**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 introduced JavaScript as "Livescript". Its official name is “ECMAScript”

• It was renamed JavaScript in 1995.

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: <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).

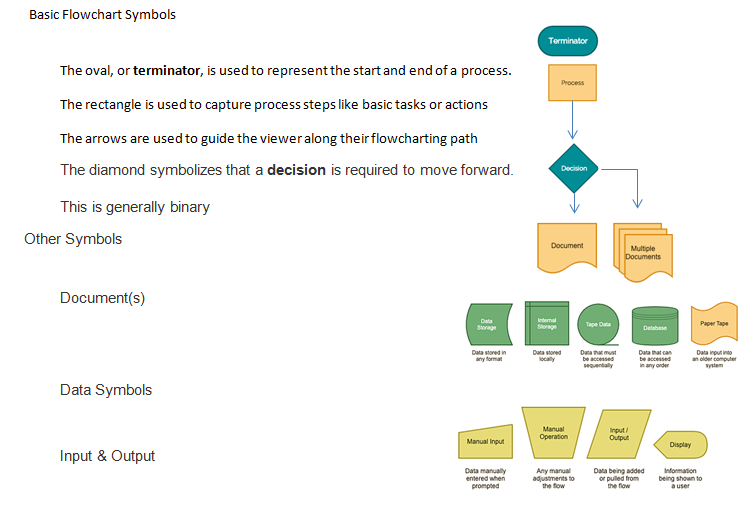
Here is a quick overview of the game

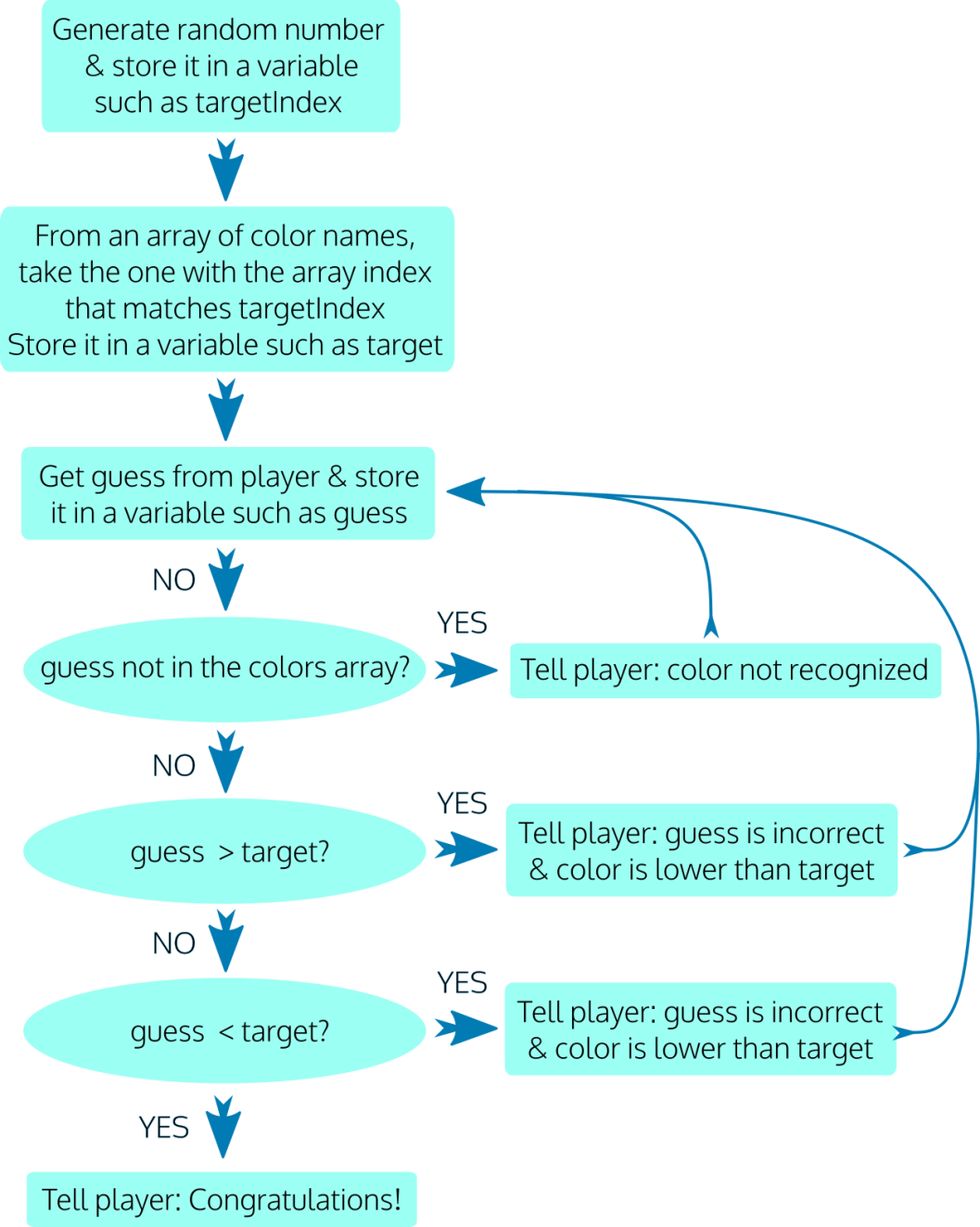
|  |  |
| --- | --- |
| You will set up an HTML page with a **button** that, when clicked, will begin the game with a prompt that looks like the screenshot below.  <button type="button" **onclick="runGame()"**>Start Game</button>  The appearance may be different depending on your browser and settings, 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 isin 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**





