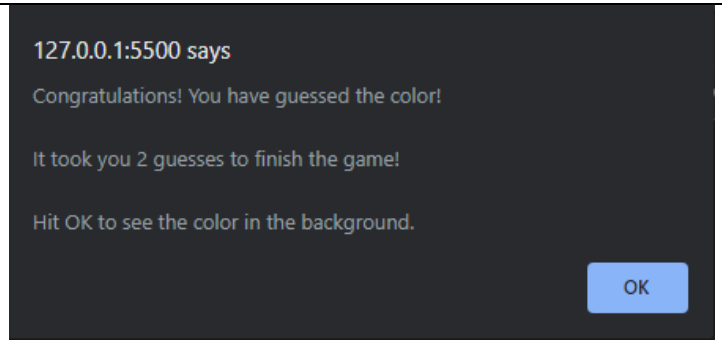
```
alert(sorryMsg + 'Hint: your color is alphabetically lower than mine.\n\n' + tryMsg);
```



4. **If the color** entered by the player **is correct**, an appropriate message will be shown, as seen below, along with a count of the total number of guesses. Once the player clicks OK, the color of the webpage background will be changed to that color.

```
alert('Congratulations! You have guessed the color!\n\n' +
        'It took you ' + numTries + ' guesses to finish the game!\n\n' +
        'Hit OK to see the color in the background.');
```

`numTries` is a js variable that holds the total number of guesses



Flowcharts

Next page shows a flowchart for the game, showing the basic behavior.

Intro to Flowcharts

Basic Flowchart Symbols

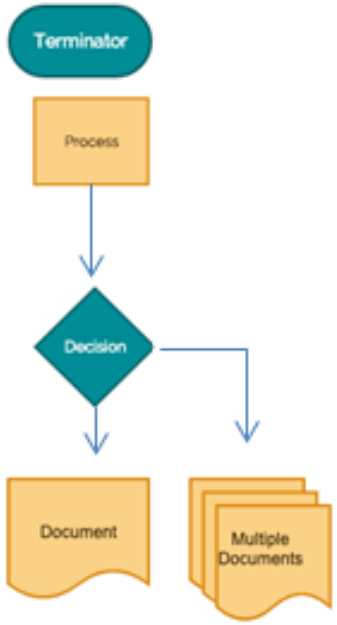
The oval, or **terminator**, is used to represent the start and end of a process.

The rectangle is used to capture process steps like basic tasks or actions

The arrows are used to guide the viewer along their flowcharting path

The diamond symbolizes that a **decision** is required to move forward.

This is generally binary

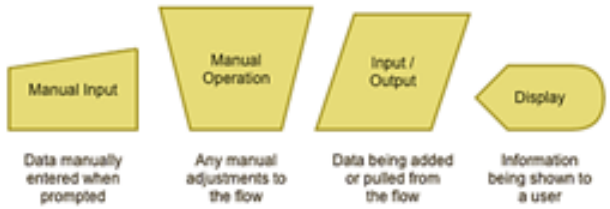


Other Symbols

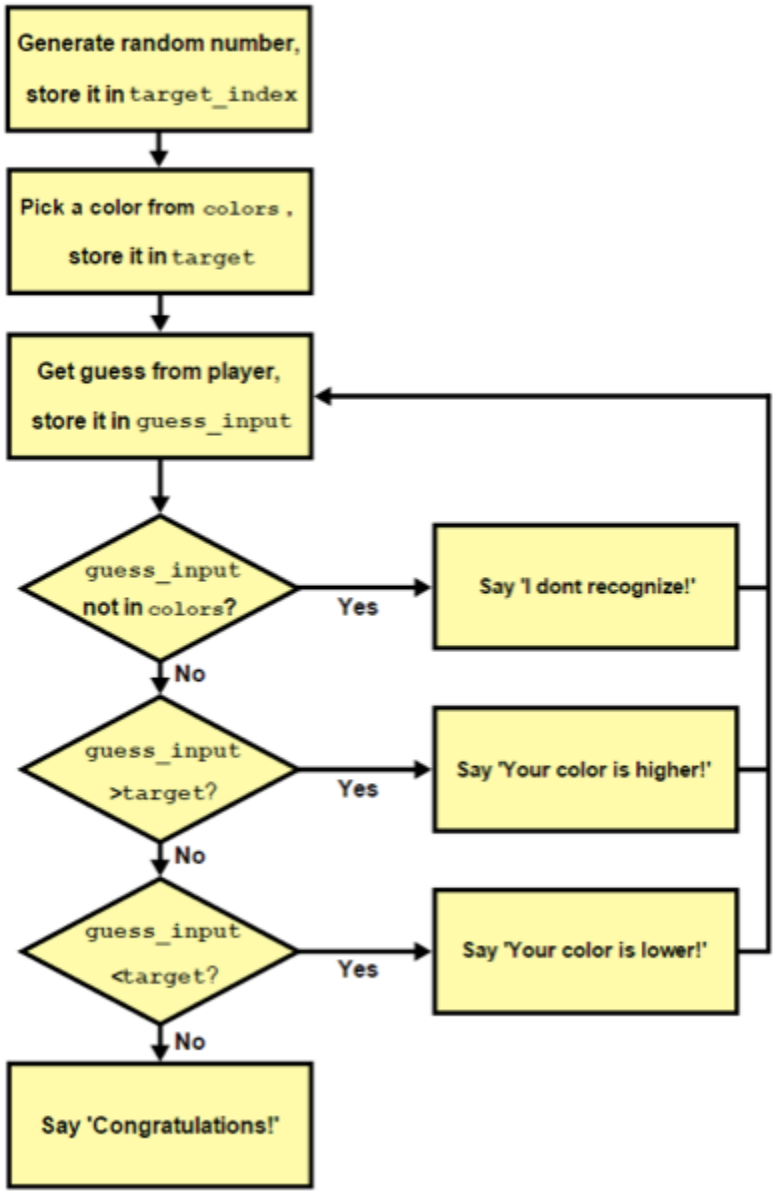
Document(s)

