Workbook Assignment LeithPrice_T3A1

Q1

Source Control is the process of managing code versions on a coding project. Source control systems are cloud based and accessible by anyone who has permissions. There are many different types of Source Control, one of the more widly accepted is GIT. GIT is an acronym for " Global Information Tracker". Good source control enables coders and management to track changes to code in a project. Relying on accurate commit descriptions and the users following best practice procedures allows multiple coders to work on single large scale projects without data version conflicts.

The key terms to know regarding source control are Repositorys, Commits, Checkout, Trunks and Branches.

Repositories- is the term to describe the location a version of code is stored. When it is in the cloud, on GitHub for example, it is called the Remote Repository, when it is on your computer, it is referred to as the Local Repository

Commit - is the term for a new version of code. With GitHub the terminal command would look something like ***git commit -m "added a new button"***. A coder would perform a commit when a particular task or stage had been achieved with the source code and they wanted to save this point. The commit message is relevant and needs to be specific to what was performed to the code so as to ensure comprehension from a third party looking at the commit history.

Trunk - is a reference to the main source code. Source control on large projects is often viewed from the POV of a tree with the trunk existing as the main code, and the multiple versions branching off the trunk as the independant work performed by the different people working on the project.

Branch - is the term for describing the the different versions of the main source code being worked on by the project members. There can be multiple versions of the source code at play at any time, and this is tracked by ensuring the work performed is set up in a new branch.

Checkout - is the term for creating a new branch of the main source code. To ensure you are not risking the main, functioning code in the remote repository, saving any changes, improvements or removals to the code is done in a new branch. Git Specific, this terminal command would look like ***git checkout -b adding-a-new-button***.

A step by step of large project source control

- Firstly a project member would download the existing source code from the main repository, or Trunk. Ths is called cloning the repository.
- Before starting the allocated task, the project member would create a new branch on their local repository. This is called Checkout and ensures that whatever actions performed are done so on the new branch and not the main branch.
- Then they would perform whatever task they needed to do, in this example they were adding a feature button.
- Once they had determined that their code was error free and wanted to send it back to the remote repository for consideration, the Project member would perform a Git add, commit (with a meaningful message) and then Push from the local repository to the remote Repository.

- The work performed will be on the Cloud and accessible by other members of the team specifically project management and senior devs in the team. It will be saved in a seperate branch of the main code with the new changes.
- The team members with authority to make the final decisions on any changes to source code would then merge the branch with the main trunk, making the new features apart of the main source code.

Q2

The standards for software quality, as set by the International Organization for Standardization (ISO) and the International Electrotechnical Commission(IEC) in the document ISO/IEC 25010:2011 are as follows:

Functional Suitability - The software product needs to do as it is intended to. It should function as needed with the accuracy level it requires to the completion level that is satisfactory.

Reliability - Software doesnt wear out so reliability needs to be considered from a user experience POV. The System needs to work when required, be designed well enough to be mostly fault resistant and if in the even it fails, needs to be able to recover and continue on.

Operability - The software product needs to be clear in what it does, have sufficient instruction and documentation in its operation. It needs to be learnable and User firendly enought to be tolerant to user mistakes.

Performance efficiency - The quality software package needs to utilize system resources as efficiently as possible, be compatitible for all systems, can be used by multiple people of different skill levels and operate without affecting the performance of other software. Primarily the software needs to be able to operate as well as possible with as little energy and time as possible.

Security - Quality software needs to be able to ensure user information is kept private, that the integrity of the system cant be compromised through external intervention, that sufficient accountability exists internally to ensure the very best efforts are maintained to enforce and update security.

Compatibility - Quality software needs to be able to ensure that it performs the intended function well and completely without affecting other software nor other systems.

Maintainability - Quality software can be updated and and adapted throught patches or other means. That it can be assessed and analysed for future modifications to ensure it stays 100% as intended for its purposes.

