**Question 1: How do you determine when this page is done?** (Refers to Step 1: Visit the bol homepage)

**Answer**: I set the "End Event" of **Step 16** ("Navigate to Bol.com Homepage") to "Action completed" in TruClient, which waits for the initial page load. Since bol.com has dynamic elements, I added **Step 18** ("Wait for Search Textbox") to wait for the search bar using its **name='searchtext'** attribute, with a 10-second timeout. This ensures the page is fully interactive, mimicking a real user’s experience.

**Question 2: How did you do this and why?** (Refers to Steps 3 and 4: Store the number of articles and turn it into a JavaScript variable)

**Answer**: For Step 3, I used **Step 25** ("Retrieve Total Number of Articles") to extract the article count (e.g., "4,224 resultaten") with JavaScript, using **object.innerText**, removing non-digits with replace**(/[^\d]/g, ''),** and converting it to an integer with **parseInt().** I stored it in **window.numberOfArticles**. For **Step 4**, I added **Step 26** ("Set Global Number of Articles") to set **window.numberOfArticlesGlobal** to this value and saved it to the **numArticles** parameter for later use. I chose this approach because the element’s text is a reliable source, and splitting the steps improves clarity while ensuring the variable is globally accessible.

**Question 3: What object recognition method did you use to click the object and what was your reasoning behind this?** (Refers to Step 4: Click the top result)

**Answer**: In Step **28.1** ("Click Top Result"), I used the XPath **//li[contains(@class, 'product-item--row')][1]//a[@data-test='product-title']** to identify the top result. This targets the first **<li> element with class product-item--row** (indicating a product in the list) and selects the **<a>** element with **data-test='product-title'**, which is the clickable title link, using the data-test attribute for reliability since it’s less likely to change compared to text or other attributes. I added Step **28.2** ‘Wait for Object’ step to ensure the element is visible before clicking, improving reliability.

**Question 4: How did you decide to solve this?** (Refers to Step 6: Click a random result from the first page)

**Answer**: For **Step 6**, I used a C function (from **Step 8**) to generate a random index, but I’ll explain the process here. In Step **28.5**, I selected all product links on the page with **document.querySelectorAll('li.product-item--row a[data-test="product-title"]')** and set their count (e.g., 24) to the **articleCount** parameter. In **Step 28.6**, I called the C function **getRandomArticleIndex** to generate a random index, which I retrieved in **Step** **28.7** to click the corresponding link. I chose this method because the **data-test** selector is reliable, and using a C function fulfills the assignment’s requirement while keeping the random selection logic separate and reusable.

**Question 5: Why did you set it up like this?** (Refers to Step 7: Handle the situation when no article is found)

**Answer**: In **Step 27**, I checked **window.numberOfArticles** and set **window.noArticlesFound to true** if no articles were found. **Step 28** uses an **If-Else block: if window.noArticlesFound is false**, it proceeds with clicking results; otherwise, it skips those steps. I set it up this way to prevent errors from interacting with non-existent results, keep the flow clear in the TruClient sidebar, and I test it with "@!xyz123abc" to confirm the logic works.

**Answer to Question 6: Can you explain your thought process on how you did this?(** (Refers to Step 8: Create a C function for a random article number)**)**

**Answer**: For **Step 8**, I created **getRandomArticleIndex** in **C-functions.c** to generate a random index from the **articleCount** parameter and save it to **randomIndex**. I used **lr\_eval\_string** to retrieve **articleCount**, converted it to an integer with **atoi**, and checked if it’s valid—returning -1 and logging an error if it’s 0 or less. I seeded the random number generator with **srand** using the current time and generated the index with **rand() % count**. The index is converted to a string with sprintf and saved to **randomIndex** using **lr\_save\_string**. Since TruClient can’t call C directly, I set articleCount in Step 28.5, called the function in Step 28.6 using an "Evaluate C" step, and retrieved randomIndex in Step 28.7 to click the article.