T3A1 Workbook

Damon Johnson

Question One

Large projects involve developers working on a large number of files that make up a codebase. As there is a lot of work that is being completed simultaneously from many developers it is essential to ensure to implement an effective version control system to track changes and allow access to timestamped backups of the codebase.

One such version control system, that is commonly adopted for smaller and larger projects alike is git. Git is an opensource distributed version control system, which means the entire codebase and version history is available to every developer who has been permission to the git repository.

The git repository contains the project source code and allows developers to branch or merge code from their personal computer to the git repository where it once again can be accessed by all developers for the project. Several companies have built cloud-based git repository hosting services that can be used by other developers. T

he most widely used service is provided by Github. Aswell as providing the infrastructure for hosting the the project codebase, github also provides a user friendly interface through its web application with various features to help developers manage their projects.

Developers can easily interact with the project repository from their command line interface by running various commands that make up the git version control process. This process has been broken down below.

Any git project must start with a developer commiting and pushing their local code to a git repository. A git commit is a command that means the developer has effectively created a timestamped ‘save’ of their local code. Once the developer enacts a push command, they are effectively uploading their local commit(s) to a repository.

This process might be enough for a solo developer but for large projects with multiple developers accessing a repository they must be able to download code from the repository to edit on their personal computer. For this to managed, a project team must decide whether they are going to use a centralized version control or distributed version control system.

A centralized version control system consists of single repository that is stored on the server (hosted by github). When a developer wants to access the codebase to start working on the project they must update their working copy to the latest version from the repository. Once they have completed the work required on their local copy, they will commit their changes.

For this example project, a distributed version control system is recommended. While both systems have their own advantages and disadvantages, a distributed version control system is arguably more suited for larger projects.

In a distributed version control system each developer who is working on the project has their own working copy of the codebase on their local device which they can freely edit as well as their own repository that is separate from the main or root project repository. To start working on the codebase, first a developer must

execute a pull request from the main repository, which in turn creates their own repository. This is referred to as branching.

The developer will then update their working copy from their own repository branch. As they make changes to the code they will commit the changes to their repostory branch. Once all of the changes are complete, and all quality control measures have been accounted for, the code from the branch repository is ready to be merged with the main project repository. To do this a developer executes a push command. Once pushed to the main repository, the code is stored on the server and is accessible by all members for the project team.

Question Two

A report by David Chappell and sponsored by Microsoft Corporation titled “The Three Aspects of Software Quality” (Chappell, 2011), defines software quality from the perspective of both stakeholders and the development team. With these perserctives in mind, software quality is broken down in to three key areas: functional quality, structural quality and process quality.

Functional Quality

Functional quality means the software that has been developed is is able to perform the tasks required by the project scope. It must satisfy the requirements of its users. The project scope is derived from the specified requirements which in turn can developed from the project’s sponsors or users. Often the functional requirements for software can change based on the needs of the user, the applicable laws and regulations and the technological context for which the software is being used.

The software must have few defects or bugs that compromise the software’s reliability, security or functionality. While achieving zero bugs is almost realistically unachievable for most projects, a project that is seen as too buggy for uses cannot be considered functional.

The software’s performance must be adequate from the user’s perspective. The expectation of software performance is constantly rising amongst users, and for this reason developers must contintually upgrade software to ensure its performance is adequate.

The software should be relatively easy to learn and easy to use. The degree of usability that is required from a software is dependent on the target users and the context of the software. For example, a professional audio engineering application should be designed to be used easily by professional audio engineers and technicians. By designing the software to be easily used for the average consumer, the overall software quality could be compromised by a recution in features in an attempt to reduce complexity for the lowest common denominator.

The usability of a software is largely determined by the user interface and experience. It is for this reason that software aesthetics should not be disregarded when discussing the requirements for quality software.

Chappell explains that software testing is focused on functional quality. Therefore, rigorously testing software and not releasing untested software is a highly necessary step in maintaining functional quality.