Transferability - Quality software can be installed repeatedly with no lowering in function, that it can run on multiple systems and still perform its intended function. That it can be replaced with alternative software without being detrimental to the initial system,

(ISO n.d)

Q3

MERN stack is the term used to describe the technologies used in a specific type application. MERN is an acronym that stands for MongoDB, Express.JS, React.JS and Node.JS. This combination is sufficient to create an entire application using only Javascript and JSON.

Outline a standard high level structure

MERN applications adopt the three tier - type architecture. This is structured in the following way:

- Front end Tier, otherwise known as Web side, is where React.JS is used.
- Application Tier, otherwise known as Server side, is where Express.JS and Node.JS are used.
- Database Tier, is where MongoDB is used.

The front tier is the webside user interface that interact with the user. Front tier then refers user requests to the Application Tier which if required, will refer to the Database tier for entering or retrieving data.

Explain the components

MongoDB: Whilst traditional relational databases are like MySQL, which use relationships between tables for ordering data. MongoDB is a document-orientated NoSQL Database. NoSQL is a term referring to "Non-Relational". MongoDB uses collections of documents which contain "Key-value Pairs" like Hashes to record and order data. Whilst a relational database may have one view of data spread across a range of database tables, in MongoDB, all that connected data is found in a single collection(Table) and is differentiated by something called embedded documents (hash format).

Express.JS: is a Web Application Framework for Node.JS. By using Express.js a lot of things you would have to do from scratch in Node.js are automatically set up. Things such as setting up ports and Route Handling. Express.JS uses the MVC platform for App logic. MVC stands for Model, Views and Controller. Express has a lot of similarities to Rails from Ruby on Rails.

React.JS: is front - end Javascript Library used for building user interfaces(UI) on the client side. React allows you to create reusable UI components that exist in a state called Virtual DOM. This means when data states in that virtual dom change, the browser page does not need to reload to reflect that new state but rather the Real DOM automatically updates to copy the virtual DOMs new state.

Node.JS: is a Javascript runtime environment. Installed on your machine it enables you to create javascript full scale apps on your computer by creating a Node server environement to base it on. Node Package Manager (NPM) was then created to allow the sharing and downloading of JS software that could assist in creating those full JS programs.

(MongoDB, n.d)

Q4

PROJECT MANAGER: Needs to know exactly what the client is hoping to have created through meetings with the client about their objectives. Needs the ability to communicate with the client through out all stages of the project. Needs to be able to plan the project from start to finish, monitor the budget and labour allocated to the task, and ensure project stays on schedule as commited to the client.

PROJECT LEAD: As there are multiple different roles involved in the project, the project lead can also be refered to as the technical lead or the Engineering Manager. Their job is to manage the technical aspects of the project, from planning the initial architecture layout to allocating "Sprints" throught the whole project. Generally the most experienced in the project, a lead wears many hats and needs the skills and knowledge of all the people underneath them like UI designers, Front and Back end developers. They are also the final one responsible for source control management.

UI DESIGNERS: Initial concepts need to be created graphically for approval before serious man hours are put into coding something. This can be done by User interface (UI) designers. They are responsable for layouts and wireframes. Using graphics design software and other programs like Balsamiq or Lucid charts for image creation would be an essential skill.

FRONT END DEVELOPERS: Front end developers are responsible for the coding of the user interface aspect of an App. Knowledge in languages such as HTML, CSS and JS would be essential. Dependant on which Dev stack the project is using, other products like REACT.js and Bootstrap can be used too.

BACK END DEVELOPERS: Back end developers are responsable for the ServerSide of an application. Again dependant on the stack being used, they would have to be well versed in using a server environment like Rails or Node.js, a framework like Express.js or again Rails as its all inclusive. And an understanding of databases and which one to use such as PostgreSQL, or MongoDB. Further more an understanding of a deployment software like Heroku or AWS for deployment of the app in its initial development stages.

Seperate to all this, everyone needs to be skilled in source control systems like GIt HUB for example to ensure best practice when having multiple different designers working on one project.

