Exercise 01\_05\_01 – Step 1

In this Exercise, we will apply DOM skills to build an app that displays a photo gallery of images and allows users to navigate through the images, as well as to view a larger version of each image.

• Open photos.html; fill in the appropriate documentation in the top comments. Scroll through the document to get familiarized with the content. Below the <header> element, the <article> element contains a <div>, three <figure> elements, and two more <div> elements.

• The first two <div> elements will be the back and forward buttons for navigating through the gallery.

• The final <div> will be the button that lets the user toggle between viewing 3 and 5 images at a time.

• Each <figure> will hold an <img> element which will display one of five images.

• Notice that no <img> has a src attribute; we will manipulate that dynamically depending upon the user actions.

• Open photos.js; fill in the appropriate documentation in the top comments. Scroll through the document to get familiarized with the content.

• Notice the “use strict” statement, we are using Best Practices.

• The script contains a photoOrder variable, to track the order of photos as users move through the gallery.

• There are rightArrow() and leftArrow() functions to shift the images to the left or right.

• There is an empty zoomFig() function, which will be used to enlarge images.

• There is a setUpPage() function and event listener code to run it when the page loads.

• There are references to functions populateFigures() and createEventListeners(), which we will create.

• Opens photos.html in the browser. The navigation buttons are displayed, but the image area is empty. Remember, there are no src values in the <img> elements yet. Check the console for any bugs. Notice that createEventListeners() is not defined. We will take care of that shortly.

• Return to photos.js. Before the setUpPage() function, define the createEventListeners() function. Be sure to comment your work with documentation that shows you understand the code and explains what it does. Code as follows:  
function createEventListeners() {  
   
}  
Save and run in the browser. The previous error should be gone in the Console, but we have a new one for something not yet built. We will take care of that later.

• Within the code block for the new function, let’s use a method of the Document object to get a reference to the left arrow button:  
 var leftArrow = document.getElementById("leftarrow");  
 alert(leftArrow);  
Notice we put in code to test right away. Save and run in the browser.

• Now we can add the pre-built event listener for the first arrow. Remove the alert() and place the following code next in the function:  
 if (leftarrow.addEventListener) {  
 leftarrow.addEventListener("click", leftArrow, false);  
 } else if (leftarrow.attachEvent) {  
 leftarrow.attachEvent("onclick", leftArrow);  
 }

• We are going to copy the left arrow code and use it to take care of the right arrow. But the professional thing to do is to test now to make sure we have no bugs that we would end up propagating to the right arrow. How about we use the alert() technique again as follows:  
function leftArrow() {  
 alert("left arrow clicked");

• Now let’s copy all of the left arrow code directly below itself. We can change all references in the copy to right arrow. We should also get rid of the left arrow alert, and move it to the rightArrow() event handler for another test. If everything is good, remove the debug code and save your work.

Exercise 01\_05\_01 – Step 2

• Return to photos.js. Below the code we already have within the code block of createEventListeners(), let’s use a different method of the Document object to get a reference to the second element of an array of the <img> elements:  
 var mainFig = document.getElementsByTagName("img")[1];  
 alert(mainFig);  
Notice we put in code to test right away. Save and run in the browser.

• Let’s add the pre-built, but empty, event listener to the center image of the photo gallery. Let’s use the following code:  
 if (mainFig.addEventListener) {  
 mainFig.addEventListener("click", zoomFig, false);  
 }  
 else if (mainFig.attachEvent) {  
 mainFig.attachEvent("onclick", zoomFig);  
 }

• We should set up an immediate test with the following:  
function zoomFig() {  
 alert("center image clicked");  
}

• Save your work, test it, and remove the debug code.

Exercise 01\_05\_01 – Step 3

• Return to photos.js. Below the global photoOrder declaration define the populateFigures() function and some variables as follows:  
function populateFigures() {  
 var filename;  
 var currentFig;  
}  
Save and test in the browser; the Console should show the undefined populateFigures() error has gone away.

• Below the variable declarations, let’s set up some code to provide the filename values for the src attributes of the three <img> elements. We can do this with a for loop that will iterate through the photoOrder array and use the middle three elements. Make sure to set up an alert() test:  
 for (var i = 1; i < 4; i++) {  
 filename = "images/IMG\_0" + photoOrder[i] + "sm.jpg";  
 currentFig = document.getElementsByTagName("img")[i - 1];  
 currentFig.src = filename;  
 alert(currentFig.src);  
 }  
The filenames should be correct and the <img> elements should populate. If not, read the alerts carefully to spot whatever bugs there may be. If working, remove the debug code.

