CHAPTER 6 – Enhancing and Validating Form

Table of Contents

[**ENHANCING AND VALIDATING FORMS** 1](#_Toc13045968)

[Using JavaScript with Forms: 1](#_Toc13045969)

[Referencing Forms and Form Elements: 3](#_Toc13045970)

[Improving Form Usability 4](#_Toc13045971)

[Designing Forms to Collect More Accurate Content 4](#_Toc13045972)

[Programming Forms to Increase Content Accuracy 4](#_Toc13045973)

[Dynamically Updating Selection List Values 6](#_Toc13045974)

[Adding Placeholder Text for Older Browsers 10](#_Toc13045975)

[Automatically Updating an Associated Field Based on a User Entry 12](#_Toc13045976)

[Transferring Duplicate Field Values 13](#_Toc13045977)

[Customizing Browser-Based Validation 14](#_Toc13045978)

[Specific Browser-Based Validation Parameters 14](#_Toc13045979)

[Customizing Browser-based Validation Feedback 15](#_Toc13045980)

[Programming Custom Validation 16](#_Toc13045981)

[Validating Submitted Data 17](#_Toc13045982)

[Validate Required Fields with Custom Functions 18](#_Toc13045983)

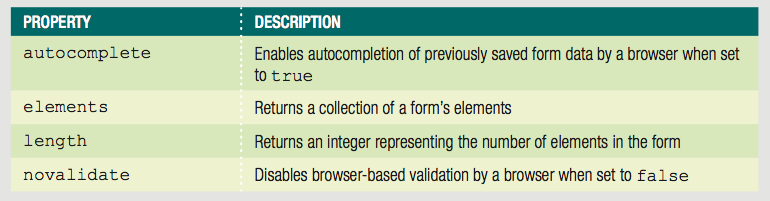
[Validating Dependent Fields with Custom Functions 24](#_Toc13045984)

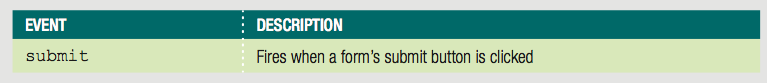
[Validating Content Type with Custom Functions 27](#_Toc13045985)

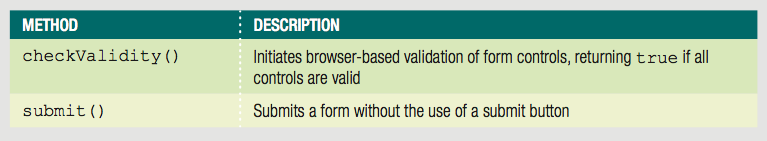
# ENHANCING AND VALIDATING FORMS

## Using JavaScript with Forms:

* ***Validation***
  + checking that form information provided by users conforms to data rules
* ***form object***
  + Represents a form in an HTML document
  + You can use the properties, events, and methods of a form object to access form and its data with JavaScript

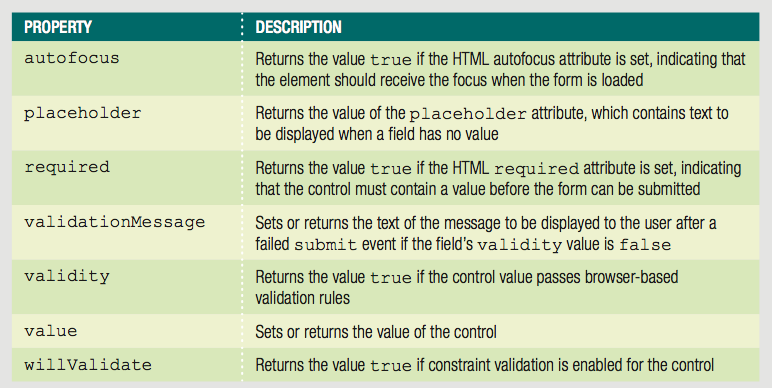


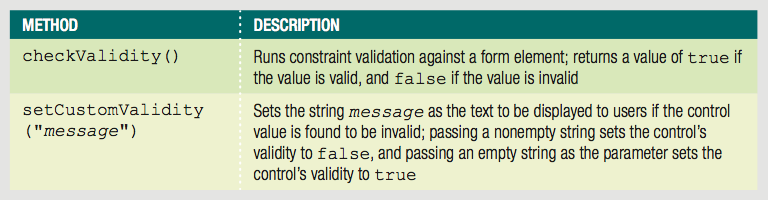


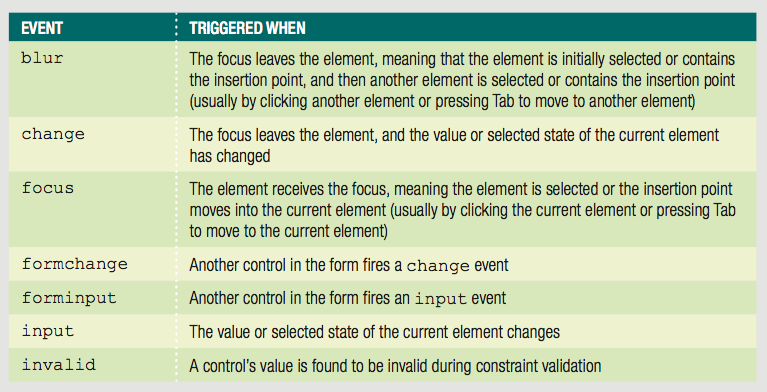


* Common elements for collecting form data:
  + input
  + select
  + option
  + textarea
  + button

Additional properties, methods, and events that are element specific:







### Referencing Forms and Form Elements:

* Can use getElementsByTagName() method:

getElementsByTagName("form")[0]

* You can also encounter the reference that refers to the first object with the default array of form object crated by the browser

document.forms[0]

* In addition, the form array contains an array of objects within a form element, known as the element array. You could reference the third object within the first form in a web document using

document.forms[0].elements[2]

Especially in a large form, using the elements array can be tedious. However, when referencing a from using the forms array, you can instead reference any element within the form by the element’s name value. This means that to reference an element with the name DeliveryZip in the first form in a web document, you could use the syntax

Document.forms[0].DeliveryZip

One advantage to reference form object using methods such as getElementById() rather than the forms array is that you don’t need to switch between using one syntax for referencing form objects and another syntax for non-form objects.

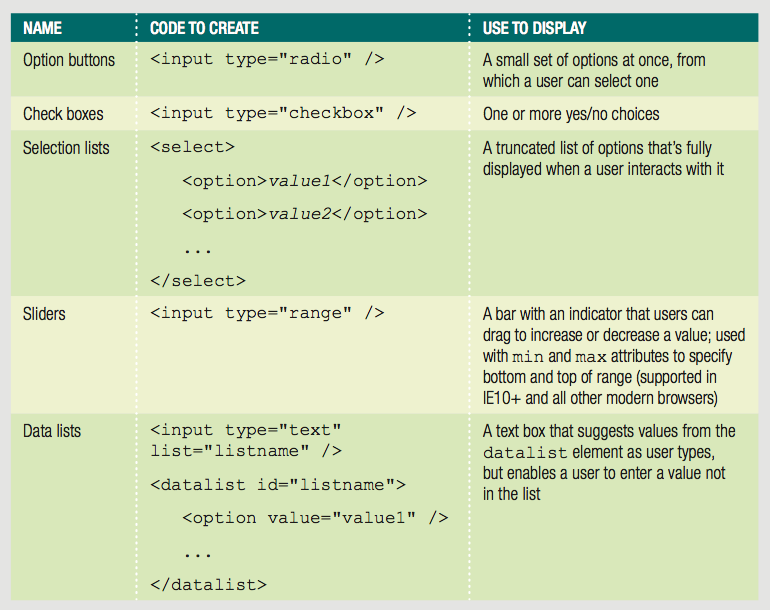
## Improving Form Usability

One of the most important form-related roles for JavaScript programs is validating user input.

### Designing Forms to Collect More Accurate Content

Most form validation involves checking data that users have typed into input and textarea boxes to ensure its accuracy.

Some alternative form elements and the types of information they’re best for.

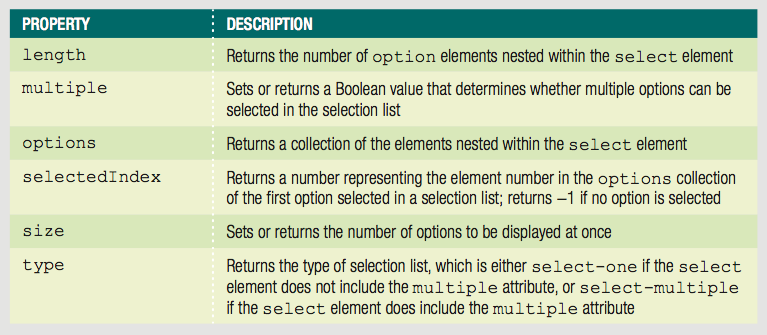


### Programming Forms to Increase Content Accuracy

Assistive functions – reduce the likelihood of user errors when completing a form. These functions do not perform validation themselves – they do not check user content for errors. Instead, these functions prevent users from entering erroneous data in the first place.