Data Symbols

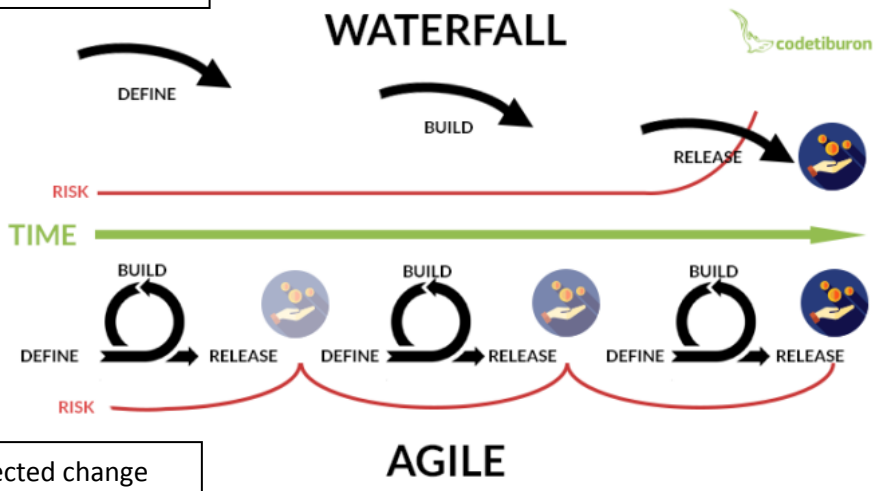
Input & Output



Traditional Flowchart

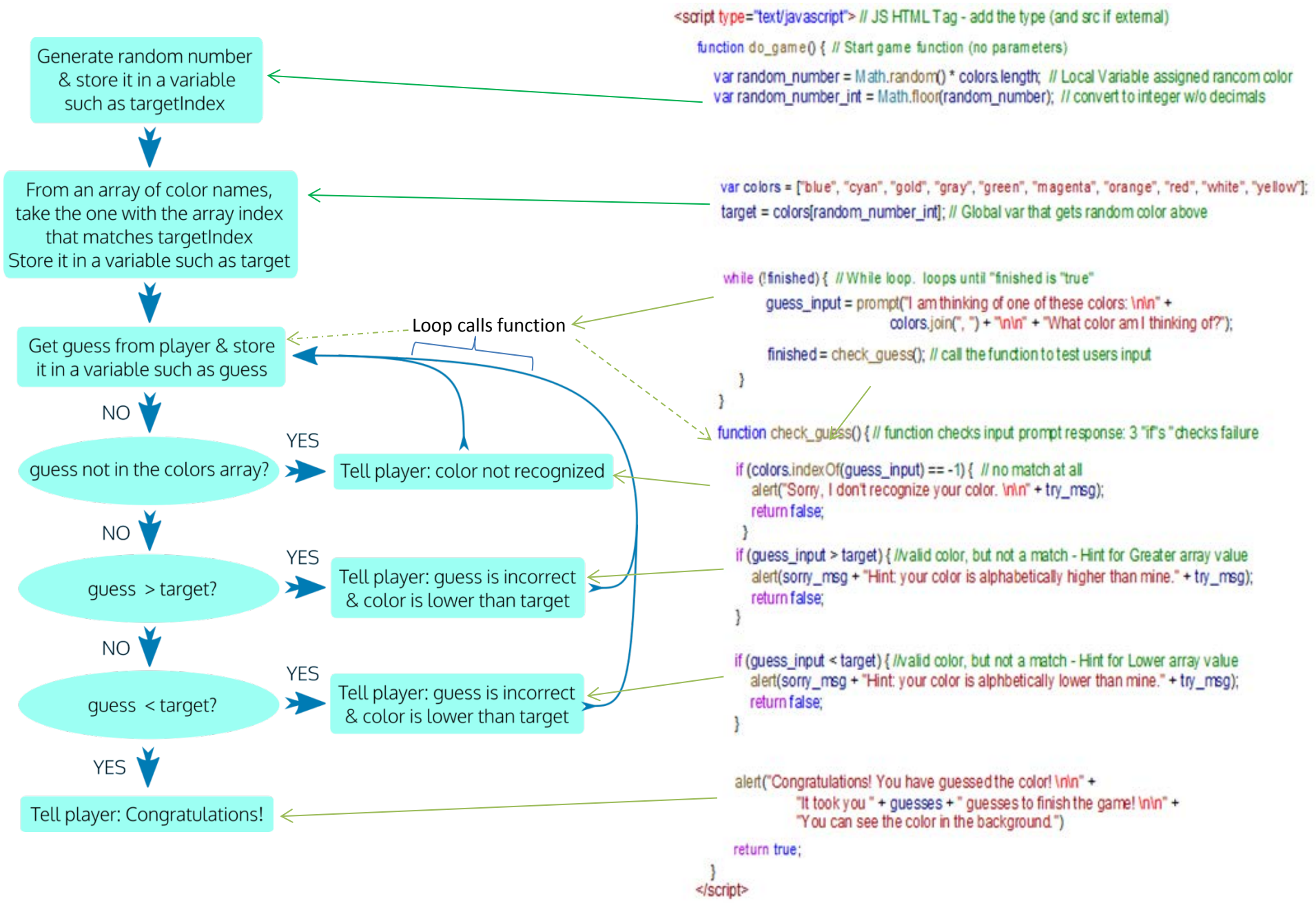


No-fuss solution



Expected change

Non-Traditional Flowchart



Assignment Tasks

To help you organize your approach, detailed instructions are provided below in the form of 2 main tasks, each separated into smaller parts.
Summary of Task 1: Set up the HTML page and the core JavaScript for the game so that you can play the game, make guesses, and receive appropriate responses.
Summary of Task 2: Update the code to add extra features, including displaying the final number of guesses and changing the background color when the correct guess is made.

How JavaScript (JS) is run

There are **three** ways to run JavaScript

1) **JavaScript can be *linked* from your HTML Document to an external ".js" file**

Note: JS Has 1 HTML Tag <script></script> *(There are actually two – there is the <noscript></noscript> tag for user w/o js.*
You Do Not need to use the *type* attribute (<script type="text/javascript">) *It is now the default, but In the old days it was required.*
For external files, the *source* attribute is required (<script **src**="*somelocation.js*">)
Example of an External File:

```
<script type="text/javascript" src="javaScriptsFolder/someJSfile.js"></script>
```

Inside the external file (in the "js" folder, under the file name someJSfile.js) you simply add your script:

```
function someScript(){
    alert("hello");
}
```

2) **JavaScript can be *embedded* into the head or body of your HTML Document**

To embed a JavaScript you must use the special HTML <script></script> tags
These tags are generally placed in the end of the body element, but may also be placed in the head element. Example of an Embedded JavaScript:

```
<script>
    function someScript(){
        alert("hello");
    }
</script>
```

