

**ICTICT449 Use version control systems in development environments**

**ICTPRG437 Build a user interface**

**ICTPRG440 Apply introductory programming skills in different languages**

**ICTPRG302 Apply introductory programming techniques**

Learner version

# Assessment task 2

JavaScript Portfolio

**Disclaimer:** This resource was developed by Holmesglen Institute of TAFE in 2024 for the State of Victoria (Department of Education and Training) for educational purposes, is © State of Victoria (Department of Education and Training) and is available under a Creative Commons Attribution Non-commercial 4.0 International [(CC BY-NC 4.0)](https://creativecommons.org/licenses/by-nc/4.0/) license.

**The license does not apply to:** Trademarks or branding, including the Victorian Government logo and any unattributed content supplied by third parties



Learner version

# Assessment task 2

JavaScript Portfolio

|  |  |
| --- | --- |
| Learner information | |
| **Learner name:** Van Minh Le | **Learner ID:** 100693330 |
|  | |

|  |  |
| --- | --- |
| Section A – Program/course details | |
| **Qualification code:** ICT40120 | **Qualification title:**  Certificate IV in Information Technology (Programming) |
| **Unit code:**  **CPRO1 - Javascript**  ICTICT449  ICTPRG437  ICTPRG440  ICTPRG302 | **Unit title:**  **JavaScript**  Use version control systems in development environments  Build a user interface  Apply introductory programming skills in different languages  Apply introductory programming techniques |
| **Pre/co-requisites:** N/A | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| Section B – Assessment task details | | | |
| **Assessment number: 2** of 2 | **Semester/year:** 1/2025 | | **Due date:** 6/06/2025 |
| **Duration of assessment:** 8 weeks | | **Assessment task results:** This assessment task will be marked as:  Ungraded result: satisfactory or not yet satisfactory | |
|  | | | |

|  |
| --- |
| Section C – Instructions to learners |
| Assessment task description:  This assessment has two parts:   * Part A – Knowledge questions * Part B – Portfolio section   For Part A, you are required to answer all questions in the boxes below each question. You should be writing at least 1 paragraph for your answers, unless the question specifies otherwise. Some questions have tables to fill out.  Part B is broken up into 6 different parts.   * Part 1: Data structures and algorithms – simple data types. * Part 2: Using basic data structures – objects * Part 3: String manipulation and files * Part 4: Data structures and algorithms – complex data types * Part 5: Build a graphical User Interface * Part 6: Version control in a team environment   While doing each part the following requirements and guidelines must be followed:  **Documentation Requirement**  You are required to fully comment the codes, you must   1. Comment all functions 2. Comment all variables   **Comply with Coding Guidelines**  You must follow the coding guidelines below when developing the program (Refer to [w3school JS style guide](https://www.w3schools.com/js/js_conventions.asp) for detailed explanation for each of the following rules)   1. Use camelCase for identifier names (variables and functions) 2. All names start with a letter 3. Always put spaces around operators (= + - \* /), and after commas 4. Always use 2 or 4 spaces for indentation of code blocks 5. Statement rules    1. Always end a simple statement with a semicolon    2. Put the opening bracket at the end of the first line.    3. Use one space before the opening bracket.    4. Put the closing bracket on a new line, without leading spaces.    5. Do not end a complex statement with a semicolon.   **Version Control**   * You are required to create a **minimum of 3 repositories** to store your code. * Each part has a section to record your repository for that part. * Suggested grouping is, parts 1 and 2, part 3, parts 4 and 5. * You are required to have at least 3 branches for each repository. * Each question has an area for you to write in your branch name and commit number. * You need to have 3 merge requests as well for each repository.   **Project setup**   * You need to create a root folder called “JSAT2” * Under this folder create the folders part1, part2, part3, part4, part5, part6.      * All work for each part should be placed inside these folders * You can create sub folders in each of these folders if you need to. I.e. one for wireframes for example. * You are required to submit all your code zipped up to Brightspace   For more detailed instructions for each part, refer to the supporting document.  {RTO to insert relevant policy and procedural information for submission of assessments} |
|  |

|  |
| --- |
| Section D – Conditions for assessment |
| ****Unit application:****   * This is an individual task. (except for Part 6 of this assessment) * You must meet all criteria listed in the marking guide to be satisfactory in this task. * You must submit all required working files, documentation, and any other assets that you feel may be required in a zipped file, including the completed and signed coversheet. The assessment must be completed and submitted electronically to Brightspace by the due date. If this is not possible, you must contact your assessor to gain written approval for an alternative arrangement for submitting the assessment. * If not successful within the enrolment period as per Holmesglen assessment procedure, you will be requested to resubmit within 7 days of receiving feedback. You will have the opportunity to resubmit if any part of the assessment is deemed unsatisfactory (you are permitted TWO (2) resubmission per assessment task). Resubmissions must be submitted by the resubmission due date provided by your teacher. * This task is open book. You may use the internet for research purposes only. All answers must be in your own words. Where a quote is used, you must cite the information source. * If you feel you require special allowance or adjustment to this task, please discuss with your assessor within one week of commencing this assessment. Any change to assessment arrangements must be reviewed by the Education Manager and approved by the Head of Department. * You can appeal an assessment decision according to the Holmesglen Assessment Complaints and Appeals Procedure. * You are expected to dedicate time to develop this assessment task both in and out of the classroom. * Leaner must contribute to and abide by organisational standards including intellectual property, privacy laws, and plagiarism and academic honesty. Further information is detailed at: https://holmesglen.edu.au/Students/Student-Resources/ |
|  |

|  |  |
| --- | --- |
| Equipment/resources learners must supply: | Equipment/resources to be provided by the RTO: |
| * **Learners opting to BYOD laptop or intending to learn remotely will require access to:** * A MAC or PC/laptop with the following minimum specification: * Quad Core CPU * • CPU with minimum 4GHz processor or faster * • RAM 16GB * • 200GB of storage * • Windows 10 OS, or virtual machine installed with Windows 10 OS (available free from https://developer.microsoft.com/en-us/windows/downloads/virtual-machines/ or https://www.microsoft.com/en-us/evalcenter/evaluate-windows-10-enterprise) * • Monitor 24" (PC only, dual monitor optional but preferred) * • Headset with microphone (webcam optional but preferred) * • Access to reliable internet connection (ADSL or cable connection desirable) * Applications are available through Holmesglen MyHorizon and some have free licences which can be downloaded via url below: * • Brightspace (Learning Management System) access - https://holmesglen.brightspace.com/ * • MyHorizon - https://myHorizon.holmesglen.edu.au - free to download * • 365 Microsoft office suite also can be downloaded via https://portal.office.com - free for all Holmesglen students * • LinkedIn Learning - free access * • WebEx - https://holmesglen.webex.com/ - free to download * • Unity - https://unity.com/ - student license free to download * • Maya - http://www.autodesk.com/education/free-software/maya - student license free to download * • Substance Painter https://substance3d.adobe.com/education/ - student license free to download * • Adobe Photoshop - https://www.adobe.com/au * • ZBrush – https://pixologic.com/ * • GitHub desktop - https://desktop.github.com/ - free to download * • 3D Coat - http:/3dcoat.com/download/ * • 7Zip or an equivalent compression utility - free to download * • Google Chrome – recommended web browser * • OneDrive - https://www.microsoft.com/en-ww/microsoft-365/onedrive/online-cloud-storage - free to downloade | **This is a blended learning course and as such a remote learner will access their own computer equipment as per the specification provided. The Institute will provide the following:**   * Holmesglen CAIT computer classroom: * data projector, whiteboard, computer with double screens, mobile Cisco Spark Board, Conference camera * HP Z1 Entry Tower G5 * • Dual Displays 24” HP Monitor * • CPU: Intel Core i7-9700K @ 4GHz * • Motherboard: 8591 * • RAM 32GB * • 1TB SSD * • OS: Windows 10 * This will vary as CAIT upgrades computer rooms. * • Internet connection * Applications available at ZENworks and Holmesglen MyHorizon and free to download via url below: * • Brightspace (Learning Management System) access - https://holmesglen.brightspace.com/ * • 365 Microsoft office suite also can be downloaded via https://portal.office.com - free to Holmesglen students * • LinkedIn Learning - free access * • WebEx - https://holmesglen.webex.com/ - free to download * • Unity - https://unity.com/ - student license free to donwload * • Maya - http://www.autodesk.com/education/free-software/maya - student license free to download * • Substance Painter https://substance3d.adobe.com/education/ - student license free to download * • Adobe Photoshop - https://www.adobe.com/au * • ZBrush – https://pixologic.com/ * • GitHub desktop - https://desktop.github.com/ - free to download * • 3D Coat - http:/3dcoat.com/download/ * • 7Zip or an equivalent compression utility - free to download * • Google Chrome – recommended web browser   • OneDrive - https://www.microsoft.com/en-ww/microsoft-365/onedrive/online-cloud-storage - free to download |

|  |  |  |  |
| --- | --- | --- | --- |
| Section E – Assessment questions/criteria | | | |
| **Assessment number:** 2 | | **Assessment title:** JavaScript Portfolio | |
| **Learner name:** Van Minh Le | | | **Learner ID:** 100693330 |
| **Unit code:**  **CPRO1 - Javascript**  ICTICT449  ICTPRG437  ICTPRG440  ICTPRG302 | **Unit title:**  JavaScript  Use version control systems in development environments  Build a user interface  Apply introductory programming skills in different languages  Apply introductory programming techniques | | |
| **Date:** Insert date |  | | |
|  | | | |

