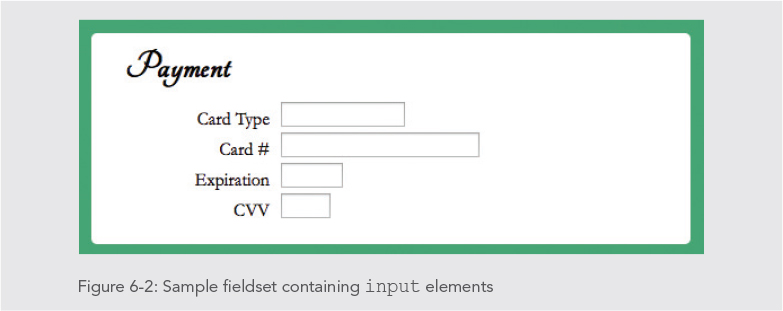
**Exercise 01\_06\_01 – Step 1**

In this Exercise, we will learn how to use JavaScript to make forms easier for users to complete, and to implement custom data validation before submission.

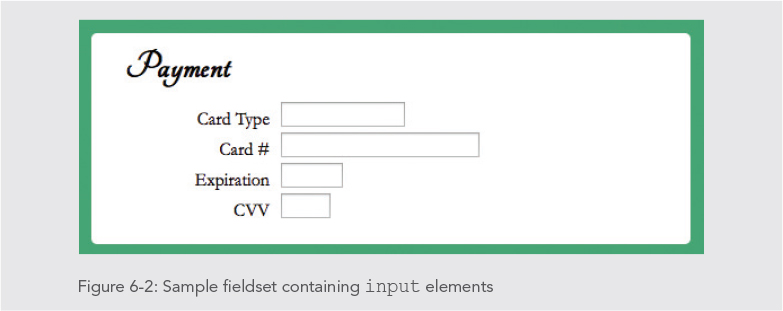
1. Open ***snoot.html***; fill in the appropriate documentation in the top comments. Scroll through the document to get familiarized with the content. The <article> contains a <form>. The <form> contains six <fieldset> elements. Open in the web browser and look at the current state of the Web page.

**Exercise 01\_06\_01 – Step 2**

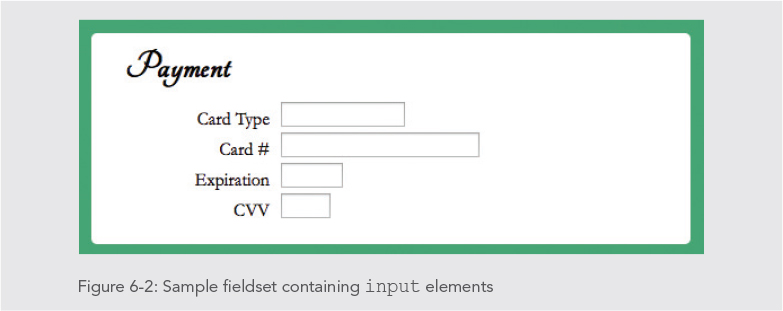
1. Open the ***snoot.html*** file and create a ***<script>*** element for an external script named ***snoot.js***:  
    ***<script type="text/javascript" src="snoot.js"></script>***
2. Create a new JavaScript document named ***snoot.js***. Add all of the appropriate documentation to the top of the file, with your name and the date. Make sure to use ***strict*** mode for JavaScript:  
   ***"use strict";***
3. We can first build a function that will remove any ***default*** values that have been preset in the selection lists. The first thing it should do is get an array of all the selection list elements. Make sure to test it, a good choice would be to ***alert()*** the ***length*** of the array produced:  
   ***function removeSelectDefaults() {  
    var emptyBoxes = document.getElementsByTagName("select");  
    alert("select lists: " + emptyBoxes.length);***  
   ***}***
4. At the bottom of ***snoot.js***, let’s add a function that will attach ***removeSelectDefaults()***as an event handler:  
   ***function createEventListener() {  
    var submitButton = document.getElementById("submit");  
    if (submitButton.addEventListener) {  
    submitButton.addEventListener("click",   
    removeSelectDefaults, false);  
    }   
    else if (submitButton.attachEvent) {  
    submitButton.attachEvent("onclick",   
    removeSelectDefaults);  
    }  
   }***  
   Save the file and run ***snoot.html*** in the browser. Make sure that everything looks good, that there are 7 ***<select>*** elements, and there are no Console errors.
5. Now let’s create a for loop that iterates through the <select> elements and remove the preset default values:  
    ***for (var i = 0; i < emptyBoxes.length; i++) {  
    emptyBoxes[i].selectedIndex = -1;  
    }***  
   Give that a test, and all the <select> elements should now be empty.