3) **JavaScript can be added as an *inline* attribute to an HTML tag**

Besides using the HTML event attributes as triggers for JavaScript functions, you can also embed your scripts directly into the attributes.
This is not generally recommended, but is often used for short scripts such as "back buttons" or "mouse over" actions.
Examples of an Inline JavaScript:

```
<button type="button" onclick="runGame()">Start Game</button>
<a href="javascript:history.go(-1)">Back Button</a>
```

TASK 1: Set up the HTML and core JavaScript for the game

Part 1: Set up the HTML

- Create a basic valid HTML5 document as you learned to do in Week 1, and give it the name **color-guessing-game.html**. Be sure to set the **DOCTYPE**, the **meta charset**, and an appropriate **title** (such as 'Color Guessing Game').
- In the **body** element, add an **h1** element with the text content of **Color Guessing Game**. Beneath this, add a **button** element with the **type** of **"button"**, an **onclick** attribute with the value of **"runGame()"**, and text content of **"Start Game"** (without the quotes).
- Add a input text box for the players name

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Color Guessing Game</title>
</head>
<body>
    <h1>Color Guessing Game</h1>
    <button type="button" onclick="runGame()">Start Game</button>
    <input type="text" id="plr" style="width: 140px;" placeholder="Your Name">
    <script>

    </script>
</body>
</html>
```

Inline JavaScript and HTML (browser) Events

Here is a list of some common HTML events that can trigger JS: (and many more here: https://www.w3schools.com/jsref/dom_obj_event.asp)

onchange	An HTML element has been changed	onmouseout	The user moves the mouse away from an HTML element
onclick	The user clicks an HTML tag/element	onkeydown	The user pushes a keyboard key
onmouseover	The user moves the mouse over an HTML element	onload	The browser has finished loading the page

JavaScript can communicate with, evaluate, and modify anything in the **DOM** (Document Object Model). **See Link**…

So, it's time to discuss: Variables, Expressions, Operators, Arrays, Functions, Loops, and Decision Logic, as we do the assignment.

(oh… And everything in OO (Object Oriented) Programming can have properties and methods …)

Next page… Some basics of JavaScript

Understanding the basics of JavaScript

Variables contain values - values are the way JavaScript determines **data types**.
- Variables can be local or global.
  - Local confines usage to a function (or block of code),
  - Global expands usage to the entire application (script).

Values of a Variable (named value pairs are converted into data types)
JavaScript is "loosely typed". This means that variables in JavaScript are not actually declared (assigned a specific data type) by the developer. Instead, JavaScript *sets* the *type* of data based upon how it interprets the value the developer provides.
Data types are:

- **Number**, (numeric data) *primitive*
- **String**, (A string value is a string of characters enclosed in ""quotes.) *primitive*
- **Boolean**, (A Boolean value is either *true* or *false* ... *1 or 0* .) *primitive*
- **Null**, (where the type is intentionally set to *null* – acts empty) *primitive*
- **Undefined** (where the type has not been set) *primitive*
- **Object**, (An Object value type is any value associated with an object: Math, Date) *Non-primitive*
- **Function** *Non-primitive*

The following values are **always falsy**:
(Boolean false)
- `false`
- `0` (zero)
- `''` or `""` (empty string)
- `null`
- `undefined`
- `NaN`

Everything else is **truthy**.
(Boolean true)
- `true`
- `'0'` (a non-empty string containing anything)
- `'false'` (a string containing the text "false")
- `[]` (an empty array)
- `{}` (an empty object)
- function(){} (an "empty" function)

In the beginning there was just one Variable. Now there are three… Think *Scope*…
var, let, const (*the reserved keywords for a variable*) are used to declare variables.

- ***var***: The Original variable. The *scope* of a variable defined with the keyword "var" is limited to the "function" within which it is defined. Otherwise, if it is defined outside of a function, the *scope* of the variable is global. It can be reassigned.
- ***let:*** is considered "block scoped". The *scope* of a variable defined with the keyword "let" is limited to the "block" defined by curly braces{} (function, loop, decision). However, if it is defined outside a function, the *scope* of the variable is global. It can be reassigned.
- ***const:*** The *scope* of a variable defined with the keyword "const" is limited to the block defined by curly braces. It cannot be reassigned. However it CAN be *mutated*.

```
const COLORS_ARRAY = ['magenta','blue','cyan','gold','gray','green','orange','red','white','yellow'];
      COLORS_ARRAY[1] = "black"; //you can change a const property , but not the object
      COLORS_ARRAY.sort();
```

```
TRUE or FALSE
<script>
  const x = undefined;
  const y = null;
  const z =["b"];
  z[0] = x == y;
  alert(z);
</script>
```

Note: Variables become expressions or "Name/Value" pairs that use the equal sign "=" to assign a value, such as:
let name="value";
An **Assignment** places a value in a variable by using the equal sign. Variable values can be *numeric, string, boolean, null, etc*.

Expressions are units of code that can be evaluated and resolve to a value.
They always use an *assignment operator*, (in JS, the "=" equal sign).
Below are variable declarations as expressions.

let **thisMonth = "January";**
let **possiblePlanets = 9 + 1;**

• *In JavaScript, expressions (statements/commands) end with a semicolon ";"*

The variable declaration

var **thisMonth = "January";**

is an expression.

So is

var **possiblePlanets = 9 + 1;**

Variables have restrictions.

- Variable names **must be unique, and** begin with a letter (no numbers or special characters… except "$"or an underscore _);
- Variable names **cannot** contain spaces, and they cannot use JavaScript reserved words.
- And, of course, they are **case sensitive**

Once you create a variable, you can use it as a modifiable expression or as a point of reference for comparisons. For example, you can change the value of a variable using math, or you can compare the value of a variable to the contents of a web control (ie., textbox).