And everyone needs to have knowledge of a project management methodolgy like Agile Methodology for example, to ensure workflows are manage properly and time targets are met efficiently.

(Yaskevich A, n.d)

Q5

In term 1 of this course, I was give a Portfolio assessment where we had to create a website about ourselves. In order to complete this project I was taught HTML and CSS. Further to CSS I was taught SASS.

HTML stands for Hyper Text Markup Language and is the format used for structuring how information is displayed on a users browser.

CSS stands for Cascading Style Sheets and is the description for how the HTML Information is display on the screen, such as colour, format, size, responsiveness etc.

SASS is an extension of CSS and allows you do perform more dynamic CSS such as utilising variables, importing conditions from one style sheet to another etc.

In order to complete the project, were also taught about how to create wireframes and sitemaps to assist in adopting best practice for creating a website from the ground up.

And further to that were given the chance to work on our customer presentation skills by doing a ten minute presentation to class about our website.

The number one skill i learnt to overcome challenges in this assessment was to "google it". Until my teachers instructed me to do so i never knew how large the online resources are for problems you come across in the software world.

Q6

In reference to the Marketplace app assessment from Term 2, being taught Ruby and everything to do with that allowed me to understand the concept of Object-Orientated Programming.

Understanding the class system of OOP allowed me to understand database relationships when setting up PostgreSQL in the Rails framework.

Being taught about about MVC architecture allowed me to better understand how Rails compartmentalises its different classes into Model and Controller and Views.

Skills taught in the original Portfolio assessment for doing presentations, sitemaps and wireframes allowed me to have a clear idea of what i was doing in this assessment. My understanding of database relationships was further cemented by entity relationship diagrams.

An improvement would be to have a clearer idea with what i wanted to achieve before coding as i had some issues in database column data type changes after DB initialization that wer'nt resolved.

Also an improvement would be to add a lot more time for the HTMl and CSS/SASS to get the final project less stock-standard and more inline with what i would like to achieve visually. Incorporating React.JS or Bootstrap for CSS would assist with this efficiently next time.

Q7

Control flow is the order in which statements are executed in a code. Normal order is from left to right and from top to bottom. However in order to make programs more reactive to differing conditions and user input, Control structures are adopted in order to be able to change the flow of script reading. This is done to allow the code to action instructions correctly and move the control flow on to where it needs to be.

For example, if the user had the option to enter A, B or C, a control structure could then have three different command options to action subject to which was selected by the user.

The following examples of control structures used to Control Flow in computer script are Javascript specific in Syntax:

- If, Else if statements : If you had a variable (age) with a numerical value, say 17 for example, you can set up conditions that assess the variable value and give you range of possible actions to execute dependant on the value of the variable. In this example if variable is over 20 then print statement "your an adult " prints to console. If the variable is under 20 then print statement "your a child" prints to console.

```
    let age = 17;

    if (age >= 20) {
        console.log('You're an adult");
    } else if (age < 20){
        console.log("You're a child");
    }
```

- Switch statements : This control structure uses CASE and BREAK commands. If you are expecting a specific input, you can set up a range of specific case conditions. In a switch control structure, each case is searched till it does or doesnt find the matching variable value. Then that case is executed and the search cycle is ended by the Break command. If you forget to have the Break command in the case statement, then it will continue to action all code after it in other case statements irregardless of the case criteria. You can end a switch statemnent with a DEFAULT command to cover the unknown.

```
    let animal = 'moose';

    switch (animal){
        case 'cow':
            console.log('Moo!');
            break;
        case 'moose':
            console.log('Groan!');
            break;
        default:
            console.log('No sound for $(animal).');

    }
```

- Loop statements : Do While and While Loop statements are a control structures that will continue to repeat until a condition is met. The only difference between the two is that the Do While loop assess the condition for repeating after the code inside the loop has been actioned whilst While loops assess the conditions at the start of loop.

  FOR Loop control structures are a drier loop syntax-wise because the conditions for looping, stopping and updating the condition variable are all written inside the single loop condition of the FOR.