**Exercise 01\_06\_01 – Step 3**

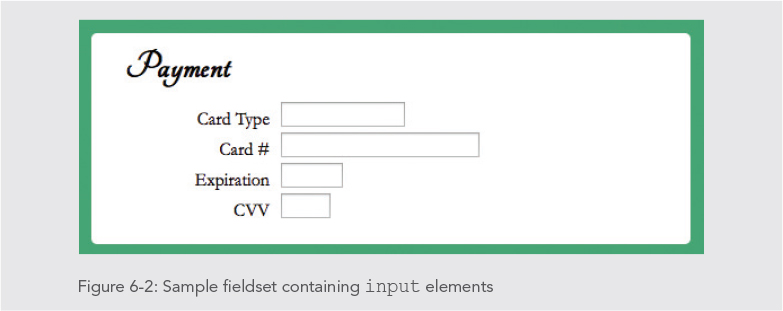
1. Open the ***snoot.html*** file in the browser and go to the ***Delivery Date*** field. Select April from the first selection list. Drop down the second selection list and scroll to the bottom. You will notice that it is possible to select day ***31***, which is incorrect. This is true of all months, and should be fixed.
2. Return to ***snoot.js***, and create 3 global variables below the ***strict*** statement. We will use a new node method to create a ***document*** ***fragment*** for each one, each of which is actually a node:  
   ***var twentyNine = document.createDocumentFragment();  
   var thirty = document.createDocumentFragment();  
   var thirtyOne = document.createDocumentFragment();***
3. First we will change the code that attaches event handlers on page load to call a new event handler name ***setUpPage()***:  
   ***/\* run setUpPage() function when page finishes loading \*/  
   if (window.addEventListener) {  
    window.addEventListener("load", setUpPage, false);  
   }   
   else if (window.attachEvent) {  
    window.attachEvent("onload", setUpPage);  
   }***
4. Below the ***removeSelectDefaults ()*** event handler, let’s start to code out the ***setUpPage()*** event handler function:  
   ***function setUpPage() {  
    removeSelectDefaults();  
   }***  
   give this a browser test and make sure we didn’t introduce any bugs and the defaults from the select lists are still being removed.
5. Below the global variables, we can now build the ***setUpDays()*** function to populate our three new document fragments. We will do this very efficiently with ***cloneNode()*** methods. It will construct us the building blocks for any changes required to the day selection list based on the chosen month:  
   ***function setUpDays() {***  
    ***var dates = document.getElementById("delivDy").  
    getElementsByTagName("option");  
    twentyNine.appendChild(dates[28].cloneNode(true));  
    thirty.appendChild(dates[28].cloneNode(true));  
    thirty.appendChild(dates[29].cloneNode(true));  
    thirtyOne.appendChild(dates[28].cloneNode(true));  
    thirtyOne.appendChild(dates[29].cloneNode(true));  
    thirtyOne.appendChild(dates[30].cloneNode(true));  
   }***  
   Test this in the browser for any possible bugs introduced.
6. Below this function, let’s build the updateDays() function that will properly set up the list of days based on the selected month. We will start off by declaring the variables that we will need:  
   ***function updateDays() {  
    var deliveryDay = document.getElementById("delivDy");  
    var dates = deliveryDay.getElementsByTagName("option");  
    var deliveryMonth = document.getElementById("delivMo");  
    var deliveryYear = document.getElementById("delivYr");  
    var selectedMonth =   
    deliveryMonth.options[deliveryMonth.selectedIndex].value;  
   }***  
   Once again test in the browser for bugs.
7. We are now going to remove the nodes from the options list that hold days ***29***, ***30***, and ***31***, using a ***while*** loop, This will set us up to add to the end of the list just the proper days for the selected month. Let’s add it to the function at the end:  
    ***while (dates[28]) {  
    deliveryDay.removeChild(dates[28]);  
    }***Keep testing for syntax bugs. We have no way yet to test for logic, so that may be difficult, depending on the results. But we know the techniques.
8. Below the ***while*** loop, let’s be sure that we have a valid year in its selection list:  
    ***if (deliveryYear.selectedIndex === -1) {  
    deliveryYear.selectedIndex = 0;  
    }***
9. Now let’s do a quick and dirty test to see if it is a ***leap*** ***year***. There is actually a good formula for this, but for our purposes we will just look for ***2018***, which is the next leap year. If we find it is, we have to add another day to one of our document fragments for February:  
    ***if (selectedMonth === "2" &&   
    deliveryYear.options[deliveryYear.selectedIndex].value === "2020") {  
    deliveryDay.appendChild(twentyNine.cloneNode(true));  
    }  
    else if (selectedMonth === "4" || selectedMonth ==="6" ||  
    selectedMonth === "9" || selectedMonth === "11") {  
    deliveryDay.appendChild(thirty.cloneNode(true));  
    }  
    else if (selectedMonth === "1" || selectedMonth ==="3" ||  
    selectedMonth === "5" || selectedMonth === "7" ||  
    selectedMonth === "8" || selectedMonth === "10" ||  
    selectedMonth === "12") {  
    deliveryDay.appendChild(thirtyOne.cloneNode(true));  
    }***  
   Wow, we really need to test soon, but we are almost there.
10. Now lets add an event listener function, called ***createEventListeners()***, after the ***setUpPage()*** function that adds the other event listeners, starting with the ***deliveryMonth***:  
    function createEventListeners() {  
     ***var deliveryMonth = document.getElementById("delivMo");  
     if (deliveryMonth.addEventListener) {  
     deliveryMonth.addEventListener("change", updateDays, false);  
     }   
     else if (deliveryMonth.attachEvent) {  
     deliveryMonth.attachEvent("onchange", updateDays);  
     }  
    }***
11. Let’s extend it right below to take care of event listeners for deliveryYear:  
     ***var deliveryYear = document.getElementById("delivYr");  
     if (deliveryYear.addEventListener) {  
     deliveryYear.addEventListener("change", updateDays, false);  
     }   
     else if (deliveryYear.attachEvent) {  
     deliveryYear.attachEvent("onchange", updateDays);  
     }***  
    Give it a syntax test. Okay just one more thing to do, at least for this part.
12. Let’s go to the ***setUpPage()*** event handler and add two more function calls:  
     ***setUpDays();  
     createEventListeners();***
13. Now it is time for some serious testing and debugging.

