**Form Validation**

Required attribute

If required attribute in HTML5 is disabled (to create personalised styling), see example below to replicate using javascript.

Function validateRequired(el) {

If(isRequired(el)) {

Var valid = !isEmpty(el); //is value not empty

If(!valid) { //if valid =false

setErrorMessage(el, ‘field is required’);

}

Return valid; //returns Boolean

}

Return true; //if not required

}

Function isRequired(el) {

Return ((typeof el.required === ‘boolean’) && el.required);

}

*\*\*The typeOf operator is used to find the type of a javascript variable – i.e. number, string, Boolean\*\**

*e.g. var number = 17.623;*

*if(typeof(number)===”number”) {*

*alert(“it worked”);*

*}*

Function isEmpty(el) {

Return !el.value;

} //returns Boolean

Creating error message

Stored in jQuerys .data() method

Function setErrorMessage(el, message) { //see 1

$(el).data(‘error message’, message); //see 2

}

1. 2 parameters
   * 1. El: the element the error message is for
     2. Message: the text the error message will display
2. 2 parameters
   * 1. The key: ‘error message’
     2. The value: error message displayed

Displaying error messages

After each element has been checked, if one or more were not valid, showErrorMessage() will display the error message on the page.

1. If an error message needs to be shown, first a <span> element will be added to the page directly after the form field with an error.
2. Message then added in to <span> element To get the text for the error message, the same jQuery .data() method that set the message is used again.

Function showErrorMessage(el) {

Var $el = $(el);

Var $errorContainer = $el.siblings(‘.error’);

If(!errorContainer.length) { //if no error currently

$errorContainer = $(‘<span class = “error”></span>’).insertAfter($el);

}

$errorContainer.text($(el).data(‘errorMessage’));

}

Validating different types of input

Function validateTypes(el) {

If(!el.value) return true;

Var type = $(el).data(‘type’) || el.getAttribute(‘type’);

If(typeof validateType[type] === ‘function’) {

Return validateType[type](el);

}else{

Return true;

};

};

1. Above checks if the element has a value – if the user has not entered any information, you cannot validate the type of data.
2. If there is a value, a variable called type holds the value of the type attribute (p618 more info on jquery .data() method)
3. This function uses an object called validateType (show on next page), to check the content of the element. The if statement checks if the validateType object has a method whose name matches the value of the type attribute. If it does then:
4. The element is passed to the object, it returns a Boolean
5. If no matching method, the object is not able to validate the form control and no error message should be set

Creating an object to validate data types

The validateType object has 3 methods:

Email: function(el) {

//check email address

}

Number: function(el) {

//check number

}

Date: function(el) {

//check date

}

Each method validates the data using something called a *regular expression*. The regular expression is the only thing that changes in each method to test the different data types.

Regular expressions (p612) allow you to check for patterns in strings, and here they are used with a method called test().

e.g. function(el) {

if valid = /[^@]+@[^@]+/ .test(el.value);

if(!valid) {

setErrorMessage(el, ‘Please enter a valid email’);

}

Return valid;

}

1. A variable called *valid* holds the result of the test using the regular expression
2. If the string doesn’t contain a match for the regular expression:
   1. An error message is set
3. The function returns the value of the valid variable. (true or false).

The test() method takes one parameter and checks whether the regular expression can be found within the string. It returns a Boolean.

Var validateType = {

Email: function(el) {

if valid = /[^@]+@[^@]+/ .test(el.value);

if(!valid) {

setErrorMessage(el, ‘Please enter a valid email’);

};

Return valid;

},

date: function(el) {

if valid = /^(\d{2}\/\d{2}\/\d{4}|(\d{4}-\d{2}-\d{2})$/ .test(el.value);

if(!valid) {

setErrorMessage(el, ‘Please enter a valid date’);

};

Return valid;

},

number: function(el) {

if valid = /^\d+$/ .test(el.value);

if(!valid) {

setErrorMessage(el, ‘Please enter a valid number’);

};

Return valid;

}

};

Regular expressions

Regular expressions search for characters that form a pattern. They can also replace those characters with new ones.