**PART A - KNOWLEDGE QUESTIONS (Written Response)**

**Provide your responses in the boxes below each question.**

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Compare GIT Hub and one other Version Control System. Look at the functionality of each system and how they work. Record your answer below.  Based on your research, which Version Control System would you pick and why? | Yes  No |
| Answer: I’m comparing GitHub and Subversion. Both platforms are Version Control Systems.  <compare>   |  |  |  | | --- | --- | --- | | **Feature** | **GitHub** | **Subversion (SVN)** | | Version Control Model | It is a **Distributed** Version Control System, meaning each developer has a full copy of the repository. | It is a **Centralized** Version Control System, mean a single central repository stored on a server. | | Branching and Merging | **Powerful** whencreating **branches and flexible** when **merging.**   * It allows us to create **lightweight** **branches** quickly and work independently without effecting the main codebase. * It supports **three main types of merging**: Fast-forward merge; three-way merge and rebase, giving **more control over history.** | **Branching** issimple **and less flexible** when **merging.**   * Branches in Subversion are just **copies of the repository**, making them heavier. * It tracks merges but it is more **complex** and can lead to conflicts. | | Offline work | It allows **work can be done offline** and commits stored locally. | It **requires an internet connection** to commit changes. | | Performance | It is **fast**, especially when working directly local repositories. | It **slow**, especially with large repositories or remote operation. | | Collaboration | It supports pull requests, forks, and **open-source contributions**. | It works primarily **with team-based collaboration** and **restricted access**. | | Conflict Resolution | It **resolves conflicts** more **easily** through merging and make it more efficient. | It is more **difficult** **to resolve** because changes need to be committed to the central server first. | | File handling | It is **inappropriate** for **large binary files**. | It is better **suited** for handling **large binary files.** | | Community and Ecosystem | It is **widely** used and has **strong open-source support**. | It is **mainly** used in **enterprises** with a **smaller** **community**. |   <Which VCS, and why?>  Based on above comparison, I would pick **GitHub**. Because GitHub more **flexible**, **supports** **offline** work. The ability to **easily branch, experiment, and merge** makes it ideal for modern software development, especially for teams. | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Define distributed and centralised version control systems and discuss the differences between the two types | Yes  No |
| Answer:  Definition:   * + - 1. Distributed Version Control Systems (DVCS) is a type of version control where every developer has a complete copy of the repository, including its full history. This means that changes can be made, committed, and reverted locally without needing an internet connection.       2. Centralized Version Control Systems (CVCS) relies on a single central repository stored on a remote server. Developers must be connected to the server to commit, update, or retrieve changes.  |  |  |  | | --- | --- | --- | | **Feature** | **Distributed Version Control Systems (DVCS)** | **Centralized Version Control Systems (CVCS)** | | Repository Structure | Each developer has a full local copy of the repository | Single central repository on a server | | Branching and Merging | Highly efficient and flexible. Branches are lightweight and local, allowing for rapid development. Supports fast-forward, three-way, and rebase merging. | Less efficient. Branches are server-side copies, making them heavier. Merging can be complex and conflict-prone. | | Offline Work | Allows offline work; network needed only for synchronization | Requires constant network connection for most operations | | Performance | Operations are generally faster, especially local operations | Operations can be slower due to network dependence | | Data Redundancy | High redundancy: each developer's local repository serves as a backup | Single point of failure; potential data loss if the central server fails | | Scalability | Scales well with a large number of users and repositories due to its distributed nature | Can become a bottleneck with a large number of users | | Security | Local control: each developer has a full copy, which can be complex to manage overall security | Centralized access control | | File Handling | Handles large binary files, but not as efficiently as text files. | Can handle large binary files slightly better than DVCS in some cases. | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| **Describe the following concepts related to Version Control System (VCS)**  **• repositories and working copies**  **• merging and merge tools**  **• working and indiscriminative commits**  **• resolving conflict and backout changes** | Yes  No |
| Answer:   |  |  | | --- | --- | |  | **Version Control System (VCS)** | | Repository | A central database storing all versions of project files and their history. | | Working Copy | A local, editable copy of files from the repository. | | Merging | Combining changes from different branches or versions. | | Merge Tools | Software applications that help visually resolve merge conflicts. Examples: KDiff3, Meld, Beyond Compare. | | Working Commits | Frequent, small, focused commits with clear messages, representing single, logical changes. | | Indiscriminate Commits | Infrequent, large commits with many unrelated changes, making it hard to understand and revert. | | Resolving Conflicts | Manually editing conflicting files to choose which changes to keep or create a merged version. | | Backing Out Changes | Undoing the effects of a previous commit using commands like git revert or SVN revert. | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| **Describe principles and techniques of creating repositories and branch workflows. (Hint: Look at the workflow in Part 6)** | Yes  No |
| Answer:   |  |  | | --- | --- | | **Principle and Technique** | **Description** | | **Principles of Creating Repositories** | | | Clear Purpose | Define the project's scope and goals before creating the repository. This ensures you're organizing your codebase effectively. | | Structure | Plan the directory structure to logically organize your code, documentation, and other assets. | | Centralized Approach | A single repository for all developers to contribute to. | | Administrator | Creates the repository and manages access. | | **Branch Workflow** | | | Simplified Model | Primarily uses feature branches and a main branch. | | Feature Branches | Create separate branches for each new feature or bug fix. This isolates changes and facilitates code review. | | Pull Requests | Used for submitting changes and getting feedback. | | **Techniques** | | | Branch Naming | Use descriptive names for branches | | Commit Messages | Write clear and concise commit messages explaining the changes made. | | Code Review | Use pull requests or merge requests to facilitate code review before merging branches. | | Merging | The manager merges approved branches into the main branch once they have been reviewed | | **Additional** | | | Testing | Encouraged before committing. | | Final Sign-off | Ensures quality assurance. | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Describe version control industry standard best practices. | Yes  No |
| Answer:  Best practices for Version Control  Use Feature Branches: Each developer should work on a separate branch to avoid conflicts with the main branch.  Follow a Clear Branching Strategy: Choose a suitable strategy like Git Flow, GitHub Flow, or Trunk-Based Development.  Commit Early and Often: Make frequent, small commits to track changes and debug issues more easily.  Write Meaningful Commit Messages: Clearly describe the commit's purpose and follow a consistent format like "[Feature] Short description".  Leverage Pull Requests and Code Reviews: Require code reviews through pull requests before merging to ensure quality and catch bugs.  Keep the Main Branch Stable: Merge only tested and reviewed code. Use CI/CD pipelines for automated testing before merging.  Tag Releases and Use Versioning: Use semantic versioning (e.g., v1.0.0) and tag releases to simplify rollbacks and troubleshooting.  Handle Merge Conflicts Properly: Regularly pull updates from the main branch to minimize large merge conflicts. Test thoroughly after resolving conflicts.  Maintain a Clean Repository: Use a .gitignore file to exclude unnecessary files like logs or temporary files to keep the repository clean.  **Automate Testing and Deployment:** Use CI/CD tools (e.g., GitHub Actions, Jenkins) to ensure the code is functional and ready before merging. | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Describe distributed version control industry standard best practices. | Yes  No |
| Answer:  Use Feature Branches: Work in separate branches to keep changes isolated and organized.  Commit Early, Commit Often: Make small, meaningful commits for better tracking and debugging.  Write Clear Commit Messages: Follow a consistent format to describe changes effectively.  Pull and Rebase Frequently: Keep your branch updated with the latest changes from the main branch.  Use Pull Requests & Code Reviews: Ensure code is reviewed and tested before merging.  Keep the Main Branch Stable: Only merge tested and approved code to avoid breaking production.  Use Tags & Versioning: Apply semantic versioning (v1.0.0) to track stable releases.  Handle Merge Conflicts Properly: Regularly sync with the main branch and resolve conflicts carefully.  Use .gitignore: Exclude unnecessary files (logs, build artifacts, etc.) to keep the repo clean.  Automate Testing & Deployment: Use CI/CD tools to maintain quality and streamline releases. | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Describe workflow processes applicable to using version control systems. You may describe the workflow studied in class. (Hint: look at part6) | Yes  No |
| Answer:  Clone Repository: Copy the central repository to a local machine.  Create a Feature Branch: Work on a separate branch for new features or fixes.  Make Changes & Commit: Edit files, stage them, and commit with clear messages.  Pull & Rebase: Update the branch with the latest main branch changes.  Push to Remote Repository: Upload changes to the remote repository.  Create a Pull Request (PR): Submit code for review and approval.  Merge to Main Branch: Integrate approved changes into the main branch.  Tag Releases (Optional): Mark stable versions for easy tracking.  Delete Merged Branches: Clean up unnecessary branches.  Automate with CI/CD: Use automated testing and deployment for stability | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| **Discuss a range of strategies when developing a user interface with HTML and CSS. Finish the table below.** | Yes  No |
| Answer:   |  |  | | --- | --- | | **Requirement** | **Strategies** | | Introduce a 20 px gap between two vertically aligned element. | Use **margin-bottom: 20px;** on the first element or **gap: 20px**; in a flex/grid container. | | Place an element in the centre of its container. | Use display: flex; justify-content: center; align-items: center; or margin: auto; with a fixed width. | | Apply certain colour to all text-based element in a container | Use **color: #yourColor;** on a parent container or target specific elements with p, h1, span *{ color: #yourColor; }.* | | Display information in a table manner | Use the **<table>** element with **<tr>** for rows and **<td>** for data cells. | | Set colour using RGB values and HEX values | Use color: **rgb(255, 0, 0);** for red or  **color: #FF0000;** as the HEX equivalent. | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| HTML + CSS + JavaScript languages are used as a combination to program web-based user interface. Discuss the role / functions of each of the languages in building web-based UI. | Yes  No |
| Answer:  **HTML (HyperText Markup Language)**   * **Role**: Structure and content of the web page. * **Function**:   + HTML provides the basic structure of the webpage by using **elements** such as headings, paragraphs, forms, links, and images.   + It defines the content hierarchy and layout using tags like <div>, <header>, <article>, <section>, and <footer>.   **CSS (Cascading Style Sheets)**   * **Role**: Design and appearance of the web page. * **Function**:   + CSS controls the **layout**, **color**, **fonts**, **spacing**, and other visual aspects of the HTML elements.   + It enhances the user experience by making the interface visually appealing and responsive across different screen sizes.   + CSS allows styling through **selectors** (class, ID, element) and **properties** like color, font-size, margin, padding, display, etc.   + It enables **responsive design** through media queries to adapt the page layout to various screen sizes.   **JavaScript**   * **Role**: Interactivity and dynamic behavior of the web page. * **Function**:   + JavaScript enables **interactivity**, allowing users to interact with the page elements (e.g., buttons, forms, menus).   + It can manipulate **DOM (Document Object Model)** elements, change the structure, or update content dynamically without needing to reload the page (AJAX).   + JavaScript can handle **events** (click, hover, submit), validate forms, create animations, and perform calculations.   + It is also used for **handling API requests** and **data fetching**. | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Describe small-size web-based UI development / application process | Yes  No |
| Answer:  **Planning**: Define the purpose, key features, and target audience. Identify necessary UI elements.  **Wireframing & Design:** Create wireframes and design the layout using tools like Figma or Sketch.  **HTML Structure:** Write the basic structure using semantic HTML tags (<header>, <main>, <footer>).  **Styling with CSS:** Apply styles to ensure a clean design. Use media queries for responsiveness.  **Adding JavaScript:** Implement interactivity (e.g., button clicks, form validation) using JavaScript.  **Testing & Debugging:** Test across browsers and screen sizes. Debug using developer tools.  **Deployment**: Choose a hosting platform (e.g., GitHub Pages, Netlify) and upload the files.  **Maintenance:** Collect feedback, fix bugs, and update features as needed. | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Identify application development languages that can be used to build the following application. And briefly describe each of them. | Yes  No |
| Answer:   |  |  |  | | --- | --- | --- | | **Application Type** | **Language** | **Description** | | **Desktop Window Application** | **C# (.NET)** | C# is a modern, object-oriented language widely used on Windows. With the .NET framework, it simplifies the development of robust desktop applications. | | **Web Application** | **JavaScript** | JavaScript, together with HTML and CSS, is the core language for interactive web development. It drives dynamic content and client-side logic on web pages. | | **Mobile Application** | **Dart (with Flutter)** | Dart is an object-oriented language developed by Google. When used with the Flutter framework, it enables building high-performance, cross-platform mobile apps. | | **Smart TV Application** | **JavaScript (Web Tech)** | Many Smart TV platforms support apps built with web technologies—HTML, CSS, and JavaScript—allowing developers to create interactive, lightweight TV applications. | | **Smart Watch Application** | **Swift** | Swift is a fast, modern language designed by Apple. It’s primarily used for developing native apps on iOS and watchOS, offering performance and safety on small devices. | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| **Describe what is UI (User Interface) prototyping and identify at least two prototyping tools.** | Yes  No |