```
let x = 0;
x = x + 1;
if(x== 1){x=0;}
```

So, the "=" equal sign is an operator in JavaScript… but there are a lot of them…

Operators are used to control the rules of JavaScript. (i.e., variables, loops, evaluations…)

Operator	Expression	Example variables / Explanation
Assignment		let x = 2; let y = 9; let z = "hi";
=	x = y; // x is 9	Assigns values from the right side operand to the left side operand
Arithmetic	**Expression**	let x = 2; let y = 9; let z = "hi";
+	x = x + y; //x is 11	Adds values
-	x = y - x; //x is 7	Subtracts values
*	x = x * y; //x is 18	Multiplies values
/	x = y / x; // y is 4.5	Divides values
%	x = y % x; // y is 1	Modulus : Gives the remainder of a division equation
Shorthand Assignment	**Expression**	let x = 2; let y = 9; let z = "hi";
+=	x += y; // x is 11	Adds the right operand to the left
-=	y -= x; // y is 7	Subtracts the right operand from the left
*=	x *= y; // x is 18	Multiplies the right operand by the left
/=	y /= x; // y is 4.5	Divides the left operand with the right
%=	y %= x; // y is 1	Gives the remainder of a division of the left operand with the right
++	x++; // x is 3	Increases (increments) an integer value by one
--	x--; // x is 1	Decreases (decrements) an integer value by one
Comparison	**Expression**	let x = 2; let y = 9; let z = "2"; let Q;
==	Q = x == z; // z is true	Checks if the value of two operands are equal or not
===	Q = x === z; // z is false	Checks both: equal value and equal type
!=	Q = x != z; // z is false	Checks if the value of two operands are not equal (negative logic)
!==	Q = x !== z; // z is true	Checks both: equal value and equal type (negative logic)
>	Q = x > y; // z is false	Checks if the value of the left operand is greater than the right
<	Q = x < y; // z is true	Checks if the value of the left operand is less than the right
>=	Q = x >= y; // z is false	Checks if the value of the left operand is greater than or equal to the right
<=	Q = x <= y; // z is true	Checks if the value of the left operand is less than or equal to the right
Logical	**Expression**	let x = 2; let y = 9; let z = "day"; let Q;
&&	Q = x == 2 && y == 5; // Q is false	And : checks if both the operands are true
||	Q = x == 2|| y == 5; // Q is true	Or : checks if either of the operands are true
!	Q = !(x == 5); // Q is true	Reverses the logical state of its operand
String Concatenation		let x = 2; let y = 9; let z = "day"; let Q;
+	Q = x + z; // Q is 2day	Joins two operands into a string (if not numeric)

Conditional Operator (? :) If Condition is true? Then value X : Otherwise value Y (discussed below…)

Part 2: Set up the script element, the colors array, and two main functions

- Add a **script** element to the HTML page, after the **button** element and before the closing **</body>** tag. *Is there a rule to placing scripts on a page? When in doubt… go to the bottom of the page…*
- Inside the **script** element, create a global constant named **COLORS_ARRAY**. Its value should be an **array** with CSS color name strings as its values. You can use these or substitute your own choice of valid CSS color names:
  `'blue', 'cyan', 'gold', 'gray', 'green', 'magenta', 'orange', 'red', 'white', 'yellow'`
  You can find a list of HTML color names here or at other places on the web.
- Beneath this, declare two separate functions. Name the first function **runGame**, and it should have an empty parameter list. Name the second function **checkGuess** - give it two parameters, **guess** and **target.**

```
  <script>
    const COLORS_ARRAY = ['blue', 'cyan', 'gold', 'gray', 'green', 'magenta', 'orange', 'red', 'white', 'yellow'];
    function runGame() {

    }

    function checkGuess(guess, target) {


    }
  </script>
</body>
</html>
```

```
// Try This
COLORS_ARRAY[2]='silver';
COLORS_ARRAY.sort();
```

JavaScript Functions are code containers

Functions are a blocks of code or organized groups of statements (or commands) that perform tasks.

- A function is a series of commands that will perform a task such as calculate a value.
- JavaScript has built-in functions (i.e., alert() prompt())
- Developers can write their own functions (i.e., myFunction(){alert("hello world") } .
- Generally functions are named (but not always). You can then trigger the function when you call its name.
- Functions can accept parameters – much like non-instantiated variables

  For example, if you pass "Dan" as a parameter to a function

  ```
  <button type="button" onclick=" sayHi('Dan')">Say Hi
  </button>
  ```

  The ***function*** **sayHi(param){alert(param);}** shows 'Dan' in an alert
- A function is reusable, and can be used to repeat a task by calling the same function rather that rewriting (duplicating) code for each instance of its use.

JavaScript Arrays are a special type of variable.

An Array is an enumerated list of variables.

There are two ways to declare an array:

```
var myArray = []; // preferred – Best practice
var myArray = new Array(10); // old school
```

– don't use it, but you may see it…

Initializing An Array

```
var x = [0,1,2,3,4,5];
```

or

```
var months = [];
months[0] = 'January';
months[1] = 'February';
months[2] = 'March';
```

Arrays have many pre-defined **methods** and **properties**.

For example, all arrays have a "length" property.

```
months.length;
```

All arrays have a sort method

```
months.sort();
```

- Functions have rules:
  - The word *function* is reserved
  - A function name must have a pair of parenthesis ()
  - You can add parameters inside the parenthesis (par1, par2,...)
  - A pair of curly brackets {} surrounds all statements in a function.

```
<body>
    <button type="button" onclick="sayHi('Dan')">Say Hi</button>
    <script>
        function sayHi(someParameter){
            alert("Hi " + someParameter);
        }
    </script>
</body>
```

Arrow functions: introduced in ES6.
Anonymous function

They allow you to write shorter function syntax:

```
let hello = function() {
  return "Hello World!";
}

let hello = () => {
  return "Hello World!";
```

Comments - Just so I don't forget...

There are two ways to create Comments in JavaScript:
The single line comment is just two slashes // (cntl + /)
The multiple line comment starts with /* and ends with */

No Script tags - Sometimes users turn off JavaScript

```
<noscript>
  Sorry...JavaScript is needed to continue.
</noscript>
```

Commenting
html and js
Windows: Shift + Alt + A;
Mac: Shift + Option + A

css and js
Windows: Ctrl + /
Mac: Command + /

Part 3: Set up variables for the *ru* **t guess**

- Declare two variables using the named **correct.** Initialize **guess** Initialize **correct** to the Boolean
- Declare a **const** variable name have learned about arrays and using **Math.random()** and **Math** is between 0 and the last index number of the **COLORS_ARRAY** array.

