T3A1 - Workbook

Q1

"Source control is a practice to track and manage changes to software code. Source control management (SCM) gives a running history of code development and helps to resolve conflicts when merging contributions from multiple sources" (AWS, viewed 30 Jul 22). Each time a change is made, the details of who and what was changed should be recorded, so that it can be reviewed later. This information can be useful in saving time to identifying and fixing bug(s) if they are introduced.

How can SCM help a large project reduce risk and make collaborating with each other more efficient? When working in an organisation setting with a large project and multiple teams there is more than one developer sharing the same codebase at the same time. Other projects and teams may be required to work on code that overlaps on the same software function. This can significantly increase the number of developers and make the project more complex to manage and co-ordinate. For this reason, consistent SCM tools and processes across the organisation is critical for maintaining a single source of truth and reduce the development cost repository (Atlassian Bitbucket, viewed 29 July 2022).

One of the main tools in SCM is source control, also known as version control. Source control tools keeps track of every change made to the code during the development life cycle. The main function for source control is to protect the source code from being compromised and damaged. Source code is like a software's blueprint, containing information on how the software works, why changes were made and details of functionality developed over time. Some of the source control tools include Git which is an open-source tool, CVS, SVN or Perforce Helix Core. Based on the organisations needs; they may choose to store changes on a single server called centralized system like VCS or Perforce or clone a copy of the repository by using a distributed version control system like GIT or Mercurial (Software Testing Help, 15 July 2022).

Source control tools, allow multiple developers to continually make changes to the same codebase and collaborate more efficiently. It tracks who, when and why the changes are made so that it is tracible. With larger projects more people are working on the same codebase. The likelihood of mistakes or code changes made not being compatible with another developer's changes can increase. When this happens developers can look back at the history, compare the code prior to their change to help solve and fix issue earlier and faster before moving forward. It gives developers comfort that they can always safely revert their changes if they have accidently overwritten over code and caused conflicts or introduced bugs in the code base.

Without SCM, more care is required to co-ordinate the timing of each release and ensure that they do not over-lap which can cause issues. With SCM it allows developers to create a separate branch, pulls the latest version of the code and work on the necessary changes independently on their local environment. When they are ready with their changes, the developer can check if there any changes from other contributor and choose to pull that code into their local repository to make sure there is no issues before merging their own changes back into the main version of the project repository. The following are SCM best practices to ensure smooth running of a large project (Atlassian Gitbucket, viewed 31 July 2022, What is source control?):

- Ensure everyone in the project have a consistent shared approach to source control workflow and practices before development begins
- Committing frequently to give quicker feedback and capture update to the codebase
- Git pull or fetch latest version before making changes or updates as this can help to reduce issue when you are merging your code
- Being diligent with making accurate notes when committing code so they can be useful and informative for future contributors
- Review changes and check what it entails before committing code in the staging area. This gives the developer a final chance to check their changes before committing their code

- Using branches as a way to enable each developer to work on separate features at the same time and merge into the main or master branch when they are ready

Q2

What standard and which aspect makes a quality software can depend on the user's perspective. This was the main reason ISO/IEC 9126 was developed. To provide a common understanding for all users in terms of what software characteristics make up a quality software. ISO/IEC 9126 standards can also help to define software development objectives and help evaluate if have been meet (Wikipedia, 18 Nov 2021). There are six main standards of software quality characteristics: Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. Each of the six aspect of quality is divided further into sub-characteristics and detailed below.

Functionality: If the software features are enough for the main task it was built for and does it work as expected according to specified requirements. The software function(s) not only needs to accurately interact with its users by handling and returning information correctly, but also accurately exchange information between each component that make up each function of the software. Interoperability relates to how different software systems interact with each other and a quality software means it can be integrated with a number of systems without requiring major changes (R. Cameron, 7 Jan 2013).

Reliability: If the software behaves consistently, is repeatable and predictable. Failures or errors should hardly occur. If the system was to crash, can it be recovered quickly in terms level of performance or data affected by the failure (R. Cameron, 7 Jan 2013).

Usability: The Software is easy to use and operate. The effort it takes for the user to learn the how to use the software and its functions should be as minimal. Information and functions should also be clearly presented. It also includes understandability standards, meaning that the software enables its users to work out what particular task and scenarios is suitable for and if it fits their needs.

Efficiency: The software to take the appropriate level of time and memory to process and respond when it is performing its function. “Efficient software accomplishes as much as possible with the least amount of resources” (Reinagel S. 15 Nov 2016).