Structural Quality

Structural quality refers to the structure of the code itself. Structural quality can be difficult to test for and should be considered as a guiding principal when developing software. High quality code structure can be measured by the following qualities:

- Testability: Code should be structured in a modular way such that each block of code can be tested separately. If a code can only be tested in large chunks, errors that arise from testing are more difficult to highlight.
- Maintainability: Again, code should be structured modularly so that blocks of code can be easily removed or edited without causing unecessary and unwanted effects to other areas of the software. Code should follow the DRY principal to ensure there are minimal repetititions within the codebase. Not only does this reduce the amount of overall code required to run software, it reduces the number of changes to the code that are required when the changes are inevitably required to maintain the software.
- Understandability: Martin Fowler, Chief Scientist of thoughtworks who has authored various software design publications wrote that “any fool can write code that a computer can understand. Good programmers write code that humans can understand.” Code should be written as simply as possible, with variable and method names that are self commenting. Where required comments should be made to assist other developers in understanding one and other’s code.
- Efficiency: Code should be efficient whilst maintaining a level of pragmatism. Algorithms should be implemented with the least amount of complexity as possible without compromising the understability/maintability of the code and without mismanaging resources.
- Security: The code should be written in a way to protect the software from common attacks such as SQL injection, buffer overuns and cross-site scripting.

Process Quality

Process quality refers to the management of the project during its entire lifecycle. Like the aboce aspects, it is important that process quality must considers the interests of users, the development team and project sponsors. The key areas that can be used to measure process quality are delivery dates and budgets.

Tools for Improving Software Quality

As mentioned previously the main tool for ensuring functional quality is testing. Testing should be done during development in the form of unit testing or test driven development aswell as for the software once it is in the hands of a user. Additional testing should be done to performance test the software under extreme conditions or with extreme usecases.

To improve structural quality developers should consider the aforementioned principals of structural quality at all times during development. When working on a difficult problem, this can be a challenge. Under these circumstances it may be appropriate for developers to focus on developing a working solution first in the form of a minimum viable product (MVP) before refactoring their code to ensure the code meets the structural requirements of the project.

Technical debt refers to the build up of code that does not meet the structural quality requirements of the project. The technical debt of the project should be tracked at all times. The project team should be managed such that there are scheduled oppportunities areas of the codebase that have been identified as technical debt.

Improving process quality is more of a management issue rather than a technical one. Process quality can be improved through well thought out leadership and management and most importantly resourcing. Management should ensure that there are enough developers with the appropriate technical capabilities to work on a problem within reasonable time contraints. Project management systems such as agile development are commonly used to address project quality. Each development team needs to work within a framework that bests suits their team and the project requirements.

Question Three

MERN is technology stack that provides the architecture to construct a full-stack web application using javascript and JSON code and a NoSQL database. The MERN acronym refers to the four technologies that comprise the stack.

- MongoDB - database
- Express.js - Node.js web framework
- React.js - Client-side framework
- Node.js Javascript web server

MongoDB is an open-source document-oriented database. Its role in the MERN stack is to provide storage for any persistent data used by the application. MongoDB supports JSON document storage (as BSON, similar to JSON) which are often created from React.js client-side rendering. MongoDB is classified as a NoSQL database meaing it doesn't use tables to store data. Instead MondoDB covnerts JSON to BSON documents that are structuredlike JSON with optional schemas to store and access data. MongoDB is well suited to modern applications with large quantities of data as it can be scaled easily by adding servers while maintaining efficiency due to its flexible document model.

Express is a web framework that works on top of the Node web server. It provides additional web development features and simplifies APIs. Express provides the process routing and HTTP requests as well as functionality for dynamic HTTP object rendering.

React is the framework responsible for client-side rendering and creating a dynamic and responsive user interface. React employs the document object model to break down the entire front end interface in to components of javascript code. These components can connect to the backend of the application to ensure to provide users with a data driven experience where the majority of processing is managed by the client rather than the server.

Node is an open-source runtime environment for executing javascript code outside of a browser. It is used to build back-end services to interact with the databse of the application.

Question Four

To develop a website for a small business, the development team would need to be well rounded in technical web development skills and general professional skills. These skills have been further broken down and described under the following subheadings.

Front-end Development

The development team would need to be well versed in at least one front end development framework. Virtually all dynamic websites rely on javascript to render client-data. It is recommended that the

development team employs a javascript based front end development framework such as react, angular or vue to streamline the front-end development for the website.