**Exercise 01\_06\_01 – Step 4**

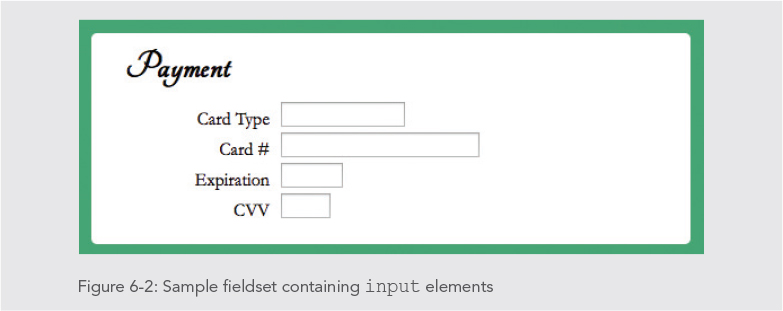
1. Our objective here will be to tie multiple fields together, so the value of one depends on the user action on another. We will create a function, ***autoCheckCustom(),*** to automatically check the ***Custom message*** checkbox if the user enters a custom message in the ***<textarea>***. We will need to create an event listener to ascertain if the data in the <textarea> has changed.Return to ***snoot.js***, and let’s create that code at the bottom of ***createEventListener():***  
    ***var messageBox = document.getElementById("customText");  
    if (messageBox.addEventListener) {  
    messageBox.addEventListener("change", autoCheckCustom, false);  
    }   
    else if (messageBox.attachEvent) {  
    messageBox.attachEvent("onchange", autoCheckCustom)***Give this a browser test for syntax.
2. Now let’s scaffold the function with a simple test for our event listener. just above the ***createEventListeners()*** function:  
   ***function autoCheckCustom() {  
    var messageBox =   
    document.getElementById("customText");  
   }***
3. Let’s build in the code that will check to see if there is custom text in the ***<textarea>***. We can do this by seeing if it is not empty and is also not equal to the placeholder text:  
   ***function autoCheckCustom() {  
    var messageBox =   
    document.getElementById("customText");  
    if (messageBox.value !== "" &&   
    messageBox.value !== messageBox.placeholder) {  
    document.getElementById("custom").checked =   
    "checked";  
    }  
   }*** ***else {  
    document.getElementById("custom").checked = "";  
    }***

**Exercise 01\_06\_01 – Step 5**

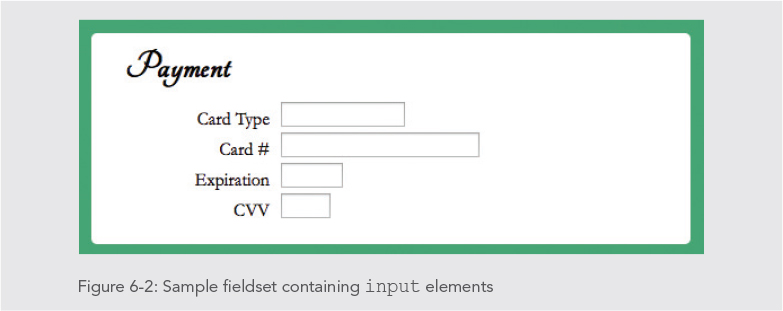
1. Now let’s use the ***same as billing address*** check box to duplicate field data. Return to ***snoot.js***, just above the ***createEventListeners()*** function. We will create a function, ***copyBillingAddress(),*** to accomplish this. This will give us a good opportunity to use the ***querySelectorAll()*** method to capture an entire group of elements:  
   ***function copyBillingAddress() {  
    var billingInputElements =   
    document.querySelectorAll("#billingAddress input");  
    var deliveryInputElements =   
    document.querySelectorAll("#deliveryAddress input");  
   }***
2. Now let’s transfer the data from one set of fields to the other if the box is checked, being careful to note that the Delivery Address group has one extra element, the check box itself. Also notice that we have to take care of the one element that is a ***<select>*** list:  
    ***if (document.getElementById("sameAddr").checked) {  
    for (var i = 0; i < billingInputElements.length; i++) {  
    deliveryInputElements[i +1].value =   
    billingInputElements[i].value;  
    }  
    document.querySelector("#deliveryAddress select").value   
    = document.querySelector("#billingAddress select").value;  
    }***Test for syntax errors with the browser Developer tools.
3. Now we need to create an event listener to call the new function when the user clicks the same as billing address check box. We can add this to the bottom or our ***createEventListeners()*** function:  
    ***var same = document.getElementById("sameAddr");  
    if (same.addEventListener) {  
    same.addEventListener("change", copyBillingAddress,   
    false);  
    }   
    else if (same.attachEvent) {  
    same.attachEvent("onchange", copyBillingAddress);  
    }***  
   Let’s give this a browser test by entering data and checking the ***same as billing address*** check box. If it is working, let’s try unchecking the box. This does not yet work.
4. Back in the ***copyBillingAddress()*** function, let’s add an ***else*** clause to the ***if*** statement, to take care of the check box getting unchecked:  
    ***else {  
    for (var i = 0; i < billingInputElements.length; i++) {  
    deliveryInputElements[i + 1].value = "";  
    }  
    document.querySelector("#deliveryAddress   
    select").selectedIndex = -1;  
    }***  
   Let’s give this a full on browser test.

