WorkBook - 22AprEnable1

This is a structured WorkBook which will take you from zero to hero with HTML, CSS and JavaScript.

The document is a guided story that will take you step by step through all of the learning required to create of a front-end that can interact with an API.

Every so often you will see references to QA-Community. These are the topics on QA-Community which contain material for the next steps of the WorkBook. If you are getting stuck a good idea would be to read through the relevent section and complete any exercises to practice and build confidence.

HTML and CSS

```
**QA-Community > HTML > Introduction to Web Development**

**QA-Community > HTML > Hypertext Merkup Language**

**QA-Community > HTML > Tags**

**QA-Community > HTML > Headings and Paragraphs**
```

To create a HTML document two elements are required: a `DOCTYPE` tag and `html`.

1. Create a HTML document using both of these elements which displays the text 'Hello world!' when opened in a web browser.

When creating a HTML document we should create two elements that are children of the `html` element; `head` and `body`. The `head` element is where we store metadata about the webpage and some links to external resources, the `body` element is where we store anything we want to be displayed on our webpage.

2. In your HTML document enclose the text 'Hello world!' in the `body` element.

3. Add a `head` element to the page containing a `title` element with the text 'My first webpage'

HTML follows 'first opened, last closed' in relation to the ***tags*** used to create each element. This feeds into the Document Object Model or DOM. There are some exceptions to the rule such as the `DOCTYPE` element.

```
OPEN DOCTYPE
OPEN html
    OPEN head
        OPEN title
            My first webpage
        CLOSE title
    CLOSE head
    OPEN body
        Hello world!
    CLOSE body
CLOSE html
```

```
├── !DOCTYPE
├── HTML
│   ├── HEAD
│   │   └── TITLE
│   └── BODY
```

4. Make sure your HTML document adheres to the Document Object Model.

There are many types of elements in HTML, the most common being the `div` element. This is the generic method for how content is **div**ided into sections.

5. Enclose the text 'Hello world!' in the `body` element inside `div` tags.

In addition to generic elements there are also some pre-defined elements. The style of which may differ slightly between browsers. One such set of elements are headings which are numbered **1** through **6**.

6. Add a heading of size 1 to the top of the `body` element containing the text 'This is the biggest heading'.

Another element which is widley used is the paragraph element which has some default formatting for a block of text.

7. Add a paragraph under the heading you just created containing the text 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean quis urna euismod, dictum dui id, porttitor leo. Sed ultrices suscipit justo, id tristique odio tempor vitae.'.

You are also able to make words **bold**, *italic*, or <u>underlined</u> in HTML.

8. From the paragraph you just created make one word **bold**.

9. From the paragraph you just created make one word *italic*.

10. From the paragraph you just created make one word <u>underlined</u>.

11. From the paragraph you just created make one word ***<u>bold, italic and underlined</u>***.

As mentioned earlier some elements do not require closing tags.

12. Add a horizontal rule to the bottom of the `body` element.

13. Add a line break above the horizontal rule.

```
**QA-Community > HTML > Lists**

**QA-Community > HTML > Hyperlinks**

**QA-Community > HTML > Attributes**

**QA-Community > HTML > Images**

**QA-Community > CSS > Introduction to CSS3**
```

HTML also has the capability to create lists, there are two kinds of lists. Ordered lists and unordered lists. Both of these lists contain list items.

14. Create an ordered list containing three list items; 'item one', 'item two', 'item three'.

15. Create an unordered list containing three list items; 'something', 'another thing', 'final item'.

You are also able to **nest** lists inside on another.

16. Create an unordered list containing two list items and another unordered list inside of the list also containing two list items.

```
├── UNORDERED LIST
│   ├── LIST ITEM
│   ├── LIST ITEM
│   └── UNORDERD LIST
│       ├── LIST ITEM
│       └── LIST ITEM
```

A key feature of HTML documents is the ability use hyperlinks. This is accomplised through the use of anchor tags.

17. Add an anchor tag to the bottom of the `body` element which links to Google.

HTML documents are also able to display images but remember, images are one of the elements which do not require a closing tag!

18. Underneath the header on your HTML document create an image with a source of 'https://qa-community.co.uk/static/qa_fill_primary.svg'

For people on e-readers and the visually impaired it is good practice to have an attribute which describes the image.

19. Add an `alt` attribute to the image describing it as the 'QAC logo'

This image appears quite large when viewed in browser.

20. Use the in-line style attribute to reduce the image's width and height to 50px.

The style attribute does not *have* to be used in-line. It can also be created as a child element of the `head` section.

21. Create a `style` element at the bottom of the `head` and change the `font-size` of the `body` element to 20px.

The final place that we can load the style for the page from is an external resourse called a cascading style sheet.