Back-end Development

The development team would need to well versed in a programming language that could be used for back-end application development. Depending on the language that is selected, the development team would require knowledge of any associated development frameworks. For instance, if the development team were to code the backend using python, it is recommended they implement a django framework.

For a basic website, the MERN/PERN stack would be a good starting point. It is recommended that development team uses express.js for the backend framework to handle routing and HTTP requests. Knowledge of web theory, and an understanding of routing and HTTP requests is also essential for the development of a business website. It is recommended that the development team implements node.js as development environment for the application.

Databases

The development team would then need to determine what database management system best suits the needs of the stakeholders. The primary decision to be made would be whether they should employ an SQL or NoSQL database. For this reason it is recommended that the development team has knowledge of both database structures as well as their advantages and disadvantages.

The development would also need to have a strong understanding of database design and entity relationship modelling and schemas. As well as how to actually create a database using a commercial database management software such as Postgresql or MongoDB.

Version Control

The development team will need ensure all members have an adequate understanding of a version control system such as a git. They will need to determine whether to use a centralized or decentralized version control system and adhere to their version control system throughout development and production.

Search Engine Optimisation

In order to encourage traffic to the business' website, the development team will need to understand and employ search engine optimisation techniques. Search engine optimisation for websites is determined by the crawlability and idexability of the website, content quality, keyword usage, HTML structure and semantics and backlinks.

Hosting

The type of hosting for the website will be determined by the expected traffic and the content of the website itself. For a simple business website with minimal traffic a conventional and cost effective web hosting provider could be implemented. For a more complex web application or a website with large amounts of traffic, the developers should have knowledge of a cloud based application platform such as Azure, Amazon AWS or heroku.

User Experience and Design

The development team should have knowledge of how to design websites with an enjoyable, effective and responsive user experience and design. Elements that make up user experience and desgin include aesthetic design, animations, graphic design, typography, sitemap, forms and front end content.

Quality Control

Throughout development and during production of website the development team must understand the requirements of project quality and hold the skills to maintain it. Project quality can be controlled by understanding the requirements of the scope, applying strong project management principals, producing reliable software through test driven development, writing maintainable, efficient, modular and DRY code and keeping technical debt to a minimum.

Project Management

The development team should have strong and effective leadership that provides sufficient project management. Project management can be structured through various methodologies but it is recommended that the development team has knowledge of and applies an agile project management methodology.

Stakeholder Communication

The development team should regularly communicate with all stakeholders. The website design should be created in direct correspondence with the business representatives, and any changes should be approved by the business representatives. Keeping a positive and professional relationship with all stakeholers is one of the most important aspects of the project. The stakeholders need to understand that their opinions are being heard and responded to. Any advice that is given to stakeholders should be delivered professionally and respectfully.

Question Five

For question five I will be discussing the two sided marketplace application that I developed using the Ruby on Rails framework with a postgresql database. In order to develop this project I required knowledge and skills in the following areas.

Ruby Programming

Ruby is the underlying programming language language for the ruby on rails framework and as a developer I needed to understand basic ruby programming and syntax. Along with basic ruby programming, various ruby gems were implemented to provide extra functionality such as devise, pundit, rolify, simple-form and ransack.

Ruby on Rails and MVC

Ruby on Rails uses the model-view-controller (MVC) framework to develop web applications. As a developer I need to understand how to use the controller as the interface between the model and the view.

Routing and HTTP requests

I needed to understand how HTTP requests from different URLs can be routed to execute controller actions and render different front end content.

HTML and CSS (Boostrap)

HTML and CSS implemented with bootstrap made up the front-end of the application. No javascript was required for client side rendering.

User Experience and Design

The site was designed to provide an an aesthetic and intuitive user experience that encourage users to sign up to and actually use the site. Wireframes were developed for mobile, tablet and desktop before being implemented with HTML, CSS and Ruby code.

Database and Hosting

A database was designed using and entity relationship diagram and eventually created through model generation with rails framework and postgresql. The website was then deployed via Heroku through their cloud services infrastructure. While only fundamental level knowedge was required for postgresql and heroku integration, troubleshooting skills were required at some stages of development.