**Exercise 01\_06\_01 – Step 6**

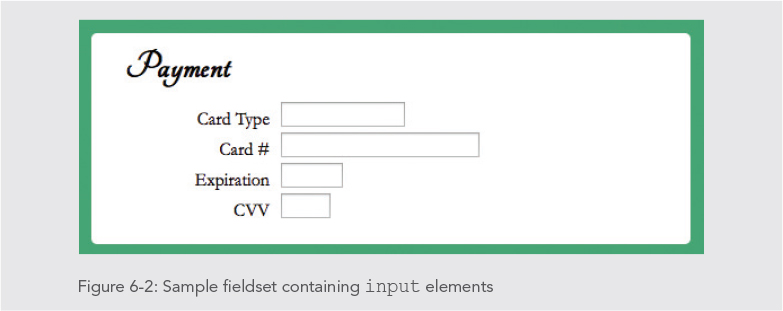
1. Return to ***snoot.html***. First we must disable browser-based validation for the form. We can do that with the form attributes:  
    ***<form action="results.htm" novalidate="novalidate">***
2. Test in the browser by not entering data into any of the fields. Then click the ***Place Order*** button. Notice that the form submits with no errors. The results page lists each field name, but none have a value.

**Exercise 01\_06\_01 – Step 7**

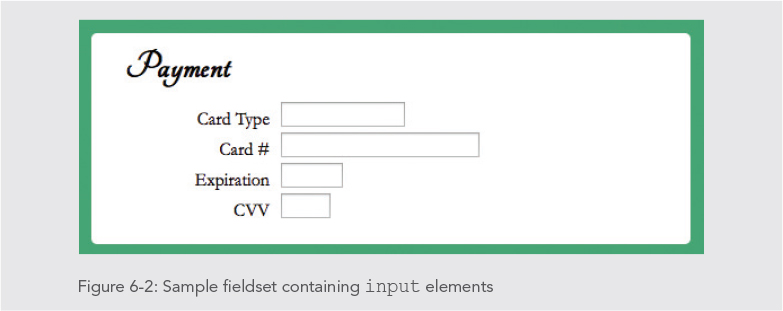
1. Return to ***snoot.js***. For starters, let’s create a global variable after the rest that we will use to keep track of the validity of the form:  
   ***var formValidity = true;***
2. Now let’s add an event listener to ***createEventListeners()*** that will call a handler on ***submit***. The event handler will be ***validateform()***, and we will use it as the main validation function:  
    ***var form = document.getElementsByTagName("form")[0];  
    if (form.addEventListener) {  
    form.addEventListener("submit", validateForm, false);  
    }   
    else if (form.attachEvent) {  
    form.attachEvent("onsubmit", validateForm);  
    }***Give it a browser test for introduced bugs. It should show that we need to define the ***validateForm()*** event handler, so let’s build that out above ***createEventListeners()***, including a test on the submit generated by the ***Place Order*** button:  
   ***function validateForm() {  
    alert("validateForm()");  
   }***It should execute on ***submit***.
3. Now let’s build it out. The function will need a parameter, ***evt***, which will receive the ***submit*** event object. We can make use of its properties and methods. We will use its ***preventDefault()*** method to disable the default behavior for the ***submit*** event. We will also put in a trap for older browsers that don’t implement this method, by just returning ***false*** to prevent the form from submitting. Also, we will reset the ***formValidity*** variable to ***true*** for now, until we build it out further:  
   . function validateForm(***evt***) {  
    ***if (evt.preventDefault) {  
    evt.preventDefault();  
    }  
    else {  
    evt.returnValue = false;  
    }  
    formValidity = true;  
   }***  
   Give this a test in the browser. The ***submit*** with the ***Place Order*** button should not produce a results page anymore. We have effectively stopped the form from submitting.
4. We are going to build a bunch of validity tests in this function. But before we do, let’s build a framework that will handle what will be the ***results*** of these tests. For starters, let’s build the code that will execute if the ***<form>*** turns out to be valid after our soon-to-be-built tests:  
    ***if (formValidity === true) {  
    document.getElementById("errorText").innerHTML = "";  
    document.getElementById("errorText").style.display = "none";  
    document.getElementsByTagName("form")[0].submit();  
    }***
5. Now let’s build the code that will execute if the ***<form>*** turns out to be invalid after our soon-to-be-built tests. We will make use of a scroll(0,0) function to move back to the top of the page to display the errors:  
    ***else {  
    document.getElementById("errorText").innerHTML =   
    "Please fix the indicated problems and then resubmit   
    your order.";  
    document.getElementById("errorText").style.display =   
    "block";  
    scroll(0,0);  
    }***A test of Place Order should now allow submission, because we flagged no errors, then explicitly called ***submit()*** in the ***if*** statement.
6. Now let’s go back to the ***validateForm()*** function and change the ***formValidity*** variable to false. This will simulate that we have flagged some errors:  
    ***formValidity = false;***  
   A test submission now should scroll to the top of the page and the error message should be visible. Change the ***formValidity*** variable back to ***true***.