• Run the application in the browser again. Check the Console for errors. Test the left and right arrows for functionality.

Exercise 01\_05\_01 – Step 4

• Return to photos.js. Let’s create the code that will turn the Photo Gallery into a five image gallery when the Show more images button is clicked. Below the leftArrow() declaration, define the previewFive() event handler function, with a test, as follows:  
function previewFive() {  
 alert("previewFive() event handler");  
}

•

• Wire up the event handler for the Show more images button. We will use a querySelector() method to get a reference to the button. Scroll down to the createEventListeners() function and add the following code:  
 var showAllButton = document.querySelector("#fiveButton p");  
 if (showAllButton.addEventListener) {  
 showAllButton.addEventListener("click", previewFive, false);  
 }   
 else if (showAllButton.attachEvent) {  
 showAllButton.attachEvent("onclick", previewFive);  
 }  
Save this and try it in the browser. A click on the Show more images button should trigger the alert() debug code. If everything is good, remove the debug code.

• Let’s create a new node for the DOM tree. It will be a <figure> element that will hold the fifth <img> element. We will do this in the previewFive() function:  
 var lastFigure = document.createElement("figure");  
Do a quick save and browser & Console test just to be sure we did not introduce any bugs.

• Now we need to add an id attribute and some style properties to the new element as follows:  
 lastFigure.id = "fig5";  
 lastFigure.style.zIndex = "5";  
 lastFigure.style.position = "absolute";  
 lastFigure.style.right = "45px";  
 lastFigure.style.top = "67px";  
Do a quick save and browser & Console test just to be sure we did not introduce any bugs.

• Let’s create another new node for the DOM tree. It will be the <img> element for the <figure> element. It should go below the previous code:  
 var lastImage = document.createElement("img");  
Do a quick save and browser & Console test just to be sure we did not introduce any bugs.

•

• Now we need to add a couple of HTML attributes to the new element as follows:  
 lastImage.width = "240";  
 lastImage.height = "135";  
Do a quick save and browser & Console test just to be sure we did not introduce any bugs.

• In Chrome, go to the Elements tab, which will show the DOM tree of the current Web page. Expand the elements until you see the <figure> elements, which will initially be fig2 through fig4. then expand one of them to see its child <img> element. Click the Show more images button. No new <figure> element appears. That is because we have not yet attached it to the DOM tree.

•

Exercise 01\_05\_01 – Step 5

• Return to photos.js. We first get a reference to the first and only <article> element. This will be the DOM tree parent node of the new elements we have created. Return to the end of previewFive():  
 var articleElem =   
 document.getElementsByTagName("article")[0];  
 alert(articleElem);  
Save and run in the browser, making sure that we have the element.

• Now let’s remove the debug and attach the new <img> element to the new <figure> element, creating a document fragment:  
 lastFigure.appendChild(lastImage);

• To attach the document fragment to the <article>, we use the following:  
 articleElem.appendChild(lastFigure);

• Save changes and run in the browser. In the Developer Tools, open the Elements tab to examine the DOM tree and expand the elements. Click the Show more images button. If everything is working, you should see the new fig5 added to the tree.

Exercise 01\_05\_01 – Step 6

• Return to photos.js. We will duplicate the new nodes we added to the DOM tree by using the cloneNode() method. We will make sure to clone the entire document fragment of parent and child nodes. At the bottom of previewFive(), let’s add the following new code:  
 var firstFigure = lastFigure.cloneNode(true);

• We need to change its id attribute and a couple of its style attribute properties:  
 firstFigure.id = "fig1";  
 firstFigure.style.right = "";  
 firstFigure.style.left = "45px"

• To attach the document fragment to the <article>, we use the following:  
 articleElem.appendChild(firstFigure);

• Save changes and run in the browser. In the Developer Tools, open the Elements tab to examine the DOM tree and expand the elements. Click the Show more images button. If everything is working, you should see the new fig1 added to the tree.

• You can see that the new elements in the gallery have no images. If you expand any of the original figure elements, you will see that we have set their src attributes. The two new elements do not have values for src. We need to take care of that at the end of previewFive():  
 document.getElementsByTagName("img")[3].src = "images/IMG\_0" + photoOrder[4] + "sm.jpg";  
 document.getElementsByTagName("img")[4].src = "images/IMG\_0" + photoOrder[0] + "sm.jpg";  
Save changes and run in the browser and click the Show more images button. You can now see the images and the Developer tools show the src attributes are now populated.