| Answer:  What is UI (User Interface) Prototyping?  UI prototyping is the process of creating a preliminary model of a user interface to simulate and validate the design, layout, and interactive elements of an application. It helps designers and stakeholders visualize the flow, gather feedback, and identify improvements before full-scale development begins.   |  |  | | --- | --- | | **Tools Name** | **Website** | | Figma | <https://www.figma.com> | | Adobe XD | <https://www.adobe.com/products/xd.html> | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| What is UI style guide, and what should a UI style guide contain (with 50+ words)? | Yes  No |
| Answer:  **What is UI Style Guide?** A UI style guide is a comprehensive reference document that outlines the design standards and visual language for a product’s user interface. It serves as a blueprint that ensures consistency, coherence, and a unified brand experience across all digital touchpoints. The style guide bridges the gap between design and development by providing clear instructions for creating and implementing user interface elements.  **What Should a UI Style Guide Contain?** A robust UI style guide should include guidelines for colors, typography, icons, spacing, and imagery, along with component libraries and layout grids. It should specify interactive behaviors, such as hover and active states, and detail responsive design principles. Additionally, it often provides usage patterns, code snippets, and tone-of-voice recommendations for content, ensuring that both design and functionality align seamlessly across different platforms and devices. | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Use example to explain the following JavaScript syntaxes. | Yes  No |
| Answer:   |  |  | | --- | --- | | **Syntax** | **Example** | | **Define variable** | let myVariable = 42;  console.log("Value of myVariable is:", myVariable); | | **Condition structure** | // If-else condition structure  if (myVariable > 40) {  console.log("Greater than 40");  } else {  console.log("40 or less");  } | | **Loop structure** | // For loop example  for (let i = 0; i < 5; i++) {  console.log("Loop iteration:", i);  } | | **Define a function** | // Function definition example  function greet(name) {  return `Hello, ${name}!`;  }  console.log(greet("Alice")); | | **Define a class** | // Class definition example  class Person {  constructor(name, age) {  this.name = name;  this.age = age;  }  introduce() {  return `Hi, I'm ${this.name} and I'm ${this.age} years old.`;  }  }  const person = new Person("Bob", 30);  console.log(person.introduce()); | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Describe organisational procedures and guidelines that can be used for the following languages. | Yes  No |
| Answer:   |  |  | | --- | --- | | **Language** | **Organisational procedures and guideline examples** | | **JavaScript** | - Code Standards: Follow consistent coding styles, such as using JavaScript Standard Style or Airbnb Style Guide.  - Linting Tools: Use tools like ESLint to enforce code quality and identify potential issues automatically.  - Modular Code Structure: Organise code using modules or components to make it reusable and maintainable.  - Versioning: Use semantic versioning for JavaScript libraries and frameworks to track releases.  - Documentation: Include JSDoc comments for documenting functions, parameters, and return values. | | **HTML** | - Semantic Markup: Use semantic tags (e.g., <header>, <article>, <section>) to improve readability and accessibility.  - Accessibility Standards: Adhere to WCAG (Web Content Accessibility Guidelines) by using proper ARIA roles and attributes.  - Validation Tools: Validate code with HTML validators to check for errors and improve compatibility.  - Consistent Structure: Use a logical file structure for organizing HTML files and components (e.g., templates, partials).  - Responsive Design: Implement responsive design practices such as viewport meta tags and flexible layouts. | | **CSS** | - Preprocessors: Use tools like SASS or LESS to manage complex styles more efficiently with variables, mixins, and nesting.  - Consistent Naming: Follow naming conventions like BEM (Block, Element, Modifier) for class names to keep the code organized and maintainable.  - Style Guidelines: Use a consistent CSS formatting style, such as indentation, spacing, and ordering of properties.  - Avoid Inline Styles: Use external stylesheets or CSS modules to separate content from presentation.  - Performance Optimization: Minify CSS, use responsive media queries, and eliminate unused styles to improve load times. | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Describe at least 3 JavaScript shorthand technique with examples | Yes  No |
| Answer:   |  |  |  | | --- | --- | --- | | **Technique name** | **Description** | **Example** | | Ternary Operator | A shorthand for an if-else statement. It evaluates a condition and returns one of two values. | let result = (age >= 18) ? "Adult" : "Minor"; | | Arrow Functions | A shorter syntax for writing function expressions, especially useful for inline functions. | const add = (a, b) => a + b; | | Template Literals | A shorthand for string concatenation, using backticks and ${} for embedding expressions. | let greeting = `Hello, ${name}!`;  // Easier than concatenating strings with + | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Identify the official standards of the following languages | Yes  No |
| Answer:   |  |  |  |  | | --- | --- | --- | --- | | **Language Name** | **Standard name** | **Description** | **Official link** | | JavaScript | ECMAScript (ES) | ECMAScript is the standard for scripting languages, including JavaScript. It defines the language's core features and behavior. | https://ecma-international.org/ | | HTML | HTML Living Standard | The HTML Living Standard is maintained by the Web Platform Working Group and defines the structure and semantics of web content. | https://html.spec.whatwg.org/ | | CSS | CSS (W3C) | The W3C CSS specifications define how CSS should be written and implemented across web browsers for styling HTML content. | https://www.w3.org/Style/CSS/ | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Describe code conversions that are used to comment JavaScript codes | Yes  No |
| Answer:  **1. Single-line Comment**   * **Syntax**: // * **Description**: This comment type is used for single-line comments. Anything after the // on that line will be ignored by the JavaScript engine.   **2. Multi-line Comment**   * **Syntax**: /\* \*/ * **Description**: This comment type is used for comments that span multiple lines. Everything between /\* and \*/ will be treated as a comment, even if it spans across multiple lines. | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| **Describe the following methodologies when programming with HTML/JavaScript/CSS languages.** | Yes  No |
| Answer:   |  |  | | --- | --- | | **Methodology** | **Description** | | **Development Process** | The development process for HTML, JavaScript, and CSS typically involves planning the project, creating the structure (HTML), styling (CSS), and adding interactivity (JavaScript). It follows a structured workflow with stages like wireframing, designing, and implementing features. Key practices include version control, code modularity, and maintaining consistent style conventions (e.g., naming conventions, code formatting). | | **Debugging Process** | Debugging in HTML, JavaScript, and CSS involves identifying and fixing issues in the code. For HTML, this could involve validating the structure with tools like W3C Validator. For JavaScript, tools like browser developer tools (console, breakpoints, etc.) and linters (e.g., ESLint) are used to identify logical or runtime errors. CSS issues are often debugged using browser inspection tools and testing for cross-browser compatibility. | | **Testing Techniques** | Testing techniques for web development include unit testing, integration testing, and end-to-end testing. JavaScript can be tested using frameworks like Jest or Mocha for unit tests. For HTML and CSS, testing might include validating HTML with W3C Validator, checking styles across multiple browsers using tools like BrowserStack, and ensuring responsive design with manual testing or automated tools like Selenium. | | **Using Data Structures** | In JavaScript, efficient use of data structures (like arrays, objects, and maps) is critical for performance and maintainability. Arrays store ordered collections, objects hold key-value pairs, and maps store keys of any type. Data structures are essential for managing dynamic content, such as handling user input, storing form data, and dynamically modifying the DOM. In HTML, data structures can be represented by lists, tables, and forms, and CSS can style them accordingly. | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Give two example of web-based DevOps lifecycle tools. Describe the tools and identify their official websites. | Yes  No |
| Answer:   |  |  |  | | --- | --- | --- | | **Tool name** | **Description** | **URL** | | Jenkins | Jenkins is an open-source automation server used to automate the parts of software development related to building, testing, and deploying. It integrates with various DevOps tools and supports continuous integration and continuous delivery (CI/CD). Jenkins can automate repetitive tasks in the software lifecycle. | https://www.jenkins.io/ | | GitLab | GitLab is a complete DevOps platform that provides version control, CI/CD, monitoring, and security. It allows teams to collaborate on code, run tests, deploy applications, and monitor performance in one place. It supports the entire DevOps lifecycle from planning to deployment. | https://about.gitlab.com/ | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Give two example of DevOps automation tools. Describe the tools and identify their official websites | Yes  No |
| Answer:   |  |  |  | | --- | --- | --- | | **Tool name** | **Description** | **URL** | | Docker | Docker is a platform that automates the deployment of applications inside lightweight, portable containers. It allows applications to run consistently across different environments by bundling them with all necessary dependencies. | https://www.docker.com/ | | Jenkins | Jenkins is one of the most popular automation tools for continuous integration and continuous delivery (CI/CD). It automates the building, testing, and deployment of applications. Jenkins supports many plugins to integrate with various other tools. | https://www.jenkins.io/ | | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| List at explain the following concepts for JavaScript   * Datatypes – list and explain at least 3 different datatypes * Operators – list and explain 3 different operators * Expression – explain what an expression is. You may use an example * Variables – list and explain two different ways to create variables | Yes  No |
| Answer:  Datatypes:  **String**: Represents a sequence of characters enclosed in single or double quotes. It is used to store text.  **Number**: Represents both integer and floating-point numbers (without quotes). It is used for numerical operations.  **Boolean**: Represents a logical value, either true or false. It is often used in conditional statements.  Operators:  **Arithmetic Operators**: Used to perform basic mathematical operations like addition, subtraction, multiplication, and division.  **Comparison Operators**: Used to compare two values, returning a Boolean (true or false) result.  **Logical Operators**: Used to combine multiple conditions in control structures like if, while, and for. Examples include && (AND), || (OR), and ! (NOT).  Expression  **Explanation**: An expression is a piece of code that evaluates to a value. It can be a combination of variables, operators, and values that return a result.  Example :  let result = 5 \* 2 + 3; // Expression that evaluates to 13  Variables:  **Using var**: The traditional way to declare a variable in JavaScript, but it is function-scoped and can lead to issues if not used carefully.  *var name = "John";*  **Using let**: A more modern way to declare variables. It is block-scoped and allows reassignment.  *let age = 30;*  *age = 31; // Variable can be reassigned* | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| List and explain the following concepts for JavaScript using examples   * What is a sequence * What is selection * What is an iteration construct | Yes  No |
| Answer:  **1. Sequence:**   * **Explanation**: A sequence refers to the execution of statements in a specific order, one after another. In JavaScript, the code runs from top to bottom, and each statement executes in sequence unless otherwise directed (e.g., using loops or conditionals). * **Example**:   let a = 5; // First statement  let b = 10; // Second statement  let sum = a + b; // Third statement  console.log(sum); // Output: 15  In this example, each statement is executed in sequence: first a = 5, then b = 10, followed by the sum calculation, and finally logging the result.  **2. Selection:**   * **Explanation**: Selection allows you to control the flow of execution based on conditions. This is typically done using if, else, and switch statements. The program evaluates a condition and selects which block of code to execute accordingly. * **Example**:   let age = 20;  if (age >= 18) {  console.log("Adult");  } else {  console.log("Minor");  }  In this example, the program evaluates whether age is greater than or equal to 18. If true, it prints "Adult"; otherwise, it prints "Minor".  **3. Iteration Construct:**   * **Explanation**: An iteration construct, or loop, allows you to repeat a block of code multiple times. Common iteration constructs in JavaScript are for, while, and do...while. Loops are useful when you need to execute the same code repeatedly, like iterating through a collection or performing repetitive tasks. * **Example** (Using for loop):   for (let i = 0; i < 5; i++) {  console.log(i); // Prints numbers 0 through 4  }  In this example, the for loop runs 5 times, starting from i = 0 and increasing by 1 until i is no longer less than 5. Each time, it prints the value of i to the console. | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Explain JavaScript debugging techniques. Using an example explain how you would find an error in your code by debugging. | Yes  No |
| Answer: JavaScript Debugging Techniques:   1. **console.log() Statements**: Use console.log() to print values and check variable states.    * **Example**:   let result = x + y;  console.log(result);   1. **debugger Statement**: Use debugger to pause the code and inspect variables step-by-step.    * **Example**:   let result = x + y;  debugger; // Pauses execution   1. **Browser DevTools**: Use DevTools to set breakpoints and step through the code.    * Open DevTools → Sources tab → Set breakpoints. 2. **try...catch for Error Handling**: Use try...catch to catch and handle errors without crashing.    * **Example**:   try {  let result = 10 / 0;  } catch (error) {  console.log("Error: " + error.message);  }  **Finding an Error Example:**  **Code with Error:**  let num1 = 10;  let num2 = "5";  let sum = num1 + num2;  console.log(sum);  **Steps:**   1. Use console.log() to check values.    * **Output**: num1 = 10, num2 = "5". 2. **Fix**: Convert num2 to a number:   let sum = num1 + Number(num2); // Output: 15 | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Explain how you would test a JavaScript application. List the steps you would follow.  Talk about what methods you would use to test your code. | Yes  No |
| Answer:  Steps to Test a JavaScript Application:   1. Unit Testing: Test individual functions using frameworks like Jest.    * Example: expect(add(1, 2)).toBe(3); 2. Integration Testing: Test how components work together (e.g., API calls) with Mocha or Supertest. 3. End-to-End Testing: Simulate user actions with tools like Cypress or Selenium. 4. Manual Testing: Interact with the UI to find bugs and compatibility issues. 5. Performance Testing: Test load times and responsiveness using Lighthouse or WebPageTest. 6. Debugging: Use console.log(), breakpoints, and DevTools to fix issues. 7. Cross-Browser Testing: Ensure compatibility on different browsers using tools like BrowserStack.   Testing Methods:   * Unit Testing: Tests individual functions. * Integration Testing: Checks interaction between components. * End-to-End Testing: Verifies the entire app flow. * Manual Testing: Human evaluation of UI. * Performance Testing: Ensures efficient performance. | Comment: |