  FOR IN and FOR OF loops are loops structures used for executing code on each element in an object and in an array respectively.

```
WHILE LOOP
    let number = 0;

    while (number < 4) {
     console.log(number);
     number++;
    }
```

```
DO WHILE LOOP
    let number = 0;

    do {
        console.log(number);
        number++;
    } while (number < 4);
```

```
FOR LOOP
    for (let i = 0; i < 4; i++){
    console.log(i);
    }
```

```
FOR IN LOOP
    const pet = {nickname: "peanut butter', animal:'dog', age:1.5}
    for (const property in pet){
        console.log('${property}: ${pet[property]}');
    }
```

```
FOR OF LOOP
    const array1 = ['a', 'b', 'c'];
    for (const element of array1){
        console.log(element);
    }
```

- Functions control statements are a block of code that can be called upon just by calling its name and if conditions are required can be entered into the call function command at the same time. Control flow would then go off to wherever this function is written, perform the code in the function and then go back to where the function was called to continue on.

```
FUNCTIONS
    function sumofTwo(num1, num2){
        return num1 + num2;
    }

    const result = sumofTwo(32, 21);
    console.log(result);
```

Q8

Type Coercion is the process of changing a unit of data from one data type to another. Data types in Javascript can be listed as the following. Boolean, Number, BigInt, String, Null and Undefined.

Because Javascript is a dynamically typed language, you dont need to specefy data types everytime. Javascript has a certain amount of common sense to assume correctly what needs to be done in order to allow the code to run error free. This is called Implicit Coercion where the data type coercion is implied.

An example of Implicit coercion from number to string would be as follows. You can't add a num to a string, so Javascript assumes for it to make sense that the number 1 in the below example must be a string. Now with string concetation, which is the merging of two string data items to become one data item, the console printed answer would be a string data type of '111'

```
const numtoString = '11' + 1;
console.log(numtoString);
```

Another example of Implicit coercion is from string to number. If you had a string of '11' and wanted to minus a number of 1 from it, it doesnt make sense. So what Javascript does is assumes for it to make sense that the string type of 11 is mislabled and is infact a number type of 11. Then the code works and 11 - 1 results in the answer of 10

```
const stringtoNum = '11' - 1;
console.log(stringtoNum);
```

The other type of data coercion is called Explicit Coercion. This is where instead of allowing Javascript to assume what changes need to occur to data types in order for the code to work, the data type is intentionally changed by the programmer. This is done by using Constructors which are built in to Javascript language and allow you to change the data type intentially.

These constructer are written like this. Number(), String() and Boolean(). The below example of code shows how they are implimented in script. The Console printed result would be '4567' as a string and not as a number data type.

```
const stringConstructor = String(4567);
console.log(stringConstructor);
```

Q9

A data type is defined by the value it can take and tells the compiler/interpretor how the data is intended to be used moving forward. Data types in Javascript can be defined as the following. Boolean, Number, String, Null, Undefined and Objects.

All but Objects are called immutable data types. Objects are mutable. This is because, in regards to the immutable data types, once you have set a data value and data type to a variable, it will never change no matter what you do to it or use it for.

Objects are called a special data type and are mutable, because even after setting the data values in it, and the data type(object), any code refering to the Object can change the contents of the Object after it is initially declared.

- Boolean: Boolean data types mean it can only have two values. True or False. These are good for control flow structures. When you need to have the direction of the control flow determined by whether the assessed condition is one way or another, true or false. In the example, it is assessed if weather as a variable is equal to hot. If this is true, then it will print to console singlet, if it is false, it will print to console jacket.

```
let weather = 'cold';
let clothing = weather === "hot" ? "singlet" : "jacket";
console.log('it is ${weather}, wear a ${clothing}');
```