Maintainability: Can the software be easily changed to cater for new enhancement or future requirements. If faults are to occur can the cause be easily identified by logs or exception reports and be fixed. Any changes made can be validated to ensure there are no bugs before deploying into production. Documentation is also critical to ensure maintainability as parties developing the code are not the ones to maintain it.

Portability: Can the software be used in different environments, hardware or operating systems other than the one it was originally intended for. Portable software can be adapted and be reused in different projects with minimal changes (Reinagel S. 15 Nov 2016).

Since ISO/IES 9126 in March 2011, ISO 25010 has since been issued in 2005 to adapt for changes in the Information System environment. This includes Security characteristics which judges the confidentiality, authentication and accountability of the software. This aspect of software quality has increasingly gained importance for users.

Q3

MERN is a JavaScript orientated web stack that consists of four main technologies. Mongo DB, Express, React and Node make up the MERN full-stack architecture that consists of three parts: Front-end, back-end and database.

React.js is responsible for the front-end or web interface. React.js is an open-source JavaScript library initially developed for Facebook and now maintained by a community of developers. It is a framework used to build dynamic and interactive user interface components in HTML. React components connect to the database on the backend server and then renders them in HMTL (MongoDB, viewed 6 Aug 2021). The service part of React.js handles changes based on user interaction and feedback results based on the user’s request and response. The service also handles data for CRUD operations of the application. Actions like clicking triggers the application to navigation to another page via the router and generating a component.

The back-end part of MERN consists of Node.js and Express.js. Express.js is a server application framework that runs inside Node.js. Its main function is to facilitate communication between the client/front-end and the server/back-end. When a HTTP request is made on the browser, React.js will send that request to Express.js which then matches it to the server responsible for that specific function. Node.js is a runtime environment built to handle asynchronous connections and interprets JavaScript code (Yeshwanthinis S, 21 March 2021).

MongoDB is the database used for MERN framework. JSON like documents is created in React.js is sent to Express.js server to be processed and stored in MongoDB. It is a NonSQL database which means it uses collections and documents instead of relational databases that use tables and rows. For this reason, MongoDB is ideal for storing, displaying and manipulating JSON files (Yeshwanthinis S, 21 March 2021).

Q4

Website development process involves more than technical skills. Developing a website for small business require careful planning and analysis to ensure it meets the client's needs, expectation, time-frame and budget. Collaborating with other team members, teams or department requires effective communication and interpersonal skills. Project management in terms of how to plan and track progress is important to make sure the project is delivered on time and the client is engage throughout the process.

Business analytical skills will help to work out who is the websites target audience, to research the business and scope out their competitors, so they know how the website can stand out. It also helps to engage with the client and understand their goals and objectives so it can be translated into the website design. Client's may also require assistance in working out what their needs are and keeping them engaged (Glorify, 1 march 2021).

Expertise in web design and user experience can be important to ensure the website looks and feel visually appealing while being responsive and functional for the user. This includes strategically making decisions for user interface features like drop down menus, tool tips, navigation or buttons to enhance user experience.

Database technology, Front-end developer and back-end development knowledge is critical in order to developing a website. Making decision on how the website application data should be organised, edited, retrieved and save so that it functions correctly require good understanding on database technology. Entity Relationship diagrams should be designed and created earlier on in the project to ensure data is stored consistently throughout the project and data integrity is maintained.

Users visiting the website mainly sees and iterate with the interface coded and built by front-end developers. Front-end developers focus on what the sites layout, pages, images and navigation should look like. Their main objective is to ensure the website is easy to navigate and intuitive for the user. This requires knowledge in coding languages such a HTML, CSS, JavaScript or Ruby and an understand what makes a responsive design across different platforms and browsers.

Back-end development expertise is critical to ensure that the website functions and behaves correctly. They focus on the database, application programming interface (API), technical architecture and servers. They write code to help the browser interact with the database. Back-end developers must have knowledge of the tools and frameworks required to develop the website such as JavaScript, Ruby or Python. They make sure the back-end performs tasks efficiently and securely while interaction between front-end users works seamlessly (Coursera, 19 July 2022).

Q5

One of my projects Marketplace applications required using Ruby on Rails framework. To develop the functionality and features for my application I firstly needed to understand Rails and the Model-View-Controller architectural pattern. As it was first time exposed to Rails, I started practicing on a few smaller projects. This helped me progressively gain confidence in knowing what component and files I needed to add or change in order to implement features I wanted. Through repetition on Rails commands and the installation order for PostgreSQL, gems, models, controllers and views, I felt more capable by the time I developed my Marketplace application. Using gems such as devise helped to build security features required for my application.