|  |  |
| --- | --- |
| Question | Satisfactory response |
| Describe how you would document your programming activities. List what documents are created and what their purpose is. (Hint: you may want to look at part 5) | Yes  No |
| Answer:  **Repository Details and Initial Commit**:   * **Purpose**: Documents the setup of the project repository and the first commit with project files. * **Content**: Repository URL, description, and initial commit message detailing the setup of the project.   **UI Requirements Gathering and Client Communication**:   * **Purpose**: Records the gathered requirements from the client and the signed approval. * **Content**: Documented user interface (UI) requirements, client communication, and sign-off confirmation.   **HTML Guidelines**:   * **Purpose**: Specifies the HTML coding standards and practices followed for the project. * **Content**: List of HTML guidelines, such as semantic tags, accessibility practices, and validation rules.   **Software Applications for Wireframe and UI Prototype**:   * **Purpose**: Identifies the tools used to create the wireframe and prototype. * **Content**: List of software tools (e.g., Figma, Adobe XD) chosen for wireframing and prototyping.   **Wireframe Creation**:   * **Purpose**: Documents the process of building the wireframe using the selected tools. * **Content**: Detailed description of the wireframe, including UI elements and layout.   **Client Feedback on Wireframe**:   * **Purpose**: Records feedback from the client on the wireframe and subsequent changes made. * **Content**: Client feedback, changes implemented, and the client's sign-off on the revised wireframe.   **Theme Creation for the UI**:   * **Purpose**: Describes the process of defining the visual theme of the UI. * **Content**: Documented color schemes, fonts, and visual design elements used   **UI Prototype Creation**:   * **Purpose**: Documents the creation of the UI prototype, incorporating feedback and theme. * **Content**: UI prototype with theme colors, structure, and interactive elements   **Client Feedback on Prototype**:   * **Purpose**: Captures feedback on the UI prototype and any required adjustments. * **Content**: Feedback from the client, changes made to the prototype, and final sign-off from the client.   **Content Flow, UI Actions, and UI Events**:   * **Purpose**: Describes the sequence of content and user interactions within the application. * **Content**: Flow of information, user interactions (clicks, inputs), and events that trigger actions.   **Languages Used to Build the UI**:   * **Purpose**: Details the languages chosen for the front-end and their specific roles. * **Content**: Description of HTML (structure), CSS (styling), JavaScript (interactivity), and any libraries or frameworks used.   **UI Construction**:   * **Purpose**: Describes the building process of the UI based on the selected languages and specifications. * **Content**: Final UI code, structure, and any challenges faced during development.   **Test Plan Development**:   * **Purpose**: Documents the approach for testing the UI functionality. * **Content**: Test cases, expected outcomes, and the testing framework used.   **Conducting Tests and Recording Results**:   * **Purpose**: Describes the testing process, results, and changes made after testing. * **Content**: Test results, issues identified, fixes made, and re-testing efforts.   **UI Demonstration to Client**:   * **Purpose**: Describes the final demonstration of the UI to the client and obtaining the final approval. * **Content**: Details of the demonstration, client feedback, and final sign-off confirmation. | Comment: |