Removing default values from Selection List – you can use HTML to set the default value for a selection list, but only to one of the menu options. However, you can use JavaScript to set the selectIndex property of a select object to -1, which corresponds to no selection.

Properties of select element that you can manipulate with JavaScript –



Example: Case Study –

/\* remove default values and formatting from state and delivery date selection

lists \*/

function removeSelectDefaults(){

//this variable references a collection that includes all the select elements

var emptyBoxes = document.getElementsByTagName("select");

//loop to run commands on the select element at position i

for(var i = 0; i < emptyBoxes.length; i++){

//enter the following code to set the selectedIndex value:

emptyBoxes[i**].selectedIndex = -1;**

}

}

/\* run setup function when page finishes loading \*/

if(window.addEventListener){

window.addEventListener("load",removeSelectDefaults,false);

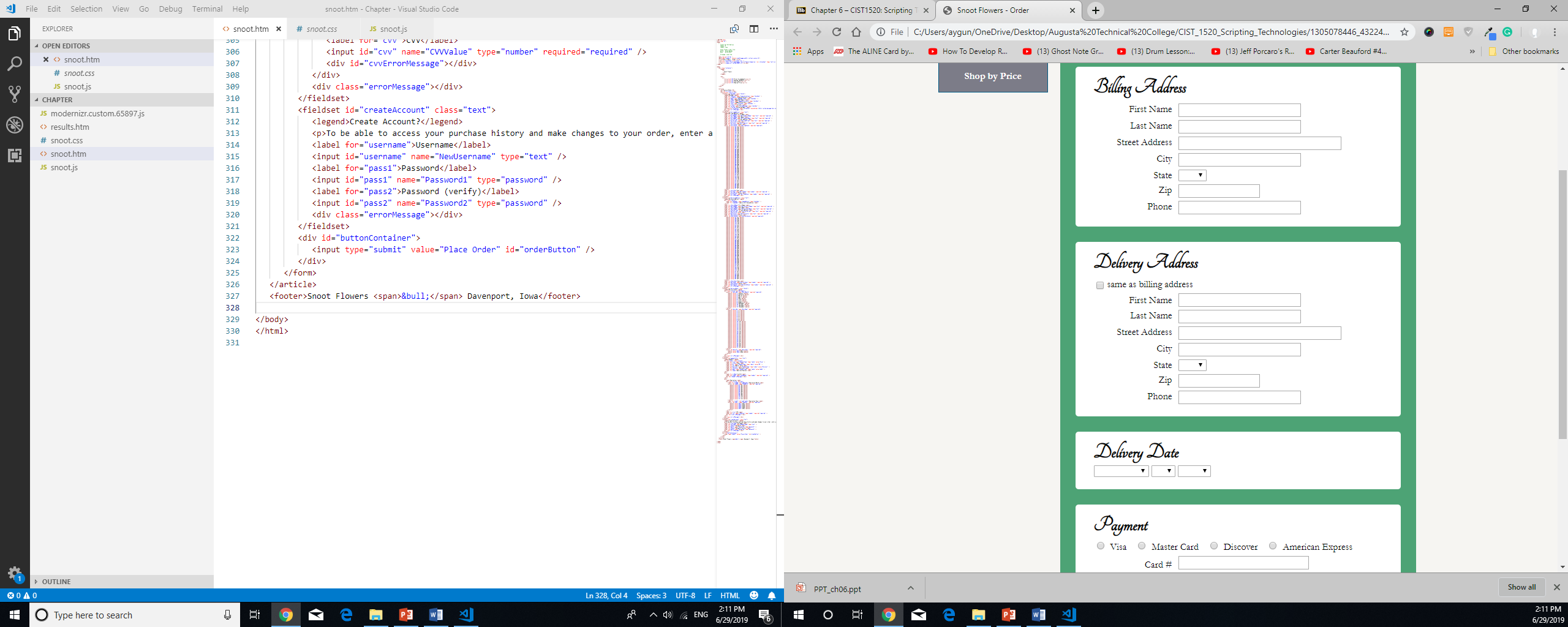
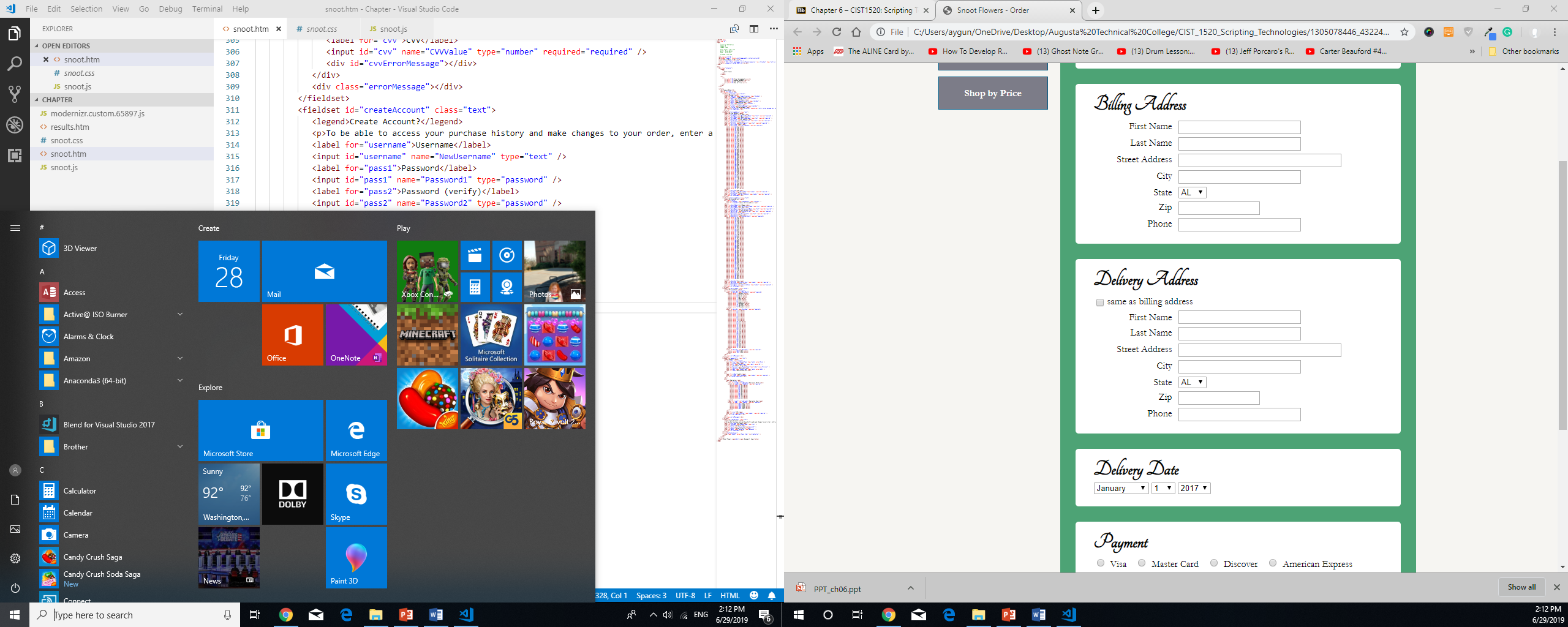
}else if(window.attachEvent){

window.attachEvent("onload",removeSelectDefaults);

}

Add the script element to reference snoot.js file you just created right before the closing body tag -

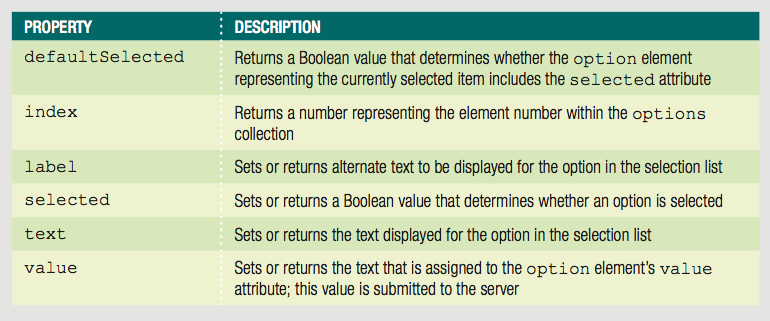
<script src="snoot.js"></script>



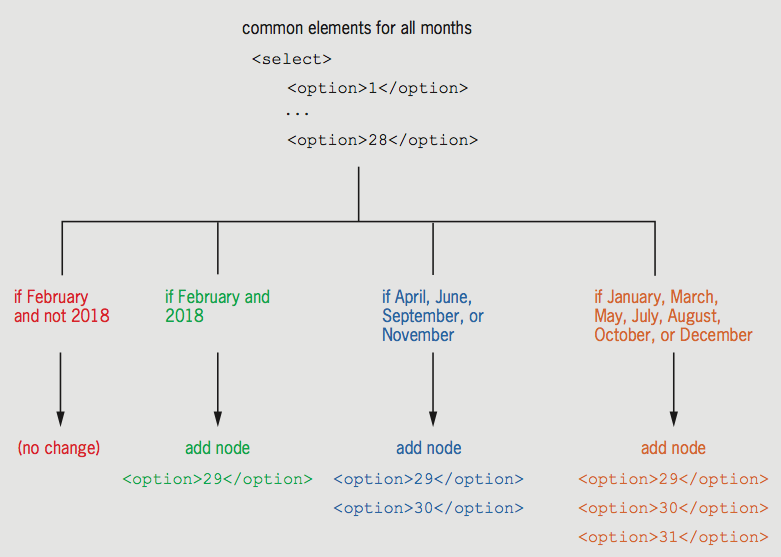
### Dynamically Updating Selection List Values

* + You can add or remove option elements from a select element using node methods

In addition to using the properties of the select element, you can also look up and set properties of option elements, which are described in the next table:



One common use of adding and removing options from a select element is to change the options shown in a list based on another selection a user has made in the same form. For instance, the number of days shown to users should depend on the selected month.