```
<script>
    const COLORS_ARRAY = ['blue', 'cyan', 'gold', 'gray', 'green', 'magenta', 'orange', 'red', 'white', 'yellow'];
     function runGame() {
        let player = document.querySelector("#plr").value;
        let guess = '';
        let correct = false;
        let numTries = 0;
        const targetIndex =
         Math.floor(Math.random() * COLORS_ARRAY.length);
```

- The max number (array length - 1) should be calculated dynamically rather than hard coded. Recall what you have learned about checking the length of the array, and that due to **zero-indexing**, the index of the last number of the array is always one less than its length.
- For example: If your array has 10 colors in it, your program should automatically generate a number between 0 and 9. If you were to add 1 more color to the array, your program should automatically generate a number between 0 and 10.

How Math.random and Math.floor work… using a two-step approach. (if you use "FLOOR" you don't need to -1)

```
var targetIndexA = Math.random() * COLORS_ARRAY.length;  // Local variable assigned a random color (random number * 10…)
var targetIndex = Math.floor(targetIndexA);  // convert to integer (i.e., 5.9999999) down to floor w/o decimals (i.e., 5)  - Similar to rounding
```

- Declare a constant variable named **target** and assign to it the value of the **COLORS_ARRAY** item that has the array index of **targetIndex**. So for example, if the random number stored in **targetIndex** is 3, the color name in **COLORS_ARRAY** with the index of 3 should now be stored in **target.** (In the example array given above, that value would be 'gray'.)
  - ***TIP:** This is not necessary for the game logic, but to make it easier for you as a developer to test the game, we suggest that you add a **console.log** at this point to log the target to the console.*

```
        const target = COLORS_ARRAY[targetIndex];

        alert('The target is: ' + target);

        // console.log('The target is: ' + target);
```

Part 4: Prompt for a guess until correct guess is made

- Write a **do ... while** loop. The condition to exit the loop should be: **!correct** (this is the same as saying **(correct === false)**)
- Inside the loop block, assign the value of the **guess** variable to the return value of a prompt, using the following string as the text for the prompt:

```
'I am thinking of one of these colors:\n\n' + COLORS_ARRAY +
'What color am I thinking of?\n'
```

- The **\n** characters will cause a newline in the text.
- Below this, still inside the loop block, assign the value of the variable **correct** to the return value of the function **checkGuess**. Pass two arguments to the **checkGuess** function: **guess** and **target**.
- After the program has exited the loop - below it, but still inside the **runGame** function, set up an **alert** that gives the user a congratulations message.

```
    do {
        guess = prompt('I am thinking of one of these colors:\n\n' +
            COLORS_ARRAY.join(', ') +
            '\n\nWhat color am I thinking of ' + player + ' ?');
        numTries += 1;

        if (guess === null) { //this is part of the enhancements
            alert('The game was ended by you.' + guess);
            return;
        }
        correct = checkGuess(guess.toLowerCase(), target);
    } while (!correct);
  // Keep looping until the function checkGuess returns a value of 'true'
}
```

Loops

Looping in all programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

- `for` - loops through a block of code a number of times
- `while` - loops through a block of code while a specified condition is true
- `do/while` - loops through a block of code while a specified condition is true

Loops offer a quick and easy way to do something repeatedly…

For Loop (has 3 parameters)

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript For Loop</h2>
    <p id="demo"></p>
    <script>
        var text = "";
        var i;
        for (i = 0; i < 5; i++) {
            text += "The number is " + i + "<br>";
        }
        document.getElementById("demo").innerHTML = text;
    </script>
</body>
</html>
```

Escape sequences in JavaScript:

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return

Escape characters were originally designed to control typewriters, teletypes, and fax machines… but still have some value in js.

You can also escape quotes and other characters: \" or \' or \{ or \}

```
onclick="alert('It\'s hot');"
```

While Loop (one parameter, but don't forget to iterate)

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript While Loop</h2>
    <p id="demo"></p>
    <script>
        var text = "";
        var i = 10;
        while (i < 10) {
            text += "<br>The number is " + i;
            i++;
        }
        document.getElementById("demo").innerHTML = text;
    </script>
</body>
</html>
```

Do While Loop (always executes at least once)

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Do/While Loop</h2>
    <p id="demo"></p>
    <script>
        var text = ""
        var i = 10;
        do {
            text += "<br>The number is " + i;
            i++;
        }
        while (i < 10);
        document.getElementById("demo").innerHTML = text;
    </script>
</body>
</html>
```

document.querySelector(#demo).innerHTML = text;

Part 5: Write the content for the *checkGuess* function

- Inside the **checkGuess** function, set up an **if** statement. For its condition, use an array method to check if **guess** is a color in **COLORS_ARRAY** at all. At this point you are not checking if the guess is correct, only if it is in the array. If it is not, show the user an appropriate message and **return false** from the function.
- Set up an **else if** block that checks if the **guess** is higher than the **target**. If so, then show the user an appropriate message and **return false**.
- Set up a second **else if** block that checks if the **guess** is lower than the **target**. If so, then show the user an appropriate message and **return false**.
- After the end of the if statement, **return true**.

```
    function checkGuess(guess, target) {
        const sorryMsg = 'Sorry, your guess is incorrect.\n\n';
        const tryMsg = '\n\nPlease try again.';
        let correct = false;
        if (!COLORS_ARRAY.includes(guess)) {
            alert('Sorry, I don\'t recognize your color. ' + tryMsg);
        } else if ( guess < target ) {
            alert(sorryMsg +
             'Hint: your color is alphabetically lower than mine.' + tryMsg);
        } else if ( guess > target ) {
            alert(sorryMsg +
             'Hint: your color is alphabetically higher than mine.' + tryMsg);
        } else {
            correct = true;
        }
        return correct;
    }
</script>
```

Note: It *is* possible to set up the ***if*** statement in a different way than outlined above, and still have it work in the same way. The above steps are a guideline - if you can set it up a different way and your code still works in the same way, that's OK.

If, If/Else - Conditional statements / Decision Logic

Decision logic in JavaScript is generally scripted, based upon very simple true or false conditions.

If something is true, then do this, else do that…

A basic if statement is much like a built in function

```
if(condition==true){execute code;}else{execute other code;}
```

For example, the following code will only run the *alert* if the condition is met:

```
let d = new Date().getHours();  // REM hours run from 0 to 23
if(d<=11){alert("Good Morning");}
```

But it won't do anything if the time is not morning (0-11).