• If we try the arrows buttons, we can see that the Photo Gallery is not yet operating correctly. We are going to have to do some surgery on the existing code. We need another global variable after the photoOrder variable. It will control our figure count:  
var figureCount = 3;

• Now we have to modify the populateFigures() function to get the carousel operating correctly. The for loop logic needs major work. There are cleaner ways to do the following, but we will just use brute force for now. Enclose the for loop in an if statement to cover 3 images as follows:  
 if (figureCount === 3) {  
 for (var i = 1; i < 4; i++) {  
 filename = "images/IMG\_0" + photoOrder[i] + "sm.jpg";  
 currentFig = document.getElementsByTagName("img")[i - 1];  
 currentFig.src = filename;  
 }  
 }

• Below the if statement, add an else statement with another for loop to cover the case of 5 images:  
 else {  
 for (var i = 0; i < 5; i++) {  
 filename = "images/IMG\_0" + photoOrder[i] + "sm.jpg";  
 currentFig = document.getElementsByTagName("img")[i];  
 currentFig.src = filename;  
 }  
 }  
A long way to go without a test. But still one more thing to do before we can test. If something messed up we are going to have a tough piece of debug, but we have the techniques. For now, let’s finish by scrolling down to the previewFive() function. Just before the closing brace, reset the figureCount global variable:  
 figureCount = 5;

• Now let’s give it a browser test. Things look good when we and click the Show more images button. All the images show up. Uh-oh, click the right arrow button and the images don’t change. Click it a second time and it works. The problem is that the problem is that the new <figure> elements are in the wrong positions in the DOM tree. Look at their order in the browser Developer Tools.

Exercise 01\_05\_01 – Step 7

• Return to photos.js, where we need to change the order of the nodes in the DOM tree. Within previewFive(), locate the statement that adds the last figure to the DOM tree and comment it out:  
 // articleElem.appendChild(lastFigure);

• Right after it, let’s change the code to:  
 articleElem.insertBefore(lastFigure,  
 document.getElementById("rightarrow"));

• Locate the statement that adds the first figure to the DOM tree and comment it out:  
 // articleElem.appendChild(firstFigure);

• Right after it, let’s change the code to:  
 articleElem.insertBefore(firstFigure,  
 document.getElementById("fig2"));

• Update the index numbers in the next two statements:  
 document.getElementsByTagName("img")[0].src = "images/IMG\_0"   
 + photoOrder[0] + "sm.jpg";  
 document.getElementsByTagName("img")[4].src = "images/IMG\_0"   
 + photoOrder[4] + "sm.jpg";

• Save changes and run in the browser. Click the Show more images button. The right arrow should now be working on the first click. In the Developer Tools, open the Elements tab to examine the DOM tree and expand the elements. The order of the DOM tree should now be correct.

• Let’s give something a try. Cycle the browser and click the Show more images button. Click it again and we are getting some bad behavior. Keep on clicking it, it gets worse. What we need to do is change the behavior of the button

Exercise 01\_05\_01 – Step 8

• To construct the code to return to the original three-image layout, return to photos.js. first we need to change the button text. Within previewFive(), add the following code to the bottom to get a reference to the button:  
 var numberButton = document.querySelector("#fiveButton p");  
 numberButton.innerHTML = "Show fewer images";  
Let’s give that a browser test.

• If the button is displaying correctly, we need to fix the behavior when it is clicked. For that, we will have to remove the event listener that adds images. Then we will set a new event listener to remove images. Below the above code, the following should accomplish that goal:  
 if (numberButton.addEventListener) {  
 numberButton.removeEventListener("click", previewFive, false);  
 numberButton.addEventListener("click", previewThree, false);  
 }   
 else if (numberButton.attachEvent) {  
 numberButton.detachEvent("onclick", previewFive);  
 numberButton.attachEvent("onclick", previewThree);  
 }  
Test that in the browser and look in the console to see if we have introduced any bugs. We can see that the previewThree() event listener is undefined.

• Below previewFive(), let’s define previewThree() with just a simple alert() test:  
function previewThree() {  
 alert("previewThree() called");  
}  
Test that in the browser. We should get our alert() message and the Console error should be gone.

• Let’s build previewThree() out with the code to remove the fig1 and fig5 elements from the DOM Tree:  
 var articleElem = document.getElementsByTagName("article")[0];  
 var numberButton = document.querySelector(  
 "#fiveButton p");  
 articleElem.removeChild(document.getElementById("fig1"));  
 articleElem.removeChild(document.getElementById("fig5"));  
Give that a browser test. The gallery should be back to three images. There should be no Console errors, and the DOM tree should be modified.