/\* change the days list dynamically based on the selected month and year \*/

/\* global variables \*/

var twentyNine = document.createDocumentFragment();

var thirty = document.createDocumentFragment();

var thirtyOne = document.createDocumentFragment();

/\* set up node building blocks for selection list of day \*/

function setupDays(){

//get all the day options

var dates = document.getElementById("delivDy").getElementsByTagName("option");

//copies the 29th option element with index no 28 to the twentyNine node

twentyNine.appendChild(dates[28].cloneNode(true));

//copies the 29th and 30th option elements

thirty.appendChild(dates[28].cloneNode(true));

thirty.appendChild(dates[29].cloneNode(true));

//copies the 29th, 30th, and 31st option elements

thirtyOne.appendChild(dates[28].cloneNode(true));

thirtyOne.appendChild(dates[29].cloneNode(true));

thirtyOne.appendChild(dates[30].cloneNode(true));

}

/\* function to update days \*/

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;

while(dates[28]){

/\*remove child with index of 28 until this index is empty(repeats this

until no child exists at index 28.) \*/

deliveryDay.removeChild(dates[28]);

}

if(deliveryYear.selectedIndex === -1){

//if no year is selected, choose the default year so length of Feb can be

determined

deliveryYear.selectedIndex = 0;

}

if(selectedMonth === "2" && deliveryYear.options[deliveryYear.selectedIndex].value === "2018"){

//if leap year, Feb has 29 days

deliveryDay.appendChild(twentyNine.cloneNode(true));

}else if(selectedMonth ==="4" || selectedMonth ==="6" || selectedMonth === "9" || selectedMonth === "11"){

//these months have 30 days

deliveryDay.appendChild(thirty.cloneNode(true));

}else if(selectedMonth ==="1" || selectedMonth ==="3" || selectedMonth === "5" || selectedMonth === "7" || selectedMonth === "8" || selectedMonth === "10" || selectedMonth === "12"){

//these months have 31 days

deliveryDay.appendChild(thirtyOne.cloneNode(true));

}

}

/\* remove default values and formatting from state and delivery date selection

lists \*/

function removeSelectDefaults(){

//this variable references a collection that includes all the select elements in the document

var emptyBoxes = document.getElementsByTagName("select");

//loop to run commands on the select element at position i

for(var i = 0; i < emptyBoxes.length; i++){

//enter the following code to set the selectedIndex value:

emptyBoxes[i].selectedIndex = -1;

}

}

/\* create event listeners \*/

/\* this code adds two event listeners: the first to call the updateDays() function when the value in the month selection list changes, and the second to call the same function when the value in the year selection list changes. \*/

function createEventListeners(){

var deliveryMonth = document.getElementById("delivMo");

if(deliveryMonth.addEventListener){

deliveryMonth.addEventListener("change",updateDays,false);

}else if(deliveryMonth.attachEvent){

deliveryMonth.attachEvent("onchange",updateDays);

}

var deliveryYear = document.getElementById("delivYr");

if(deliveryYear.addEventListener){

deliveryYear.addEventListener("change",updateDays,false);

}else if(deliveryYear.attachEvent){

deliveryYear.attachEvent("onchange",updateDays);

}

}

/\* run initial form configuration functions when page is loaded first \*/

function setUpPage(){

removeSelectDefaults();

setupDays();

createEventListeners();

}

/\* run setup function when page finishes loading \*/

if(window.addEventListener){

window.addEventListener("load",setUpPage,false);

}else if(window.attachEvent){

window.attachEvent("onload",setUpPage);

}

### Adding Placeholder Text for Older Browsers

One popular recent feature in form creation is the placeholder attribute of the input and textarea elements. You can specify a word or phrase as the value of the placeholder attribute; this word or phrase is then displayed in the input or textarea box, generally in gray characters. When the user clicks in an input or textarea element containing placeholder text, the placeholder text is removed, and the user can make an entry in the box.

In this case study’s order form, you’ll add code to replicate the behavior of the placeholder attribute in older browsers. To do so, you need to replicate three separate behaviors that are automatic in modern browsers:

Add placeholder text when page finishes loading

Remove placeholder text when user selects field

Add back placeholder text if user makes no entry

You’ll start by creating a function to add the placeholder text in older browsers. The order form page already includes the Modernizer library from making semantic HTML elements available in older browsers. You’ll check the value of the of the Modernizr.input.placeholder value to see if the browser opening the page supports the placeholder attribute. If so, your function will add the text of the placeholder attribute as the value of the textarea control.

//to add placeholder text to the Custom message box in older browser:

function generatePlaceholder(){

if(!Modernizr.input.placeholder){

var messageBox = document.getElementById("customText");

messageBox.value = messageBox.placeholder;

messageBox.style.color = "rgb(178,184,183)";

}

}

Scroll down to the setUpPage() function, and add a call to the generatePlaceholder() function:

/\* run initial form configuration functions when page is loaded first \*/

function setUpPage(){

removeSelectDefaults();

setupDays();

createEventListeners();

generatePlaceholder();

}

Next you’ll add code to clear the value in the textarea box when a user clicks in the box.

//remove fallback placeholder text

function zeroPlaceholder(){

var messageBox = document.getElementById("customText");

messageBox.style.color = "black";

if(messageBox.value === messageBox.placeholder){

messageBox.value = "";

}

}

Within the generatePlaceholder() function you created just before the closing } for the if statement, create an event listener to call the zeroPlaceholder() function.

//to add placeholder text to the Custom message box in older browser:

function generatePlaceholder(){

if(!Modernizr.input.placeholder){

var messageBox = document.getElementById("customText");

messageBox.value = messageBox.placeholder;

messageBox.style.color = "rgb(178,184,183)";

if(messageBox.addEventListener){

messageBox.addEventListener("focus",zeroPlaceholder,false);

}else if(messageBox.attachEvent){

messageBox.attachEvent("onfocus",zeroPlaceholder);

}

}

}

Add code to restore the placeholder text when a user leaves the textarea field without typing anything.

//restore placeholder text if box contains no user entry

function checkPlaceholder(){

var messageBox = document.getElementById("customText");

if(messageBox.value === ""){

messageBox.style.color = "rgb(178,184,183)";

messageBox.value = messageBox.placeholder;

}

}

Within the generatePlaceholder() function you created add the code to create an event.

//to add placeholder text to the Custom message box in older browser:

function generatePlaceholder(){

if(!Modernizr.input.placeholder){

var messageBox = document.getElementById("customText");

messageBox.value = messageBox.placeholder;

messageBox.style.color = "rgb(178,184,183)";

if(messageBox.addEventListener){

messageBox.addEventListener("focus",zeroPlaceholder,false);

messageBox.addEventListener("blur",checkPlaceholder,false);

}else if(messageBox.attachEvent){

messageBox.attachEvent("onfocus",zeroPlaceholder);

messageBox.attachEvent("onblur",checkPlaceholder);