22. Create a cascading style sheet in the same directory as your HTML file called 'style.css' and change the `color` (watch the spelling) to `white` and the `background-color` to `blue`.

23. Add a link to the bottom of your `head` element with a hypertext reference of './style.css' and a relationship of 'stylesheet'

```
**QA-Community > HTML > Forms**

**QA-Community > CSS > Bootstrap Introduction**
```

HTML also has some ability to send data using `forms`. These are made up of a method; how to send the information, and the action; where to send the information.

The `get` method sends a form's input to the URL bar with a 'name : value' pair and is used like a hyperlink.

24. Create a form at the bottom of the `body` element of your HTML document with a method of `get` and an action of `search.html`

Forms cannot work by themselves, they need an input for data and something which will trigger the form to perform it's functionality. Often this is a button.

25. Add an input to your form with the name `q` and the type `text`

26. Add a button to your form with the type `submit` and containing the text 'SEARCH'

Now that you have created a search form, try it and look at how the URL bar of your browser changes. Once you have, navigate to Google or Bing and search for something. These search forms are also using the `get` method, an action of `search` and an input with the name `q`!

Another common type of form is the `post` form. it is used to send data to an action URL which processes the data. These are often used for mail forms.

You do not **need** to specift an action or a method to create a form. They can exist to simply organise a group of inputs.

27. Create a new form with no action or method
28. Add an input to the form with a type `checkbox` and name `checkbox` and value `1`.
29. Add a `fieldset` element to the form containing four inputs of type `radio` and name `choice` give each of these a different value.
30. Add an input to the form of type `date`
31. Add an input to the form of type `color`

There are many types of input and each browser will have it's own way of styling those inputs.

In order to unify a HTML document's style accross multiple browsers and devices we use CSS. One of the most popular CSS frameworks is called Bootstrap.

32. Navigate to getbootstrap.com and add bootstrap to your HTML document. There are **TWO** different things to add to your file.

```
## CV Challenge

### Part One

1. Create a web page for an imaginary CV (if you need content you can head to
[loremipsum.io](https://loremipsum.io/) and fill it in with fake latin). It must
include sections for:
  - Contact Information

  - Personal Statement

  - Work Experience

  - Education

  - Skills


2. Include custom css. This can be in the `head`, using in-line styling or in a .css
file.

3. Upload your work to GitHub.

### Bonus

If you have completed the challenge, refactor your website to use bootstrap!

Additional sections you might want to include:

- Professional certifications

- Hobbies and Interests

- Languages

- Volunteering

- Projects

- Publications

- Awards and Conferences
```

JavaScript

```
**QA-Community > JavaScript > What is JavaScript**

**QA-Community > JavaScript > Strict Mode**

**QA-Community > JavaScript > Getting started with JS**
```

JavaScript does not come with it's own runtime environment, and as such it needs to be run through a browser (for now).

1. Create a new HTML document with a `DOCTYPE` a `head` and a `body`. Then create a file called `script.js`. Load the .js file at the bottom of the `body` element in the HTML document with the `script` element.

JavaScript is a very forgiving language. To force us to use best practice we can include 'use strict' at the top of our script.js file.

2. Include `'use strict'` at the top of your `script.js` file.

A simple way to check that our script is loading correctly is to output some text.

3. Use `console.log` to write `'Hello world!'` to the console
4. check the console in your browser to make sure that it is outputting `Hello world!`

The console can be used for different kinds of output.

5. Try using `console.log`, `console.info`, `console.warn` and `console.error`

The console output can also be styled using css.

6. Using `let`, create a variable called `text` with a value of `'This is a message'`
7. Create a `console.error` that outputs the variable with `color` set to `black` and `background-color` set to `white`

```
**QA-Community > JavaScript > Data Types**

**QA-Community > JavaScript > Variables**
```

There are a few different types of variable in JavaScript, best practice it to use `let` to create mutable variables but there are two others.

8. Create a constant variable and output it using `console.log`

We no longuer use `var` when developing in JavaScript though you might still see it pop up in older codebases from time to time.

Variables in JavaScript can be used to store many different **Data Types** there are two 'kinds' of Data Types; primitives and objects.

9. Create variables with `let` for every different type of primitive DataType and output them to the console with `console.log`

Objects are how we store unordered 'key : value' pairs in JavaScript.

10. Add two more 'key : value' pairs to the object below, they must be comma separated. Then output it to the console using `console.log`

    ```
    let myFirstObject = {
        key : `value`,
        name : `my first object`,
        age : 42,
        awesome : true
    }
    ```

Strings can be mutated in two ways, concatenation and interpolation. Concatenation is the joining of strings end-to-end, it is quite memory intensive as multiple strings must be created and stored in memory to output the final result.