Question Six

Ruby Programming

Overall my knowledge of ruby programming and syntax did not inhibit my ability to produce a functional marketplace application. There were definitely areas of my code that were not efficient or were not written in accordance with best practice software development principles. During the process I did refactor the code to reduce the level of repetitive code in an effort to conform with DRY software principles. Due to the time constraints of the project I was unable to refactor the code to a professional standard.

Exception handling and error handling was by and large well managed across the site with the exception of bad dates causing crashes to the application. This should have been identified with testing prior to production.

Ruby gems were implemented correctly and assisted the application in handling user authentication, authorisation and item searching.

Ruby on Rails and MVC

The marketplace application was the first complete application that I had developed using the Ruby on Rails framework. With this in mind I believe the end product was of a reasonable standard. I found that while the overall project was still delivered, the project took longer than expected due to gaps of knowledge with the rails framework. I was able to adhere to the MVC architecture ensure there was no data transferred from the model directly to the views of the application.

One area that could definitely have been improved would be to reduce the number of calls to model. This is an area that should have been addressed during refactoring however I was unable to achieve this with the time constraints.

Routing and HTTP requests

Across the sites routing was generally well managed and the routes file was well structured and maintainable. Site URLs were concise and meaningful and the site map was intuitive. Overall the end product, was not hindered by knowledge of routing and HTTP requests.

A better understanding of HTTP requests from the beginning of the project may have enabled me to be more efficient with application testing using postman during development. If this process was more efficient I may have been able to use my time more effectively to fix bugs and refactor the code.

HTML and CSS (Boostrap)

The HTML was generally semantic and well structured. Ruby code was embedded in to HTML fles without issues and created a functional application. Bootstrap was applied for CSS styling and made the styling process more efficient than I would have been if I implemented pure CSS or SASS.

User Experience and Design

While the user experience and design was not the primary focus for the application, the site was reasonably aesthetic and intuitive to use. By putting effort in to the front-end design of the application, I felt more compelled as a developer to work on the project.

I also found that by applying attention to detail to the front-end helped to create a high standard for quality that was carried through the rest of the application.

Database

The database design was the area that caused me the most difficulties during development. During development there were four iterations of the entity relationship diagram and database schema. This caused signficant delays to the project and resulted in a reduction of features at delivery as well as an overall reduction in code quality. In future projects as a developer, I aim to be more critical of my database schema in the planning phase to ensure it will be functional prior to writing any code.

Deployment and Hosting

Approximately half a day was spent troubleshooting issues during deployment of the application to Heroku. In future projects I would have deployed the project earlier and more often through development to minimise the chance of errors and setbacks. No issues were encountered when setting up the database on postgresql.

Question Seven

Control flow is the order of how a computer executes code for a given file. As javascript is an interpreted programming language, control flow determines the order in which code is interpreted prior to execution. By default javascript will be interpreted and executed line by line from, from the top to the bottom of a file. In

order to change the default control flow of a program, programmers must use one of the following control flow statements: loops, conditions or functions. These control flow statements have been broken down further under the following subheadings.

Loops

Loops are iterative statements that are specified only once but will continue to run until there are no elements within a given set to loop over. If there are no restrictions on the loops programmers can (generally by accident) greate an infinite loop that will continue to run until the program is interrupted generally by a force stop or quit. There are two primary ways of implementing loops in javascript.

The for loop, takes three arguments: intializer, condition and incrementer.

- `i` is the variable declaration of the the index for the array that we are iterating over.
- The initializer represents the initial value of `i`.
- The condition sets the constraints for the for loop. The loop will continue to run until the condition is true.
- The incrementer changes the value of `i` to track how many iterations of the loop have been completed. Often the incrementer is simply set to add 1 to the value of `i` after each iteration to simply count the number of iterations however more complex expressions can also be used.

For Loop Example:

```
const genres = ['jazz', 'rock', 'country', 'classical']

for (let i = 0; i < genres.length; i ++) {
    console.log(`Let's listen to some ${genres[i]}`)
}

// Let's listen to some jazz
// Let's listen to some rock
// Let's listen to some country
// Let's listen to some classical
```

The other main method of implementing loops in javascript is the for of loop used to iterate through arrays. The for of loop iterates through an array with reference to each element of the array.