}

}

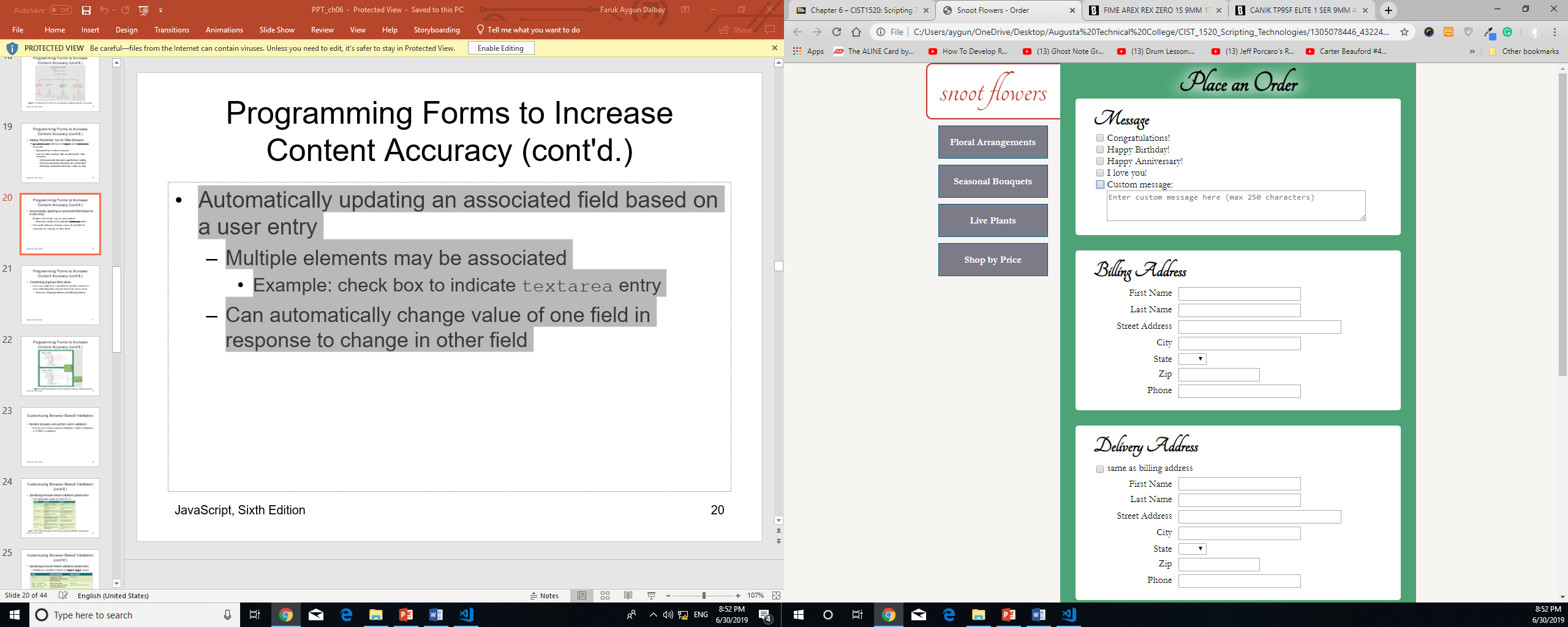
}

### Automatically Updating an Associated Field Based on a User Entry

* + Multiple elements may be associated

Example: check box to indicate textarea entry

* + Can automatically change value of one field in response to change in another field



Some commonly used properties and methods associated with textarea elements –

* Placeholder – Value of the placeholder attribute
* defaultValue – Default value displayed in the textarea element when the page loads
* select() – Selects the element contents

To create a function to automatically check the Custom message check box when a custom message has been entered:

//automatically check Custom message check box if user makes entry in customText box

function autockeckCustom(){

var messageBox = document.getElementById("customText");

if(messageBox.value !== "" && messageBox.value !== messageBox.placeholder){

//if user entry in textarea, check Custom check box

document.getElementById("custom").checked = "checked";

}

}

Within the createEventListeners() function, add the event listener that calls autocheckedCustom () function when a user leaves the textarea control:

var messageBox = document.getElementById("customText");

if(messageBox.addEventListener){

messageBox.addEventListener("blur",autockeckCustom,false);

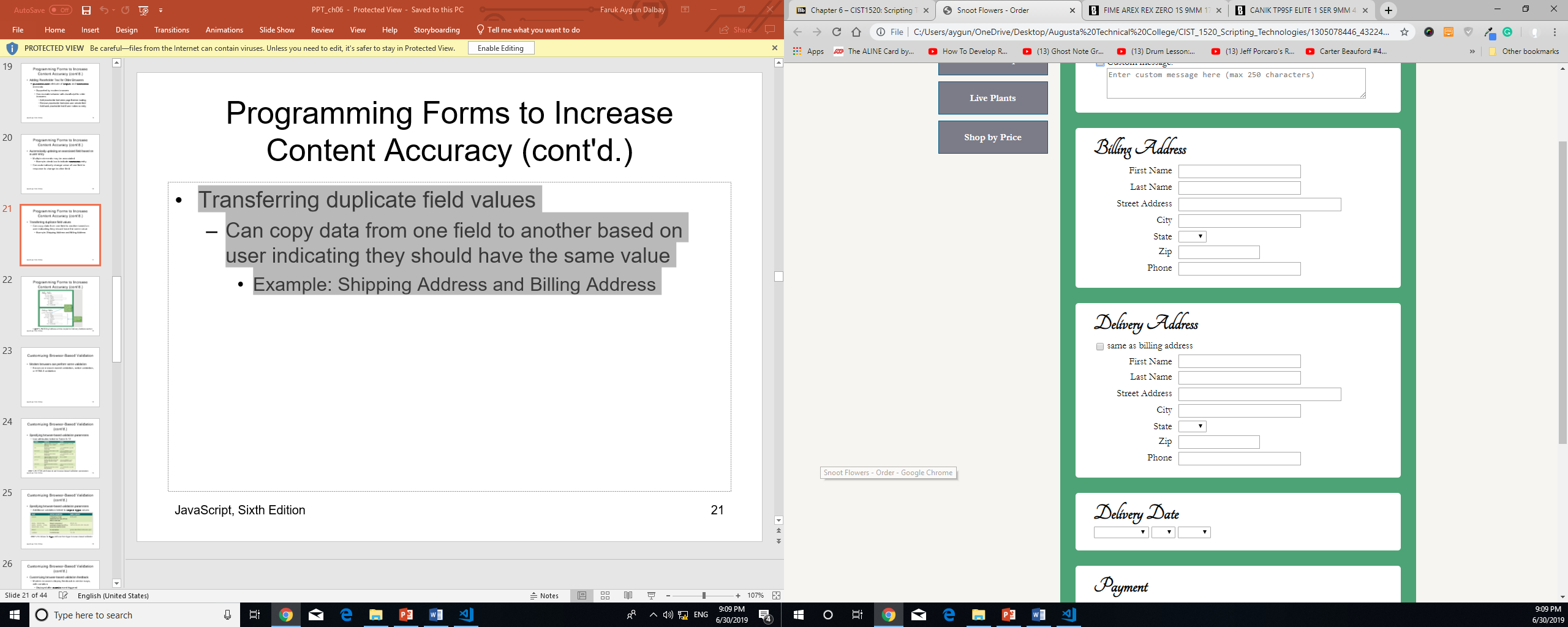
}else if(messageBox.attachEvent){

messageBox.attachEvent("onblur",autockeckCustom);

}

### Transferring Duplicate Field Values

* + Can copy data from one field to another based on user indicating they should have the same value
    - Example: Shipping Address and Billing Address



To enable users to use billing address information for the shipping address:

//copy values for Billing Address fields to Delivery Address fields

function copyBillingAddress(){

var billingInputElements = document.querySelectorAll("#billingAddress input");

var deliveryInputElements = document.querySelectorAll("#deliveryAddress input");

if(document.getElementById("sameAddr").checked){

//copy all input values

for(var i = 0; i < billingInputElements.length; i++){

deliveryInputElements[i+1].value = billingInputElements[i].value;

}

//copy the single select element

document.querySelector("#deliveryAddress select").value = document.querySelector("#billingAddress select").value;

}

}

Within the command block for the createListeners() function, create an event listener for the copyBillingAddress() function.

var same = document.getElementById("sameAddr");

if(same.addEventListener){

same.addEventListener("click",copyBillingAddress,false);

}else if(same.attachEvent){

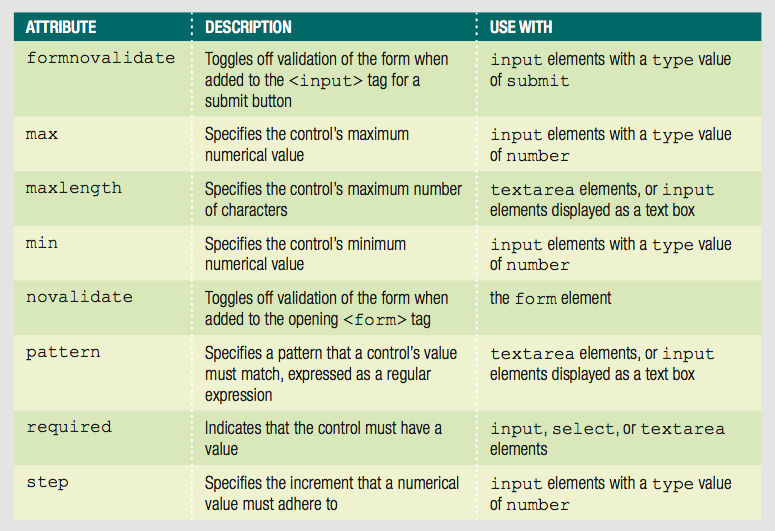
same.attachEvent("onclick",copyBillingAddress);

}

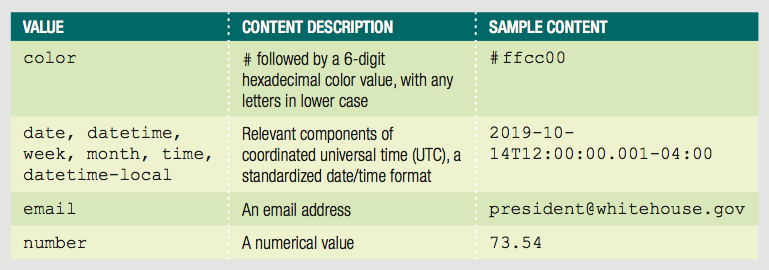
## Customizing Browser-Based Validation

Modern browsers can perform some validation without any extra JavaScript. This type of validation is known as ***browser-based validation***, ***native validation***, or ***HTML5 validation.***

### Specific Browser-Based Validation Parameters



Additional browser-based validation is linked to several values for the type attribute of the input element. For instance, if a field is created with the tag <input type=”number”/>, modern browsers automatically reject nonnumeric content.