I used Bootstrap and Simple forms CSS frameworks to style the user interface of my application to look more appealing. Both these CSS frameworks had default built-in templates for the user Interface, making it quicker to develop my website. To tailor the theme, colours and style of my application it still required knowledge of CSS and HTML. When integrated with my application I required to know where and what changes I needed to make in order for it to fit with my application features and look and feel. It took a little experimenting, referring to the documents and checking the internet for resources but being an open-source framework, it was not hard to find information. I still find the front-end time consuming and fiddly but taking advantage of Bootstrap and Simple form did save me time.

The Marketplace Application used PostgreSQL as the database. Understanding how each table used on the application in terms of the relationship they have with each other impacted the feature implemented on my application. Knowledge on how to query, change, add or manipulate each table on PostgreSQL assisted during the development and testing phase. An example was when I added a field or column on an entity incorrectly. This meant that it was not displaying correctly on the front end. To overcome this challenge, I had to search for online resources and learning material. Eventually through practice I was able to fix the issue.

Having Ruby knowledge was important to drive the behaviour and logic on my application. This drove what was displayed on the front end and the features available to make the application more user friendly. Knowledge on Ruby code was integral to make sure that the correct data was shown on the front-end accurately to the user, time and page. Ruby code also influenced weather user interaction with the application was stored correctly and was retrieved at the right time and location.

Project management skills was important to make sure that I completed what features that I planned to complete by the due date. Using tools like Trello made it easier to plan and track my time line and get an overview on what tasks I still needed to complete. Breaking down the project into smaller manageable task, with details, check-list and due date at the beginning of the project made it easier to manage. I think this skill is important to make sure that I stayed on target and allowed me to plan what features I could and need to delivered at a minimum and clearly allowed me to see where I was at any point of the time.

Q6

I believe my knowledge with Ruby on Rails was sufficient to achieve the minimum viable features that I aimed to develop for my Marketplace application but more practice on using the framework and Ruby language could have improved my application in the following ways.

Being able to include searching, sorting and filtering capability. I think these features would have allowed the user to interact with the website more. Allowing the user to target their search would have been more engaging for the user. I started to code for the sorting feature but couldn't get my function to extract the information according to category. I think being able to refine my Ruby knowledge and practicing how on how to solve these problems in terms of Ruby code would help with my future projects.

Minimize database calls to make running my application faster. I did not plan time to refactor my code. I believe there are ways to make sure I am not making multiple repeated queries when only selected data is required. I know that making my code more efficient will be helpful but exploring ways or tools to measuring the performance would be useful for my future projects.

Lastly having more practice and exposure to CSS styling, tools or frameworks can further improve my front-end development skills. I believe this knowledge will not only make it more efficient but it makes the applications I develop for future projects more intuitive and personal for the target users.

Q7

Control flow in JavaScript is where one piece of code controls when another piece of code is going to run. JavaScript executes code from top to bottom unless it is changed with control flow structures such as conditionals, loops, or functions (Cleary R. 18 July 2020). Control flow structures give the program the ability to respond to changing conditions to the order in which the code is executed.

Some of the most common structures used in JavaScript are similar to other languages but only with different syntax and names. Conditional statements check to see if a certain condition is either true or false. If the condition is true then run a specific code, if it's false then run another piece of code. The following example demonstrates an if, else if and else statement. Including else if statements allow for more complicated logic and multiple conditions to be checked and executed. Often if there is only one of each condition that results in true and false, the else if conditions are not required.

```
let completedQuestion = 7;

if (completedQuestion = 13) {
          console.log('Amazing job! You are DONE!`);
} else if (completedQuestion = 7) {
          console.log('You are half way. Keep up the good work.');
} else {
          console.log(`You are making progress!');
}
```

Variance of if statements are switch or ternary operators. The Ternary operator provides a cleaner and easier way to write and if else statement. The ternary operator consists of three parts. The example below checks if completedQuestion equals to 13. If true then 'Amazing job! You are DONE!' will print, otherwise it will print 'You are making progress!'

```
let completedQuestion = 7;

if (completedQuestion = 13) {
          console.log('Amazing job! You are DONE!`);
} else {
          console.log(`You are making progress!');
}

completedQuestion = 13 ? [true] : [false];
```