**Exercise 01\_06\_01 – Step 8**

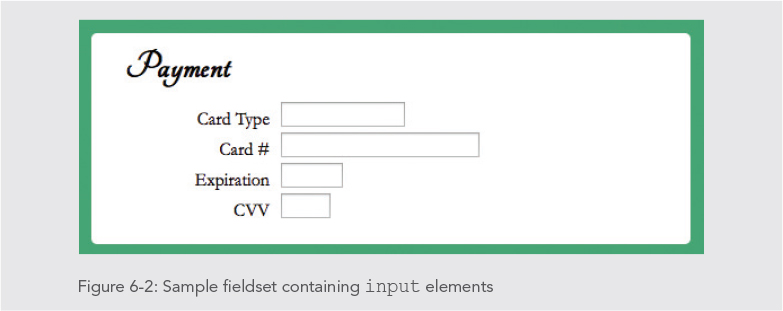
1. Return to ***snoot.js***. We will start our required field validity check by building a function called ***validateAddress()***. We will give it a parameter to hold the id of the <fieldset>. This will allow us to use the same code for the delivery and billing groups. Let’s set up the function first above the ***validateForm()*** function:  
   ***function validateAddress(fieldsetId) {  
     
   }***
2. Now let’s set up some variables for support, in particular one that will grab all of the ***<input>*** fields and group them in an array we will be able to loop through. Another will grab the ***<div>*** for the fieldset that will hold error messages. We will also need to track the validity of the <fieldset>, the number of elements, and the current element:  
    ***var inputElements = document.querySelectorAll("#" +   
    fieldsetId + " input");  
    var errorDiv = document.querySelectorAll("#" + fieldsetId +   
    " .errorMessage")[0];  
    var fieldsetValidity = true;  
    var elementCount = inputElements.length;  
    var currentElement;***
3. We need to set up a ***try*** / ***catch*** structure to handle this. Our intention will be to have the ***try*** clause loop through the fields, testing for validity. It will ***throw*** an error if necessary. The ***catch*** clause will handle the mechanics of the error. For starters, let’s get the structure going. We’ll start with just the scaffolding of the ***try*** and catch clauses:  
    ***try {  
    alert("I am executing the try clause");  
    }  
    catch(msg) {  
    errorDiv.style.display = "block";  
    errorDiv.innerHTML = msg;  
    formValidity = false;  
    }***Check in the browser for any syntax errors.
4. Now let’s see if we can test the structure. We want to do this before we add too much code in that will make it much harder to debug. Go back to the ***validateform()*** function, directly below where the ***formValidity*** variable is set. Enter the following code to see if we get some results:  
    ***validateAddress("billingAddress");  
    validateAddress("deliveryAddress");***  
   submitting the form should give us two alerts, one for each time it goes through the try clause to check billing address, then delivery address. This give us the confidence to proceed with more code.
5. We are going to need a ***for*** loop in the ***try*** clause to iterate through all of the ***<input>*** fields, checking for validity. In this case, validity means there is a value in each, they are ***required***. Every one that we find that is invalid, we will change its ***background*** property, and we will also mark the ***fieldsetValidity*** variable as ***false***:. This time let’s use a debugger command to monitor results:  
    ***for (var i = 0; i < elementCount; i++) {  
    currentElement = inputElements[i];  
    if (currentElement.value === "") {  
    debugger;  
    currentElement.style.background = "rgb(255,233,233)";  
    fieldsetValidity = false;  
    }  
    else {  
    currentElement.style.background = "white";  
    }  
    }***A browser test with Developer Tools on and with no data should give us 12 debugger pauses on submission. If we set the screen correctly, we should be able to see the field backgrounds change. The field that is missing is the ***<select>*** field which we have not messed with yet. Now lets put data int the first field and check the ***same as billing address*** check box. A submission should now give us only 10 alerts.
6. Now let’s set up throwing the error in the ***try*** clause, right below the ***for*** loop:  
    ***if (fieldsetValidity === false) {  
    if (fieldsetId === "billingAddress") {  
    throw "Please complete all Billing Address   
    information.";  
    }  
    else {  
    throw "Please complete all Delivery Address   
    information.";  
    }  
    }  
    else {  
    errorDiv.style.display = "none";  
    errorDiv.innerHTML = "";  
    }***  
   If everything appears to be working in the browser test, remove the debugger statement and let’s giv it some significant testing.