Note – HTML also includes the values tel (telephone number) and url as values for the input type attribute. However, browsers do not enforce any rules for content validity on elements that use these types.

Browser-based validation with missing or invalid data in the order form-

Note that the input elements for zip codes and phone numbers use a type value of number; and the elements in the form are marked as required.

<label for="delivZip">Zip</label>

<input id="delivZip" name="DeliveryZip" type="number" required="required" />

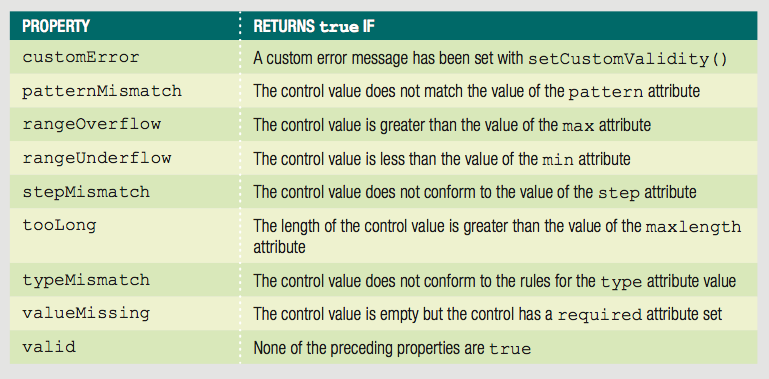
<label for="delivPhone">Phone</label>

<input id="delivPhone" name="DeliveryPhone" type="number" required="required" />

### Customizing Browser-based Validation Feedback

* + Modern browsers display feedback in similar ways, with variation
    - Displayed after submit event triggered
    - Invalid controls highlighted
    - Bubble displayed next to first control

Many aspects of the way browsers present browser-based validation feedback are customizable through the properties and methods of the constraint validation API. In addition to the validationMessage and willValidate properties, the validity object is also a part of this API. The validity object contains several properties; if all of these properties have a value of false, then the value of the validity object is true.



Methods of the constraint validation API –

* checkValidity() – Returns true for a control if the control value is valid
* setCustomValidity() – Sets custom validation message and rules

In conjunction with these properties and methods ,you can use the CSS :invalid and :valid pseudo-classes to change the properties of form elements based on their validity status. Using all of these tools, you can customize the behavior of browser-based validation so users get similar feedback in all modern browsers.

Example – The following code checks the valueMissing property of the fname element, and if the value is true, it sets the bubble text to “Please fill out this field”.

var fname = document.getElementById("firstName");

if(fname.valueMissing){

setCustomValidity("Please fill out this field");

}

The addition of the following CSS sets the background of the field to a pink color as long as the validity state of the field is invalid:

#firstName:invalid{

background: rgb(255, 233, 233);

}

Customizing browser-based validation feedback (cont'd.)

* + Bubble appearance varies among browsers
  + Cannot set multiple validation messages for a single field at once
  + If creating a uniform appearance in every browser is a goal, you can instead use the preventDefault() method and the invalid event to keep the Bubbles from being displayed in any browser. If disabled, must program custom validation for displaying feedback to users.

In preparation for writing custom validation functions for all browsers, you’ll add the novalidate attribute to the form element to disable browser-based validation in modern browsers.

To disable browser-based validation for the order form:

Within the opening <form> tag, add the attribute novalidate=”novalidate”.

<form action="results.htm" novalidate="novalidate">

## Programming Custom Validation

* Common validation functions:
  + Checking that required fields contain entries
  + Checking values dependent on other fields
  + Checking for appropriate content type

### 

### Validating Submitted Data

In chapter 1 and 2, you learned about many of the events that you can listen for in JavaScript. One event, submit, is available for use with the form element. The submit event fires when a form is submitted. A form data is often verified or validated when the submit event fires, before the data is sent to a server.

You’ll call all of your custom validation functions from a main function, which will determine whether the form passes validation – if so, the function will submit the form contents.

To create the main validation function:

Within the createEventListener() function add the following code –

var form = document.getElementsByTagName("form")[0];

if(form.addEventListener){

form.addEventListener("submit",validateForm,false);

}else if(form.attachEvent){

form.attachEvent("onsubmit",validateForm);

}

This code triggers the validateForm() function, which you’ll create in a later step.

In the global variable section at the top of the snoot.js document, add the following –

var formValidity = true;

Create the validateForm() function –

/\* validate form \*/

function validateForm(evt){

if(evt.preventDefault){

evt.preventDefault();//prevent form from submitting(prevents the default behaviour )

}else{

evt.returnValue = false;//prevent form from submitting in IE8

}

formValidity = true;//reset value for revalidation

if(formValidity === true){

document.getElementById("errorText").innerHTML = "";

document.getElementById("errorText").style.display = "none";

document.getElementsByTagName("form")[0].submit(); //if data is valid

}else{

document.getElementById("errorText").innerHTML = "Please fix the indicated problems and then resubmit your order.";

document.getElementById("errorText").style.display = "block";

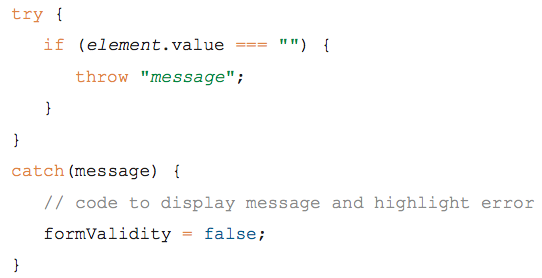
scroll(0,0);

}

}

### Validate Required Fields with Custom Functions

Retrieve values of required fields, then check if any is empty



* Checking for Empty Text Input Field

You use different properties to validate each form element and input type. For text boxes – including any object created with an input element can use the value property to check for a value.

Most commonly used properties of input elements –

Accept, alt, autocomplete, checked, defaultChecked, defaultValue, files, list, max, min, multiple, placeholder, size, type,…

Most commonly used input element methods –

Blur(), click(), focus(), select()

Example – to check if the input element with the id value firstName is blank, you could use the following statement:

if (document.getElementById("firstName").value === "") {

// code to run if the field is blank

}

Use loop statement to check each field in a group.

To create a custom validation function for input field in the Billing Address fieldset:

/\* validate address fieldsets \*/

function validateAddress(fieldsetId){

var inputElements = document.querySelectorAll("#" + fieldsetId + " input");

var errorDiv = document.querySelectorAll("#" + fieldsetId + "

.errorMessage")[0];

var fieldsetValidity = true;

var elementCount = inputElements.length;

var currentElement;

try{

for(var i = 0; i < elementCount; i++){

//validate all input elements in fieldset

currentElement = inputElements[i];

if(currentElement.value === ""){

currentElement.style.background = "rgb(255,233,233)";

fieldsetValidity = false;

}else{

currentElement.style.background = "white";

}

}

if(fieldsetValidity === false){

//throw appropriate message based on current fieldset

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 = "";

}

}catch(msg){

errorDiv.style.display = "block";

errorDiv.innerHTML = "msg";

formValidity = false;

}

}

In the validateForm() function, replace the code with the two method calls validateAddress() with the fieldsets Id “billingAddress” and “deliveryAddress”

/\* validate form \*/

function validateForm(evt){

if(evt.preventDefault){

evt.preventDefault();//prevent form from submitting(prevents the default behaviour )

}else{

evt.returnValue = false;//prevent form from submitting in IE8

}

formValidity = true;//reset value for revalidation

validateAddress("billingAddress");

validateAddress("deliveryAddress");

}

* Checking for Selection Lists with No Values

To validate whether a selection has been made in a selection list, you check the value of the selectIndex property. Recall from earlier in the chapter that the selectedIndex property represents the index value of the selected item within the array of option elements. If no option is selected in a selection list, the selectedIndex property for the select element is equal to -1. Retrieving this value is the easiest means to check whether a required selection list has been completed.