11. Create two variables using `let`, a string with a value of `'1 + 1 = '` and a number with a value of `1 + 1`. Use string interpolation to output tge string and the number to the console using `console.log`

The reason this will work is that Data Types in JavaScript are mutable and stored as text in memory until they are used, so the number can be converted to a string with ease.

The other method of mutating a string is interpolation where string literals containing placeholders are used. These are called template literals.

12. Use a template literal to output the calculation in question 11

```
**QA Community > Javascript > Iteration**
```

Like most programming languages, iteration is an integral part of JavaScript. There are three main types of loops we use; the **for** loop, the **while** loop and the **do-while** loop.

13. Create a **for** loop which uses an iterator `i` starting with a value of `0` which loops as long as `i` is less than `10`, increase the value of `i` by `1` every loop. Every time the loops runs output the value of `i` to the console using `console.log`

While loops work a little differently, they use booleans and tun until a condition is no longer true.

14. Create a variable with `let` called `condition` with a value of `true`. Create a variable with `let` called `increment` with a value of `0`.

15. Create a **while** loop with an argurment of `condition`, every loop increase increment by `1` and outputs increment to the console using `console.log`. To stop the loop from executing infinitely create an **if** statment inside the while loop which will change the value of `condition` to `false` if the value of `increment` is equal to `3`.

Do-While loops are similar to while loops but will always run at-least once.

16. Create a variable with `const` called `notThis` with a value of `false`. Then create a **do-while** loop with an argument of `notThis` which will output the string `'this is a do-while loop'` to the console using `console.log`

The final kind of loop is called a **switch case** and though it is not technically a loop and is often used to replace large if/else if/else trees, it has looping functionality. Switch cases will only exit a loop on a `break` or `default`.

17. Create a variable with `let` called `num` with a value of `1`, then create a **switch** with the argment of `num`. Inside the switch create a `case` for `0`, for `1`, for `2`, and a `default`. Inside each of these output a unique message the console using `console.log`

```
**QA Community > Javascript > Conditionals with Truthy/Falsey**
```

All data in JavaScript has an inherent feature of being truthy or falsey, that means resolving to true or false when mutated.

18. Try running the code below and observing the output.

```
if ('') {
    console.log(true);
} else {
    console.log(false);
}

if (' ') {
    console.log(true);
} else {
    console.log(false);
}
```

19. Try the above code with different values to check if they are truthy or falsey.

In JavaScript we can use **if** statements to check a condition.

20. Create a variable with `let` named `str` and a value of `'hello'`. Then create an **if** statement to check that `str` is equal to `'hello'`. Output to console `'string is equal to hello'` using `console.log`.

What if a condition is not met? In JavaScript we can use the **else if** statement to run additional checks.

21. Change the value of `str` to `'goodbye'` then add an **else if** statement to the end of the existing **if** statment to check that `str` is equal to `goodbye`. Output to console `'string is equal to goodbye'` using `console.log`.

To catch all other outcomes we can use an **else** statement.

22. Change the value of `str` to `' '` then add an **else** statement to the end of the **else if**. Output to console `'string is not equal to hello or goodbye'` using `console.log`.

There is a shorthand for **if / else** statments in JavaScript called a **ternary if**.

23. Convert the code below into a **ternary if** statement.

```
let x = 1;
if (x == 1) {
    console.log(`x is 1`);
} else {
    console.log(`x is not 1`);
}
```

As type is mutable in JavaScript it will try to work out what you are trying to compare when running comparisons, this means that you can compare seemingly different data types. The way we stop this from happening is with the **strictly equal operator**.

24. convert the code below to use the strictly equal operator.

```
42 == `42` ? console.log(true) : console.log(false);
```

```
## FizzBuzz Challenge

Write a short program that prints each number from 1 to 100 on a new line.

For each multiple of 3, print "Fizz" instead of the number.

For each multiple of 5, print "Buzz" instead of the number.

For numbers which are multiples of both 3 and 5, print "FizzBuzz" instead

### Hint

- use a for loop

- And either a switch case or if / else if / else statements

### Bonus

Convert your fizzbuzz to use ternery if statements (these can be nested)
```

```
**QA Community > Javascript > Objects, Arrays + JSON**
```

Objects in JavaScript are an unordered collection of **key : value** pairs. One way we can create object in JavaScript is like so:

```
let petDog = new Object();
petDog["name"] = "Clifford";
petDog["type"] = "Dog";
petDog["size"] = "Big";
```

25. Try creating the object above using **literal** notation.

Objects can be stored in an array.

26. Create an array called `pets` and another object called `petCat` using literal notation. Add `petDog` and `petCat` to the `pets` array.