JavaScript allows another way of testing multiple conditions using switch statements. Switch statements is best suited when there are more than 3 or 4 conditions. It takes a single value and looks through the list of cases until the case matching the value is found. If any of the proceeding cases match the value, then the corresponding code within that value will be executed. In the example below, the function takes in a number as an argument. The switch statement has different cases which return strings corresponding with the number of questions completed (Cleary R. 18 July 2020). At the end of each case the break will break out of the code. For the example below, the console.log function(getPercentageOfCompletion) passes in seven. This will print out 'Half way there. Keep up the good work!' to as it is the corresponding code to case seven.

```
function getPercentageOfCompletion(num) {
     switch (num) {
          case 1:
               return ‘Good start’;
          case 3:
                return ‘You are making progress’;
       break;
          case 7:
                return ‘Half way there. Keep up the good work!’;
           break;
          case 11:
                return ‘Two more questions to go!’;
       break;
          case 13:
                return ‘Amazing job! You are DONE!`;
     }
}

Console.log(getPercentageOfCompletion(7))
// Output: Half way there. Keep up the good work!
```

Loops are ideal to perform repetitive tasks with less code. The statement within the loop code will continue to run until there it runs out thing to loop over or if the condition is false. For example, using loops to iterate over an array and perform some sort of action on each piece of data in the array, or using a loop to compile a list of all the elements. There are many types of loops in JavaScript but the most commonly used are for loops and while loops.

The for loop is used to go over a piece of code for a set number of times each with a different value. A for loop consists of three parts. ¬¬¬ First is an initialiser which is executed first. Secondly the condition which defines when the code block is executed. As long as the condition is true, then the code block will run. The last part is the final expression, which is run after the code block is run. In the example below the initializer is 'i' equalling the value of zero. The condition is checked to see if the value of 'i' is less than or equal to ten, then the code block will run and print out the value of 'i'. Finally, each time the loop is run, the code takes the value of 'i' and increments it by one. This will continue until eleven where the condition equals to false and the loop will end. 'i' is a convention when writing a for loop but it can also be replaced with a variable such as number. The final expression can also be used to decrement instead of incrementing (freeCodeCamp, 15 Feb 2020).

```
for (let i = 0; i <= 10, i++) {
          console.log(i);
}

// Output :
// 1
// 2
// 3 ...10
```

Variables of the for loop are for in and for of loops. For in loops can be used to loop through properties of an object where the code within loop will be executed once for each property of an object. The for of loop was introduced as part of the ES6 update and iterates over the values of many types of iterables data structures like arrays, and special collections such as Map and Set. The code block will be executed for each value in the iterable object (freeCodeCamp, 15 Feb 2020).

While loop is similar to for loop as it repeats a block of code while the condition equals true. While loop is used when you do not know how many times the loop should be run. Like the for loop, a while loop has an initializer, condition and final expression. The do while loop is similar to the while loop in that it runs when the condition is true, however it executes the code block first before it checks the condition.

The example below is for the while loop. It allows someone to complete up to ninety-nine percent of the Workbook. The initial value is set as zero, while percent is less than one hundred it will increment the percent by one and print out "You are making good progress." This will keep running until the block code stops running at 100. It will exit the while loop and print "You have completed your workbook. Well done!".

```
let percent = 0;

while (percent  <  100) {
    percent++;
    console.log('You are making good progress.');
}

console.log('You have completed your workbook.  Well done!');
```

Functions are blocks of code that can be run whenever we want. They are also known as first-class objects meaning they can be treated like objects in that it can be stored inside variables, can be passed as parameters, used as arguments and can return functions from another functions. The order of which function is called first, controls the flow on which set of tasks is executed regardless if it was declared first in the code. The function has a name and takes one or more arguments. Returning values are optional and you need to explicitly write 'return' the keyword if you want a value returned (freeCodeCamp, 15 Feb 2020). There are also anonymous functions which is a function without a name.

The following example is of an anonymous function where the keyword 'function' is used to declared and a parameter 'input' follows after it. Once the anonymous function is being declared it is being stored inside the const. The function can be called in the same way as the named function. In this case the 'show' function is retrieved from the variable, called and print out "This is an anonymous function" (Etherington M, 2020).

```
const show = function(input) {
     console.log('input');
};

show('This is an anonymous function');
```

Q8

Type coercion is when JavaScript automatically converts the values of one data type into another data type. JavaScript only coerces to the string, number and Boolean primitive types and often when using different value types to perform a particular task. Type conversion is often confused with the type coercion process. The difference is that type conversion is where a developer write code to explicitly convert a value type to a specific type (Maldonado L. 19 July 2021). Type coercion on the other hand is automatically or implicitly done by JavaScript.