Example – to check if a user has made a selection in a select element

<label for="billState">State</label>

<select id="billState" name="BillingState" required="required">

<option value="AL">AL</option>

<option value="AK">AK</option>

<option value="AZ">AZ</option>

<option … .

if (document.getElementById("state").selectedIndex === -1 {

// code to run if the field is blank

}

To add validation for the State selection list to the validation function for the address fieldsets:

In the validateAddresss() function insert the following code –

currentElement = document.querySelector("#" + fieldsetId + " select");

//validate state select element

if(currentElement.selectedIndex === -1){

currentElement.style.border = "1px solid red";

fieldsetValidity = false;

}else{

currentElement.style.border = "";

}

* Checking for Option Button Sets with No Selection
  + Check value of checked property
  + Use And (&&) operators to check if no option button is selected

Example – check if none of the three option buttons with the name Color are selected:

var buttons = document.getElementsByName("Color");

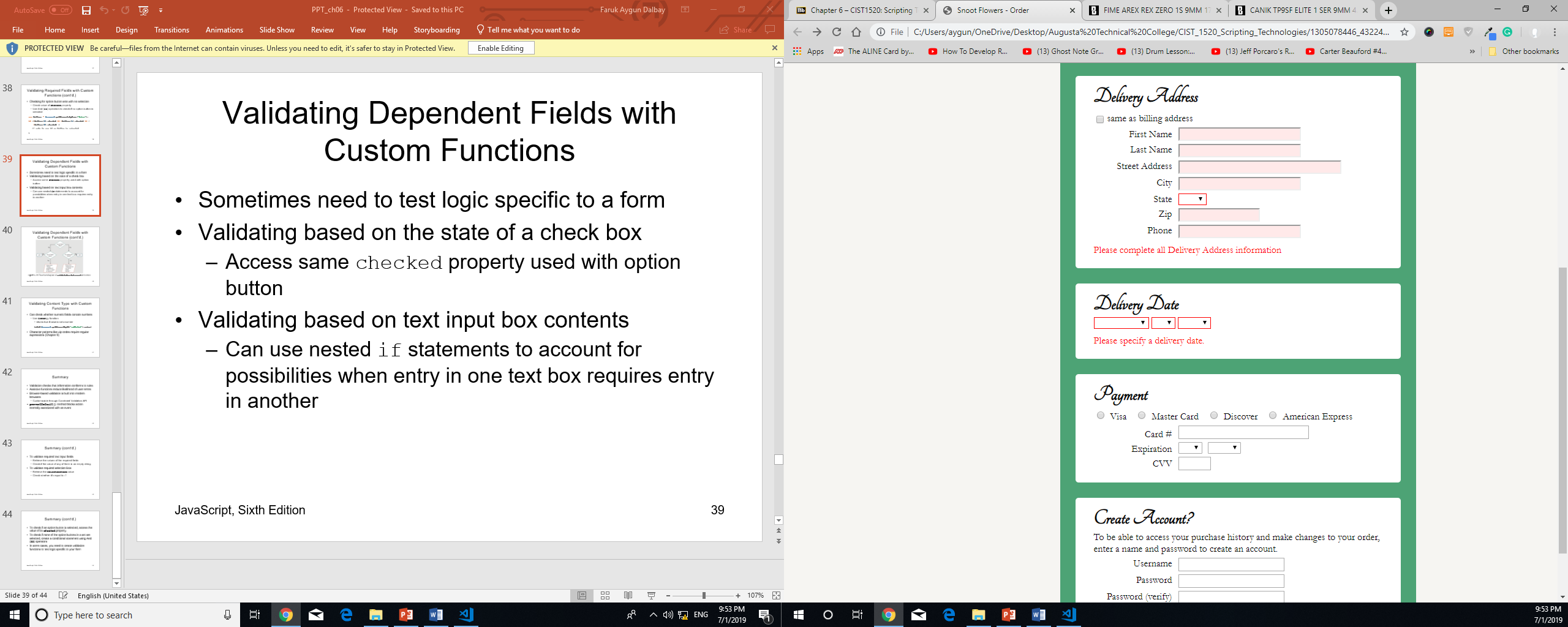
if (!buttons[0].checked && !buttons[1].checked && ↵

!buttons[2].checked) {

// code to run if no button is selected

}

To create a validation function for the Payment fieldset:



Above the validateForm() function, enter the following code:

<fieldset id="paymentInfo" class="text">

<legend>Payment</legend>

<div id="cards" class="inline">

<input id="visa" name="PaymentType" type="radio" value="Visa" />

<label for="visa">Visa</label>

<input id="mc" name="PaymentType" type="radio" value="MC" />

<label for="mc">Master Card</label>

<input id="discover" name="PaymentType" type="radio" value="Discover" />

<label for="discover">Discover</label>

<input id="amex" name="PaymentType" type="radio" value="AmEx" />

<label for="amex">American Express</label>

</div>

<div>

<label for="ccNum">Card #</label>

<input id="ccNum" name="CardNumber" type="number" required="required" />

<div id="ccNumErrorMessage"></div>

</div>

<div>

<label>Expiration</label>

<div class="inline" id="exp">

<label for="expMo" id="expMoLabel">Expiration Month</label>

<select id="expMo" name="ExpMonth" required="required">

<option value="01">01</option>

<option value="02">02</option>

<option value="03">03</option>

<option …

</select>

<label for="expYr" id="expYrLabel">Expiration Year</label>

<select id="expYr" name="ExpYear" required="required">

<option value="2017">2017</option>

<option value="2018">2018</option>

<option …

</select>

</div>

<label for="cvv">CVV</label>

<input id="cvv" name="CVVValue" type="number" required="required" />

<div id="cvvErrorMessage"></div>

</div>

<div class="errorMessage"></div>

</fieldset>

/\* validate payment fieldset \*/

function validatePayment(){

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;

try{

//verify that a card is selected

if(!cards[0].checked && !cards[1].checked && !cards[2].checked && !cards[3].checked){

for(i = 0; i < 4; i++){

cards[i].style.outline = "1px solid red";

}

fieldsetValidity = false;

}else{

for(i = 0; i < 4; i++){

cards[i].style.outline = "";

}

}

//verify that a card number has been entered

if(ccNumElement.value === ""){

ccNumElement.style.background = "rgb(255,233,233)";

fieldsetId = false;

}else{

ccNumElement.style.background = "white";

}

//verify that a month and year have been selected

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 = "";

}

}

//verify that a cvv value has been entered

if(cvvElement.value === ""){

cvvElement.style.background = "rgb(255,233,233)";

fieldsetValidity = false;

}else{

cvvElement.style.background = "white";

}

//check if any field is blank

if(!fieldsetValidity){

throw "Please complete all payment infromation";

}else{

errorDiv.style.display = "none";

}

}catch(msg){

errorDiv.style.display = "block";

errorDiv.innerHTML = msg;

formValidity = false;

}

}

In the validateForm() function, add a call to the validatePayment() function –

/\* validate form \*/

function validateForm(evt){

if(evt.preventDefault){

evt.preventDefault();//prevent form from submitting(prevents the default behaviour )

}else{

evt.returnValue = false;//prevent form from submitting in IE8

}

formValidity = true;//reset value for revalidation

validateAddress("billingAddress");

validateAddress("deliveryAddress");

validateDeliveryDate();

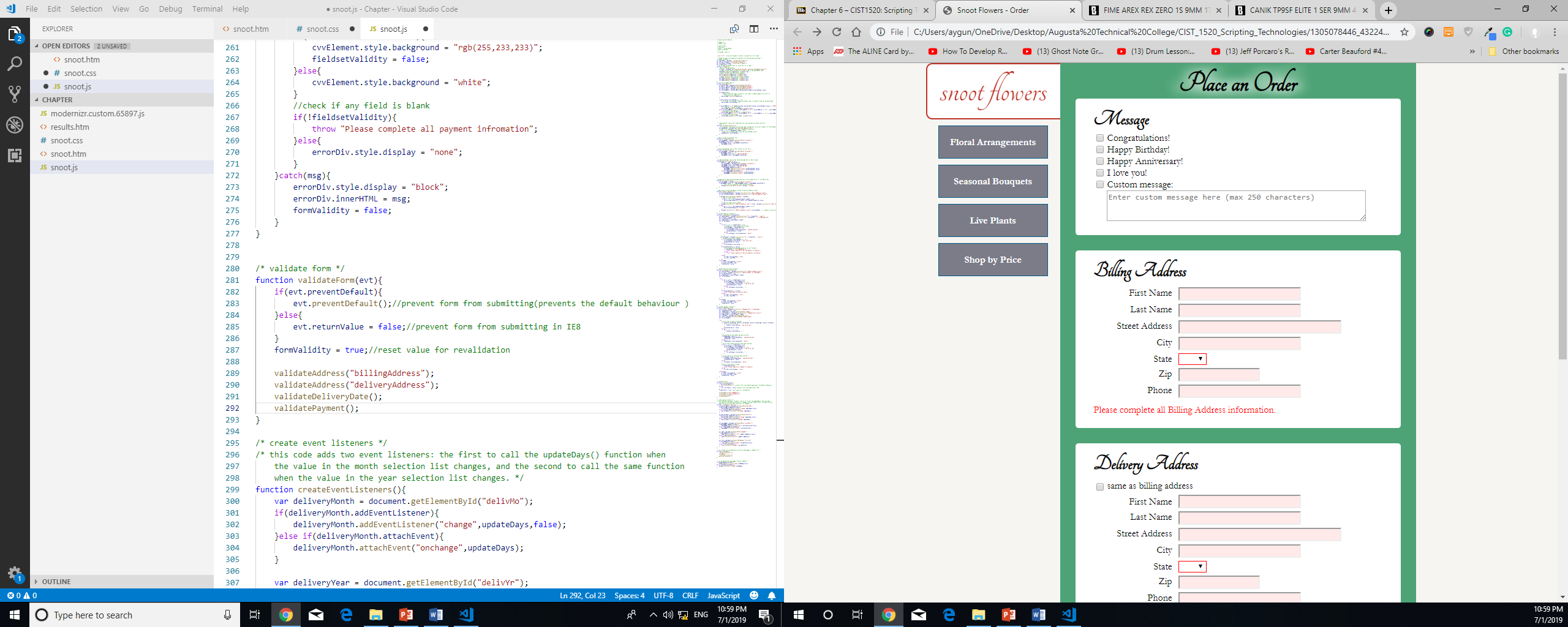
validatePayment();

}

### Validating Dependent Fields with Custom Functions

For instance, we could add validation logic that verifies that the text box contains an entry, but only if the “other” option button is selected.