For of Loop Example:

```
const letters = ['a', 'b', 'c', 'd']
for (letter of letters) {
    console.log(letter)
}

// a
// b
```

```
// c
// d
```

While Loop

Similar to a for loop, a while loop continues to run as long as a specified condition is true. The main of differnece between a while loop and a for loop is that a while loop is suited to a scenario where we do not know how many times a loop needs to be executed.

While Loop Example:

```
let sunny = true

while (sunny === true) {
    console.log("don't forget a hat")
    checkWeather()
}
```

Do-while Loop

The do while loop serves the same purpose as a while loop but has been restructured as follows.

```
let sunny = true

do {
console.log("don't forget a hat")
checkWeather()
} while (sunny === true)
```

Conditionals

Conditionals are `if, else, else if and switch` statements that are used to check a condition and run a command based on the result of the checked condition. The if statement will run if the condition is found to be true. If the condition is found to be false, and else if statement acts as additional if statements. If else statements are optional conditional statements. Finally, the else statement acts as a catch all. If all condition checks were found to be false, the else statement will be executed. If, else if and else have been combined in the following code example. Note, multiple else if statements can be implemented one after the other. They are checked sequentially from top to bottom. Once a condition is found to be true, the remaining checks are not considered.

Conditional Statements Example

```
const card = 17

if (card === 21) {
    console.log("blackjack!")
} else if (card === 20) {
    console.log("I'll split")
{ else if (card > 16) {
    console.log("I'll stay")
} else {
    console.log("Hit me")
}

// I'll stay**If, Else if, Else**
```

Switch

Switch statements are similar to if statements in that they check the value of a condition. The switch statement does this by taking in an expression and evaluating it, and then checking if the result of the expression matches any of the cases that have been specified as a part of the switch statement. Breaks are another control flow feature of javascript. The switch statement will continue to check all cases even if a match is found. To counter this, a break is added to end of each block following a case statement. The break exits outs of the switch statement and ceases to check the remaining cases. The default acts as a catch all similar to and else statement. If no match is found within the cases the default block will be executed.

Switch Statement Example:

```
const food = "soup"

switch (food) {
  case "pizza":
  case "sandwich":
    console.log("Use your hands")
    break
  case "sushi":
    console.log("Use chopsticks")
    break
  case "pasta":
    console.log("Use a fork")
    break
  case "soup":
    console.log("Use a spoon")
    break
    default:
      console.log("Have you tried a spork?")
}
```

Functions

Finally, functions are blocks of code that can be called from other parts of the program. There are many ways that we can declare functions which can also influence whether or not a function is hoisted. A hoisted function can be declared after it is called without causing an error. This is typically an unwanted issue. The following example shows two ways to declare a function. The first will allow hoisting while the second, known as a fat arrow function will not allow the function to be hoisted. Below the function declaration is an example of how a function can be called with parameters passed in to the function as arguments.

```
// Typical function declaration (hoisting enabled)
function adder(x,y) {
    return x + y
}

// Fat arrow function declaration (hoisting disabled)
function adder = (x,y) => x + y

// Calling a function
console.log(adder(10,5))
// 15
```

Question Eight

Type coercion is the conversion of values between data types. By definition data types are immutable. So type coercion is not a change of the data type itself, but a change of the data type of the variable. This is best explained by example. All examples for this response will be shown in javascript. In below code snippet, a variable is declared as a string. The variable is then converted to a number through type coersion.

```
let number = '12'
console.log(typeof number)
// string

let number = 12
console.log(typeof number)
// number
```

Type coercion can be carried out explictly like the above example or automatically. In the below code snippet automatic type coercion is shown. Automatic type coercion can be carried out by a programming language such as a javscript during a comparison of two variables.

```
let numberA = 12
let numberB = '12'

if (numberA == numberB) {
    console.log('type coercion has occured')
} else {
```

```
    console.log('type coercion has not occurred')
}

// type coercion has occured
```

The code can be adjusted using `===` in the comparison to remove automatic type coercion. This is generally good practice when comparing values in javascript.

```
let numberA = 12
let numberB = '12'

if (numberA === numberB) {
    console.log('type coercion has occured')
} else {
    console.log('type coercion has not occurred')
}

// type coercion has not occured
```

The following code snippets a how explicity type coercion methods can be applied to convert from string to numbers and vice versa.

```
// Converting from string to number

let value1 = "5"
console.log(typeof value1)
// string

value1 = Number(value1)
console.log(typeof value1)
// number
```

Question Nine

All programming languages use built in data types. Differentiating between datatypes offers a necessary level of complexity required to make programming efficient. By classifying data in to a distinct type, it allow the computer to make assumptions about the data that assist in manipulation, processing and abstraction.