Type system is a set of rules assigns to specific data types in computer programs. Understanding JavaScript's set of default type system rules may make writing code more efficient and shorter without having to explicitly convert to specific data types. At the same time, it can cause unexpected results and bugs.

String Coercion: When a string is added to a number using the plus(+) operator, JavaScript first convert the data types to string. The '+' operator acts as a concatenation of two values when instead you would expect an error to occur as you shouldn't be able to add a string with a number (Sebhastian N, 22 Jan 2021). Below are examples of string coercion.

```
console.log('1' + 1);  //  Output: '11'
console.log('1' + 0.222);  //  Output: '0.2221'
console.log('1' + true); // Output:  1true
console.log('1' + undefined);  // Output: '12undefined'
```

Number Coercion: The following examples is how JavaScript performs type coercion on strings with operations like subtraction (-), multiplication (*), division (/) and modulus (%). JavaScript takes the string and coverts it implicitly into a number. The plus(+) does not occur with number coercion(Gutte N, 30 May 2021).

```
console.log('12' - 1); // Output: 11
console.log('2' * 2); // Output: 4
console.log('12' / 2); // Output: 6
console.log('10' % 2); // Output: 0
console.log('Work' * 2) // Output: NaN
```

Boolean Coercion: This is where Boolean values such as true and false is implicitly converted to a number. JavaScript converts true value to 1 and false value to 0.

```
console.log(true + 1); // Output: 2
console.log(false - 1); // Output: 1
console.log(true * undefined); // Output: NaN
console.log(false / null); // Output: NaN
console.log(true % Nan); // Output: NaN
```

Q9

In JavaScript a value is a certain data type. Data types specify what kind of data can be stored within the program. JavaScript data types can be grouped into primitive or non-primitive category. Primitives is another word for basic and the data that they hold are single data values that can't be changed. The following is their characteristics and their examples:

Number: JavaScript has only one kind of number data type which include both numeric and decimal numbers. Mathematical operations can be performed on numbers such as; addition (+), subtraction (-), multiplication (*), division (/) and modulus (%). Nan and Infinity are also non numeric values that is included in the number data type. Infinity being a value that is greater than any number and NaN is a computational error resulting from an undefined mathematical operation(Javascript.Info, 10 July 2022). For example:

```
let num = 7;
let num = 7.55;
console.log( 1 /0); // Output: infinity
console.log('not a number' / 2 - 1); //Output: NaN
```

String: Are always surrounded by single quotes, double quotes or backticks. Backticks can be used to embed variables and expressions into a string(Javascript.Info, 10 July 2022). They are text values in single letter or a paragraph.

```
let str = 'green';
let favFruit = 'My favourite fruit is strawberry';
console.log( `My favourite colour is ${str}` ); // Output: My favourite colour is green
```

Boolean: Has only two values, true and false. Boolean data type can be used to compare and assess a condition and then perform an action depending if the outcome is true or false.

```
console.log(100 > 1) // Output: true
console.log(1 > 100) // Output: false
```

BigInt: Was create to handle extra-large numbers. The "number" data type can't store any numbers greater than and less than (253-1) or an error will occur as all the digits can't fit into a fixed 64-bit storage. Numbers larger than that can only be stored as a BigInt data type. Adding 'n' at the end of a long number can declared at BigInt. This may be rarely used but it is available in cases it is required(Javascript.Info, 10 July 2022)..

```
let bigNumber = 100003497663456494949n;
console.log(typeof bigNumber); // Output: bigint
```

Symbol: Can be created by invoking the symbol function and is used to create unique identifiers for objects. This data type was added in ES6.

Undefined: A variable that has not been assigned a value. The example below shows that the variable 'dType' has been declared but not initialized it with a value. When you log the typeof variable, the console prints 'undefined'.

```
let dType;
console.log(typeof dType); // Output: undefined
```

Null: Is used to declare a value as empty. Unlike undefined, it is specifically declared as the value.