To create a validation function for the Custom message check box and its dependent textarea control:



<fieldset id="message" class="checks">

<legend>Message</legend>

<input id="congrats" name="Congratulations" type="checkbox" />

<label for="congrats">Congratulations!</label>

<input id="bday" name="HappyBirthday" type="checkbox" />

<label for="bday">Happy Birthday!</label>

<input id="anniv" name="HappyAnniversary" type="checkbox" />

<label for="anniv">Happy Anniversary!</label>

<input id="love" name="ILoveYou" type="checkbox" />

<label for="love">I love you!</label>

<input id="custom" name="CustomMessage" type="checkbox" />

<label for="custom">Custom message:</label>

<textarea id="customText" name="CustomText" placeholder="Enter custom

message here (max 250 characters)"></textarea>

<div class="errorMessage"></div>

</fieldset>

/\* validate message fieldset \*/

function validateMessage(){

var errorDiv = document.querySelector("#message .errorMessage");

var msgBox = document.getElementById("customText");

try{

if(document.getElementById("custom").checked && ((msgBox.value === "") || (msgBox.value === msgBox.placeholder))){

//custom checked but message box empty

throw "Please enter your message text.";

}else{

errorDiv.style.display = "none";

msgBox.style.background = "white";

}

}catch(msg){

errorDiv.style.display = "block";

errorDiv.innerHTML = msg;

msgBox.style.background = "rgb(255,233,233)";

formValidity = false;

}

}

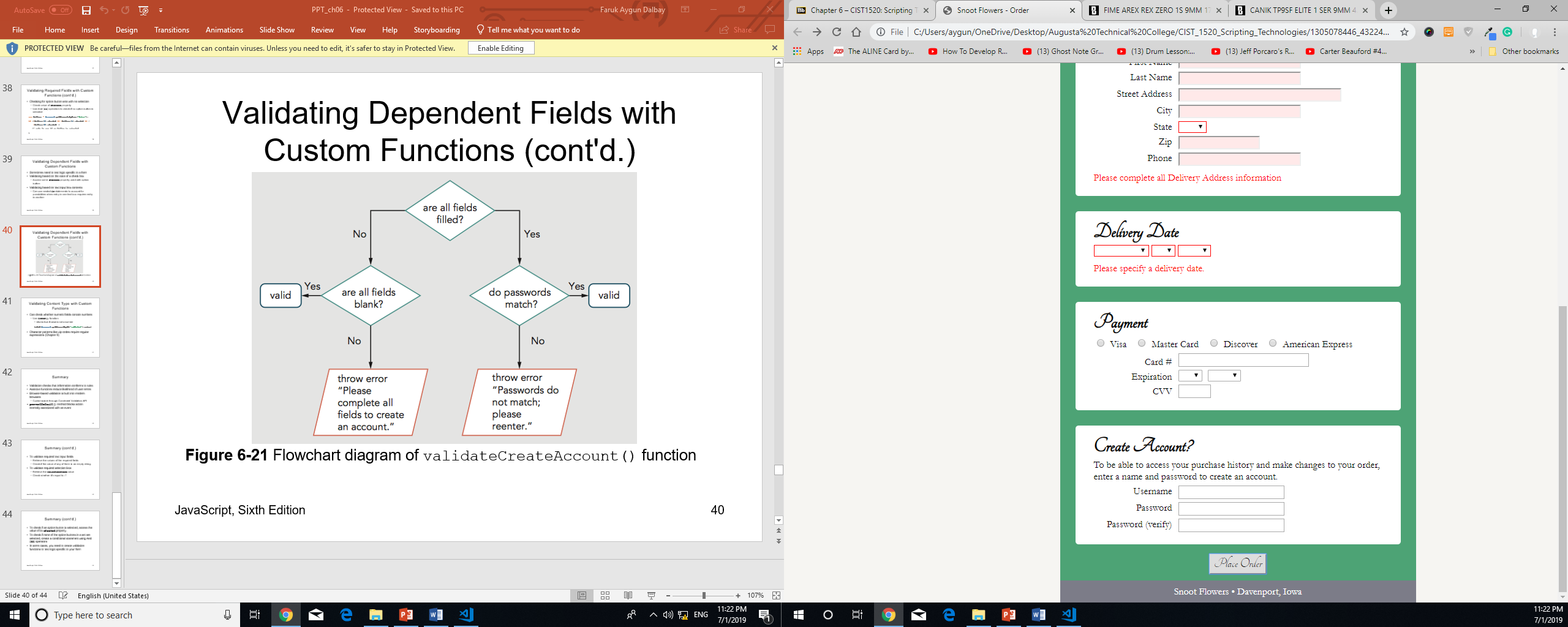
In the validForm() function, add a call to the validateMessage() function after the call to the validatePayment() function –

…;

validateMessage();

}

To create a validation function for the Create Account Fieldset:



<fieldset id="createAccount" class="text">

<legend>Create Account?</legend>

<p>To be able to access your purchase history and make changes to

your order, enter a name and password to create an account.</p>

<label for="username">Username</label>

<input id="username" name="NewUsername" type="text" />

<label for="pass1">Password</label>

<input id="pass1" name="Password1" type="password" />

<label for="pass2">Password (verify)</label>

<input id="pass2" name="Password2" type="password" />

<div class="errorMessage"></div>

</fieldset>

/\* validate create account fieldset \*/

function validateCreateAccount(){

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)";

try{

//reset styles to valid state

usernameElement.style.background = "";

pass1Element.style.background = "";

pass2Element.style.background = "";

errorDiv.style.display = "none";

if(usernameElement.value !== "" && pass1Element.value !== "" && pass2Element.value !== ""){

//all fields are filled

if(pass1Element.value !== pass2Element.value){

//password don't match

passwordMismatch = true;

throw "Passwords entered do not match; please reenter.";

}

}

if(!(usernameElement.value === "" && pass1Element.value === "" && pass2Element.value === "")){

//not all fields are blank

throw "Please complete all fields to create an account.";

}

}catch(msg){

errorDiv.innerHTML = msg;

errorDiv.style.display = "block";

if(passwordMismatch){

usernameElement.style.background = "";

pass1Element.style.background = invColor;

pass2Element.style.background = invColor;

}else{

if(usernameElement.value === ""){

usernameElement.style.background = invColor;

}

if(pass1Element.value === ""){

pass1Element.style.background = invColor;

}

if(pass2Element.value === ""){

pass2Element.style.background = invColor;

}

}

formValidity = false;

}

}

In the validationFrom() function, add a call to the validateCreateAccount() function after the call to the validateMessage() function –

validateCreateAccount();

### Validating Content Type with Custom Functions

* Can check whether numeric fields contain numbers
  + Use isNaN() function - returns true if value is not a number

isNaN(document.getElementById("subtotal").value)

* Character patterns like zip codes require regular expressions (Chapter 8)

To validate the Card # and CVV numeric fields:

/\* validate number fields for older browsers \*/

function validateNumbers(){

var ccNotNum;

var cvvNotNum;

var ccNumElement = document.getElementById("ccNum");

var cvvElement = document.getElementById("cvv");

var ccNumErrMsg = document.getElementById("ccNumErrorMessage");

var cvvErrMsg = document.getElementById("cvvErrorMessage");

try{

if(isNaN(ccNumElement.value) || ccNumElement.value === ""){

ccNotNum = true;

}else{

//ccNum value is a number

ccNumElement.style.background = "";

ccNumErrMsg.style.display = "none";

}

if(isNaN(cvvElement.value) || cvvElement.value === ""){

cvvNotNum = true;

}else{

//cvv values is a number

cvvElement.style.background = "";

cvvErrMsg.style.background = "none";

}

if(ccNotNum || cvvNotNum){

throw "must contain numbers only.";

}

}catch(msg){

if(ccNotNum){

ccNumElement.style.background = "rgb(255,233,233)";

ccNumErrMsg.style.display = "block";

ccNumErrMsg.innerHTML = "The card number " + msg;