They also check for sequences of upper/lowercase characters, numbers, punctuations and other symbols.

|  |  |
| --- | --- |
| . | Any single character (except new line) |
| [ ] | Single character contained within brackets |
| [^ ] | Single characters not contained within brackets |
| ^ | The starting position in any line |
| $ | The ending position in any line |
| ( ) | sub expressions |
| \* | Preceding element zero or more times |
| \n | Nth marked sub expression (n is digit 1-9) |
| {m, n} | Preceding element at least m, but no more than n, times |
| \d | Digit |
| \D | Non-digit character |
| \s | White space character |
| \S | Anything but white space character |
| \w | Alpha numeric character (A-Z, a-z, 0-9) |
| \W | Non-alpha numeric character (except \_) |

Common regular expressions

|  |  |
| --- | --- |
| /^\d+$/ | Number |
| /^[\s]+/ | Whitespace at start of line |
| /[^@]+@[^@]+/ | Email |
| /^#[a-fA-fO-9]{6}$/ | Hex color value |
| /^(\d{2}\/\d{2}\/\d{4}|(\d{4}-\d{2}-\d{2})$/ | Date yy-mm-dd |

Custom validation

The final part of the script performs 3 checks that apply to individual form elements; each live in a named function.

1. validateBio()
2. validatePassword()
3. validateParentsConsent()

if(!validateBio()) {

showErrorMessage(document.getElementById(‘bio’));

valid.bio=false;

}else{

removeErrorMessage(document.getElementById(‘bio’));

}

1. call validateBio(), if not valid…
2. Show error message and valid object set to false
3. If function returns true then error message is removed

Bio Validation

Function validateBio() {

Var bio = document.getElementById(‘bio’);

Var valid = bio.value.length <=140;

If(!valid) {

setErrorMessage(bio, ‘your bio should not exceed 140 characters’);

}

Return valid; //returns Boolean value

}

Password validation

Function validatePassword() {

Var password = document.getElementById(‘password’);

Var valid = password.value.length >=8;

If(!valid) {

setErrorMessage(password, ‘Password must be at least 8 characters’);

}

Return valid;

}

Parental consent validation

Function validateParentsConsent() {

Var parentsConsent = document.getElementById(‘parents-consent’);

Var consentContainer = document.getElementById(‘consent-container’)

Var valid = true;

If(!consentContainer.className.indexOf(‘hide’)=== -1) {

Valid = parentsConsent.checked;

If(!valid) {

setErrorMessage(parentsConsent, ‘you need your parents\’ consent’);

};

};

Return valid;

};

1. Stores checkbox and containing elements in variables.
2. Variable valid is set to true.
3. If statement checks if container is hidden, using indexOf() (p218) – if value not found, then indexOf() will return ‘-1’.
4. If not hidden, the user is under 13, so, if the checkbox is selected the variable is set to true, and if it was not selected, it will be set to false.
5. If not valid – an error message is added to the element.
6. The function returns the value of the valid variable to indicate whether the consent was given.

Hide parental consent / age confirmation

As you saw in the previous example, the subscription form uses two extra scripts to enhance the user experience.

* 1. Uses JQuery UI date picket to show a consistent date picker across browsers
  2. Checks whether the parental consent checkbox should be shown when the user leaves the date input (if under 13yrs old)

(function() {

Var $birth = $(‘#birthday’);

Var $parentsConsent = $(‘#parents-consent’);

Var $consentContainer = $(‘#consent-container’);

//create the date picker using JQuery UI

$birth.prop(‘type’, ‘text’).data(‘type’,’date’).datepicker( {

dateFormat:’yy-mm-dd’

});

$birth.on(‘blur change’, checkDate);

Function checkDate() {

Var dob = this.value.split(‘-‘);

//pass toggleParentsConsent() the date of birth as a date object

toggleParentsConsent(new Date(dob[0], dob[1] -1, dob[2]));

};

Function toggleParentsConsent(date) {

If(isNaN(date)) return;

Var now = new Date();

//if difference (now minus date of birth, is less than 13 years

//show parents consent checkbox (does not account for leap years

//to get 13 years ms\*secs\*mins\*hrs\*days\*years

If((now – date) < (1000\*60\*60\*24\*13)) {

$consentContainer.removeClass(‘hide’);

$parentsConsent.focus();

}else{

$consentContainer.addClass(‘hide’);

$parentsConsent.prop(‘checked’, false);

}

}

}());

\*\*look at p618-619 for full breakdown\*\*

Password feedback

* 1. Provides feedback to the users as they leave either of the password inputs. It changes the value of the class attribute for the password inputs, offering feedback to show whether or not the password is long enough and whether or not the value of the password and its confirmation box match.