**Exercise 01\_06\_01 – Step 9**

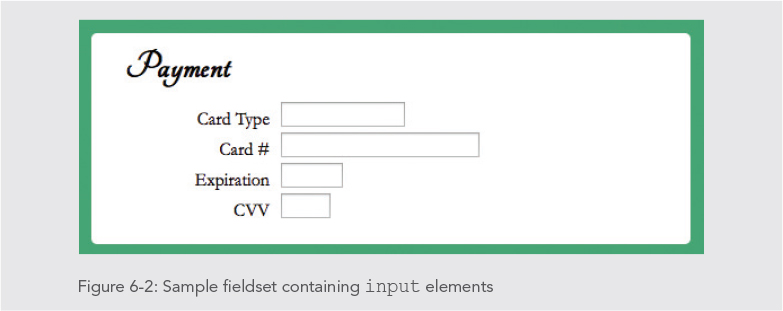
1. Return to ***snoot.js***. We have just the ***<select>*** fields in both field sets to validate. Go back to ***validateAddress()*** and locate the statement   
   ***if (fieldsetValidity === false)***. Before it, let’s put in some code to handle the <select> fields. We will use a red border on select lists to indicate errors:   
    ***currentElement = document.querySelector("#" +   
    fieldsetId + " select");  
    if (currentElement.selectedIndex === -1) {  
    currentElement.style.border = "1px solid red";  
    fieldsetValidity = false;  
    }  
    else {  
    currentElement.style.border = "";  
    }***
2. Test this with no data to see if we get the effect that we want. Then do a thorough test of the select lists in both field sets.

**Exercise 01\_06\_01 – Step 10**

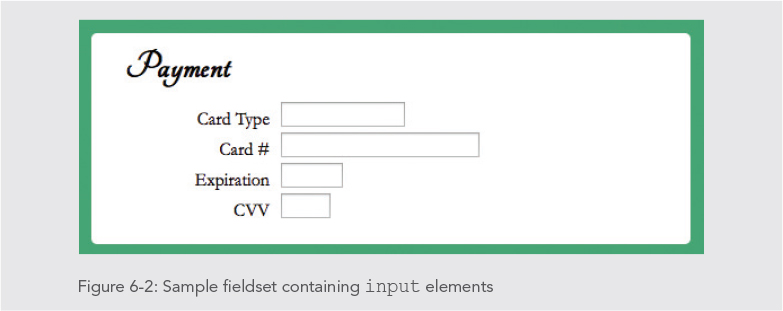
1. Return to ***snoot.js***. Let’s validate the three ***<select>*** lists that make up the ***Delivery Date*** <fieldset>. We should be able to use a lot of the code from the previous section. Let’s copy all of the ***validateAddress()*** function to right below itself. Let’s rename it and remove the parameter, it won’t be necessary, as this validation is for just one ***<fieldset>***:  
   ***function validateDeliveryDate() {  
     
   }***
2. Now let’s modify the support variables:  
    ***var selectElements =   
    document.querySelectorAll("#deliveryDate select");  
    var errorDiv = document.querySelector("#deliveryDate   
    .errorMessage");  
    var fieldsetValidity = true;  
    var elementCount = selectElements.length;  
    var currentElement;***
3. Now let’s do some work on the ***try*** / ***catch*** structure to handle this ***<fieldset>***. Just a different type of element, but similar logic:  
    ***try {  
    for (var i = 0; i < elementCount; i++) {  
    currentElement = selectElements[i];  
    if (currentElement.selectedIndex === -1) {  
    currentElement.style.border = "1px solid red";  
    fieldsetValidity = false;  
    } else {  
    currentElement.style.border = "";  
    }  
    }  
    if (fieldsetValidity === false) {  
    throw "Please specify a Delivery Date.";  
    } else {  
    errorDiv.style.display = "none";  
    errorDiv.innerHTML = "";  
    }  
    } catch (msg) {  
    errorDiv.style.display = "block";  
    errorDiv.innerHTML = msg;  
    formValidity = false;*** }Check in the browser for any syntax errors.
4. Now let’s see if we can test the structure. Go back to the ***validateform()*** function, directly below the function call to ***validateAddress("deliveryAddress");*** , enter the following code to see if we get some results:  
    ***validateDeliveryDate();***  
   Let’s give this a significant browser test.
5. Notice that if we recycle the browser, then just enter a year, we show a syntax error in the Console tab. This is because no month has been selected. Return to the ***updateDays()*** function and enter the following code to fix the bug:  
    var deliveryYear = document.getElementById("delivYr");  
    ***if (deliveryMonth.selectedIndex === -1) {  
    return  
    }***  
   Now let’s test this again in the browser.