- Number: Number data types are items of information that need to be able to be calculated against. Whether they be counted in iterations, or added together for example. Whilst a string data type would treat the number 1 as a word, a number data type would treat the number 1 as a number. Another characteristic of JS number data types is that JS does not discern between the different types. Whilst in Ruby Language you need to be clear whether the number has decimal places or not by designating the data type as either a INT or a FLOAT. In JS, it is all inclusive under Number. A Number data type can only be 16 digits long because JS numbers are 64 bit floating numbers. The below example shows where JS is smart enough to assume that the amounts are number data types and treat them as so. The result would be printing the number 104 to the screen.

```
const result = 52 + 52;
console.log(result);
```

- String: String Data types are textual data. Represented inside quotation marks '', it shows to JS, that no matter what format the information takes, if it's between the two quotation marks that is exactly how the data is to be represented. The below example shows how a piece of data is saved to a variable as a string data type, and how it can be manipulated, in this case the amount of characters in the variable are counted to be 5 and printed to the screen.

```
const b = 'hello';
console.log(b.length);
```

- Null: Null data type means there is no value, It is empty. More specifically Null is actually an 'object' data type. This means that the variable that is denoted as Null exists and has been assigned a value, that value being NULL. Its a way to initialize variables without assignment value to them. The following example initializes the variable items, but has no value assigned yet.

```
let items = null;
```

- Undefined: Undefined data type means it is undefined for now but should be some value in there. It is a data type thats covers a variable that has been initialized but does not contain any value. For example, if you were to initialize variableExample and assigning nothing to it, when tested it would return undefined.

```
var variableExample;
console.log(variableExample);
```

- Objects : Object data types are key-value pairs in hashes. It is a data type that can store multiple collections of data and as explained before, is mutable in that it can be manipulated and changed after initialization. Below is an example of a Object,progress, being assigned its keys and its values. Keys being Program and Language, values being Programming 1 and Ruby.

```
let progress = {
    program: "Programming 1",
    language: "Ruby"
};
```

Q10

Arrays are data structures, a collection of data assigned to a single name. This data is listed in order, using an index system from zero to Highest Number allowed. Every piece of data in the array is allowed to be of differing data type, it can even be an array of arrays, or array of objects. Below is an example of an array with three data elements of the string data type. I will use this example as reference for all examples below.

```
let tool = [ "Hammer", "Screwdriver", "Spanner"];
```

Javascript has a series of inbuilt methods that you can add to array commands to achieve specific manipulation results.

push() - If you want to add new data elements to the end of an existing array, The below example will now add 'Ratchet' as a string data type to the end of the tools array, it will appear after 'Spanner'.

```
tool.push("ratchet");
```

pop() - If you want to remove the last data element in an array. The below example applied to the tools array will remove 'Spanner' leaving an array that only has 'Hammer' and 'Screwdriver' in it.

```
tool.pop();
```

shift() - If you wanted to to remove the first data element in the array. The below example applied to tools would leave you an array with only 'Screwdriver' and 'spanner' in it.

```
tool.shift();
```

map() - If you wanted to perform an action to every item in an array then return a new array with the manipulated data, this inbuilt method acts like a loop. The below example will create a new array called pluralizedTools which contains the original data elements but they will now be pluralized using string concatentation.

```
const pluralizedTools = tool.map( tools => tools + 's');
```

forEach() - Similiar to map(), however mainly used for creating a loop of actions to an array data elements without returning a new array value. Once complete this method returns undefined. The below example would run through each element in the 'tool' array and action the function written inside the forEach method which is to print to console the data element.

```
tool.forEach((item) => console.log(item));
```

includes() - If you needed to determine if a specific data element was in an array and you knew the element your were searching for you could use includes(). In this example,

```
tool.includes('Hammer')
```

Q11

A Javascript Object is a collection of data assigned to a specific name. Referred to as Hashes in Ruby, in Javascript they are called objects and are structured in the same way, consisting of Key value pairs. The best way to understand an object is to use an example. If you had a electric drill, what properties would it have. It would have a brand, a model, a price, a power rating, a colour etc. So below is an example of how you would write out the object drill in Javascript.

```
const Drill = {
    brand: 'Makita',
    model: 'T1000',
    price: 299.99,
    power_rating: '1000nm',
    colour: 'Blue'
};
```