(function() {

Var password = document.getElementById(‘password’);

Var passwordConfirm = document.getElementById(‘conf-password’);

Function setErrorHighlighter(e) {

Var target = e.target;

If(target.value.length<8) {

Target.className = ‘fail’;

}else{

Target.className = ‘pass’;

}

}

Function removeErrorHighlighter(e) {

Var target = e.target;

If(target.className === ‘fail’) {

Target.className = ‘’;

}

}

Function passwordsMatch(e) {

Var target = e.target;

//if value matches pwd and if it is longer than 8 characters

If((password.value === target.value) && target.value.length>=8){

Target.className = ‘pass’;

}else{

Target.className = ‘fail’;

}

}

addEvent(password, ‘focus’, removeErrorHighlighter);

addEvent(password, ‘blur’, setErrorHighlighter);

addEvent(passwordConfirm, ‘focus’, removeErrorHighlighter);

addEvent(passwordConfirm, ‘blur’, passwordsMatch);

}());

\*\*Look at P620-621 for full breakdown\*\*

**Full code**

// JavaScript validation of subscription form.

// A. Anonymous function triggered by submit event

// B. Functions called to perform generic checks by anon function in section A

// C. Functions called to perform generic checks by anon function in section A

// D. Functions to get / set / show / remove error messages

// E. Object to check type of data using RegEx called by validateTypes in section B

// Dependencies: jQuery, jQueryUI, birthday.js, styles.css

(function () {

document.forms.register.noValidate = true; // Disable HTML5 validation - using JavaScript instead

// -------------------------------------------------------------------------

// A) ANONYMOUS FUNCTION TRIGGERERD BY THE SUBMIT EVENT

// -------------------------------------------------------------------------

$('form').on('submit', function (e) { // When form is submitted

var elements = this.elements; // Collection of form controls

var valid = {}; // Custom valid object

var isValid; // isValid: checks form controls

var isFormValid; // isFormValid: checks entire form

// PERFORM GENERIC CHECKS (calls functions outside the event handler)

var i;

for (i = 0, l = elements.length; i < l; i++) {

// Next line calls validateRequired() validateTypes()

isValid = validateRequired(elements[i]) && validateTypes(elements[i]);

if (!isValid) { // If it does not pass these two tests

showErrorMessage(elements[i]); // Show error messages

} else { // Otherwise

removeErrorMessage(elements[i]); // Remove error messages

} // End if statement

valid[elements[i].id] = isValid; // Add element to the valid object

} // End for loop

// PERFORM CUSTOM VALIDATION

// bio (you could cache bio input in variable here)

if (!validateBio()) { // Call validateBio(), and if not valid

showErrorMessage(document.getElementById('bio')); // Show error message

valid.bio = false; // Update valid object - this element is not valid

} else { // Otherwise remove error message

removeErrorMessage(document.getElementById('bio'));

}

// password (you could cache password input in variable here)

if (!validatePassword()) { // Call validatePassword(), and if not valid

showErrorMessage(document.getElementById('password')); // Show error message

valid.password = false; // Update the valid object - this element is not valid

} else { // Otherwise remove error message

removeErrorMessage(document.getElementById('password'));

}

// parental consent (you could cache parent-consent in variable here)

if (!validateParentsConsent()) { // Call validateParentalConsent(), and if not valid

showErrorMessage(document.getElementById('parents-consent')); // Show error message

valid.parentsConsent = false; // Update the valid object - this is not valid

} else { // Otherwise remove error message

removeErrorMessage(document.getElementById('parents-consent'));

}

// DID IT PASS / CAN IT SUBMIT THE FORM?

// Loop through valid object, if there are errors set isFormValid to false

for (var field in valid) { // Check properties of the valid object

if (!valid[field]) { // If it is not valid

isFormValid = false; // Set isFormValid variable to false

break; // Stop the for loop, an error was found

} // Otherwise

isFormValid = true; // The form is valid and OK to submit

}

// If the form did not validate, prevent it being submitted

if (!isFormValid) { // If isFormValid is not true

e.preventDefault(); // Prevent the form being submitted

}

}); // End of event handler anon function

// END: anonymous function triggered by the submit button

// -------------------------------------------------------------------------

// B) FUNCTIONS FOR GENERIC CHECKS

// -------------------------------------------------------------------------

// CHECK IF THE FIELD IS REQUIRED AND IF SO DOES IT HAVE A VALUE

// Relies on isRequired() and isEmpty() both shown below, and setErrorMessage - shown later.

function validateRequired(el) {

if (isRequired(el)) { // Is this element required?

var valid = !isEmpty(el); // Is value not empty (true / false)?

if (!valid) { // If valid variable holds false

setErrorMessage(el, 'Field is required'); // Set the error message

}

return valid; // Return valid variable (true or false)?

}

return true; // If not required, all is ok

}

// CHECK IF THE ELEMENT IS REQUIRED

// It is called by validateRequired()

function isRequired(el) {

return ((typeof el.required === 'boolean') && el.required) ||

(typeof el.required === 'string');

}

// CHECK IF THE ELEMENT IS EMPTY (or its value is the same as the placeholder text)

// HTML5 browsers do allow users to enter the same text as placeholder, but in this case users should not need to

// It is called by validateRequired()

function isEmpty(el) {

return !el.value || el.value === el.placeholder;

}

// CHECK IF THE VALUE FITS WITH THE TYPE ATTRIBUTE

// Relies on the validateType object (shown at end of IIFE)

function validateTypes(el) {

if (!el.value) return true; // If element has no value, return true

// Otherwise get the value from .data()

var type = $(el).data('type') || el.getAttribute('type'); // OR get the type of input

if (typeof validateType[type] === 'function') { // Is the type a method of validate object?

return validateType[type](el); // If yes, check if the value validates

} else { // If not

return true; // Return true because it cannot be tested

}

}

// -------------------------------------------------------------------------

// C) FUNCTIONS FOR CUSTOM VALIDATION

// -------------------------------------------------------------------------

// IF USER IS UNDER 13, CHECK THAT PARENTS HAVE TICKED THE CONSENT CHECKBOX

// Dependency: birthday.js (otherwise check does not work)

function validateParentsConsent() {

var parentsConsent = document.getElementById('parents-consent');

var consentContainer = document.getElementById('consent-container');

var valid = true; // Variable: valid set to true

if (consentContainer.className.indexOf('hide') === -1) { // If checkbox shown

valid = parentsConsent.checked; // Update valid: is it checked/not

if (!valid) { // If not, set the error message

setErrorMessage(parentsConsent, 'You need your parents\' consent');

}

}

return valid; // Return whether valid or not

}

// Check if the bio is less than or equal to 140 characters

function validateBio() {

var bio = document.getElementById('bio');

var valid = bio.value.length <= 140;

if (!valid) {

setErrorMessage(bio, 'Please make sure your bio does not exceed 140 characters');

}

return valid;

}

// Check that the passwords both match and are 8 characters or more

function validatePassword() {

var password = document.getElementById('password');

var valid = password.value.length >= 8;

if (!valid) {

setErrorMessage(password, 'Please make sure your password has at least 8 characters');

}

return valid;

}

// -------------------------------------------------------------------------

// D) FUNCTIONS TO SET / GET / SHOW / REMOVE ERROR MESSAGES

// -------------------------------------------------------------------------

function setErrorMessage(el, message) {

$(el).data('errorMessage', message); // Store error message with element

}

function getErrorMessage(el) {

return $(el).data('errorMessage') || el.title; // Get error message or title of element

}

function showErrorMessage(el) {

var $el = $(el); // The element with the error

var errorContainer = $el.siblings('.error.message'); // Any siblings holding an error message

if (!errorContainer.length) { // If no errors exist with the element

// Create a <span> element to hold the error and add it after the element with the error

errorContainer = $('<span class="error message"></span>').insertAfter($el);

}

errorContainer.text(getErrorMessage(el)); // Add error message

}

function removeErrorMessage(el) {

var errorContainer = $(el).siblings('.error.message'); // Get the sibling of this form control used to hold the error message

errorContainer.remove(); // Remove the element that contains the error message

}

// -------------------------------------------------------------------------

// E) OBJECT FOR CHECKING TYPES

// -------------------------------------------------------------------------

// Checks whether data is valid, if not set error message

// Returns true if valid, false if invalid

var validateType = {

email: function (el) { // Create email() method

// Rudimentary regular expression that checks for a single @ in the email

var valid = /[^@]+@[^@]+/.test(el.value); // Store result of test in valid

if (!valid) { // If the value of valid is not true

setErrorMessage(el, 'Please enter a valid email'); // Set error message

}

return valid; // Return the valid variable

},

number: function (el) { // Create number() method

var valid = /^\d+$/.test(el.value); // Store result of test in valid

if (!valid) {

setErrorMessage(el, 'Please enter a valid number');

}

return valid;

},

date: function (el) { // Create date() method

// Store result of test in valid

var valid = /^(\d{2}\/\d{2}\/\d{4})|(\d{4}-\d{2}-\d{2})$/.test(el.value);

if (!valid) { // If the value of valid is not true

setErrorMessage(el, 'Please enter a valid date'); // Set error message

}

return valid; // Return the valid variable

}

};

}()); // End of IIFE