There is one non-primitive data type in JavaScript called Objects. This includes an array, a function and objects itself. It is called non-primitives as the data type is used to store multiple and more complex data and values. An array is a list of different values. To access a value in the array, The following is an example of an array:

```
Let myArray = ['Nhi', 40, 'Developer soon', 2.2];
console.log(myArray); // Output: (4) 'Nhi', 40, 'Developer soon', 2.2
console.log(myArray[0]); // Output: Nhi
```

Function is another object data type. The example below shows how a function can be created. By using the parenthesis at the end of the function name calls the function.

```
function hello() {
       Console.log('hello');
}

hello() // Output: hello
``

Objects can be used to store a collection of properties.  The properties of an object define
```

let person = { name = 'Nhi', age = 35, student = true, address: { number: '6', street: 'High street', city: 'Melbourne', state: 'Victoria', postcode: '3000' }

} console.log(person.name); // Output: Nhi

Q10

An array is an ordered list of values that can be of any data types stored under a single variable name. Each of the values in the array can be accessed by the variable by specifying an index. The data in arrays can be manipulated using a variety of methods to make it accessible to perform specific tasks. The following are some of many JavaScript methods used to add, remove and alter the data in an array (Walker J, 9 March 2021).

Adding to Arrays: push() – Adds new elements to the end of the array

```
let colours = [‘green’, ‘red’]
colours.push(‘yellow’); // Output: green, red, yellow
```

unshift() – Adds new elements to the start of the array.

```
let colours = [ ‘green’, ‘red’]
colours.unshift(‘yellow’);  // Output: yellow, green, red
```

splice() – Inserts elements to a specific index. The first argument (1) is the index to start or insert new elements. The second argument is the number of arguments to delete from the array. ‘0’ in the example below will retain the existing elements. If the second argument is not declared, elements after the starting from the given index in first argument will be deleted. The third argument is the element to be added(Walker J, 9 March 2021). .

```
let colours = [‘green’, ’red’]
colours.splice(1, 0, ‘yellow’); // Output: green, yellow, red
```

Removing from Arrays: pop() – Removes the last element from the array

```
let colours = [‘green’, ‘red’, ‘yellow’]
colours.pop() // Output: green, red
```

shift() – Removes the first element from the array

```
let colours = [‘green’, ‘red’, ‘yellow’]
colours.shift() // Output:  red, yellow
```

splice() – Removes one or more elements from a specific index. ‘Yellow’ is the value to remove. The second argument ‘2’ remove two elements from the end of the array.

```
let colours = [‘green’, ‘red’, ‘yellow’, ‘blue’]
colours.splice(colours.indexOf(‘yellow’), 2);  // Output:  green, red
```

Mapping Arrays: map() – Is a powerful method commonly used to iterate over the array and change its elements using a callback function. The callback function will then be executed on each of the elements in the array. A new array containing the results of each function call is then returned (Walker J, 9 March 2021).

```
let arr = [2, 3, 4, 5];

let modifedArr = arr.map(function(element) {
        return element * 3
});

console.log(modifiedArr); // Output: 4, 6, 8, 10
```

Filtering Arrays: filter() method also takes on a callback function and creates a new array with all the elements that pass a certain condition implemented by the callback function(FAM, 5 May 2021.

```
let colours = [‘green’, ‘red, ‘yellow’, ‘grey]
let gColours = colours.filter( word => word.charAt(0) === ‘g’);

console.log(gColours);  // Output: green, grey
```

Altering Arrays: The examples below show how the value of an array can be changed. The first element is assessed by the ‘0’ index number and changed to ‘emerald’.

```
let colours = [‘green’, ‘red’, ‘yellow’]
colours[0] = ‘emerald’;

console.log(colours);  // Output: emerald, red, yellow
```

concat() – Combines two arrays together and then return a new array.

```
let primColours = [ ‘red’, ‘yellow, ‘blue’]
let secondColours = [ ‘orange, ‘green, ‘violet’]
let allColours = primColurs.concat(secondColours);

console.log(allColours);  // Output: red, yellow, blue, orange, green, violet
```

Some of many useful methods available to manipulate JavaScript arrays are listed below:

join() – Combines all array elements into a string.

toString() – Converts an array to a string data type separated by a comma.

split() – Divides a string into substrings and returns a new array.

sort() – Sorts an array in ascending or alphabetical order and returns a sorted array..

includes() – Checks if an array contains a certain element.

reduce()- Used to return the sum of all the elements in an array.

Q11

In JavaScript there are a variety of operations that can be used to add, remove and alter objects.

