WEBD-1000

Project 2 Form Validation

📜 *ePortfolio Candidate*

**Download the necessary starter files from Learn.** Failure to use the appropriate starting files will result in a mark of 0 for this project.**Note: in this project there is no need to make any changes to the markup or the styles**. All you have to do is add code to **formvalidate.js.**

Code should be indented appropriately and semicolons always used. All other coding practices are up to the discretion of your instructor and will be covered in class.

Before you start coding, study the markup and look for the different classes and ids assigned to the elements.

Also included in the starting files:

**Order Form Cheatsheet.pdf**

The document contains methods and techniques that could prove very helpful in completing this project.

**utilityfunctions.js**

This file contains methods you can use to make validation easier. Review these functions by reading the document. If you do not understand how these functions work, ask your instructor.

**formvalidate.js**

This file contains variables, started functions and other code to start you off on this project.

### Tasks

1. Ensure that at least one product has been added to the shopping cart.

2. Validate the shipping and payment fields. If *any* errors are made during the form submission process, the appropriate error should be displayed (check the markup and demos).

When the function loads, call a function called **hideErrors()** that will hide all the error messages.

**load()**

Create event listeners for the submit and reset buttons. The submit event should trigger the **validate()** function, and the reset button should trigger the **resetForm()** function, which has already been coded for you (still, look it over and see what it does).

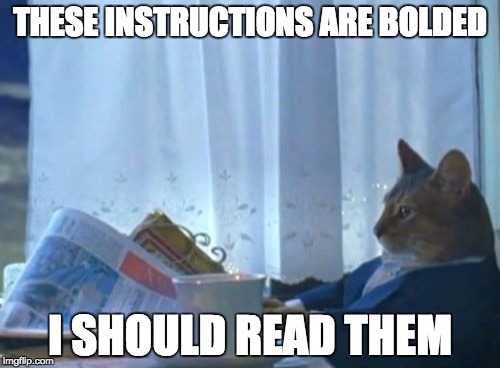
There are 5 buttons that will add their own product to the cart. Write event listeners for each of these that will call a function called **addItemToCart()**, and pass to the function its unique item number (“1” for Mac, “2” for Razer Mouse, “3” for WD EHDD, “4” for Nexus, and “5” for Drums).

**Hint:** to do this, you’re going to have to use anonymous functions (check the demos!).

**validate() and formHasErrors()**

Like the demo for form validation, these two functions work together to ensure only valid data is passed to the server.

The validate() function is to call the formHasErrors() function, and based on the value returned, will determine if the submission process should continue.

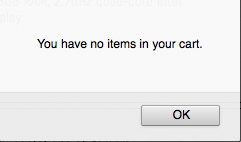


**\*Remember, even one error means the form should NOT submit. Failure to meet this criteria will result in the immediate loss of a mark. In other words, halting the submission process is crucial when there are errors in the user data.**

formHasErrors(), again like the demo, is where the majority of your code for this project will go. Check below to see the specifics of what and how you are to evaluate the user entered data.

**Shopping Cart Validation**

The shopping cart must contain at least one item to process an order. If the form is submitted without an item in the shopping cart, display the following message box:



After the message box disappears, set focus to the first quantity field. You should set focus to the first error on the form just like we did in the example.

**Shipping Information Validation**

Validate the customer’s shipping information as follows:

| Full Name | Required field |
| --- | --- |
| Address | Required field |
| City | Required field |
| Postal Code | Required field, Valid Postal Code |
| Email | Required field, Valid Email Address |

Use Regular Expressions to validate the Postal Code and Email text inputs.

**Payment Information Validation**

Validate the customer’s payment information as follows:

| Card Type | Required that one is selected |
| --- | --- |
| Name on Card | Required field |
| Expiry Date (Month) | Required field |
| Expiry Date | Valid expiry date to ensure the card being used has not expired |
| Card Number | Required field, Valid Card Number |

**NOTE:** regarding the expiry date, you MUST use the [date object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date) to ensure that your date check is dynamic.

**Modulus Check Logic**

For this project, we will be taking in 10 digit credit card numbers. Ensure the user entered only 10 numbers. If they pass this check, perform a modulus check to ensure that the user has submitted a valid credit card number.

Note that the **checking factors** below are the *literal* checking factors you will use in your solution.

| Sample Credit Card Number | 9434578423 |
| --- | --- |
| Checking Factors | 432765432 |
| Multiply Checking Factors by Credit Card digits and Sum | (9 \* 4) + (4 \* 3) + (3 \* 2) + (4 \* 7) + (5 \* 6) + (7 \* 5) + (8 \* 4) + (4 \* 3) + (2 \* 2) = 195 |
| Divide the Sum by 11 | 195 / 11 = 17, gives a remainder of 8 |
| Subtract | 11 - 8 = 3 (this is the check digit) |
| Compare the check digit to the last digit in the credit card number | 9434578423 |

The credit card number is valid when the check digit and the last digit in the credit card number are equal.

The following valid credit card numbers can be used for testing:

1111111118

2222222225

4111313135

2234321326

Note:

The “%” is the mathematical operator used to get the remainder.

Examples:

10 % 5 = 0, there is no remainder

10 % 3 = 1, there is a remainder of 1

## Project Suggestions

* This can be a long project. Be patient and **ask questions!** Don’t get yourself stuck for too long without asking for help.
* Anytime in class ask your instructor to see the full working version if you need to.
* Test as often as possible and after every section of code you write. Try to fix any errors before moving onto the next section of code.
* If you wish to create further functions, feel free!

## Rubric

| **Deduction Level** | **5 Marks** | **2 Marks** | **1 Mark** | **0 Marks** |
| --- | --- | --- | --- | --- |
| Javascript | Syntax Errors  Starting files not used  Project not submitted | Form submits despite errors in data  Form doesn’t catch errors and submits | Focus not applied to the first error in the form  Form submits despite one error in data  Error messages not displayed  No modulus check done for credit card  Regular expressions not used (postal code and email) | All validations done correctly  Error messages displayed properly  Form submits when all errors have been fixed  Form doesn’t submit when there’s an error |