Timed Mathematical Equation Game

In this module, we are going to create a maths game where the user needs to answer 10 equations to pass the round and get to the winners page. We are using variables, functions, conditional statements and arrays, and will be introducing JavaScript Math functions too! We will once again specify the controls for our page in HTML, add our functionality in JavaScript, and styling our page in CSS.

Go to your newly created math.html file which you created in the previous module (if you didn't create math.html then, you can do it now). See if you can set up this HTML file like you did for the Guessing Game and the Login Page, adding the correct HTML tags.

Here are some things to remember:

- Create a file called 'math.js' and save it in the 'js' folder. Reference this in the HTML.
- You will be using the same css file 'style.css'. Reference this in the HTML.
- You will also want to put your controls in the middle of the page like the previous screens. Add the correct element and class to do this.

Once you have done that, your math.html file should look like this:

```
<html>

<head>

          <link rel="stylesheet" type="text/css" href="css/style.css">

           <script src="js/math.js"></script>

</head>

<body>

          <div class="main">

          </div>

</body>

</html>
```

Now your page is set up, with your CSS and JavaScript already referenced!

How The Math Game works!

We are going to create a game which requires the user to solve maths puzzles in a given amount of time.

When the page loads, three equations will show on the “main” div, with a textbox next to each one in which the user will input their answer to the equation. There will be one equation each for:

- Subtraction
- Multiplication
- Addition

Each time the user gets an answer correct they get one point. To finish the game they need to have 10 points, so we need a variable to keep a tally of this (like we did with ‘attempts’ in our login page, and ‘correctlyGuessed’ in our guessing game).

There is a timer on the page which counts down from 10. Every 10 seconds, the equations are refreshed so the user needs to guess the answer to the equations within 10 seconds or they will be replaced with more numbers and they will need to mentally calculate these new numbers again.

We can make this game easy / hard / extremely hard by changing how big the numbers are, and how fast the timer goes. I will show you how to do this at the end.

Designing your game controls

Just as we did with our previous screen, let’s think about what controls we might want on this page

- A label to show the timer countdown, with an ID of ‘countdown’
- Three labels, one for each equation. With IDs of ‘subtract’, ‘multiply’ and ‘add’
- Three textboxes, one for each equation. With IDs of ‘subtractAnswer’, ‘subtractAnswer’ and ‘addAnswer’.
- A label to show the count of correct guesses with an ID of ‘progress’.

Task: Without looking at the solution below, can you design this page in HTML?

Go through the list above carefully, adding each of the elements and their required attributes.

Load the page.

Can you see that all of the elements are grouped together tightly? We need to separate these out.

Do you remember how we fixed this in the login page? We used a container around elements that we wanted to group together. Try adding tags around elements that need to be together.

Answer

Compare your HTML with this, and see where yours is different. Make changes where necessary.

```
<body>
    <div class="main">
        <div>
            <label id="countdown"></label>
        </div>
        <div>
            <label id="subtract"></label>
            <input id="subtractAnswer" type="text">
        </div>
        <div>
            <label id="multiply"></label>
            <input id="multiplyAnswer" type="text">
        </div>
        <div>
            <label id="add"></label>
            <input id="addAnswer" type="text">
        </div>
        <div>
            <label id="progress"></label>
        </div>
    </div>
</body>
```

How did you do??

Adding your functionality

Next, we are going to add our functionality and make our game come to life.

You have already learned about **variables**, **functions**, **conditional statements**, and **arrays**. Now you will learn about **objects, math functions** and **timers**, and how these are implemented in JavaScript. This game takes a little bit more code, so rather than typing it all out - I want you to start with pre-written code this time.