Adding to Objects: The examples below demonstrate how dot notation or square bracket notation can be used to add new key-value pairs to an object.

```
let obj = {
     name: ‘Nhi’,
     age: 40,
}

// Using dot notation
obj.state = ‘Victoria’;
Console.log(obj).  //Output: { name: ‘Nhi’, age: 40, state: ‘Victoria’}

// Using bracket notation
obj[‘state’] = ‘Victoria’;
console.log(obj);    //Output: { name: ‘Nhi’, age: 40, state: ‘Victoria’}
```

Removing from Objects: The delete keyword is used to remove properties inside an object. Note that if undefined or null . . . .

```
let obj = {
     name: ‘Nhi’,
     age: 40,
     state: ‘Victoria’
}

Delete obj.name;
console.log(obj); //Output: { age: 40, state: ‘Victoria’}
```

Altering Objects: To change some values in an object the dot notation or square bracket notation can be used. This is demonstrated by the examples below:

```
let obj = {
     name: ‘Nhi’,
     age: 40,
     state: ‘Victoria’
}

// Using dot notation
obj.state = ‘Queensland’; // Output: { name: ‘Nhi’, age: 40, state: ‘Queensland’}

// Using bracket notation
obj[‘state’] = ‘Queensland’;
```

```
// Output: { name: ‘Nhi’, age: 40, state: ‘Queensland’}
```

The spread operation can be used to merge two objects and return a new object(Talha A, 4 Nov 2018). If two objects have the same keys, the value of the object merged in last will override the earlier one.

```
let obj = {
     name: ‘Nhi’,
     age: 40,
}

let newObj = {
     ...obj,
     state: ‘Victoria’
}
console.log(newObj);  // Output:  newObj = { name: ‘Nhi’, age: 40, state: ‘Victoria’}
```

The object.assign() is used to combine objects together.

```
let obj = {
     name: ‘Nhi’,
     age: 40,
}

let obj2 = {
     state: ‘Victoria’
}

let combineObj = Object.assign( obj, obj2)
console.log(combineObj);  // Output:  combineObj = { name: ‘Nhi’, age: 40, state: ‘Victoria
```

Q12

JavaScript Object Notation(JSON) is a text-based format data structure based on JavaScript object syntax. JSON data are mainly returned from a web server, external source or loaded from a file in the form of strings. There are more you can do to manipulate objects then strings, so for this reason the JSON.parse() method can be used to convert JSON string into a JavaScript object. Once converted into JavaScript object, the values and properties can then be accessed and manipulated using JavaScript. The example below shows what a JSON file looks like, how JSON.parse() and JSON.stringify() methods are used and ways to manipulate the data(Gu M, 24 May 2018).

```
// JSON data from data.json file
{ "name": "Nhi", "age": 40, "state": "Victoria" }

// JSON.parse()
let obj = JSON.parse(data);
```

```
// Manipulation
obj.name = "Sue"; // change the name value
Delete obj.age; // delete the age value
let mergeObj = Object.assign(obj, obj2) // merge two jsons

// JSON.stringigy() converts JavaScript objects or arrays to strings
let str = JSON.stringify(obj);
```

Data from JSON files can also be extracted, filtered or transformed to be more meaningful and to suit a specific task.

Q13

```
/* Use class keyword to declare an object from ES6. Initialize the properties of
an instance and pass in 'brand' as the parameter */
  class Car {
  constructor(brand) {
    //Create instance variable and assign value matching the paramater(brand)
    this.carname = brand;
  }

  //Declaring a method called present. Specify what to print - 'I have a {brand}'
  present() {
    return 'I have a ' + this.carname;
  }
}

/* Declaring a class called 'Model'.
Using extend keyword to inherit property and methods from the 'Car' class */
class Model extends Car {

  // initialize the properties of an instance and pass in 'brand' as the parameter
  constructor(brand, mod) {

    // Give access to parent class 'Car' parameter 'brand'
    super(brand);

    //Create instance variable and assign value matching the paramater(mod)
    this.model = mod; //Create instance variable and assign value matching the paramater(moc
}
  }
  show() { // Declaring a method called show

    /* Specify what to print - 'I have a {brand}' inherited from the Car class present()
    and concatenate ', it was made in {model}' */
    return this.present() + ', it was made in ' + this.model;
  }
}

let makes = ["Ford", "Holden", "Toyota"]  // Declare a variable called 'Make' and giving 3 v

/* Declare a variable called 'model' to generate a sequance of years using the Array.from me
(x) and it's index(i), 3rd using arrow function to add the range derived from index of each
let models = Array.from(new Array(40), (x,i) => i + 1980)

// Declare function called radomIntFromInverval passing in 2 parameters min and max
function randomIntFromInterval(min,max) { // Declare function called radomIntFromInverval pa

    /* Math.radom function returns random integers between max and min value inclusive of mi
    Result of Math.random function is passed into Math.floor which then truncates decimal nu
    return Math.floor(Math.random()*(max-min+1)+min);
}

// Looping through each element of the models object
for (model of models) {

  // Declare 'make' varable to equal range from randIntFromInteval starting from 0 & end at
  make = makes[randomIntFromInterval(0,makes.length-1)]

  // Declare model varable and pass in same values as the make variable
  model = models[randomIntFromInterval(0,makes.length-1)]

  // Create new instance of Model using it's property and methods and pass in make and model
  mycar = new Model(make, model);

  // Printing sentence from 'show' method inserting make and model value derived from random
  console.log(mycar.show())
}
```