The way in which data types are handled differs across programming languages but this response will discuss the data types that are built in to the javascript programming language. First we must understand that javascript is a dynamically typed language. Meaning any variable can be assigned and reassigned values of all types.

```
let value = true // value is a boolean
value = 'pizza // value is now a string
value = 22 // value is now a number
```

```
// The data type of var has changed with each line
// and these are all valid declarations in javascript.
```

This response will only discuss primitive data types. Primitive data in javscript is data that is not an object and has no methods or properties. Because of this the data type is immutable and cannot be altered. The value of the variable can be altered but the data type itself cannot be altered. For instance, arrays are not considered a primitive data type in javascript. They are instead considered container objects which are dynamically created. The seven primitive datatypes in javascript have been described below.

Booleans

Booleans represent a logical datatype that can have one of two values: true or false. For this reason booleans are commonly used in control flow.

```
let rain = true
if (rain === true) {
    console.log('Pack umbrella')
}
```

Null

Null represents a lack of value, or a nonexistent or invalid object or adress.

```
let  number = null
```

Undefined

Undefined refers to a variable that has not been assigned to datatype. This does not meed the variable itself has not been declared.

```
let number
console.log(number)
// undefined
```

Numeric - Number

Numbers are the first of the numeric primitives data types in javascript. Number is capable of storing a range of floating point numbers and itnegers within a set range. Numbers outside of the range are classified as either +Infinity or -Infinity.

```
let number = 22
let number = 0.2412497832423

// both of these are valid numbers
```

Numeric - BigInt

The BigInt data type is the second of the numeric primitive data types in javscript. It is used to represent integers with arbitrary precision beyond the integer limits used for the number data type. That is to say, numbers larger than 2^53 - 1 which is represented by the constant: MAX_SAFE_INTEGER. Standard mathematical operations like + - * ** % and / can be used on BigInts however they cannot be used in the same mathematical expression as number data types.

```
let bigNumber = BigInt("112347037512280721484647909315925481 2")
```

String

The string data type is used to represent text data. Like an array, each element of the string holds a unique position or index within the string.

```
let number = '22'
let food = 'apple'
```

Symbol

Symbols are immutable (unchangeable) values that can be used as the property of key of a javascript object.

```
let mySymbol = Symbol("mySymbol")
```

Q10) Talk about at least 2 ways

Arrays are an ordered list of elements. The element is assigned a value that can hold a variety of datatypes including strings, numbers and booleans. Arrays can also stored multiple data types and repeating values in a single array. Arrays can also be nested in other arrays or objects.

The order of an array is determined by the index value of each element. The index value of the first element is 0 and the index value of each element is sequentially incremented from here.

There a multiple ways that arrays can be created in javascript. The simplest method has been included in the example below.

```
// Creates an array called numbers ranging from 1 to 5.
let numbers = [1, 2, 3, 4, 5]

// Creates an array of letters (strings) ranging from a to e.
let letters = ['a', 'b', 'c', 'd', 'e']
```

A key part of manipulating arrays is to use the index numbers of the elements. The following example shows how we can access elements of array.

```
letters[0]
// a

letters[3]
// c
```

Methods

Array methods are built in actions that can be applied to manipulate arrays. Methods can accept parameters and functions as arguments to add complexity to the manipulation of arrays. The following array methods are some of the more common ways of manipulating arrays in javascript.

Slice

The slice method extracts a a shallow copy of a subset of elements from an array. It accepts the index of the first value to be extracted as well as the index of element at the end of the subset. The end value is not extracted. It does not modify the original array.

```
letters.slice(1,3)
//['b','c']
```

Pop

The pop method removes and returns the last element of an array. The pop method does alter the original array.

```
letters.pop()
// e

//letters after pop
[ 'a', 'b', 'c', 'd' ]
```

Push

The push method adds additional elements (taken as argument(s)) to the end of an array. It also returns the length of the new array with the additional element(s).

```
const moreLetters = letters.push('f', 'g')
// returns 7

letters
// returns ['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

Shift

The shift method removes the first element of an array and returns the element. The shift method does modify the original array by removing the element.