}

if(ccNotNum){

cvvElement.style.background = "rgb(255,233,233)";

cvvErrMsg.style.display = "block";

cvvErrMsg.innerHTML = "The cvv number " + msg;

}

formValidity = false;

}

}

In the validateForm() function, add a call to the validateNumbers() functions after the call to the validateCreateAccount() function –

validateNumbers();

Hands-on Project 6-1

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Hands-on Project 6-1</title>

<link rel="stylesheet" href="styles.css" />

<script src="modernizr.custom.65897.js"></script>

</head>

<body>

<header>

<h1>

Hands-on Project 6-1

</h1>

</header>

<article>

<h2>Personal Information</h2>

<!--disable browser based validation-->

<form action="results.htm" novalidate="novalidate" >

<div id="errorText"></div>

<div id="numErrorText"></div>

<fieldset id="contactinfo">

<label for="addrinput">

Street Address

</label>

<input type="text" id="addrinput" name="Address" required="required" placeholder="number and street name" />

<label for="cityinput">

City

</label>

<input type="text" id="cityinput" name="City" required="required" />

<label for="stateinput">

State/Province

</label>

<input type="text" id="stateinput" name="State" required="required" />

<label for="zipinput">

Zip/Postal Code

</label>

<input type="number" id="zipinput" name="Zip" required="required" />

<label for="ssn1">

Social Security Number

</label>

<input type="number" id="ssn1" name="SSN1" class="ssn" maxlength="3" required="required" />

<label for="ssn2" id="ssn2label">

Social Security Number (continued)

</label>

<input type="number" id="ssn2" name="SSN2" class="ssn" maxlength="2" required="required" />

<label for="ssn3" id="ssn3label">

Social Security Number (end)

</label>

<input type="number" id="ssn3" name="SSN3" class="ssn" maxlength="4" required="required" />

</fieldset>

<fieldset id="submitsection">

<input type="submit" id="submitBtn" value="Submit" />

</fieldset>

</form>

</article>

<script src="script.js"></script>

</body>

</html>

---script.js –

"use strict";//interpret document contents in JavaScript strict mode

/\* global variables \*/

var formValidity = true;

/\* validate required fields \*/

function validateRequired(){

var inputElements = document.querySelectorAll("#contactinfo input");

var errorDiv = document.getElementById("errorText");

var elementCount = inputElements.length;

var requiredValidity = true;

var currentElement;

//validate all input elements in fieldset

try{

for(var i = 0; i < elementCount; i++){

currentElement = inputElements[i];

if(currentElement.value === ""){

currentElement.style.background = "rgb(255,233,233)";

requiredValidity = false;

}else{

currentElement.style.background = "white";

}

}

if(requiredValidity === false){

throw "Please complete all fields";

}

errorDiv.style.display = "none";

errorDiv.innerHTML = "";

}catch(msg){

errorDiv.style.display = "block";

errorDiv.innerHTML = msg;

formValidity = false;

}

}

/\* create event listeners \*/

function createEventListeners(){

var form = document.getElementsByTagName("form")[0];

if(form.addEventListener){

form.addEventListener("submit",validateForm,false);

}else if(form.attachEvent){

form.attachEvent("onsubmit",validateForm);

}

}

/\* trigger validation of required fields when the submit button is clicked \*/

function validateForm(evt){

if(evt.preventDefault){

evt.preventDefault(); //prevent form from submitting(prevents default behaviour)

}else{

evt.returnValue = false; //prevent form from submitting in IE8

}

formValidity = true; //reset value for revalidation

validateRequired(); // will validate or fail and set the value for validation

if(formValidity === true){

document.getElementsByName("form")[0].submit();

}

}

/\* run the setup function when page finishes loading \*/

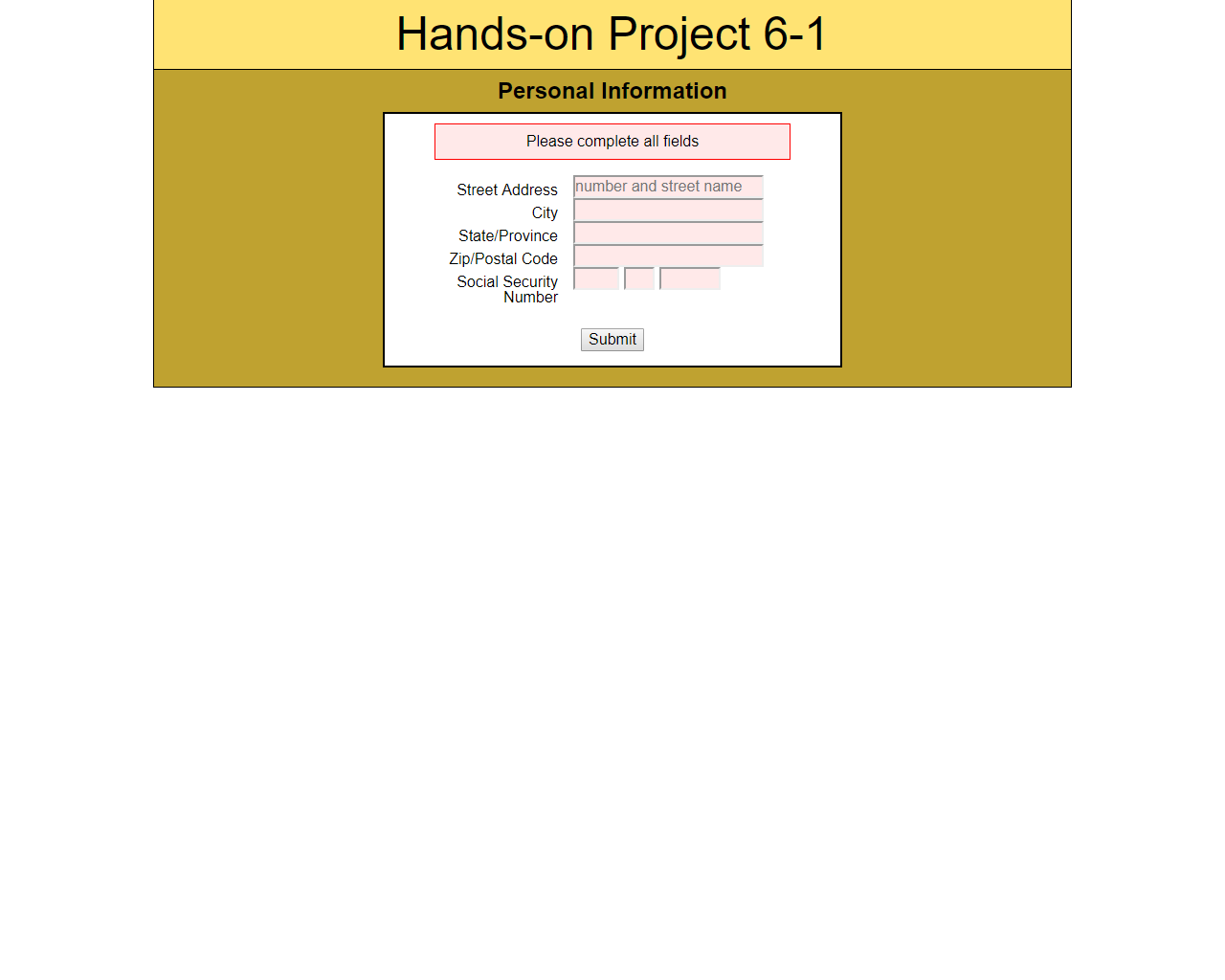
if(window.addEventListener){

window.addEventListener("load",createEventListeners,false);

}else if(window.attachEvent){

window.attachEvent("onload",createEventListeners);

}



To validate input elements with the number type for older browsers -

/\* validate number fields for older browsers \*/

function validateNumbers(){

var numberInputs = document.querySelectorAll("#contactinfo **input[type=number]"**);

var elementCount = numberInputs.length;

var numErrorDiv = document.getElementById("numErrorText");

var numbersValidity = true;

var currentElement;

try{

for(var i = 0; i < elementCount; i++){

//validate all input elements of type "number" in fieldset

currentElement = numberInputs[i];

if(isNaN(currentElement.value) || (currentElement.value === "")){

currentElement.style.background = "rgb(255,233,233)";

numbersValidity = false;

}else{

currentElement.style.background = "white";

}

}

if(numbersValidity === false){

throw "Zip and Social Security values must be numbers.";

}

numErrorDiv.style.display = "none";

numErrorDiv.innerHTML = "";

}catch(msg){

numErrorDiv.style.display = "block";

numErrorDiv.innerHTML = msg;

formValidity = false;

}

}

In the validateForm() function, add a call to the validateNumbers() function as follows:

/\* trigger validation of required fields when the submit button is clicked \*/

function validateForm(evt){

if(evt.preventDefault){

evt.preventDefault(); //prevent form from submitting(prevents default behaviour)

}else{

evt.returnValue = false; //prevent form from submitting in IE8

}

formValidity = true; //reset value for revalidation

validateRequired(); // will validate or fail and set the value for validation

**validateNumbers();**

if(formValidity === true){

document.getElementsByName("form")[0].submit();

}

}