I believe that the radomIntFromInterval function only require the length of the array to be passed in to the Math.random(). It is currently repeating the same year between 1980 to 1982 multiple times as the randomise number added into 1980 is an integar before being trancated by the Math.floor(). I feel the range is already derived from the ‘model’ variable above. This can make the code more dry and efficient.

Reference Source

Atlassian Bitbucket, viewed 29 July 2022, What is version control? https://www.atlassian.com/git/tutorials/what-is-version-control

Atlassian Bitbucket, viewed 29 July 2022, Source control management, < https://www.atlassian.com/git/tutorials/source-code-management>

AWS, viewed 30 July 2022, What is source control? https://aws.amazon.com/devops/source-control/#:~:text=Source%20control%20(or%20version%20control,merging%20contributions%20from%20multiple%20sources.

Cameron R, 7 January 2013, Software Quality Characteristics, < https://www2.cs.sfu.ca/~cameron/Teaching/473/quality_characteristics.html >

Cleary R, 18 July 2020, Control Flow in JavaScript, < https://medium.com/@rianna.cleary/control-flow-in-javascript-9c63d0c98bb9#:~:text=Control%20flow%20in%20JavaScript%20is,loops%2C%20conditionals%
>

Coursera, 19 July 2022, What Does a Back-End Developer Do? https://www.coursera.org/articles/back-end-developer

Etherinton M, 2020, JS Functions, Academy of Information Technology, https://ait.instructure.com/courses/3971/pages/js-functions?module_item_id=336499

FAM, 5 may 2021, How to manipulate arrays in JS like a pro, https://levelup.gitconnected.com/how-to-manipulate-arrays-in-js-like-a-pro-b8f6f5cff5ac

Free Code Camp, 15 February 2020, JavaScript Loops Explained: For Loop, While Loop, Do...while Loop, and More, < https://www.freecodecamp.org/news/javascript-loops-explained-for-loop-for/>

Glorify, 1 March 2021, How to develop a website: A Details guide to website design and development for small businesses in 2021, < https://www.glorify.com/learn/website-development-for-small-business >

Gu M, 24 May 2018, JSON Manipulation with JavaScript, < https://mossgreen.github.io/JSON-manipulation-with-javascript/>

Gutte N, 30 May 2021, Type Conversion and Coercion – JavaScript concepts explained in-depth, < https://medium.com/theleanprogrammer/type-conversion-and-coercion-8974afe03b85>

Javascript.Info, 10 July 2022, JavaScript Fundamentals – Data types, < https://javascript.info/types>

Maldonado L, 19 July 2021, Type coercion in JavaScript, https://blog.logrocket.com/type-coercion-in-javascript/

Mdn web docs, viewed 18 August 2022, Working with Objects, < https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects >

MongoDB, viewed 6 August 2022, MERN Stack Explained, https://www.mongodb.com/mern-stack

Reinagel S, 15 Nov 2016, The Seven Aspects of Software Quality, https://www.silasreinagel.com/blog/2016/11/15/the-seven-aspects-of-software-quality/

Sabhastian N, 22 January 2021, Learning JavaScript type coercion, < https://sebhastian.com/javascript-type-coercion/>

Software Testing Help, 15 July 2022, 15 Best Version Control Software (Source Code Management Tools, < https://www.softwaretestinghelp.com/version-control-software/>

Talha A, 8 November 2018, JavaScript Object Manipulation, < https://medium.com/infancyit/javascript-object-manipulation-5d1145cf06ef>

Walker J, 9 March 2021, How to Manipulate JavaScript Arrays, < https://www.howtogeek.com/devops/how-to-manipulate-javascript-arrays/ >

Wikipedia, 18 November 2021, ISO/IEC 9126, < https://en.wikipedia.org/wiki/ISO/IEC_9126#Developments >

Yeshwanthini S, 21 March 2021, Illustration about MERN tack, https://medium .com/techiepedia/what-exactly-a-mern-stack-is-60c304bffbe4