**Exercise 01\_06\_01 – Step 11**

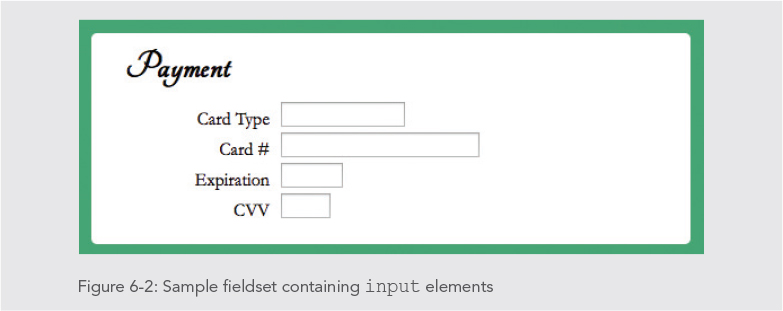
1. Return to ***snoot.js***. Let’s validate the Payment <fieldset>. This has got quite a mix of different form controls, so we can mix up our code a little bit. We can save some code, but some needs major surgery. Let’s start by copying all of the ***validateDeliveryDate()*** function to right below itself and let’s rename it:  
   ***function validatePayment() {  
     
   }***
2. Now let’s modify the support variables, noticing that we have to do a bit more to get our hands on the different control types:  
    ***var errorDiv = document.querySelector("#paymentInfo   
    .errorMessage");  
    var fieldsetValidity = true;  
    var ccNumElement = document.getElementById("ccNum");  
    var selectElements =   
    document.querySelectorAll("#paymentInfo select");  
    var elementCount = selectElements.length;  
    var cvvElement = document.getElementById("cvv");  
    var cards = document.getElementsByName("PaymentType");  
    var currentElement;***
3. Now let’s do some work on the ***try*** / ***catch*** structure to handle this ***<fieldset>***. This one is quite a bit different so let’s just wipe out most of it and leave the scaffolding for the try block, along with the entire catch clause:  
    ***try {  
    alert("In the try block");  
    if (fieldsetValidity === false) {  
    throw "Please complete all Payment information.";  
    } else {  
    errorDiv.style.display = "none";  
    errorDiv.innerHTML = "";  
    }  
    }   
    catch (msg) {  
    errorDiv.style.display = "block";  
    errorDiv.innerHTML = msg;  
    formValidity = false;  
    }***Check in the browser for any syntax errors.
4. Now let’s see if we can test the structure. Go back to the ***validateform()*** function, directly below the function call to ***validateDeliveryDate();***, enter the following code to see if we get some results:  
    ***validatePayment();***  
   Let’s give this a browser test. We should get an alert when we ***Place Order*** that indicates the validation of this ***<fieldset>*** is proceeding.
5. The first part of this ***<fieldset>*** we will work on is the set of option buttons that control the credit card type. One must be checked for the ***<form>*** to be valid:  
    ***if (!cards[0].checked && !cards[1].checked &&   
    !cards[2].checked && !cards[3].checked) {  
    for (var i = 0; i < cards.length; i++) {  
    cards[i].style.outline = "1px solid red";  
    }  
    fieldsetValidity = false;  
    }  
    else {  
    for (var i = 0; i < cards.length; i++) {  
    cards[i].style.outline = "";  
    }  
    }***Give that a browser test with a ***Place Order*** with no card type checked, then try it with one checked.
6. Now we need to check that there is a ***Card #*** entered as well. There are well-known algorithms to check to see if it is a valid number of the particular card type. That is beyond where we will go in this exercise. On the client side, we cannot, of course, check to see if it is approved. Below the previous code, let’s get that into place below the previous try code:  
    ***if (ccNumElement.value === "") {  
    ccNumElement.style.background = "rgb(255,233,233)";  
    fieldsetValidity = false;  
    } else {  
    ccNumElement.style.background = "white";  
    }***Give that one a browser test.
7. Let’s get the Expiration Date ***<select>*** fields, which are basically nothing new at this point:  
    ***for (var i = 0; i < selectElements.length; i++) {  
    currentElement = selectElements[i];  
    if (currentElement.selectedIndex === -1) {  
    currentElement.style.border = "1px solid red";  
    fieldsetValidity = false;  
    } else {  
    currentElement.style.border = "";  
    }  
    }***Another test and we are almost there.
8. Okay one more to go in this ***<fieldset>***, the ***CVV*** code:  
    ***if (cvvElement.value === "") {  
    cvvElement.style.background = "rgb(255,233,233)";  
    fieldsetValidity = false;  
    } else {  
    cvvElement.style.background = "white";  
    }***Let’s give a thorough browser test to the whole ***<form>*** as it stands so far.