So you can add an else statement for a default condition

```
let d = new Date().getHours();
if(d<=11){alert("Good Morning");}
else{alert("Good Afternoon");}
```

But you may want to test for more than one condition…

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript</h2>
    <p id="demo"></p>
    <script>
        let d = new Date().getHours();
        let result = (d < 12) ? "Good Morning" : "Good Day";
        alert(result);
        document.getElementById("demo").innerHTML = "the hour is: " + d;
    </script>
</body>
</html>
```

See if/else code below…

To test multiple conditions, add an *else if* statement

```
var d = new Date().getHours();
if(d<=11){alert("Good Morning");}
else if (d<17){alert("Good Afternoon");}
else{alert("Good Night");}
```

Note: the conditions are tested in top down order - if the first is not met, check the next, and finally the ending else is met.

There is another way to test conditions in JavaScript…

Switch Case - Conditional logic

The Switch statement provides logic to evaluate many conditions.

- Each evaluation is a “case”.
- You start with “case 0:” and evaluate as many cases as you like…
- Evaluations are tested top down.
- If your condition is met, you can use a “break” clause to stop anymore evaluations from executing in the code.
- You can group cases by not adding a “break” after the “case”.
- You can also add a “default” statement if none of the cases are met.

The switch statement is best used when there are many conditions to test.

Another approach to evaluating cases is to evaluate true or false

```
var d = new Date().getHours();
switch(true){
    case d<=10:
        alert("Good Morning");
        break;
    case d<=17: alert("Good Afternoon");
        break;
    default: alert("Good Night");
}
```

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript</h2>
    <p id="demo"></p>
    <script>
        var d = new Date().getHours();
        if(d<=11){
            alert("Good Morning");
        }
        else if (d<17){
            alert("Good Afternoon");
        }
        else{
            alert("Good Night");
        }
       document.getElementById("demo").innerHTML = "hour is: " + d;
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript</h2>
    <p id="demo"></p>
    <script>
        var d = new Date().getHours();
        switch(d){
           case 0: alert("Very Early Morning");
              break;
           case 1: alert("Early Morning");
              break;
           case 2:case 3:case 4:case 5:alert("Mid-Morning");
              break;
           case 6: case 7: case 8: case 9:
           case 10: alert("Good Morning");
              break;
           case 11: case 12: case 13: case 14:
           case 15: case 16: case 17:alert("Good Afternoon");
              break;
           default: alert("Good Night");
        }
        document.getElementById("demo").innerHTML = "the hour is: " + d;
    </script>
</body>
</html>
```

At the end of Task 1, test your game and make sure that it runs as demonstrated in the workshop video.

TASK 2: Update the code

Update the code with more features, including displaying the total number of guesses and changing the background color when the correct guess is made, and more.

Part 1: Display the total number of guesses when the correct guess has been made:

- In the **runGame** function, at the top where the other variables were declared, declare a new variable using **let** named **numTries** or a similar name. Initialize it to **0**.
- Increment **numTries** by 1 inside the **do ... while** loop block by, for example, using the addition assignment operator.
- Note that it is important that the **let** declaration for this variable is outside of the **do ... while** loop. If it was declared inside the loop, then it would be reset at every iteration of the loop.
- In the congratulations message to the user, show the number of tries.

Part 2: Change the background color to the correctly guessed color:

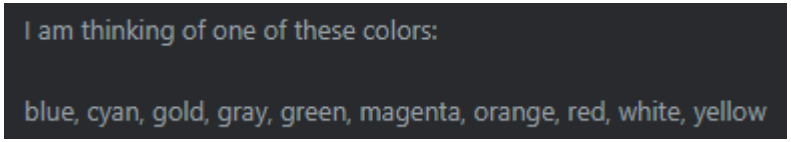
- Use the code below in an appropriate place to let the player know their guess has been successful:

```
document.body.style.background = guess;
```

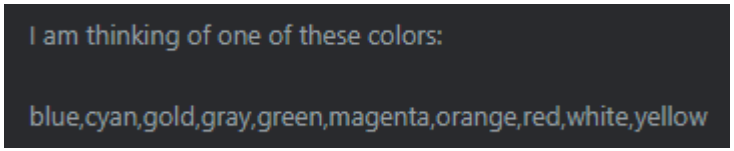
```
<script>
    const COLORS_ARRAY = ['blue', 'cyan', 'gold', 'gray', 'green', 'magenta', 'orange', 'red', 'white', 'yellow'];

    function runGame() {
        let guess = '';
        let correct = false;
        let numTries = 0;

        const targetIndex =
             Math.floor(Math.random() * COLORS_ARRAY.length );
        const target = COLORS_ARRAY[targetIndex];
        alert('The target is: ' + target);
        //console.log('The target is: ' + target);
        do {
          guess = prompt('I am thinkin of 1 of these colors:\n\n' +
                COLORS_ARRAY.join(', ') +
               '\n\nWhat color am I thinking of ' + player + ' ?');
            numTries += 1;

          if (guess === null) { //this is part of the enhancements
              alert('The game was ended by you.' + guess);
              return;
          }
            correct = checkGuess(guess.toLowerCase(), target);
        } while (!correct);
        // Keep looping until the function checkGuess returns a value of 'true'

        document.body.style.background = guess;
        alert('Congratulations! You have guessed the color!\n\n' +
            'It took ' + player + ' ' + numTries +
            ' guesses to finish the game!\n\n' +
            'Hit OK to see the color in the background.');
    }
```

Part 3: Display the color names with each separated by a comma and a space

- Look into using the *array*.join() method to see how you can show the color names from the array using a comma and a space as a separator, so that it shows up like this:



instead of like this:




```
guess = prompt('I am thinking of one of these colors:\n\n' +
    COLORS_ARRAY.join(', ') + '\n\nWhat color am I thinking of?\n');
```

Part 4: Allow users to abort the game by clicking Cancel.

- Set up an **if** condition that follows the **prompt** inside the **do ... while** loop. This if condition should check if **guess** contains a falsy value (such as null or an empty string). You can do this by simply checking: **if (!guess)**
- If so, display an appropriate message to the user and use a **return** statement with no return value. This will exit the **runGame** function and end the game.

```
// this is the same as if(guess <> true) or if(guess == false)
if (!guess) {
    alert('The game has been cancelled.');
    return;
}
```

At the end of Task 2, test your game and make sure that it runs as demonstrated at the end of the workshop video.

BONUS CHALLENGES

The following challenges are not required by the assignment. If you have time left, try them out!

- to sort the list of colors alphabetically when showing them to the user. You'll want to make sure that your colors are *not* in alphabetical order in the array when you test this; otherwise, you won't be able to tell if it's working.

```
COLORS_ARRAY.sort();          REM: sort() is very quirky Caps and numbers don't act the way you think (a, b, c, A, B, C) (1,11,2,22,3)
```

- Research the use of the *string*.toLowerCase() method to help make the guess case insensitive. You can do so by making sure that both the target and the guess are both in the same case before being compared.

```
correct = checkGuess(guess.toLowerCase(), target);
```

- Instead of only showing the number of tries at the end of the game, how would you show it for each try, even for incorrect guesses?

```
correct = checkGuess(guess.toLowerCase(), target, numTries);
function checkGuess(guess, target , numTries) {
alert('Sorry, I don\'t recognize your color. ' + tryMsg + " tries: " + numTries);
```

- Add more HTML and CSS to the page to improve its appearance. Your choice in how to carry this out!