Go to your 'project files' folder, and copy the **math-duplicate.js** file into the 'js' folder in your project. Now change your math.html file so that it references this new file.

```
<script src="js/math-duplicate.js"></script>
```

Take a look through this file, and see what it is doing. At the top of the page we are declaring our variables, and below this there are four functions:

- **Initialise**: This is called when the page loads, and it calls the **calculate**() function and starts the **timer**. Load math.html in your browser to see this.
- **Calculate**: this is the function that generates the 'add' equation which is displayed both when the page loads and the timer gets to 0.
- **Add**: this is called when an 'onkeyup' event happens in the 'add' textbox, and checks if the entered value matches the actual answer to the equation. If the answer is correct, it increases the count of correct answers.
- **checkCorrectGuesses**: This checks if the number of guesses has reached 10, updates the number of guesses left, and disables the textbox if the user got the guess right.

To see this code working, you will have to add the 'add()' function to the onkeyup event for the 'addAnswer' text box in the html, and the'initialise()' function to the body tag.

Update your <body> tag and your "addAnswer" input tag to look like this;

```
<body onload="initialise()">
```

```
  <input id="addAnswer" type="text" onkeyup="add()">
```

Save the JavaScript and html, and load the page. Adding these events means that the browser will go and kick off these functions when the events occur (loading the page, and typing in the 'addAnswer' textbox.)

You will see that the Add label is populated with an equation when the page loads, and the textbox checks your answers. You can also see that the timer is counting down, and the equations change when the timer reaches 0.

Let me show you how the functionality here works:

Timer

At the top of our math.js file, we have declared a variable called timeLeft. This is set to 10, as we want our timer to refresh every 10 seconds.

In the **initialise()** function, look at this code:

```
var startTimer = setInterval(function () {
        if (timeleft <= 0) {
            calculate();
            timeleft = 10;
        }
        document.getElementById("countdown").innerHTML = timeleft;
        timeleft -= 1;
    }, 1000);
```

Here, we are using a built-in Javascript function called **'setInterval'**. What this function does is run a specified task at a specified interval, until told to stop. This is the syntax

`setInterval(function(){ //task to do }, milliseconds per interval);`

So you can see in the above code we're saying set an interval of 1000 milliseconds (1 second), and every second, perform this conditional statement.

(If the value of the timeleft variable is zero)

{

// Call the calculate function to refresh the numbers

// Reset the clock to 10

}

then

Show time left to the user

Take one second off the timeleft variable

So you can see that all it does is reduce the timeleft variable by 1 every second, and display the value on a timer label. Then when it gets to zero, it calls a function to refresh the numbers, and sets the clock to 10. It's pretty simple right?

Math

The JavaScript Math object allows you to perform mathematical tasks quickly and easily, without writing very much code. You can round any number just by calling Math.round() on the number, and you can find the lowest number in a list of numbers by calling Math.min() on a range of numbers. This saves you having to write lots of lines of code to do these small tasks. [You can read more about Math objects here.]

Math.random is a great function that comes in very useful - it generates a random number between 0 and 1. This is what we're going to use to generate the numbers used in our equations, along with Math.Floor - which rounds our numbers down so the equations don't have lots of decimal places.

Using Math.random, we can generate new equations each time the **'calculate'** function is called.

Let's go through the code and see how this works:

First we declare two variables, one for each number in our equation. Next, we assign these variables a number using Math functions.

```
    Math.floor(Math.random() * difficulty) + 1;
```

When you are reading this piece of the code, solve the equation in the brackets first and then move outside the brackets.

- Math.random() generates a number between 0 and 1. So it is a decimal number.
- Difficulty is a variable we have set at the top of the page, and it's set to 10
- We multiply our decimal number by 10 and we have a number between 0 and 10.
- Next we floor the number, so that we aren't asking the user for '4.575953 + 7.9835987'
- Next we add 1, so that the number is never 0 - that would make things too easy!

So that's what this single line of code is doing! To write this yourself would take much more time and effort, so this is really helpful for developers!

After we have assigned the values to **'one'** and **'two'**, we add these variables together and assign the value to our **'result'** variable which we declared at the top of the page. The 'result' variable will be used by multiple functions in our code.