```
const firstLetter = letters.shift()

return firstLetter
// a

return letters
// ['b', 'c', 'd', 'e']
```

Find

The find method returns the first element of an array that complies with the requirements of a checking function passed in to the find method as an argument.

```
const firstEven = numbers.find(x => x % 2 === 0)
console.log(firstEven)
// 2
```

Includes

The includes method searches an array for a value supplied to the method as an argument. If the value is found within the array, the method returns true, otherwise the method returns false. Even

```
console.log(letters.includes('a'))
// true

console.log(letters.includes('x'))
// false
```

Map

The map method iterates through an array and applies a function to each array element. The results of the function are then used to create a new array.

```
const doubles = (numbers.map(x => x * 2))
console.log(doubles)
// [ 2, 4, 6, 8, 10 ]
```

forEach

Similar to the map method, the for each method accepts a function as an argument and applies the method to each array element.

```
numbers.forEach(number => console.log(number))
// [1, 2, 3, 4, 5]
```

Filter

The filter method combines the effect of the find method and the map method. The filter method takes a test function as an argument and creates a new array of all elements of satisfy the test function.

```
let evenNumbers = numbers.filter(x => x % 2 === 0)
console.log(evenNumbers)
// [2, 4]
```

Reduce

The reduce method accepts a callback function as argument and applies the function to each element of the array in order. Each time the function is applied to the array it takes in the result of the previous function iteration as well as the next element value as parameters. The simple application of the reduce function is to sum numbers as shown in the below example. The callback function can take an optional third parameter representing the index of the current element.

```
let numbers = [1, 2, 3, 4, 5]
console.log(numbers.reduce((previousValue, currentValue, currentIndex) =>
previousValue + currentValue))
// 15
```

Question Eleven

Objects are a javascript datatype that are used to form hash tables of key-value pairs. In javascript the key is also referred to as a property. Almost all objects are an instance of, and thereby inherit properties from the object prototype however properties can be overridden.

Objects can be used to describe 'real world' objects. The understand how to manipulate an object, first we need to understand how an object is structured. In the example below, the object is declared and consists of 3 properties with their respective values separated by commas and contained in braces.

```
let table = {legs: 4, material: 'timber', colour: 'brown'}
```

To access an object's value we call the objects property. This can be done through dot notation or square bracket notation.

```
console.log(table.legs)
// 4

console.log(table['material'])
// timber
```

The same notation can be used to change the value of an object property.

```
table.legs = 6
console.log(table.legs)
// 6

table['material'] = 'plastic'
console.log(table['material'])
// plastic
```

In javascript we can destructure objects to extract the properties from an object in to separate variables. To do this we need to

```
const table = {legs: 4, material: 'timber', colour: 'brown'}

// Normal variable assignment
const legs = table.legs
// 4

const material = table.material
// timber

const colour = table.colour
// brown
```

```
// This is the same as this shorthand method of destructuring:
const { legs, material, colour } = table
```

Javascript provides numerous built in methods to iterate through objects. To obtain a set of all keys for an object we can use the Object.keys method.

```
console.log(Object.keys(table))
// [ 'legs', 'material', 'colour' ]
```

To obtain a set of all values for an object we can use the Object.values method.

```
console.log(Object.values(table))
// [ 4, 'timber', 'brown' ]
```

To obtain a set of all key-value pairs in a multi-dimensional array we can use the Object.entries method.

```
console.log(Object.entries(table))
// [ [ 'legs', 4 ], [ 'material', 'timber' ], [ 'colour', 'brown' ] ]
```

To iterate through an object's enumerable properties we can implement a for-in loop.

```
for (property in table) {
    console.log(`My table has ${table[property]} ${property}`)
}

// My table has 4 legs
// My table has timber material
// My table has brown colour
```

We can also use the freeze method to prevent any modification of an object's properties. It also prevents new properties being added to an object. Obce an object has been frozen there is no built in method to unfreeze the object. A work around could be achieved by cloning the existing object for mutation.