Add Confetti

```
<button type="button" onclick="runGame()" onmouseover="hide()">Start Game</button>
    <div id="confetti"> <h1>You Won!</h1>
        <h3>Click on Start Game to play again</h3>
        <img src="./confetti-32.gif" alt="confetti" id="bravo">

    </div>

    <style>
      #confetti{display:none;}
    </style>
```

```
<!-- https://giphy.com/search/bravo-bravo-stickers  bravo_9.gif
https://www.w3schools.com/js/js_examples.asp        -->

//document.querySelector("#bravo").src = "./Old/Bravo_" + targetIndex + ".gif"
```

```
        function hide() {
            document.getElementById('confetti').style.display = "none";
        }
        function checkGuess
        else{
                correct = true ;
                document.getElementById('confetti').style.display = 'block';
                document.getElementById("plr").focus();//Rem this only works if you added the input box for players name
            }
```

A Note on Lexicographical Order

In the game, we use lexicographical order to provide hints to the player. When comparing lower case strings of characters from the alphabet, lexicographical order works in this way:

The first characters of each string are compared alphabetically, with **'a'** at the lowest end and **'z'** at the highest end. If the characters are the same, then the next character of each string is compared. Thus**, 'ant' < 'zoo**', **'meow' < 'merman'**, and so on.

Here are some examples of strings where the first string is higher/greater than the second string:

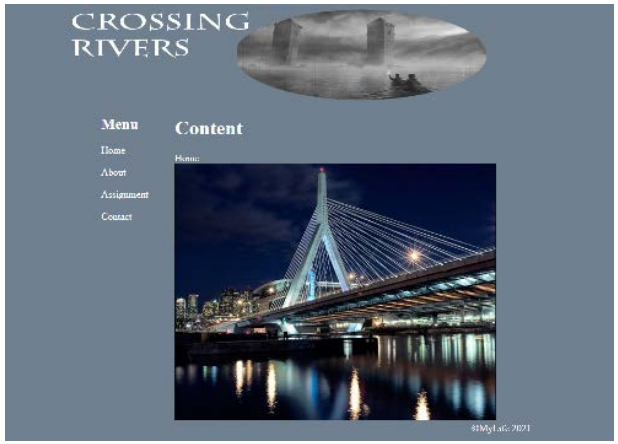
- `sat > sad`
- `bags > bag`
- `thin > fat`
- `good > bad`

Here are more examples of strings where the first string is lower/less than the second string.

- `rag < rat`
- `bit < bite`
- `food < water`
- `potato < potatoes`

A Simple JavaScript Show Hide Menu – <u>NOT Required</u> for submission

Header Div	
Menu Div Home About Assignment Contact	Content Div container (Actually contains 4 Divs) Home About Week 1 Assignment Contact
Footer Div	



First Build a quick HTML Web Page	<body> <main>

Build a <main> section with four sections:

- One for the Header,
  - Add a nice banner image,
- One for the Menu, In that div
  - Add 4 <a> tags with empty href attributes "#"
  - One for Home, About, Assignment, and Contact
  - Add a JS Trigger attribute to each: onclick="…"
  - The onclick attribute will call the JS Function: ShowHide
  - ShowHide has one parameter (tagId) – we will pass the name of the id that we want to show or hide. These will also correspond to the menu names…
- One for the Content, In that add 4 child divs:
  - One for: Home, About, Assignment, and Contact
  - Give each a unique Id with those same names
  - Give each a shared class of "content"
  - Add a nice image in each (or any content you like),
- One for the Foot – give it a little inline style

```
            <img id="castleImg" src="../images/BoatToTheCastle.jpg" style="" />
        </div>
        <div id="Menu">
            <h2>Menu</h2>
            <a href="#" onclick="ShowHide('Home')">Home</a> <br><br>
            <a href="#" onclick="ShowHide('About')">About</a> <br><br>
            <a href="#" onclick="ShowHide('Assignment')">Assignment</a> <br><br>
            <a href="#" onclick="ShowHide('Contact')">Contact</a> <br><br>
        </div>
        <div id="Content">
            <h1>Content</h1>
            <div id="Home" class="content"> Home <br>
                <img src="http://www.drpage.net/nucamp/images/bridge.jpg" />
            </div>
            <div id="About" class="content"> About <br>
                <img src="http://www.drpage.net/nucamp/images/tree.jpg" />
            </div>
            <div id="Assignment" class="content"> Class 1 Assignment <br>
                <img src="http://www.drpage.net/nucamp/images/Rome.jpg" />
            </div>
            <div id="Contact" class="content"> Contact <br>
                <img src="http://www.drpage.net/nucamp/images/DOM.jpg" />
            </div>
        </div>
        <div id="Foot" style="text-align: right;
            padding-right: 33%;">&copy;MyLife 2021</div>
    </main>
</body>
```

Next Give it a Style

The only required style for the ShowHide function is the following:

```
/* Below is to hide all content except home  */
    #Home {display: block;}
    #About,
    #Assignment,
    #Contact {display: none;}
/* Above is to hide all content except home  */
```

**When the page loads this style will show only the Home **

And will hid the About, Assignment and Contact 's

Everything else in the CSS is just to make the page look nice.

```
<style type="text/css">
    body {
        background-color: slategrey; /* #708090 Color Hex Slate Gray */
        color: #ffffff;
    }
    main { width: 66%; margin: auto;}
    #Menu {
        float: left;
        color: #fff;
        width: 120px;
        height: 450px;
        background-color: slategrey;
    }
    a:link {color: #fff; text-decoration: none;}
    a:visited {color: #fff; text-decoration: none;}
    a:hover {color: #111; text-decoration: none;}
    a:active {color: #fff; text-decoration: none; }
    #Content{background: slategrey;}
/* Below is to hide all content except home  */
```

You can modify it as you like…

However…

Your menu should have a `float: left;` so that the Content is to the right of it.

And the Foot should have a `clear: both;` so that it breaks

```
    #Home {display: block;}
    #About,
    #Assignment,
    #Contact {display: none;}
/* Above is to hide all content except home  */
    #Foot {
        clear: both;
        color: white;
    }

    img {width: 50%; height: auto; }
    #castleImg {
        position: relative;
        top: 1%;
        left: -5%;
        height: auto;
        width: 100%;
        max-height: 150px;
        max-width: 750px;
    }
</style>
```

Now add the JavaScript

The function is really pretty simple…

It has a parameter `(tagId)` that can communicate with any unique id on the page with a class of `("content")`.