Next, we set the text on our **'add'** label to be our equation: (value of variable **'one'** "+" value of variable **two'**). This is what the user will see on the page.

```
    document.getElementById("add").innerText = one + " + " + two;
```

Refresh the page. You can see this label is updated when you load the page, and when the time goes to zero. In the last two lines we make sure the answers textbox is enabled, and that it is empty and the old guesses are cleared out, ready for new guesses!

Adding the Multiplication and Subtraction functionality to our code

Now that you can see how the functionality works for the ‘add’ section of the game, can you think about how we would add the multiplication and subtraction sections?

Task

Go through the math-duplicate.js code, and see how you would change this code to get the logic working for the **multiply** and **subtract** sections of the game.

We need to:

- Calculate the equation for multiply and subtract and display it when we load the page
- Calculate the correct result for both equations, to compare the users input with later
- Check the user input for both equations and see if it matches the correct result
- Disable the text boxes for each control if the user gets the answer correct.

Think about how you would do this, perhaps even give it a try and see how it would work.

Do you feel like adding these two sections would involve copying and pasting a lot of the existing functions, and then changing the logic slightly? This would lead to a lot of code duplication, which is not a good thing. I am going to explain to you why code duplication is not good, and how we can avoid it.

Code Duplication

Code duplication is when parts of your code are repeated over and over in the same project or JavaScript file. There are many problems with it:

- It makes your code longer and harder to read.
- If you need to make a change to one part of your code, you need to make the same change to every part of the code that contains the same logic.
- It makes your code harder to update or to fix if there are bugs.
- When you're working on a large project with multiple developers it means that your code can become slow.

Code duplication often happens when you have multiple controls on your page that do *almost* the same thing. For controls that do the **exact** same thing you can re-use a function. But if something differs slightly, many people just copy and paste the code and then change it slightly. It's tempting to do here, but we're going to find a way around it using better coding practices!

With a project like this, we are doing *almost* the same thing three times - for addition, subtraction, and multiplication. We are:

- Generating random numbers for all three equations which will be shown when the page loads
- Comparing the users guessed answer with the correct answer for each calculation.

This means we would essentially have three 'add' functions, and three 'calculate' functions, each performing a slightly different task using some similar and some identical code.

Instead of creating 4 extra functions with lots of code duplication, we are to use JavaScript objects to allow us to use the 'add' and 'calculate' functions for all three calculations.

In your 'project files' folder, copy the 'math.js' file into the 'js' folder of your project. Now update your <**script**> </**script**> tag in the math.html file to reference **'math.js'** instead of 'math-duplicate.js'.

```
<head>
    <link rel="stylesheet" type="text/css" href="css/style.css">
    <script src="js/math.js"></script>
</head>
```

Let's take a look at this code and see what it does**: (if you don't want to see how it works right now, and just want to get your game working, skip to page 15 for the next step)**

In the new math.js file, you can see that the **initialise** and **checkCorrectGuesses** functions are pretty much the same as the math-duplicate.js file. You can see that our variables are also the same.

Below our variables are three JavaScript Objects: **results**, **functions**, and **symbols**. These are what we will use to make the existing functions for all three of our equations, instead of copying and pasting code.

JavaScript Objects

JavaScript objects are variables, but objects can contain many values, like arrays.

However, while arrays store one value per row, objects store a pair of values within them. They can assign a value to each property that is within the object. An object acts like a container for a number of related variables, so that you can access all of these variables at once in the code by calling this object.

Let's use the example of a car for this.

Variable

var car = “mercedes”;

Array

var typesOfCars = [“Mercedes”, “Toyota”, “Fiat”]

Object

```
var car = {
    Type: “Fiat”,
    Model: “500”,
    Color: “Blue”
}
```

All cars have pretty much the same properties - type, model, color, number of wheels, engine size etc. So we can create an object called ‘car’ which can contain all of the properties that our car has. We can pass this object around the code and check all the features of our car in one line of code, instead of checking multiple variables.

Values inside an object are called key : value pairs (name and value separated by a colon).