```
Object.freeze(table)

table.material = 'plastic'

console.log(table.material)
// timber
```

Similar to the freeze method, the seal method can be applied to prevent new properties from being added to an object. However the point of difference is that the seal method allows existing properties to be changed.

Question Twelve

JSON (Javascript Object Notation) is a standardised format for text data that is based on the Javascript language syntax for objects. When handled using javascript, JSON is interpreted as an object. However, during transactions of JSON files, it is interpretted as a string.

The following methods have been built in to javascript to allow for the conversion of a javascript object to a string and vice versa.

```
// Converting from JSON to string
JSON.stringify(jsonData)

// Converting from
JSON.parse(stringData)
```

To obtain JSON keys values and entries we can use the following methods.

```
shoppingList.keys()
shoppingList.values()
```

To obtain an array of JSON keys and their respective values we can use the entries method.

```
shoppingList.entries()
```

JSON can be iterated through like any other javascript object using javascripts built in methods for objects. To iterate through JSON we can use a for loop.

```
for (item in shoppingList) {
    console.log(item)
}
```

Question Thirteen

```


// Declaration of Car class.
// Class takes one argument (brand) which is used to declare the property
carname.
// Constructor function is executed when new car object is created.
  constructor(brand) {
```

```
// Sets carname property to the brand argument that is passed in when
creating a new Car object.
    this.carname = brand;
  }

// Class function present() returns the carname property.
  present() {
    return 'I have a ' + this.carname;
  }
}

// Declaration of Model class to inherit from Car Class.
class Model extends Car {
// Constructor class takes two arguments and is executed when new model
object is created.
  constructor(brand, mod) {
// Super method refers to the parent class (Car).
// Brand property inherits from parent Car class.
    super(brand);
// Sets model property to the mod argument that is passed in when creating
a new Model object.
    this.model = mod;
  }

// Class function show() calls the present() function from the parent Car
class
// and adds the string containing the model property to
// output a description of the car's name and model.
  show() {
    return this.present() + ', it was made in ' + this.model;
  }
}

// Declaring an array of car makes
let makes = ["Ford", "Holden", "Toyota"]

// Declaring a new array of models using the Array() constructor to create
a new
// array object of integers with a range of 1980 - 2020 representing car
model years.
let models = Array.from(new Array(40), (x,i) => i + 1980)

// Declaring a function randomIntFromInterval that takes a minimum and
maximum
// number as arguments. These numbers represent the minimum and maximum
number
// of a range that the returned interval can be randomly created from.
// Math.floor is used to round down the random number to an integer value.
function randomIntFromInterval(min,max) { // min and max included
    return Math.floor(Math.random()*(max-min+1)+min);
}

// For loop that iterates through each year of models with a range of 1980
to 2020
```

```
for (model of models) {

// Setting the make variable to a randomly selected make from the
previously
// declared make array. The index used to select the array element is
selected
// at random using the randomIntFromnterval function with a range of 0
(1st element)to the
// to the length of the makes array-1 which refers to the last element of
the array.
  make = makes[randomIntFromInterval(0,makes.length-1)]

// Setting the model variable to a randomly selected model from the
previously
// declared model array. The index used to select the array element is
selected
// at random using the randomIntFromnterval function with a range of 0
(1st element) to the
// to the length of the model array-1 which refers to the last element of
the array.
  model = models[randomIntFromInterval(0,makes.length-1)]

// Creates a new Model object called mycar and passes in the randomly
selected
// make and model from the previous lines as arguments. These arguments
are passed
// in the to constructor function for the class to set the make = carname
property
// and the model = model property.
  mycar = new Model(make, model);

// Logs the execution of the show method for the mycar model object.
// Expected output: "I have a <make>, it was made in <model>" where
// <make> could be one of the following models: "Ford, Holden, Toyota"
// <model> could be any year within the range of 1980 to 2020.
  console.log(mycar.show())
}
```

References

Chappell, D., 2011. *THE THREE ASPECTS OF SOFTWARE QUALITY : FUNCTIONAL , STRUCTURAL , AND PROCESS*. [ebook] David Chappell & Associates. Available at: http://www.davidchappell.com/writing/white_papers/The_Three_Aspects_of_Software_Quality_v1.0-Chappell.pdf [Accessed 20 June 2022].