It is possible to loop through arrays in JavaScript, the standard way is with a **for loop**. But there is also an **enhanced for loop**

```
let x = [a,b,c,d,e];

for(let i = 0; i < x.length; i++){
    console.log(x[i]);
}
```

27. Convert the above code to use an enhanced for loop.

There are also several **array object methods** that can be used on an array to carry out predetermined functionality.

28. Reverse the array `x`

29. Output the array `x` as one long `-` separated string using `console.log`

A subset of literal notation for object in JavaScript is JSON or JavaScript Object Notation. The main difference being that the information does not need to be stored in a variable and the *keys* must be surrounded by quotation marks `"`.

30. Create a JSON object with two key : value pairs

JavaScript has an inbuilt method to parse JSON data.

31. Using `JSON.parse` convert your JSON object into a JavaScript Object.

There is also a method to convert a JavaScript Object into a JSON string.

32. Using `JSON.stringify` convert `petDog` into JSON.

```
**QA Community > Javascript > Functions, function expressions and arrow functions**
```

Functions in JavaScript are a block of code that can be evoked when called by something. We use these to better adhere to **DRY** principles. They can take arguments and are globally scoped.

33. Create a **function** called `output` which takes in one argument. When the function is called `console.log` the argument.

When a function reaches a **return** statement it stops.

34. Create a function called `request`. When the function is called have it return a string of value `'response'`.

The return statment can be used with arithmetic operators.

35. Create a function called `sum` which takes in two arguments. When the function is called have it return the sum of the arguments.

In JavaScript ES6 we can use arrow function expressions as an alternative, more copact, regular function expression.

36. Convert the regular function below into an arrow function.

    ```
    function log(str) {
        console.log(str);
    }
    ```

```
**QA Community > Javascript > DOM Manipulation**
```

JavaScript is able to interact with webpages through **DOM** (document object model) **manipulation**.

37. Add a `button` element to the top of the `body` element with an `id` of `button` a `type` of `button` and the text 'press me'.
38. Create a new variable using `const` in your `script.js` file with the name `button` and a value of `document.getElementById('button')`.
39. Create a function that will change the `innerHTML` of the button to 'I have been pressed!'
40. Use the `onclick` method with the `button` variable to call the function you just created.

```
## Calculator Challenge

- Using the files provided or creating your own from scratch, build a working
calculator in HTML, CSS and JS.
- Try adding bootstrap to your calculator.
- Provide it two inputs for calculations.
- Use a separate JS file to manage your DOM.
- When loading the calculator script, make sure the script type in your .html file is
"module".
- You must be able to add, subtract, multiply and divide the inputs.
- Every new result should be written to the page as a new list item in the Output
List.
- Upload your work to GitHub.

### Bonus

- Display the full calculation in the output list (e.g: 1 + 1 = 2).
- Try converting any functions that you can into arrow functions.
- Add an 'equals' button and store the calculation in memory, only writing it to the
Output List when the equals button is pressed.
- Create a function to clear the Output List and both inputs.
- Hook that new functionality up to a button.
```

calculator challenge files

```
**QA Community > Javascript > Fetch API**
```

JavaScript also has the capability to make requests to APIs and to perform actions on the responses. It does this through the use of the fetch method and promises. The promise we use is then which can take two option arguments, a callback for success and a callback for failure.

41. In your HTML document create an input text field, a button of type button and an output paragraph. Then create variables for each of these in your `script.js` file using `const`.
42. Create a function called `read` that takes an argument of `url`. Implement the fetch method using the URL, on a successful callback write the response to the output paragraph. On a failure send the error to the console.
43. Add an event listener for the button's onclick event to call the read function. Test the functionality by inserting 'https://emojihub.herokuapp.com/api/random' into the input text field on your HTML document and pressing the button.

A powerful JavaScript library that can streamline fetch requests is axios. Add the line

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

above where you are currently loading script.js in your HTML document.

44. Comment out the `read` function in your script file and insert the following code as a replacement:

    ```
    read = URL => {
      axios
        .get(URL)
        .then((response) => {
          output.innerHTML = JSON.stringify(response.data);
        }).catch((err) => {
          console.error(err);
        })
    }
    ```

45. If the above code works then axios is correctly installed. Try writing a `create` function under the `read` function targeting the URL https://reqres.in/api/users. Send an object containing two keys, one of `first_name` with a value of your name and one of `email` with a value of a random email.

```markdown
## API Challenge

Design a product using one of the APIs listed here: [Public API Lists | public-api-lists](https://public-api-lists.github.io/public-api-lists/)

### Specification

Your product offering should include the following:

- use case for product

- working front-end that interacts with API

- clear reasoning for UI/UX choices
```