On the left of the colon we have the key (Type), and on the right of the colon we have the value for our key (Fiat).

You access your values by searching your object for the key. The syntax is ***objectName[key]***

In this example: **car[“Color”]** returns **“Blue”.**

Let's look at the objects we have in our new ‘math.js’ file.

The first object we have is **results**()

We are going to use this object to store the correct result of our equations. This way, when we need to check if a result is correct - we can just call the ‘results’ object and ask it for the result that we need.

To set the result of the addition equation you type: ***results[“add”] = 17 + 14;***

To get the result later in the code, you type: ***results[“add”]*** to get the assigned value to this key.

```
var results = {
    add: 0,
    multiply: 0,
    subtract: 0
}
```

The next object is **symbols**()

Here we are storing the different symbols we need for our equations. We can call the object and ask for the symbol we need depending on our calculation.

```
var symbols = {
    add: '+',
    subtract: '-',
    multiply: '*'
}
```

When we are populating our equation labels on the screen, we need to get the symbol to put between our two randomly generated numbers. To do that we do the following:

```
document.getElementById(“add”).innerText = one + " " + symbols[“add”] + " " + two;
```

Lastly, we have an object called **functions.**

Here, instead of storing a **value** for our **key**, it stores a function.

Each function is the calculation that corresponds to the key.

You pass this function 2 values, and it adds, multiplies, or subtracts the values depending on which key you provide.

```
var functions = {
    add: function(number1, number2) {
        return number1 + number2;
    },
    subtract: function(number1, number2) {
        return number1 - number2;
    },
    multiply: function(number1, number2) {
        return number1 * number2;
    }
}
```

Can you see that all of these objects have the same name for their **keys:** add, multiply, and subtract .

This is important, and is what will allow us to update our **'calculate'** and **'add'** functions to work for all three calculations instead of creating 4 more functions.

Let’s take a look at the calculate function:

```
function calculate() {
    var one = 0;
    var two = 0;
    Object.keys(results).forEach(key => {
      one = Math.floor(Math.random() * difficulty) + 1;
      two = Math.floor(Math.random() * difficulty) + 1;
      results[key] = functions[key](one, two);
      document.getElementById(key).innerText = one + " " + symbols[key] + " " + two;
      document.getElementById(key + "Answer").disabled = false;
      document.getElementById(key + "Answer").value = null;
    });
}
```

It is almost the same, except any reference to ‘add’ is replaced with ‘key’, and we have an extra line:

Object.keys(results).forEach(key => {

What this line of code does is loop through our ‘results’ object and get the keys (**add, multiply, subtract**).

Then using something called a **‘forEach’** loop, it loops through the existing code we already wrote for math-duplicate.js 3 times, replacing the word ”key” each time with the name of one of the keys (“add”, “multiply”, “subtract”) .

The code above will loop through this code 3 times, once each for add, multiply, subtract, and perform the same tasks but using the correct add, multiply and subtract HTML elements, functions, symbols and results.

All we need to do is use:

- function[key] - to get the correct function for the key
- Symbol[key] - to get the correct symbol for the key
- Result[key] - to set the correct result for the key, which we will check when the user types in their answer

So when the code is running in your browser, it runs through this code 3 times.

First, it replaces the word ‘key’ first with “add”:

```
  one = Math.floor(Math.random() * difficulty) + 1;
  two = Math.floor(Math.random() * difficulty) + 1;
  results[“add”] = functions[“add”](one, two);
  document.getElementById("add”).innerText = one + " " + symbols[“add”] + " " + two;
  document.getElementById("addAnswer").disabled = false;
  document.getElementById("addAnswer").value = null;
```

Then it runs the code again, replacing the word ‘key’ with “multiply”:

```
  one = Math.floor(Math.random() * difficulty) + 1;
  two = Math.floor(Math.random() * difficulty) + 1;
  results[“multiply”] = functions[“multiply”](one, two);
  document.getElementById("multiply”).innerText = one + " " + symbols[“multiply”] +
" " + two;
  document.getElementById("multiplyAnswer").disabled = false;
  document.getElementById("multiplyAnswer").value = null;
```