**PART B - PORTFOLIO**

|  |  |  |
| --- | --- | --- |
| Criteria for assessment | SatisfactoryYes No | Comment |
| **The following has been submitted for assessment:** | | |
| Part 1 |  |  |
| 1A Learner has listed repository details |  |  |
| 1.1 Learner has built the array to specifications |  |  |
| 1.2 Learner has sorted the array |  |  |
| 1.3 Learner has inserted the required numbers into the array and kept the array in ascending order |  |  |
| 1.4 Learner has removed the required numbers from the array and kept the array in ascending order. |  |  |
| 1.5 Learner has implemented sequential search algorithm |  |  |
| 1.6 Learner has implemented binary search algorithm |  |  |
| 1.7 Learner has developed a test plan for parts 1.5 and 1.6 |  |  |
| 1.8 Learner has conducted and recorded the results of the tests created in 1.7 |  |  |
| 1.9 Learner has documented the JavaScript code file as per requirements and organisational requirements |  |  |
| 1.10 Learner has used debugging tools to set a break point, step through code and look at variable contents |  |  |
| 1.11 Learner has followed all guidelines and organisational requirements. |  |  |
| Part 2 |  |  |
| 2A Learner has listed repository details |  |  |
| 2.1 Learner has built the object to specifications |  |  |
| 2.2 Learner has added the new properties to the object |  |  |
| 2.3 Learner has modified the required properties on the object |  |  |
| 2.4 Learn has removed the required property on the object |  |  |
| Part 3 |  |  |
| 3A Learner has listed repository details |  |  |
| 3.1 Learner has created the required strings |  |  |
| 3.2 Learner has used length and charAt to get the required information |  |  |
| 3.3 Learner has used slice and substring |  |  |
| 3.4 Learner has used toUpperCase and toLowerCase |  |  |
| 3.5 Learner has used concat, trim, replace and split as required |  |  |
| 3.6 Learner has created algorithms to save to a text file and read from a text file |  |  |
| 3.7 Learner has written a string to a text file |  |  |
| 3.8 Learner has read text from a text file |  |  |
| Part 4 |  |  |
| 4A Learner has listed repository details |  |  |
| 4.1 Learner has implemented the Movie class |  |  |
| 4.2 Learner has created an array of 10 movies, unordered |  |  |
| 4.3 Learner has sorted the array by Movie ID |  |  |
| 4.4 Learner has implement either sequential or binary search algorithm |  |  |
| Part 5 |  |  |
| 5.1 Learner has listed repository details and initial commit |  |  |
| 5.2 Learner has gathered and understood the UI requirements. The learner has communicated with the client about the requirements and get a sign off. |  |  |
| 5.3 Learner has listed the HTML guidelines the are following and described them |  |  |
| 5.4 Learner has identified the software applications to be used to create the wireframe and UI prototype |  |  |
| 5.5 Learner has created the UI wireframe using the tools listed in 5.4 |  |  |
| 5.6 Learner has met the client to get feedback on the wireframe, made changes due to feedback and gotten client sign off to the changes made. |  |  |
| 5.7 Learner has created a theme for the UI |  |  |
| 5.8 Learner created a UI prototype using the tools listed in 5.4. The learner enhanced the wireframe with the theme colours defined in 5.7 |  |  |
| 5.9 Learner has gotten feedback and made changes to the prototype, then gotten the client sign off. |  |  |
| 5.10 Learner has described the content flow, UI actions and UI events for the application |  |  |
| 5.11 Learner has determined the languages to be used to build the UI. The learner has described the languages and their purpose in building the UI |  |  |
| 5.12 The Learner has built the UI with the languages listed in part 5.11 and to specifications. |  |  |
| 5.13 The learner had developed a test plan to test the UI functionality |  |  |
| 5.14 The learner has conducted the tests in the test plan, recorded the results and made changes where needed. |  |  |
| 5.15 The learner has demonstrated the UI to the client and obtained the final sign off. |  |  |
| Part 6 |  |  |
| 6.1 Learner has created the GitHub account and communicated any difficulties or disruptions while completing this task |  |  |
| 6.2 Learner has cloned the required repository, created their team directory and required files. The learner has recorded the merge request, branch name and commit number |  |  |
| 6.3 The Learner has received and recorded feedback from the teacher about their page. |  |  |
| 6.4 The Learner has actioned changes from feedback and recorded the merge request, branch name and commit number |  |  |
| 6.5 The learner has updated the myscript.js file to add in their details. |  |  |
| 6.6 The learner has obtained the final sign off. |  |  |

|  |
| --- |
| Section F – Feedback to Learner |
| **Has the learner successfully completed this assessment task?** Yes No |
| **Assessor feedback (as appropriate):** Insert feedback |
| **Resubmission allowed:** Yes No  **Resubmission due date:** Insert date |
| **Assessor name:** Insert name |
| **Assessor signature:** |
| Learner acknowledgement: |
| **Learner Signature:** |
| **Date:** Insert date |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Part 1: Data structure and algorithm – simple data type** | | | |
| **Programming Requirement**  In this part, you are required to design and implement a range of algorithms to operate an array with given input. The following operations need to be done using JavaScript programming language and the result need to be verified using browser.   1. Build an array with the following numbers.    * 11, 5, 8, 3, 25, 16, 31, 45, 14, 20 2. Sort the array in ascending order (means to arrange them from smallest to largest) 3. Insert three numbers 19, 23, 30 into the sorted array in step 2, and keep the new array sorted in ascending order. You could decide where to insert. 4. Remove two numbers 8, 31 from the array generated in step 3, and keep the new array sorted in ascending order 5. Create a JavaScript function called “sequentialSearch”    * This function accepts two parameters: an array and a value    * This function searches the input array for the input value using sequential search    * You cannot use the JavaScript build-in “find” function. You must implement the algorithm with you own code.    * This function returns the index of the target value if found and or -1 if not found 6. Create a JavaScript function called “binarySearch”    * This function accepts two parameters: an array and a value    * This function searches the input array for the input value using binary search    * You cannot use the JavaScript build-in “find” function. You must implement the algorithm with you own code.    * This function returns the index of the target array element if found or returns a -1 if not found   **Documentation Requirement**  You are required to fully comment the codes, you must   1. Comment all functions 2. Comment all variables   **Comply with Coding Guidelines**  You must follow the coding guidelines below when developing the program (Refer to [w3school JS style guide](https://www.w3schools.com/js/js_conventions.asp) for detailed explanation for each of the following rules)   1. Use camelCase for identifier names (variables and functions) 2. All names start with a letter 3. Always put spaces around operators (= + - \* /), and after commas 4. Always use 2 or 4 spaces for indentation of code blocks 5. Statement rules    1. Always end a simple statement with a semicolon    2. Put the opening bracket at the end of the first line.    3. Use one space before the opening bracket.    4. Put the closing bracket on a new line, without leading spaces.    5. Do not end a complex statement with a semicolon.   **Debugging**  You are required to use the developer tool in modern web browser (e.g. google chrome) to debug the code. You are required to demonstrate the following skills   * Setup break point to pause the execution * Use code step through functions to step-by-step trace the code execution * Examine variable contents in runtime   **Testing**   * You are required to develop a test plan for this programming task, using required test plan template. * The test case shall have a full coverage of function “sequentialSearch” and “binarySearch” * You are required to test your code according to test plan and record the test result. * If any of the test cases failed, you are required to debug the codes, fix the issue and test again, until all test cases pass.   **Version Control**   * You are required to create a repository to store your code. * You are required to have at least 3 branches for this part. * Each question has an allowance for you to write in your branch name and commit number. * You need to have 3 merge requests as well.   **Submit for review**  You are required to submit the following for review   1. This assessment document with all questions answered 2. All source code files 3. Test plan & test result   You are required to compress all files should in a zip file and submit the zip file for review. | | | |
| **Q1A Record GitHub repository details for this part:** | | | |
| **Record the GitHub repository name:** **JSAT2-parts1and2**  **Record the GitHub repository URL: https://github.com/vanminhle50/JSAT2-parts1and2.git** | | | |
| **Q1.1 Build an array with the following numbers in JavaScript and print the array in the console: 11, 5, 8, 3, 25, 16, 31, 45, 14, 20** | | | |
| < Insert the screenshot JavaScript code that built the array>    <Insert the screenshot of the array printed in the console >    Is your result correct?  Yes  (If No, fix your code and correct the result) | | | |
| Branch name | part-1 | Commit Number: | 9fc243e |
| **Q1.2 Sort the array created in Q1.1 above in “ascending” order (ascending is smallest to largest)** | | | |
| < Insert the screenshot JavaScript code that sorts the array>    <Insert the screenshot of the sorted array printed in the console>    Is your result correct?  Yes / No.  (If No, fix your code and correct the result) | | | |
| Branch name | part-1 | Commit Number: | 04d1442 |
| **Q1.3 Insert three numbers 19, 23, 30 into the sorted array above, and keep the new array in ascending order** | | | |
| < Insert the screenshot JavaScript code that inserts the new numbers>  (Note: remember that you can choose where you place the numbers or sort the array again.)    <Insert the screenshot of the new array printed in the console >    Is your result correct?  Yes  (If No, fix your code and correct the result) | | | |
| Branch name | part-1 | Commit Number: | 40bdf29 |
| **Q1.4 Remove two numbers 8, 31 from the array generated in Q1.3, and keep the new array in ascending order.** | | | |
| < Insert the screenshot JavaScript code that removes the numbers from the array>  (Note: remember you can choose what to delete)    <Insert the screenshot of the array printed in the console >    Is your result correct?  Yes  (If No, fix your code and correct the result) | | | |
| Branch name | part-1 | Commit Number: | d61a47f |
| **Q1.5 Implement the function “seqentialSearch” as per part 5 of the task description** | | | |
| <Insert a screenshot of the JavaScript code of the function “sequentialSearch”>    <Insert a screenshot of the console when the target element is found>    <Insert a screenshot of the console when the target element was not found>    Is your result correct?  Yes  (If No, fix your code and correct the result) | | | |
| Branch name | part-1 | Commit Number: | 41fd178 |
| **Q1.6 Implement the function “binarySearch” as per part 6 of the task description** | | | |
| <Insert a screenshot of the JavaScript code of the function “binarySearch”>    <Insert a screenshot of the console when the target element is found>    <Insert a screenshot of the console when the target element was not found>    Is your result correct?  Yes  (If No, fix your code and correct the result) | | | |
| Branch name | part-1 | Commit Number: | 0e98ac8 |
| **Q1.7 You are required to develop a test plan that covers the functions “sequentialSearch” and “binarySearch” The test plan should be in the submission for review.** | | | |
| Your test plan will need to test the functionality of these two functions.  <Record the name of your test plan file:> **TestPlanPart1.docx** | | | |
| Branch name | part-1 | Commit Number: | f08e2d7 |
| **Q1.8 Conduct the tests in your test plan and record the results in your test plan file.** | | | |
| Record the results of your test plans in your test plan document.  <Record the name of your test plan file:> **TestResultPart1.docx** | | | |
| Branch name | part-1 | Commit Number: | 70ca837 |
| **Q1.9 You are required to properly document your JavaScript code. The JavaScript code files must be submitted for review.** | | | |
| <Record the name of the JavaScript code files for this part:> **part1.js** | | | |
| Branch name | main | Commit Number: | 751af91 |
| **Q1.10 While development of the functions in Q1.5 and Q1.6 use the debugging tools to show the following:** | | | |
| <A screenshot of a break point set up>  <Screenshots of a step-by-step flow of one of the functions>  <A screenshot of variable contents>  **Details in the file with name: Use the debugging tools .docx** | | | |
| Branch name | main | Commit Number: | 011e388 |
| **Q1.11 Confirm that all coding guidelines have been followed** | | | |
| Yes | | | |
| Branch name |  | Commit Number: |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Part 2 – Using basic data structures – Objects** | | | |
| **Programming Requirement**  In this part, you are required to design and implement a range of algorithms to operate an object with given input. The following operations need to be done using JavaScript programming language and the result need to be verified using browser.   1. Build an object called “myMovie” with the following properties and data.    * Title: The longest day    * Year: 1964 2. Add in the properties:    * rating: 5    * Summary: World War II movie about the Normandy landings 3. Change the rating to 4 and change the year to 1962 4. Remove the Summary property   As per the last part, each question will have a space for you to record a branch name and commit code. | | | |
| **Q2A Record GitHub repository details for this part:**  **(Note: This repository can be the same as part1)** | | | |
| **Record the GitHub repository name: JSAT2-parts1and2**  **Record the GitHub repository URL: https://github.com/vanminhle50/JSAT2-parts1and2.git** | | | |
| **Q2.1 Build an object called “myMovie” with the following properties and data:**   * + **Title: The longest day**   + **Year: 1964** | | | |
| <Insert the screenshot JavaScript code that built the object>    <Insert the screenshot of the object printed in the console>    Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | part-2 | Commit Number: | a60e8aa |
| **Q2.2 Add the following properties and data to the “myMovie” object:**   * + **rating: 5**   + **Summary: World War II movie about the Normandy landings** | | | |
| < Insert the screenshot JavaScript code that inserted new properties in the object>    < Insert the screenshot of the new object printed in the console >    Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | part-2 | Commit Number: | a60e8aa |
| **Q2.3 Change the data of the properties listed below:**   * + **Rating: 4**   + **Year: 1962** | | | |
| < Insert the screenshot JavaScript code that updated the properties in the object >    <Insert the screenshot of the new object printed in the console>    Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | part-2 | Commit Number: | 49d7e40 |
| **Q2.4 Remove the summary property from “myMovie”:**   * + **Title: The longest day**   + **Year: 1964** | | | |
| < Insert the screenshot JavaScript code that removed the property in the object >    < Insert the screenshot of the new object printed in the console >    Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | part-2 | Commit Number: | 2fd5900 |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Part 3 – String Manipulation and Files** | | | |
| **Programming Requirement**  In this part, you are required to design and implement a range of algorithms to manipulate strings. The following operations need to be done using JavaScript programming language and the result need to be verified using node.  You will then need to create an algorithm and implement this to write a string to a text file and read in a text file. Reading and writing to file can be done in either C# or JavaScript. If using JavaScript, you will need to verify the results using Node.js.  As per the pervious parts, there are spaces for you to record your version control. The repository, branch name and commit code.  You should start a new repository for this part.   1. Define the following strings:    * myString = “This is a string”    * anotherString = “ Another string” (note there are 3 blank spaces at the beginning)    * hello = “Hello there!    * myName = your first name 2. Use the utility functions length and chat to find out information    * Find out the length of myString    * Find out the first character of myString    * Find out the 11th character of myString 3. Use the slice and substring functions to create new strings.    * Slice “is a” from myString    * Use substring to get “the” from anotherString 4. Change the case of the myName string    * Print out the myName string in all upper case    * Print out the myName string in all lower case 5. Use concat, trim, replace and split to create new strings    * Concat hello and myName together    * Use the trim function to remove the spaces in anotherString    * Use the replace function to change “is a” to an empty space in myString    * Split myString on the spaces. 6. Create algorithms    * Create algorithm for saving string to a text file    * Create algorithm for reading in the text file 7. Write a text file    * Name the text file output.txt    * Create a string of data to save to a text file    * Write the string to the text file 8. Read text from a text file    * Get the file name output.txt    * Read in the text file    * Prin the text that was read in, to the screen.   **Submit for review**  You are required to submit the following for review   1. This assessment document with all questions answered 2. All source code files   You are required to compress all files should in a zip file and submit the zip file for review. | | | |
| **Q3A Record GitHub repository details for this part:** | | | |
| **Record the GitHub repository name:** **JSAT2-part3**  **Record the GitHub repository URL: https://github.com/vanminhle50/JSAT2-part3.git** | | | |
| **Q3.1 Define the following strings:**   * + **myString = “This is a string”**   + **anotherString = “ Another string” (note there are 3 blank spaces at the beginning)**   + **hello = “Hello there!**   + **myName = your first name** | | | |
| <Insert the screenshot JavaScript code that built the strings>    <Insert the screenshot of the strings printed in the console>    Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | Part3-Step1-5 | Commit Number: | e25a2d7 |
| **Q3.2 Use the utility functions to find out the following information**   * + **Find out the length of myString**   + **Find out the first character of myString**   + **Find out the 11th character of myString** | | | |
| <Insert a screenshot of the JavaScript code for the length>    <Insert a screenshot of the JavaScript code for first character>    <Insert a screenshot of the JavaScript code for 11th character>    <Insert a screenshot of length in the console>    <Insert a screenshot of the for the first character shown the console>    <Insert a screenshot of the for the 11th character shown the console>  Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | Part3-Step1-5 | Commit Number: | b61ca63 |
| **Q3.3 Use the functions slice and substring to create new strings**   * + **Slice “is a” from myString**   + **Use substring to get “the” from “notherString** | | | |
| <Insert a screenshot of the JavaScript code for slice function>    <Insert a screenshot of the JavaScript code for substring function>    <Insert a screenshot of the slice shown in the console>    <Insert a screenshot of the substring shown the console>  Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | Part3-Step1-5 | Commit Number: | 988c77d |
| **Q3.4 Change the case of the myName string**   * + **Change myName string to upper case**   + **Change myName string to lower case** | | | |
| <Insert a screenshot of the JavaScript code>    <Insert a screenshot of the console showing myName in upper case and in lower case>  Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | Part3-Step1-5 | Commit Number: | 24a4a77 |
| **Q3.5 Use various utility methods to create new strings**   * + **Use the concat method to join the hello string and myName string**   + **Use the trim method to remove the spaces in anotherString**   + **Use the replace method to change “is a” to an empty space in myString**   + **Use the split method on myString using the spaces to break the string** | | | |
| <Insert a screenshot of the JavaScript code>  <Insert a screenshot of the console showing the new strings>    <concat>    <trim>    <replace>    <split>  Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | Part3-Step1-5 | Commit Number: | 66bd0b9 |
| **Q3.6 Create algorithms for file handling**   * + **Create an algorithm for saving a string to a text file**   + **Create an algorithm for reading in a text file** | | | |
| <Saving a text file algorithm>    <Reading a text file algorithm> | | | |
| Branch name | main | Commit Number: | e25a2d7 |
| **Q3.7 Write a text file**   * + **Name the text file output.txt**   + **Create a string to save to the text file**   + **Write the text file**   **Note: For this question you can use JavaScript or C#** | | | |
| <Insert a screenshot of the JavaScript code>    <Insert a screenshot showing the created text file>    <Insert a screenshot showing the contents of the text file>  Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | Part3-Step7-8 | Commit Number: | f6d9e1d |
| **Q3.8 Read the contents of a text file**   * + **Read in the text file output.txt from Q3.7**   + **Read in the contents of the file**   + **Display what was read in from the text file to the screen**   **Note: For this question you can use JavaScript or C#** | | | |
| <Insert a screenshot of the JavaScript code>    <Insert a screenshot showing the console >  Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | Part3-Step7-8 | Commit Number: | 0182bfe |

|  |  |  |  |
| --- | --- | --- | --- |
| **Part 4 – Data structures and algorithms – complex data types** | | | |
| **Programming Requirement**  In this part, you are required to design and implement a range of algorithms to operate an array of objects. The following operations need to be done using JavaScript programming language and the result need to be verified using browser.  You will be required to create a GitHub repository for this part. As per the previous parts, branch names an commit numbers can be recorded in each question below.   1. Define a movie class with the following properties.    * Movie ID    * Title    * Year    * Rating 2. Create an array of at least 10 movies with mock values    * Movie ID must be unique for each movie.    * Movie ID must not be sorted in the array 3. Sort array by Movie ID in ascending order 4. Implement at least one search algorithm (sequential search or binary search)    * This function accepts two parameters: an array and a movie ID    * This function searches the input array for the input movie ID using selected searching algorithm    * You cannot use the JavaScript build-in “find” function. You must implement the algorithm with your own code.    * This function returns the movie object found or return “null” if not found   Note: Students may use <https://www.mockaroo.com/> to help generate mock data for array.  **Submit for review**  You are required to submit the following for review   1. This assessment document with all questions answered 2. All source code files   You are required to compress all files should in a zip file and submit the zip file for review. | | | |
| **Q4A Record GitHub repository details for this part:** | | | |
| **Record the GitHub repository name:** **JSAT2-parts4-5**  **Record the GitHub repository URL: https://github.com/vanminhle50/JSAT2-parts4-5.git** | | | |
| **Q4.1 Define the movie class as required using JavaScript. Insert the screenshot of your code below** | | | |
| <Insert a screenshot of the JavaScript code that built the class>    <Insert a screenshot showing the movie class in the console > | | | |
| Branch name | part4 | Commit Number: | 3e218bf |
| **Q4.2 Create an array of at least 10 movies with mock values and print the array in console.**  **Remember: Movie ID must be unique for each movie and not sorted in the array.** | | | |
| <Insert a screenshot of the JavaScript code that built the array>    <Insert a screenshot showing the array in the console >  Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | part4 | Commit Number: | a153584 |
| **Q4.3 Sort the array in “ascending” order by Movie ID.**  **Remember: Movie ID must be unique for each movie and not sorted in the array.** | | | |
| <Insert a screenshot of the JavaScript code that sorted the array>    <Insert a screenshot showing the sorted array in the console >  Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | part4 | Commit Number: | 45811a0 |
| **Q4.4 Implement at least one search algorithm (either sequential search or binary search) according to the requirements in the task description.** | | | |
| Which search algorithm was chosen: **binarySearch and sequentialSearch**  **1. BinarySearch**    <Insert a screenshot of the JavaScript code of selected search function>    < Insert a screenshot of the search function when the target ID was found in the console >    < Insert a screenshot of the search function when the target ID was not found in the console>  **1. SequentialSearch**    <Insert a screenshot of the JavaScript code of selected search function>    < Insert a screenshot of the search function when the target ID was found in the console >    < Insert a screenshot of the search function when the target ID was not found in the console>  Is the result correct?  Yes  (If no, fix your code till the result is correct) | | | |
| Branch name | part4 | Commit Number: | 271c910 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Part 5 – Build a graphical user interface** | | | | | | | | | | | | |
| **UI Requirement**  You are required to create a JavaScript array to **maintain a list of the movie object defined in Part 4** and build a web-based UI to manage the movie list using HTML, CSS and JavaScript. This move list needs to be created as a class.  Also, you are required to create and maintain a repository for this project. You need to upload or push your changes after you complete each question. The repository name is completely up to you, but it must be public.  Remember to use branches and insert your commit numbers in each question  The web-based UI has three sections   1. Add movie section: this UI section allow user to insert one movie to the array. In this UI, the following elements are required    * **A Movie ID input**    * **A Title input**    * **A Year input**    * **A Rating input**    * **A Submit button** – when user click on the Submit button, a movie object will be created using the information get from the input UI above, and this object will be added to the movie JavaScript array 2. Display movie list: this UI section allow user to view the current movie list. In this UI, the following elements are required    * **A list view** – this view is used to display information stored in the movie JavaScript array    * **A Refresh button** – when user click on the Refresh button, all data in the list view will be refreshed with the current data stored in JavaScript 3. Search movie: this UI section allow user to search movie information using movie ID, the following element are required    * **A movie ID input** – user can input a movie ID for searching    * **A Search button** – when user click on Search button, the program will search the movie list using the input ID and return result    * **A search result view** – this view is used to display the search result. The movie details will be displayed if found or a “0 result” information will be displayed if not found.    * **Search by Title** – when this is used search the movie list for something that contains the search string typed in. The list should update with all names shown in the list. 4. Sort movie: this UI section allow user to sort the list in both a-z or z-a fashion and render to the screen,    * **Sort A-Z -** This button should show the movie list sorted by Title A-Z and displayed to the screen    * **Sort Z-A** – This button should show the movie list sorted by Title Z-A and displayed to the screen.    * **Best Movies** - This button should show the movie list sorted by Rating Z – A and displayed to the screen.   **Testing:**  You are required to test both the UI and your JavaScript.   * You need test that your UI shows everything as expected and that it behaves as expected * You also need to test that all your JavaScript functions work as expected.   For the UI defined above you need to style the layout with CSS for better user experience. Pick a theme (colours) to use and stick with it through the page(s).  **Workflow**  You are required to follow the workflow blow to finish the development.   1. Establish UI requirement 2. Build a prototype UI (wireframe or graphic design) 3. Design and implement the UI 4. Test the UI 5. Documentation 6. Obtain client sign-off   **Coding guidelines**  HTML & CSS - <https://www.w3schools.com/html/html5_syntax.asp>  JavaScript - <https://www.w3schools.com/js/js_conventions.asp>  **Mock Data**  Mockaroo: <https://www.mockaroo.com/>  **Submit for review**  You are required to submit the following for review   1. This assessment document with all questions answered 2. All source code files 3. Wireframes in .png or .jpg format | | | | | | | | | | | | |
| **Q5.1 Record GitHub repository details for this part:** | | | | | | | | | | | | |
| **Record the GitHub repository name:**  **Record the GitHub repository URL:**  **The initial commit number:** | | | | | | | | | | | | |
| **Q5.2 Gather the UI requirements from the description above and communicate with the client. Confirm the requirements with your client and obtain client sign off.** | | | | | | | | | | | | |
| List all requirements in the table below:   |  |  | | --- | --- | | **Requirement #** | **Requirement Details** | | 1 | Build a web-based UI to manage the movie list using HTML, CSS and JavaScript.  The web-based UI has four sections:   * 1. Add movie section   2. Display movie list   3. Search movie   4. Sort movie | | 2 | **Add Movie Section:**  Functionality: User should be able to insert a new movie by providing the following details:   * Movie ID: A unique identifier for the movie (Movie ID should be a non-empty string). * Title: The title of the movie (The title should be a non-empty string). * Year: The release year of the movie. (the year must be between 1895 - current). * Rating: The movie’s rating, It’s a number possibly a scale like 0.1-10.0. * Submit Button: When clicked, it should add and show a movie object to the movie list array. | | 3 | **Display Movie List Section:**  **Functionality**: The user should be able to view the entire movie list stored in the JavaScript array.  The list be displayed in a table format:   * **List View:** A visual display of all the movies with their details (ID, Title, Year, Rating). * **Refresh Button:** When clicked, it will refresh the displayed list to match the current JavaScript array. | | 4 | **Search Movie Section:**  **Functionality:** The user should be able to search for a movie.  This section includes:  **1**. **Two buttons:**   * **Search by ID Button:** When clicked, it searches for the movie with the matching ID. * **Search by Title:** When clicked, it searches for the movie with the matching Title.   **2. One Input box:**   * **Search Input:** Enter a specific Movie ID or Title to search.   **3. Search ID Result View:**   * Displays the details of the movie is found * Show the number of the result is found and "0 result" message if not found. | | 5 | ****Sort Movie Section:********Functionality:**** The user should be able to sort the movie list:****Sort A-Z Button****: Sort movies by title in ascending alphabetical order.****Sort Z-A Button****: Sort movies by title in descending alphabetical order.****Best Movies Button****: Sort movies by rating, from highest to lowest. **Bonus**:   * **Sort A-Z by ID Button:** Sort movies by title in ascending alphabetical order.  Sort Z-A by ID Button: Sort movies by title in descending alphabetical order. | | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Checklist and sign off** | | | | | | | | | | | | |
| **Skills to be observed during this task to the required standard.** Checklist (To be completed by the learner’s facilitator)The following tasks are to be completed in relation to the brief for this project. Each of the skills must be observed on at least one occasion. | | | | | | **Date 1** | | | | **Date 2** | | |
|  | | | |  | | |
| **Satisfactory** | | | | **Satisfactory** | | |
| **Yes** | | | **No** | **Yes** | | **No** |
| **Leaner has expressed UI requirements to the client** | | | | | |  | | |  |  | |  |
| **Learner has confirmed and understood the UI requirements** | | | | | |  | | |  |  | |  |
| **Learner has confirmed the task with the client** | | | | | |  | | |  |  | |  |
| **Learn has confirmed the specifications, standards and guidelines to be followed with the client.** | | | | | |  | | |  |  | |  |
| **Assessor Name** | | *Amberle Seidl* | **Assessor Signature** | | *Signature* | | **Date** | | | |  | | |
| **Q5.3 What are the HTML guideline documents that are to be followed? List the filename or URL of the guidelines. Describe the guidelines you are following (at least 1 paragraph)** | | | | | | | | | | | | |
| <HTML guideline / file name or URL>  <HTML guideline description> | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.4 What are the software applications that are to be used to create the prototype UI** (wireframe and prototype) | | | | | | | | | | | | |
| <List tools to use here>  1. Wireframe  - Figma  - InVision  - Photoshop  2. Prototype  VS  <List which tool you are using > | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.5 Build the wireframe of your UI using the tool you identified in question 5.4** | | | | | | | | | | | | |
| <insert the screenshot of your wireframe> | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.6 Review the wireframe with the client and get feedback. Revise and update the wireframe from feedback given.** | | | | | | | | | | | | |
| <List feedback>  <Insert the screenshot of your updated wireframe> | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.7 Create the theme for your UI. Record the colours and font below** | | | | | | | | | | | | |
| <Colour 1 >  <Colour 2>  <Colour 3>  <Font> | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.8 Create prototype of the application using tools listed in Q5.4. Enhance the wireframe with the theme colours.** | | | | | | | | | | | | |
| <Insert screenshot of the prototype > | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.9 Review the prototype with the client and get feedback. Revise and update the wireframe from feedback given. Obtain signoff** | | | | | | | | | | | | |
| <List feedback>  <Insert the screenshot of your updated prototype> | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Client Name** | *Amberle Seidl* | **Client Signature** | *Insert Signature* | **Date** | *Insert Date* |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Q5.10 Describe the content flow / UI Actions / UI Events of each of the UI sections required.**   * **Content flow: Describe the content flow from the user’s perspective** * **UI Actions: describe the UI actions based on the UI design from the user’s perspective** * **UI Events: describe the events triggered in HTML / JavaScript** | | | | | | | | | | | | | |
| |  |  |  |  | | --- | --- | --- | --- | | **UI Section** | **Content flow** | **UI Actions** | **UI Events** | | Add a movie | User input movie id, title, rating and year in the web form, and then click the submit button to add the movie. |  |  | | Display all movies |  | Action 1: user click refresh button  Action 2: list view area display all movies |  | | Search by ID |  |  | Button onclick event is trigger upon clicking the search button | | | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.11 Determine the languages used to build the UI. List all the languages used and describe what they are used for. (at least 1 paragraph for each description.** | | | | | | | | | | | | | |
| |  |  | | --- | --- | | **Language Name** | **Description** | |  |  | |  |  | |  |  | | | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.12 Design and build the UI with the languages listed in Q5.11** | | | | | | | | | | | | |
| List all files and file paths for your UI  <Insert the screenshot of the UI as shown in a browser>  <Get feedback from the client and make changes. List changes made below>  (You may need multiple branches for this question) | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.13 Develop the test cases the verify the UI Functionality** | | | | | | | | | | | | |
| <Insert the name of your test plan file> | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.14 Test the UI according to the test cases created in Q5.13. You must iterate your UI design and build, until the test results meet the requirements.** | | | | | | | | | | | | |
| Record the results of the tests you outlined in 5.13  <Insert the name of your test plan file>  If any tests are not successful, make changes to the code and test again. Record the results of the updates made in your test plan file. | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Q5.15 You are required to demonstrate your application to the client and obtain final sign-off** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| Branch name |  | | | Commit Number: | | | |  | | | | | |
| **Checklist and sign off** | | | | | | | | | | | | |
| **Skills to be observed during this task to the required standard.** Checklist (To be completed by the learner’s facilitator)The following tasks are to be completed in relation to the brief for this project. Each of the skills must be observed on at least one occasion. | | | | | | **Date 1** | | | | **Date 2** | | |
|  | | | |  | | |
| **Satisfactory** | | | | **Satisfactory** | | |
| **Yes** | | | **No** | **Yes** | | **No** |
| **Leaner has used debugging techniques to detect and correct error** | | | | | |  | | |  |  | |  |
| **Learner has developed the application in accordance with client requirements and guidelines** | | | | | |  | | |  |  | |  |
|  | | | | | |  | | |  |  | |  |
| **Assessor Name** | | *Amberle Seidl* | **Assessor Signature** | | *Signature* | | **Date** | | | |  | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Part 6 – Version control in a team environment** | | | | | | | | |
| **Task Requirements**  You are required to contribute to a project and work in a team collaboration environment using GitHub. The project is a website. And the URL of the website is <https://github.com/Amberside/the-programming-team-202510>  The home page of the website has been done. Your task is to create your own page on this website.  Each grid in the home page links to the page of a team member. The link has been defined in the home page (e.g., /team/amberle/index.html), but the member’s page has not been created yet. You need to join the team to get your page created, tested locally and then merge your changes to the project.   * Your teacher’s page (Amberle Seidl) has been created. Click on Amberle’s name to check the detail. * Your page has not been created yet. Find and click on your name to check the detail, which is a 404 file does not exist error.   **1. Requirement of your page**  You shall create a sub-folder, the name of which is made of first name, under the “team” folder. In this folder, you shall create a file called “index.html”. Refer to Amberle’s page as an example.  A screen shot of a computer screen  Description automatically generated  You must include your name.  You need to include some information about yourself.  You can think of this as a portfolio piece about yourself. Include things like experience and projects worked on.  **2. VCS requirement**  You must use GitHub as the VCS system for this task.  You shall provide your GitHub username to your teacher to add you a collaborator of the project, so that you could submit code.  You will also document the use of GitHub and the repositories here as well. Fill out the table provided. You must create at least 3 repositories while completing assessment task 2. These repositories must have at least 3 branches and be updated / merged 3 times.  You are not able to directly push updates to the “main” branch of the repository, instead, a new branch must be created for each push. And a pull request must be created for review. Your submitted code shall be reviewed and approved before merging to the main branch  **3. Edit JavaScript & Feedback**  You will need to update your name in the JavaScript array so that your page can be seen.  Lastly you will need to get feedback and make changes to your page.  **4. Workflow & procedure**  Step 1. Visit <https://github.com/Amberside/the-programming-team-202510>  Step 2. Click on Amberle’s name (the first grid) to view Amberle’s page (page can be viewed)  Step 3. Find your name, click on your name to check your page (page not found)  Step 4. Provide your GitHub username to your teacher to be added to the repository  Step 5. After your teacher invited you to the repository, accept the invitation in your email.  Step 6. Clone the repository to your local environment  Step 7. Create a new branch  Step 8. In folder “team”, create a sub-folder and name it using your first name  Step 9. In the sub-folder created above, create a file named “index.html”.  Step 10. Develop the content  Step 11. Test the result – Your page shall be displayed after clicking your name from the home page  Step 12. You may ask your teacher to confirm the test result before committing the codes  Step 13. Commit the codes  Step 14. Push/publish the commit to the server  Step 15. In GitHub website, create a pull request then notify your teacher to review the codes  Step 16 Respond to feedback.  Step 17. If your code is approved, your teacher will merge the code. Then you could view the website again to confirm the final result.  Step 18. Update the JavaScript file  Step 19. Obtain final sign-off from a supervisor  **5, Upload your pages and this document to Brightpspace** | | | | | | | | |
| **Q6.1 GitHub Details and config** | | | | | | | | |
| <Record your GitHub username>  <record any difficulties with using GitHub>  <Let the teacher know if there are any disruptions while you are doing this part> | | | | | | | | |
| **Q6.2 Create your team page** | | | | | | | | |
| Clone <https://github.com/Amberside/the-programming-team-202510>  <insert a screenshot of GitHub desktop showing the cloned repository>  Add in your directory under the team directory  Add in your name, information and portfolio details  Send the merge request to the teacher.  Merge request number: | | | | | | | | |
| Branch name |  | | | Commit Number: | | |  | |
| **Q6.3 Get feedback from your teacher** | | | | | | | | |
| <Insert feedback from the teacher> | | | | | | | | |
| **Q6.4 Make changes to your page based on the feedback.** | | | | | | | | |
| Make changes and create a new merge request  Merge request number: | | | | | | | | |
| Branch name |  | | | Commit Number: | | |  | |
| **Q6.5 Edit myscipt.js to add your details.** | | | | | | | | |
| Edit the required JS file and create a merge request.  Merge request number: | | | | | | | | |
| Branch name |  | | | Commit Number: | | |  | |
| **Q6.6 Obtain final sign off.** | | | | | | | | |
| <Project has been signed off and approval given>  Yes / no | | | | | | | | |
| **Assessor Name** | | *Amberle Seidl* | **Assessor Signature** | | *Signature* | **Date** | |  |