We will only be using the id's we cave to the div's in our content section.

When the user clicks on one of the Menu Links with the unique id names… `<a href="#" onclick="ShowHide('About')">About</a>`

↓

It passes that value to `function ShowHide(tagId) { …`

Next, create a variable-array of all elements with the class: `content`

`let cnt = document.getElementsByClassName("content");`

```
<script>
//bring in 1 param (The content to show)
    function ShowHide(tagId) {
        // build an array of content classes
        let cnt = document.getElementsByClassName("content");
       // let cnt = document.querySelector(".content");
        let i;  // incrementor variable
        //Loop through the array of content classes
        for (i = 0; i < cnt.length; i++) {
            // set them all to hidden
            cnt[i].style.display = "none";
```

or

```
let cnt = document. querySelector(".content");
```

We need to loop through the array setting **every** element's style display rule to hidden (*none*).

```
for (i = 0; i < cnt.length; i++) {
     cnt[i].style.display = "none";
}
```

REM: an array **length** is a number from 1 to *n*. But its **index** is 0 to n

Finally, set the style display rule of the div id you want show (block), to what the user passed to the parameter (`tagId`).

```
document.getElementById(tagId).style.display = 'block';
```

That's it…

```
        }
        // show just the param
        document.getElementById(tagId).style.display = 'block';
    }
</script>
```

Final Code - Class 3 Assignment

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Color Guessing Game</title>
</head>
<body>
    <h1>Color Guessing Game</h1>
    <button type="button" onclick="runGame()">Start Game</button>

    <script>
        const COLORS_ARRAY = ['blue','cyan','gold','gray','green','magenta','orange','red','white','yellow'];
```

```
      COLORS_ARRAY[2]='silver';
      COLORS_ARRAY.sort();
function runGame() {
  let guess = '';
  let correct = false;
  let numTries = 0;
  const targetIndex = Math.floor(Math.random() * COLORS_ARRAY.length);
  const target = COLORS_ARRAY[targetIndex];
  alert('The target is: ' + target);
  do {
      guess = prompt('I am thinking of one of these colors:\n\n' +
          COLORS_ARRAY.join(', ') +
          '\n\nWhat color am I thinking of?\n');
      numTries += 1;

      if (guess === null) { //this is part of the enhancements
          alert('The game has been aborted.');
          return;
      }

      correct = checkGuess(guess.toLowerCase(), target, numTries);
  } while (!correct);
  document.body.style.background = guess;
  // Keep looping until the function checkGuess returns a value of 'true'
  //alert('Congratulations!');
  alert('Congratulations! You have guessed the color!\n\n' +
      'It took you ' + numTries +
      ' guesses to finish the game!\n\n' +
      'Hit OK to see the color in the background.');
}

function checkGuess(guess, target , numTries) {
    alert(guess + " " + target)
    const sorryMsg = 'Sorry, your guess is incorrect.\n\n';
    const tryMsg = '\n\nPlease try again.';
```

```
            let correct = false;
            if (!COLORS_ARRAY.includes(guess)) {
                alert('Sorry, I don\'t recognize your color. ' + tryMsg + " tries: " + numTries);
            } else if ( guess < target ) {
                alert(sorryMsg +
                 'Hint: your color is alphabetically lower than mine.' + tryMsg + " tries: " + numTries);
            } else if ( guess > target ) {
                alert(sorryMsg +
                 'Hint: your color is alphabetically higher than mine.' + tryMsg + " tries: " + numTries);
            } else {
                correct = true;
            }
            return correct;
        }

    </script>
</body>
</html>
```

ShowHide Code – Not Required

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple JS Show/Hide Menu</title>
    <style type="text/css">
        body {
            background-color: slategrey; /* #708090 Color Hex Slate Gray */
```

```
        color: #ffffff;
    }
    main { width: 66%; margin: auto;} /* Center the page */
    #Menu {
        float: left; /* move the next element to the right */
        color: #fff;
        width: 120px;
        height: 450px;
        background-color: slategrey;
    }
    a:link {color: #fff; text-decoration: none;}
    a:visited {color: #fff; text-decoration: none;}
    a:hover {color: #111; text-decoration: none;}
    a:active {color: #ffF; text-decoration: none; }
/* Below is to hide all content except home  */
    #Content{background: slategrey;}
    #Home {display: block;}
    #About,
    #Assignment,
    #Contact {display: none;}
/* Above is to hide all content except home  */

    #Foot {
        clear: both; /* turn off float */
        color: white;
    }

    img {width: 50%; height: auto; }
    #castleImg {
        position: relative;
        top: 1%;
        left: -5%;
        height: auto;
        width: 100%;
        max-height: 150px;
```

```
            max-width: 750px;
        }
    </style>

    <script type="text/javascript">
        //bring in 1 param (The content to show)
        function ShowHide(tagId) {
            // build an array of content classes
            var cnt = document.getElementsByClassName("content");
            var i;  // incrementor variable
            //Loop through the array of content classes
            for (i = 0; i < cnt.length; i++) {
                // set them all to hidden
                cnt[i].style.display = "none";
            }
            // show just the param
            document.getElementById(tagId).style.display = 'block';
        }
    </script>
</head>


<body>
    <main>
        <div id="Header">
            <img id="castleImg" src="../images/BoatToTheCastle.jpg" style="" />
        </div>
        <div id="Menu">
            <h2>Menu</h2>
            <a href="#" onclick="ShowHide('Home')">Home</a> <br><br>
            <a href="#" onclick="ShowHide('About')">About</a> <br><br>
            <a href="#" onclick="ShowHide('Assignment')">Assignment</a> <br><br>
            <a href="#" onclick="ShowHide('Contact')">Contact</a> <br><br>
        </div>
        <div id="Content">
```

```
            <h1>Content</h1>
            <div id="Home" class="content"> Home <br>
                <img src="http://www.drpage.net/nucamp/images/bridge.jpg" />
            </div>
            <div id="About" class="content"> About <br>
                <img src="http://www.drpage.net/nucamp/images/tree.jpg" />
            </div>
            <div id="Assignment" class="content"> Class 1 Assignment <br>
                <img src="http://www.drpage.net/nucamp/images/Rome.jpg" />
            </div>
            <div id="Contact" class="content"> Contact <br><br/><br/>
                <img src="http://www.drpage.net/nucamp/images/DOM.jpg" /><br/><br/><br/>
            </div>
        </div>
        <div id="Foot" style="text-align: right; padding-right: 33%;">&copy;MyLife 2021</div>
    </main>
</body>
</html>
```