And it runs the code one final time, replacing the word ‘key’ with “subtract”:

```
  one = Math.floor(Math.random() * difficulty) + 1;
  two = Math.floor(Math.random() * difficulty) + 1;
  results[“subtract”] = functions[“subtract”](one, two);
  document.getElementById("subtract”).innerText = one + " " + symbols[“subtract”] +
" " + two;
  document.getElementById("subtractAnswer").disabled = false;
  document.getElementById("subtractAnswer").value = null;
```

By doing this it creates three labels with equations,and saves three results in our result object waiting to be checked when the user types in their answer.

We have managed to populate all three labels in just one function!

Now let’s look at how we check our answers….

In the new math.js file, we have replaced our **Add**() function with a function called **check**(), because we are now using the function for more than just addition.

You can see below that we have declared the function differently to how we usually do

```
function check(type) {
    if (document.getElementById(type + "Answer").value == results[type]) {
        CorrectGuesses++;
        document.getElementById(type + "Answer").disabled = true;
        document.getElementById("progress").innerText = 10 - CorrectGuesses + " Left
To Proceed";
        checkCorrectGuesses();
    }
}
```

This function is now expecting a **parameter** named 'type' when it is being called. This means that to call 'check()' you need to supply a 'type' in the function call.

The code will replace the word 'type' within the function with whatever value you send the function. From reading the code, can you tell what the 'type' should be?

If you guessed "add", "multiply", "subtract" then you are correct!

Look through this code again, and imagine how it would look if 'type ' were replaced with "add", then "multiply", then "subtract". Try copying and pasting the code and replacing the word to see what the code looks like when the function is called - similar to what I did above with the calculate() function.

Updating the HTML with our new function calls

In the math.html, replace the <body> </body> tags and everything within them with the following code:

You can see that we are now calling our ‘check()’ function from each of our textboxes, and passing through our check ‘type’ to tell this function which type of check we want to do.

```
<body class="mathBody" onload="initialise()">
    <div class="main">
        <div class="mathElement">
            <label class="timer" id="countdown"></label>
        </div>
        <div class="mathElement">
            <label>Answer 10 equations correctly to move to the next
page.</label>
        </div>
        <div class="mathElement">
            <label id="subtract"></label>
            <input id="subtractAnswer" type="text" onkeyup="check('subtract')">
        </div>
        <div class="mathElement">
            <label id="multiply"></label>
            <input id="multiplyAnswer" type="text" onkeyup="check('multiply')">
        </div>
        <div class="mathElement">
            <label id="add"></label>
            <input id="addAnswer" type="text" onkeyup="check('add')">
        </div>
        <div class="mathElement">
            <label id="progress"></label>
        </div>
    </div>
</body>
```

Save your JavaScript and HTML file, and refresh the page - now your game should work as expected!

Styling your game

Copy the following code into your style.css file, and refresh the page to pick up your changes

```
.mathBody {
    background: url("../img/numbers.jpg") 50% 50% / 100% no-repeat fixed;
}

.mathElement {
    padding-bottom: 20px;
}

.mathElement input {
    height: 30px;
    width: 50px;
    margin-left: 20px;
}

.timer {
    color: red;
    font-size: xx-large;
}
```

- The **mathBody** class sets the background image of the page
- The **mathElement** class adds some spacing to the elements so they're not too close together
- The .**mathElement input** class, applies a height, width and margin to all input elements
- And the **timer** class makes the font of the timer red and the font size bigger.

Tasks:

- Play around with these classes and see how you can style the page differently.
- See if you can change one of the equations to be a subtraction equation.
- Change the code so that you only have 5 seconds to answer the questions
- Change the count to be higher or lower than 10.
- Increase the ‘difficulty’ variable level to 100. See what happens with the numbers?
- See if you can create an entirely new equation for subtraction. Update your existing objects and HTML, and see if you can do this!