If you wanted to access any of the information within the Object Drill, in a similiar format to the inbuilt methods of arrays you can call the indiviual keys. See below example of the command to print to console the value of the drill model key to screen. The result would be displaying the string T1000.

```
console.log(Drill.model);
```

Objects can contain objects within themselves too. For example if drill.brand had two names, Makita and Power range you would structure the object with a second object inside brand. In order to print 'Power range' to screen, you would call the object key brand.secondname as seen below.

```
const Drill = {
      brand: {
          firstname: 'Makita',
          secondname: 'Power range',
          },
      model: 'T1000',
      price: 299.99,
      power_rating: '1000nm',
      colour: 'Blue'
  };

console.log(Drill.brand.secondname)
```

You can also update existing values in an array with the following command. object.key = newvalue;. Below is an example where the drill price is updated to 199.99 from the original 299.99

```
Drill.price = 199.99
```

You can also also create new Keys in the object after the fact if new information is needed to be recorded. The below example adds a new key:value pair to the object Drill, in this case adding a Key for costprice and a value of 99.99. .

```
Drill.costprice = 99.99
```

The existing object Drill would now look like this

```
 const Drill = {
    brand: 'Makita',
    model: 'T1000',
    price: 299.99,
    power_rating: '1000nm',
    colour: 'Blue',
    costprice: 99.99
};
```

The next level for object manipulation in Javascript is Constructors. Also called classes this enables you to creat multiple objects with multiple properties without having to make object specific code. In the example we have been using, we made the object Drill. But if we had gone back one step and made the Constructor class Powertool for example, then we could use the object structure for all types of power tools including Drill. The below example is how we would set up the Class Powertool.

```
class powerTools {
  constructor(brand, model, price, power_rating, colour, costprice) {
      this.brand = brand;
      this.model = model;
      this.price = price;
      this.power_rating = power_rating;
      this.colour = colour;
      this.costprice = costprice;
  }
}
```

The command 'This.' is used to refer to the object the code is being written inside off.

To enter the example information we have on Drill object we would enter the line

```
const Drill = new powerTools('Makita','T1000', 299.99, '1000nm', 'Blue',
99.99)
```

This process now allows you to do this with multiple different power tools in this example and use the code and specifically the format written in the one class - powerTools.

(MDN web docs, n.d)

Q12

JSON is a term that stands for Javascript Object Notation. JSON format is used for the transmission of data from servers to browser terminals and return. JSON is very similiar in syntax to Javascript Objects, allowing complex data to be structured in a key:value pair series. However using built in methods for JSON manipulation, the JSON can be converted from object to string data type for data transmission. This converstion to string for network transmission is called 'Serialization'.

The two inbuilt methods for JSON conversion are `STRINGIFY` and `PARSE`.

1. Stringify - The inbuilt method Stringify will manipulate the JSON object and turn it into a string data type format. This serialization of a JSON object will allow the object to take a string data format and be ready for transmission. The below example shows an object of several key:pair values. Once the object undergoes Stringify, the output will be the entire object as a string.

```
  const Drill = {
      brand: 'Makita',
      model: 'T1000',
      price: 299.99,
      power_rating: '1000nm',
      colour: 'Blue',
      costprice: 99.99
  };
  const drillString = JSON.stringify(obj);
```

The output in the variable of 'drillString' would look like the foollowing.

```
  '{ "brand": "Makita", "model": "T1000","price": 299.99, "power_rating":
"1000nm", "colour": "Blue", "costprice": 99.99}'
```

2. Parse - The inbuilt method Parse will manipulate the JSON string back into a JSON object where the data will be structured in the format it needs to be useful in. The below example shows the syntax for converting incoming JSON string data by the name of drillString back into a usable JSON object.

```
  const drillObject = JSON.parse(drillString);
```

This result will be an object formatted to look exactly like object DRill in previous example.