Loop calls function

**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) works**  There are **three** ways to run JavaScript  1) **JavaScript can be *linked* from your HTML** **Document** **to an external file**  Note: JS Has 1 HTML Tag <script></script>  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"> )  Example of an External File:  <script type="text/javascript" src="javaScriptsFolder/someJSfile.js"></script>  Inside the external file (in the javaScriptsFolder 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). | <!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>  </body>  </html> |

**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  onclick The user clicks an HTML element  onmouseover The user moves the mouse over an HTML element | onmouseout The user moves the mouse away from an HTML element  onkeydown The user pushes a keyboard key  onload The browser has finished loading the page |

Next page… The 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 and data types)  JavaScript is "loosely typed".  This means that variables in JavaScript are not actually assigned a specific data type by the developer. Instead, JavaScript will set 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 null/empty) *primitive* * **Undefined** (where the type has not been set) *primitive* * **Object**, (An Object value type is any value associated with an object. ) *Non-primitive* * **Function** *Non-primitive*   **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();    **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 “;” .*    **Variables have restrictions**.   * Variable names **must** begin with a letter or an underscore (no numbers or special characters); * 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).  var 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…) |

|  |  |
| --- | --- |
| **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](https://www.w3schools.com/colors/colors_names.asp)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> |

|  |  |
| --- | --- |
| **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 the function  <button type="button" onclick=" sayHi(‘Dan’)">Say Hi</button>  The **function** **sayHi(x){alert(x);}** will show Dan in the 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. * Functions have rules:   + ƒ the word *function* is reserved   + ƒ a function name must have a pair of parenthesis ()   + ƒ you can add parameters inside the parenthesis (par1, par2,...)   + ƒ A pair of curly brackets {} surrounds all statements in a function.   For example:  function sayHi(someName){  alert(someName);  } | **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** = [*10*]; // *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 = [12];  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();  **Comments -** Just so I don't forget...  There are two ways to create Comments in JavaScript:  The single line comment is just two slashes //  The multiple line comment starts with /\* and ends with \*/ |

|  |  |
| --- | --- |
| **Part 3: Set up variables for the *runGame* function & determine the correct guess**   * Declare two variables using the **let**keyword: one named **guess** and one named **correct.**Initialize **guess** to the value of an empty string. Initialize **correct**to the Boolean value of **false**. * Declare a **const** variable named **targetIndex**. For its value, use what you have learned about arrays and generating random numbers using **Math.random()** and **Math.floor()**to generate a random number that 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 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” DO YOU NEED TO USE LENGTH -1 ???)    var targetIndexR = Math.random() \* COLORS\_ARRAY.length - 1;  // Local variable assigned a random color (random num \* 10… -1)  var targetIndex = Math.floor(targetIndexR);  // convert to integer (5.9999999) down to floor w/o decimals (5) - Similar to rounding | |
| * Declare a const 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?\n');                  numTries += 1;                    if (guess === null) { //this is part of the enhancements                      alert('The game has been aborted.');                      return;                  }                  correct = checkGuess(guess.toLowerCase(), target);              } while (!correct);  // Keep looping until the function checkGuess returns a value of 'true'        } |

|  |
| --- |
| **Loops**   * 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   **For**  <!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>  **While**  <!DOCTYPE html>  <html>  <body>  <h2>JavaScript While Loop</h2>  <p id="demo"></p>  <script>  var text = "";  var i = 0;  **while** (i < 10) {  text += "<br>The number is " + i;  i++;  }  document.getElementById("demo").innerHTML = text;  </script>  </body>  </html>  **Do While**  <!DOCTYPE html>  <html>  <body>  <h2>JavaScript Do/While Loop</h2>  <p id="demo"></p>  <script>  var text = ""  var i = 0;  **do** {  text += "<br>The number is " + i;  i++;  }  **while** (i < 10);  document.getElementById("demo").innerHTML = text;  </script>  </body>  </html> |

|  |  |
| --- | --- |
| **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){then execute code;}else{execute some other code;}  For example, the following code will only run code 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.  So you can add an additional else statement for any other 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…  So, if want to test multiple conditions you can add an else if statement          var d = new Date().getHours();          if(d<=10){alert("Good Morning");}          else if (d<17){alert("Good Afternoon");}          else{alert("Good Night");}  Note: the conditions are tested in top down order, so if the first is not met, we check the next, and finally the ending else is met.  There is another way to test conditions in JavaScript…  **Switch Case -** Conditional statements  The Switch statement provides logic for evaluating 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.           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");          }  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");          } |

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

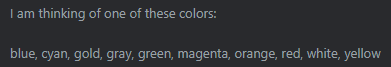
**TASK 2: Update the code**

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.

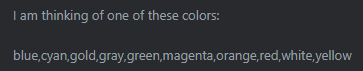
|  |  |
| --- | --- |
| **Part 1: Display the total number of guesses when the correct guess has been made:**   * In the **runGame**function, at the top where the other variables were declared, declare a new variable using **let** named **numTries**or a similar name. Initialize it to **0**. * Increment **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?\n');                  numTries += 1;                    if (!guess) {                      alert('The game has been aborted.');                      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 you ' + 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)

                if (!guess) {

                    alert('The game has been aborted.');

                    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!

**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

Final Code

<!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>

|  |  |
| --- | --- |
| The following values are **always falsy**: (Boolean false)   * false * 0 (zero) * '' or "" (empty string) * null * undefined * NaN | Everything else is **truthy**. (Boolean true)   * '0' (a string containing a single zero) * 'false' (a string containing the text “false”) * [] (an empty array) * {} (an empty object) * function(){} (an “empty” function) |