**Exercise 01\_06\_01 – Step 12**

1. Return to ***snoot.js***. the ***Message*** ***<fieldset>*** has some field dependent logic. None of the check boxes are required. However, if the ***Custom message*** check box is checked, there must be a custom message in the <textarea>. We can again save some code, but some needs major surgery. Let’s start by copying all of the ***validateDeliveryDate()*** function to right before the ***validateForm(evt)*** function below itself and let’s rename it:  
   ***function validateMessage() {  
     
   }***
2. Now let’s modify the support variables to give us what we need for this ***<fieldset>***:  
    ***var errorDiv = document.querySelector("#message   
    .errorMessage");  
    var msgBox = document.getElementById("customText");***
3. Now let’s do some work on the ***try*** / ***catch*** structure to handle this ***<fieldset>***. This one is quite a bit different so let’s just wipe out most of it and leave the scaffolding for the try block. Note carefully that we are making a slight change to what has been our standard catch block:   
    ***try {  
    alert("In the try block");  
    }   
    catch (msg) {  
    errorDiv.style.display = "block";  
    errorDiv.innerHTML = msg;  
    msgBox.style.background ="rgb(255,233,233)";  
    formValidity = false;  
    }***Check in the browser for any syntax errors.
4. Now let’s see if we can test the structure. Go back to the ***validateform()*** function, directly below the function call to ***validatePayment();***, enter the following code to see if we get some results:  
    ***validateMessage();***  
   Let’s give this a browser test. We should get an alert when we ***Place Order*** that indicates the validation of this ***<fieldset>*** is proceeding.
5. Let’s check the ***Custom message*** check box, it is the only one we really care about for validity:  
    ***if (document.getElementById("custom").checked &&   
    ((msgBox.value === "") || (msgBox.value ===   
    msgBox.placeholder))) {  
    throw "Please enter your Message text.";  
    } else {  
    errorDiv.style.display = "none";  
    msgBox.style.background = "white";  
    }***Give that a browser test with a ***Place Order*** and various combinations of checked boxes and custom messages.

**Exercise 01\_06\_01 – Step 13**

1. Return to ***snoot.js***. the ***Create Account?*** ***<fieldset>*** has some more field dependent logic. The <fieldset> as a whole is not required. However, if data is entered into any of the fields, multiple checks must happen. 1) If any field has data, all must have data. 2) The password fields must match. Let’s follow the patter and start by copying all of the ***validateMessage()*** function to right below itself and let’s rename it:  
   ***function validateCreateAccount() {  
     
   }***
2. Now let’s see if we can test the structure. Go back to the ***validateform()*** function, directly below the function call to ***validateMessage();***, enter the following code to see if we get some results:  
    ***validateCreateAccount();***  
   Let’s give this a browser test. We should get an alert when we ***Place Order*** that indicates the validation of this ***<fieldset>*** is proceeding.
3. Now let’s modify the support variables to give us what we need for this ***<fieldset>***:  
    ***var errorDiv = document.querySelector("#createAccount   
    .errorMessage");  
    var usernameElement =   
    document.getElementById("username");  
    var pass1Element = document.getElementById("pass1");  
    var pass2Element = document.getElementById("pass2");  
    var passwordMismatch = false;  
    var invColor = "rgb(255,233,233)";  
    var fieldsetValidity = true;***
4. Now we can get to work on the validation in the ***try*** clause. First we will turn off any pre-existing error markings:  
    ***usernameElement.style.background = "";  
    pass1Element.style.background = "";  
    pass2Element.style.background = "";  
    errorDiv.style.display = "none";***
5. Next let’s build out the part of the ***try*** clause that will handle all fields filled in and testing for passwords matching:  
    ***if (usernameElement.value !== "" &&   
    pass1Element.value !== "" &&   
    pass2Element.value !== "") {  
    if (pass1Element.value !== pass2Element.value) {  
    passwordMismatch = true;  
    throw "Passwords entered do not match, please   
    reenter.";  
    }  
    }***
6. We need to do a little work on the ***catch*** clause this time to try out this first part of the validation:  
    ***} catch (msg) {  
    errorDiv.style.display = "block";  
    errorDiv.innerHTML = msg;  
    formValidity = false;  
    if (passwordMismatch) {  
    usernameElement.style.background = "";  
    pass1Element.style.background = invColor;  
    pass2Element.style.background = invColor;  
    }***Let’s give that a browser test for syntax.
7. Now we will work on the ***try*** clause to take care of the situation in which only some fields are filled in:  
    ***else if (usernameElement.value === "" &&   
    pass1Element.value === "" && pass2Element.value ===   
    "") {  
    fieldsetValidity = true;  
    }  
    else {  
    fieldsetValidity = false;  
    throw "Please enter all fields to Create Account.";  
    }***
8. One last modification to the catch should take care of the validation:  
    if (passwordMismatch) {  
    usernameElement.style.background = "";  
    }   
    ***else {  
    usernameElement.style.background = invColor;  
    }***