Q13

The below code snippet for Q13 is printing to console 40 iterations of the same sentance. "I have a" and "it was made in". The only difference each time is a randomly decided carname, picked from a predetermined array placed between the two string statements and a year of manufacture placed, again randomly determined at the end of the second string statement. The following example is how it would look for a single iteration to console. "I have a Ford, it was made in 1980"

```

  class Car { // This is an object class for Car objects
    constructor(brand) { // This is a method for initializing a object
instance of the Car class and the variable brand
      this.carname = brand;  // assigns the variable brand to
this.carname, this is in reference to whatever object the code is inside
of when it is written.
    }
    present() {  // This is the creation of a new function called
present()
      return 'I have a ' + this.carname;  // using string concatentation,
it is mergeing a statement of string, whatever value is assigned to
'this.carname' when it is called and returning the result to the main
control flow.
    }
  }

  class Model extends Car {  // This is an object class for Model objects
plus extends allows inheritence from the Car class and all its specific
details.
    constructor(brand, mod) {  // This is a method for initializing a
object instance of the Car class and the variables brand and mod.
      super(brand); // is a method that refers to the parent class and
allows the current Model class to inherit all the functions and specifics
from the Parent class, Car.
      this.model = mod;  // assigns the variable mod to this.model, this
is in reference to whatever object the code is inside of when it is
written.
```

```
    }
    show() {
      return this.present() + ', it was made in ' + this.model;  // using
string concatentation,  it is mergeing a statement of string, whatever
value is assigned to 'this.model' when it is called and returning the
result to the main control flow.
    }
  }

  let makes = ["Ford", "Holden", "Toyota"]  // this statement is
initializing and proliferating an array called makes with three car model
names in string format.
  let models = Array.from(new Array(40), (x,i) => i + 1980)  // this
statement is initializing and proliferating an array called model.
Array.from sets the dimensions and conditions, with Array size being
limited to 40 index postions, and that each value from 0 index onwards is
started from the year date 1980.

  function randomIntFromInterval(min,max) { // min and max included //
this is the declaration of a function, and stating there are two
conditions that need to be entered for it to work, labeling them min and
max,.
      return Math.floor(Math.random()*(max-min+1)+min); // this is to
return a result when this function is called upon. The result is derived
from the Calculation. Math.floor will truncate the decimal and ensure the
return number you get will always be a whole number. Math.random will
provide a randomly generated number within the parameters that are set.
End result is generating a randomly generated number withing the called
parameters.
  }

  for (model of models) {  // this is the setup of a loop action,
specifically looping for very model in the array models. which is 40
times.

    make = makes[randomIntFromInterval(0,makes.length-1)]  // this is
saving a result to the variable make. The result will be 1 of the data
elements in the array Makes. Instead of entering a specific number or name
to call in the array,  it calls to the function randomIntFromInterval to
ensure the final result is random.
    model = models[randomIntFromInterval(0,makes.length-1)]  // // this is
saving a result to the variable model. The result will be 1 of the data
elements in the array Models. Instead of entering a specific number or
name to call in the array,  it calls to the function randomIntFromInterval
to ensure the final result is random. It is using the function paramters
to be set off Makes.length, which is only 3 data elemnts long.

    mycar = new Model(make, model);  // now the variables make and model
have been randomly determined,  they are entered into the Model Class. the
result is assigned to the variable mycar
    console.log(mycar.show())  //instructs the computer to print to
console the  result  of variable  mycar and prewritten method show() in
the class.
  }
```

References

- ISO n.d, ISO/IEC 25010:2011(en), Online Browsing Platform, viewed 27 July 2022, https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en
- MongoDB n.d, MERN Stack Explained, viewed 2 August 2022, https://www.mongodb.com/mern-stack
- Yaskevich, A n.d, "How to assemble a good web development team", Sciencesoft, viewed 2 August 2022, https://www.scnsoft.com/blog/how-to-assemble-a-good-web-development-team
- Mdn web docs, n.d, "JavaScript object basics", viewed 17 August 2022, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics