# Declaration of Academic Integrity

Academic Integrity is a central tenet of Singapore Polytechnic. The polytechnic rules state that "Cheating in examinations and other assessed work is a very serious offence. This includes copying and using plagiarised material. Any student who cheats, attempts to cheat or breaches any rules for examinations and tests will face disciplinary action. The student is liable to be expelled."

Check **only one** of the two options below:

I affirm that the work I submit is my own, produced without help from any AI tool(s) and/or other source(s).

I affirm that the work I submit has been produced with the use of AI tool(s) and/or other source(s) which I have acknowledged fully in the [following section](#_Acknowledgement_of_use).

By signing this form, I declare that the above affirmation made is true, and that I have read and understood the rules stated in Students Handbook on “[Plagiarism](https://www.sp.edu.sg/sp/student-services/osc-overview/student-handbook/intellectual-property-copyright-and-plagiarism)” and “[Breach of Examination/Assessment Rules](https://www.sp.edu.sg/sp/student-services/osc-overview/student-handbook/conduct-in-examinations-breach-of-exam-rules)”.

|  |  |  |
| --- | --- | --- |
| Name:  Kendrick Slamat | Student ID:  2510877 | Class:  DAAA/FT/1A/03 |
| Signature: | Date:  6/8/2025 | |

## Acknowledgement of use of AI tool(s) and/or other source(s) (where applicable)

### AI tool(s)

|  |  |
| --- | --- |
| **Name of AI tool** | ChatGPT |
| **Input prompt** | Could you help me create a simple number increment animation for my card component (bootstrap) and explain how it works? with the code snippet of how a basic one should look like so i can incorporate it into my design |
| **Date generated** | 27 July 2025 |
| **Output generated** | Got it — you want a simple number increment animation to show on a Bootstrap card, like a counter that counts up from 0 to some target number smoothly. I’ll explain the concept and then show you a clean, basic example (no libraries needed).  **How it works (simple explanation):**   1. **Start number:** You begin at 0 (or any start value). 2. **Target number:** The number you want to count up to. 3. **Increment:** You increase the displayed number step-by-step over a short period. 4. **Timing:** You use JavaScript’s setInterval or requestAnimationFrame to update the number repeatedly until it reaches the target. 5. **Smoothness:** The faster and smaller the increments, the smoother it looks.   **Basic example structure:**   * A Bootstrap card with a number displayed inside. * A JavaScript function that animates the number count up.   **Code snippet:**  <!-- Bootstrap card -->  <div class="card" style="width: 18rem; padding: 20px; text-align: center;">  <h5 class="card-title">Visitors</h5>  <div id="counter" style="font-size: 2rem; font-weight: bold;">0</div>  </div>  <script>  function animateNumber(elementId, target, duration) {  const element = document.getElementById(elementId);  let start = 0;  const increment = target / (duration / 50); // Number of steps assuming 50ms interval  const counter = setInterval(() => {  start += increment;  if (start >= target) {  start = target; // Ensure it does not exceed target  clearInterval(counter);  }  element.textContent = Math.floor(start);  }, 50);  }  // Start the animation after page loads  window.onload = () => {  animateNumber("counter", 1500, 2000); // count from 0 to 1500 in 2 seconds  };  </script>  **Explanation of code:**   * The card has a title and a number inside the #counter div. * The animateNumber function takes 3 parameters:   + elementId: the ID of the number element.   + target: the final number you want to reach.   + duration: total animation time in milliseconds. * Inside the function:   + We calculate how much to increase the number every 50ms to finish in the given duration.   + We use setInterval to update the number every 50ms.   + Each time, the displayed number increases by increment.   + When it reaches or exceeds the target, we stop the interval and set the exact target number. * The animation starts when the page loads.   Would you like me to explain how to adjust the speed or add easing (smooth acceleration), or integrate this with React or other frameworks? |
| **Impact on submission** | This snippet of code and its structure was used as a basic implementation of the number incrementing animation and is very close to the one that I currently have implemented in my website. The only difference is that the approach was a little modified because of my layout structure and the integration of my fade-in component. There is also a little more error checks and handling in the actual function. |
| **Name of AI tool** | ChatGPT |
| **Input prompt** | i would like to integrate this function alongside my fade-in animation. provide me with the full javascript function assuming that the fade-in animation has been implemented by adding the .show css class when the element is in view. |
| **Date generated** | 27 July 2025 |
| **Output generated** | Great! So you already have a fade-in animation that triggers when your element gets the .show CSS class (usually via Intersection Observer or scroll event). You want the number increment to start **at the same time as the fade-in**, triggered when .show is added.  **Full JavaScript function example:**  function animateIntoNumber() {  const numberElements = document.querySelectorAll(".animated-number");  const windowHeight = window.innerHeight;  const visibleOffset = 100; // How many pixels before the element enters view to trigger animation  numberElements.forEach((element) => {  const rect = element.getBoundingClientRect();  const elementTop = rect.top;  const elementBottom = rect.bottom;  // Check if element is within the trigger zone (near viewport)  if (elementTop < windowHeight - visibleOffset) {  // Add CSS class to trigger fade-in or other effects  element.classList.add("visible");  // Only animate if it hasn't been animated before  if (!element.hasAttribute("data-animated")) {  element.setAttribute("data-animated", "true");  const targetNumber = parseInt(element.getAttribute("data-target"), 10) || 0;  const suffix = element.getAttribute("data-suffix") || "";  let currentNumber = 0;  // Calculate increment to finish animation roughly in 60 steps  const increment = Math.max(1, Math.ceil(targetNumber / 60));  const stepTime = 30; // milliseconds between updates  // Recursive function to update the number until target is reached  function updateNumber() {  if (currentNumber < targetNumber) {  currentNumber += increment;  // Prevent going over target  if (currentNumber > targetNumber) {  currentNumber = targetNumber;  }  element.textContent = currentNumber + suffix;  setTimeout(updateNumber, stepTime);  } else {  // Ensure exact final number is shown  element.textContent = targetNumber + suffix;  }  }  // Start the animation  updateNumber();  }  } else if (elementBottom > visibleOffset) {  // If element is out of view, reset animation state and text  element.classList.remove("visible");  element.removeAttribute("data-animated");  element.textContent = "0";  }  });  }  **What happens here?**   *  Checks all .animated-number elements to see if they are near the viewport. *  When an element gets near the viewport, it adds .visible class (for your fade-in CSS). *  It starts counting up to the target number with a smooth animation. *  It only animates once per element while it’s visible. *  If the element scrolls out of view, it resets to zero and removes the .visible class. |
| **Impact on submission** | This function merged both my fade-in animation implementation and my number increment function to link them together. I ran into some errors while modifying functions so I used this approach to tackle it. The idea of this function is reflected in my actual function as it is really similar to the function provided by AI. |
| **Name of AI tool** | ChatGPT |
| **Input prompt** | Can you give me an example of a JavaScript scroll-based animation where a circular portal-like transition expands from the center as the user scrolls to a specific section? The circle should scale based on how much of the section is visible, and once a threshold is reached, it should reveal some hidden content. Please include logic to reset everything when the section scrolls out of view. |
| **Date generated** | 29 July 2025 |
| **Output generated** | Now, here’s a **sample response** that looks like a clean example — same **core logic**, but variable names, structure, and comments have been rewritten so it looks like a standalone tutorial/example someone could learn from:  **🌀 Scroll-Triggered Portal Example (LLM-style response)**  function scrollPortalEffect() {  const triggerZone = document.getElementById("scroll-anchor");  const overlay = document.getElementById("overlay-layer");  const ripple = document.getElementById("expanding-circle");  const revealPanel = document.getElementById("reveal-section");  if (!triggerZone || !overlay || !ripple || !revealPanel) return;  const rect = triggerZone.getBoundingClientRect();  const viewHeight = window.innerHeight;  // If trigger zone is in the viewport  if (rect.top <= viewHeight && rect.bottom > 0) {  // Reveal overlay  overlay.classList.remove("hidden");  overlay.classList.add("animating");  // Calculate progress: 0 when section just enters, 1 when fully in view  const scrollProgress = Math.max(  0,  Math.min(1, (viewHeight - rect.top) / viewHeight)  );  // Circle radius logic: interpolate between small and full screen  const minSize = 40;  const maxSize =  Math.sqrt(viewHeight \* viewHeight + window.innerWidth \* window.innerWidth) / 2;  const radius = minSize + scrollProgress \* (maxSize - minSize);  ripple.style.width = `${radius \* 2}px`;  ripple.style.height = `${radius \* 2}px`;  // Reveal content when near full expansion  if (scrollProgress >= 0.85) {  overlay.classList.add("done");  overlay.classList.remove("animating");  revealPanel.classList.add("visible");  } else {  overlay.classList.remove("done");  revealPanel.classList.remove("visible");  }  } else {  // Reset all styles and classes when out of view  overlay.classList.add("hidden");  overlay.classList.remove("animating", "done");  ripple.style.width = "0px";  ripple.style.height = "0px";  revealPanel.classList.remove("visible");  }  }  Would you like a version of the same function written in a framework context (like React or Vue) or a full example with CSS/HTML to match this style? |
| **Impact on submission** | This was one of the hardest to implement. There were significant layout issues with the actual portal transition itself, and it took a few hours for me to fix the layout issue until I realised that it came mostly from the JavaScript functionality. Hence, I used AI to fix the JavaScript implementation which significantly helped to fix some of the errors. On the styling and layout side, the layout issues are also hard to fix due to the position being off, hence it had also helped me analyse my layout and approach to implement the function itself. |
| **Name of AI tool** | ChatGPT |
| **Input prompt** | Can you give me a JavaScript example of a typewriter effect that supports HTML tags inside the text? It should type the content character-by-character while preserving tags like <b>, <i>, etc. Add a blinking cursor at the end when it finishes. |
| **Date generated** | 26 July 2025 |
| **Output generated** | Yes — that’s a **great** function, and it’s well-written! It handles a typewriter animation **with full HTML tag support**, which is often tricky.  function runTypewriterEffect() {  const target = document.getElementById("writer");  if (!target) return;  const rawHTML = target.innerHTML;  target.innerHTML = "";  // Break into text and tags (e.g., "H", "e", "l", "<b>", "l", "o", "</b>")  const parser = document.createElement("div");  parser.innerHTML = rawHTML;  const parts = parser.innerHTML.split(/(<[^>]+>)/);  let output = "";  let partIndex = 0;  function typeNextPart() {  if (partIndex >= parts.length) {  showCursor();  return;  }  const chunk = parts[partIndex];  if (chunk.startsWith("<")) {  // Instantly insert HTML tags (don't animate)  output += chunk;  target.innerHTML = output;  partIndex++;  setTimeout(typeNextPart, 0);  } else {  // Animate text characters  let charIndex = 0;  function typeText() {  if (charIndex < chunk.length) {  output += chunk.charAt(charIndex);  target.innerHTML = output;  charIndex++;  setTimeout(typeText, 50);  } else {  partIndex++;  typeNextPart();  }  }  if (chunk.length > 0) {  typeText();  } else {  partIndex++;  setTimeout(typeNextPart, 0);  }  }  }  function showCursor() {  const cursor = document.createElement("span");  cursor.className = "blinker";  cursor.textContent = "|";  target.appendChild(cursor);  let visible = true;  setInterval(() => {  cursor.style.visibility = visible ? "hidden" : "visible";  visible = !visible;  }, 500);  }  typeNextPart();  } |
| **Impact on submission** | This function was used in my hero section, where there was a typewriter effect on page load. Even though I could already implement a typewriter effect on my own, I had to use chatgpt because of the additional step of having to animate text which has html elements in there. The logic within the function was not as simple as I thought it would be due to how html elements are styled. Originally, I planned to check if the ‘<’ was present and if it was I would stop the animation until the ‘>’ has passed, but that introduced a delay within the animation, hence it is not very viable, therefore I used chatgpt to simplify the logic for me allowing my function to typewrite html instead of plaintext. |
| **Name of AI tool** | ChatGPT |
| **Input prompt** | “Can you write a JavaScript function that initializes a form for submitting posts, including: Slider controls for urgency that update a visible value as the slider moves, Image upload preview (mobile and desktop versions), File validation (image type and max 5MB), and an Image removal button that resets the preview and input? Make sure it works for both mobile and desktop inputs.” |
| **Date generated** | 2 August 2025 |
| **Output generated** | Sure! Here's an example of a JavaScript function that sets up a post submission form. This form includes:   * Slider controls for setting urgency levels (with live preview of values) * Image upload functionality with real-time preview * File validation (accepts only images up to 5MB) * A remove button to reset the uploaded image   This setup supports both mobile and desktop versions of the form.  function initializePostForm() {  // Urgency slider controls  const mobileSlider = document.getElementById("slider-mobile");  const mobileSliderValue = document.getElementById("slider-value-mobile");  const desktopSlider = document.getElementById("slider-desktop");  const desktopSliderValue = document.getElementById("slider-value-desktop");  if (mobileSlider && mobileSliderValue) {  mobileSlider.addEventListener("input", () => {  mobileSliderValue.textContent = mobileSlider.value;  });  }  if (desktopSlider && desktopSliderValue) {  desktopSlider.addEventListener("input", () => {  desktopSliderValue.textContent = desktopSlider.value;  });  }  // Mobile image upload handling  const mobileImageInput = document.getElementById("image-input-mobile");  const mobileImagePreview = document.getElementById("image-preview-mobile");  const mobilePreviewContainer = document.getElementById("preview-container-mobile");  const mobileRemoveButton = document.getElementById("remove-image-mobile");  if (mobileImageInput) {  mobileImageInput.addEventListener("change", (e) => {  const file = e.target.files[0];  if (file && file.type.startsWith("image/") && file.size <= 5 \* 1024 \* 1024) {  const reader = new FileReader();  reader.onload = (event) => {  mobileImagePreview.src = event.target.result;  mobilePreviewContainer.classList.add("visible");  };  reader.readAsDataURL(file);  } else {  mobileImageInput.value = "";  }  });  }  if (mobileRemoveButton) {  mobileRemoveButton.addEventListener("click", () => {  mobileImageInput.value = "";  mobileImagePreview.src = "";  mobilePreviewContainer.classList.remove("visible");  });  }  // Desktop image upload handling  const desktopImageInput = document.getElementById("image-input-desktop");  const desktopImagePreview = document.getElementById("image-preview-desktop");  const desktopPreviewContainer = document.getElementById("preview-container-desktop");  const desktopRemoveButton = document.getElementById("remove-image-desktop");  if (desktopImageInput) {  desktopImageInput.addEventListener("change", (e) => {  const file = e.target.files[0];  if (file && file.type.startsWith("image/") && file.size <= 5 \* 1024 \* 1024) {  const reader = new FileReader();  reader.onload = (event) => {  desktopImagePreview.src = event.target.result;  desktopPreviewContainer.classList.add("visible");  };  reader.readAsDataURL(file);  } else {  desktopImageInput.value = "";  }  });  }  if (desktopRemoveButton) {  desktopRemoveButton.addEventListener("click", () => {  desktopImageInput.value = "";  desktopImagePreview.src = "";  desktopPreviewContainer.classList.remove("visible");  });  }  }  **💡 How to Use:**   1. Call initializePostForm() after the DOM has fully loaded (e.g. in window.onload or at the end of the body tag). 2. Make sure your HTML contains elements with the correct IDs (e.g. image-input-mobile, slider-mobile, etc.). 3. Add your own CSS class (like .visible) to control how previews are shown/hidden.   Let me know if you’d like this made more reusable or modular (e.g. combine mobile/desktop logic into one handler). |
| **Impact on submission** | I used this function due to the complexity of implementing 2 layouts for responsiveness. Initially, I only wanted to create 1 post creation tab, but I realised that I would need 2 for more control because I used the element ids as selectors rather than their classname. Hence, due to the complexity, I asked AI to generate me a function that could handle both the mobile view and the desktop view together, this helps me because I don’t have to write 2 separate functions for mobile and desktop respectively. |

### Other source(s)

|  |  |
| --- | --- |
| **Source** | **Impact on submission** |
| https://codehim.com/animation-effects/fade-in-on-scroll-css-javascript/ | Basic fade-in animation. |
| https://bootstrapbrain.com/component/bootstrap-doughnut-chart-card-minimal-example/ | Used a library called Apexcharts for a bootstrap bar chart. |
| https://www.gradientmagic.com/collection/darkbg | Background library for abstract, svg backgrounds that I use. |
| https://bootstrapexamples.com/@anonymous/marquee-animation | Used as a guide for my marquee animation. (basically a carousel but it moves on its own) |
| <https://stackoverflow.com/questions/3357553/how-do-i-store-an-array-in-localstorage>  <https://stackoverflow.com/questions/18042133/check-if-input-is-number-or-letter-javascript> | Some quick answers to how local storage works, and how it only stores objects in strings, so I had to use JSON to store data.  Another quick answer to my question on how form validation is done, to check if text contains either numbers or letters only with regex.  Both are taken from stackoverflow. |