• If we start clicking the right arrow, we see that we have some more fix up to do to make this work properly. Let’s add the cleanup code to the bottom of previewThree():  
 figureCount = 3;  
 numberButton.innerHTML = "Show more images";  
 if (numberButton.addEventListener) {  
 numberButton.removeEventListener("click", previewthree, false);  
 numberButton.addEventListener("click", previewFive, false);  
 }   
 else if (numberButton.attachEvent) {  
 numberButton.detachEvent("onclick", previewThree);  
 numberButton.attachEvent("onclick", previewFive);  
 }  
Let’s give that a test in the browser for functionality, console errors, and DOM tree structure.

Exercise 01\_05\_01 – Step 9

• Open zoom.html; fill in the appropriate documentation in the top comments. Scroll through the document to get familiarized with the content. The <body> section contains only two elements. There is a <figure> element containing an <img> element. There is also a <footer> element.

• Open zoom.js; fill in the appropriate documentation in the top comments. Scroll through the document to get familiarized with the content. The fill contains some global variable declarations and a pageSetup() event listener function.

• Let’s open the zoom.html file in a new window. Return to photos.js, find the zoomFig() function, and add the following code to its code block:  
var zoomWindow = window.open("zoom.html", "zoomwin",   
 "width=960,height=600");  
Save and test in the browser, making sure there are no console errors. Notice that clicking Close Window has no effect, at least not yet.

• Let’s do one final thing. We will add a focus() method to zoomFig(). This will solve the possible problem of the new window getting lost behind the original window with inadvertent mouse clicks. Easy enough, add a final statement to zoomFig():  
 zoomWindow.focus();  
Let’s give it a final test.

Exercise 01\_05\_01 – Step 10

• Let’s return to zoom.js to implement the Close Window feature. Below the pageSetup() function, let’s declare the closeWin() event handler function:  
function closeWin() {  
 window.close();  
}

• Below the new function, let’s build a createEventListener() function:  
function createEventListener() {  
 var closeWindowDiv = document.getElementsByTagName("p")[0];  
 if (closeWindowDiv.addEventListener) {  
 closeWindowDiv.addEventListener("click", closeWin, false);  
 } else if (closeWindowDiv.attachEvent) {  
 closeWindowDiv.attachEvent("onclick", closeWin);  
 }  
}

• Let’s finish this up with a call to createEventListener(). Scroll to the pageSetup() event listener and add the following as its last line:  
 createEventListener();  
Save it and test in the browser, making sure Close Window is working and there are no bugs in the Console.

Exercise 01\_05\_01 – Step 11

• Let’s return to photos.js and implement an interval timer to make the Photo Gallery images advance automatically every 5 seconds. Close Window feature. Add the following code to the end of the global variables section:  
var autoAdvance = setInterval(rightArrow, 5000);  
Save and give this the usual browser and developer tools test and the gallery should be automated.

• The only problem is that there is no way to shut it off. Let’s use a click on the Left Arrow to take care of that. Scroll to the leftArrow() function and add the following code to the top of its code block:  
 clearInterval(autoAdvance);  
Give this a browser test and we should have our shutoff switch.

• Enabling a stop switch for the right arrow is a little more tricky. This is because the original setInterval() call is using the rightArrow() function. The fix to this is to change the call to rightAdvance in the setInterval() call, then rename the rightArrow() function to rightAdvance():  
var autoAdvance = setInterval(rightAdvance, 5000);  
  
function rightAdvance() {  
If we test this, the image gallery is working, but the console tells us that rightArrow() is no longer defined.

• Just above the rightAdvance(), let’s build a pretty simple new rightArrow() function, that just shuts down the interval timer, then calls rightAdvance() to shift one image:  
function rightArrow() {  
 clearInterval(autoAdvance);  
 rightAdvance();  
}

• Let’s give this a full on browser test and we should have interval shutdowns with both arrows.

Exercise 01\_05\_01 – Step 12

• Let’s return to photos.js to implement a final touch to our enlarged window feature. We can use the screen object to center the new window on the screen. Place the following code at the top of the zoomFig() function:  
 var propertyWidth = 960;  
 var propertyHeight =600;  
 var winLeft = ((screen.width - propertyWidth) / 2);  
 var winTop = ((screen.height - propertyHeight) / 2);  
 var winOptions = "width=960,height=600,";  
 winOptions += ",left=" + winLeft;  
 winOptions += ",top=" + winTop;

• Finally, let’s change the window.open() statement as follows:  
 var zoomWindow = window.open("zoom.html", "zoomwin",   
 winOptions);  
Let’s save this and give the site a good browser test. One thing to note, Chrome has a weird bug and ignores the left option if running on a second monitor.