Project 3: Interactive Form Study Guide

Sections Of This Guide:

- **How to approach this project** includes detailed guidance to help you think about how to organize your code, project, and files.
- **How to succeed at this project** lists the grading requirements for the project, with links to course videos to refresh your memory and helpful resources.

How To Approach This Project

With a lot of moving parts, user interaction, and complex DOM structures, forms are all about the details. A good development process for this and other projects is to write one or two lines of code, and then log out your values to ensure the code is what you think it is, and that it's doing what you think it's doing. Log out variables, iterators, and conditional clause values to get a look at what's actually going on with your code every step of the way. What you want to avoid is writing ten or fifteen lines of code just to find out that somewhere in your code hides a problem that needs to be hunted down. For additional powerful debugging techniques, check out this great workshop, **Debugging JavaScript in the Browser**.

This project, like most, can be broken down into smaller steps or sections that can be completed mostly independent of one another. So just take it slow, one small step at a time. And start at the beginning.

Setup your files

- Create a `script.js` file in the `js` folder, and make sure it is linked to the `index.html` file with a `<script></script>` tag, just before the closing `</body>` tag in the HTML file.

Put the first field in the `focus` state

- Use JavaScript to select the 'Name' input element and place focus on it.

Add an "Other" option to the Job Role section

This is the one and only section of the project where you will have to make changes directly in the `index.html` file.

Need help? Visit the unit-03 Slack channel

- In the `index.html` file, just below the 'Job Role' `select` element, create a text input element, set its `name` attribute to "job_role_other", set its `placeholder` attribute to "Your Job Role", and give it an `id` attribute of "other-title" so you can easily target this element in your JS file.
- In your JavaScript file, target the 'Other' input field, and hide it initially, so that it will display if JavaScript is disabled, but be hidden initially with JS.

T-Shirt section

The goal for the t-shirt section is to filter the available "Color" options by the selected theme in the "Design" field. Doing this ensures that the user cannot select an invalid combination of values for the "Design" and "Color" fields.

When the form is initially loaded, we need to update the "Color" field so that it's clear to the user that they need to select a theme before selecting a color. Use javaScript to:

- Update the "Color" field to read "Please select a T-shirt theme".
- Hide the colors in the "Color" drop-down menu.
- NOTE: Be sure to check out the helpful links in the second section of this Study Guide if you're unsure of how to accomplish these steps.

Then, when one of the two themes is selected, only the appropriate colors should show in the "Color" drop-down menu, and the "Color" field should update to the first available color. You'll use a `change` event listener on the "Design" menu `select` element to listen for changes. And inside the event listener, you'll use a conditional to determine what to hide, show, and update.

- If "js puns" is selected, hide the three "heart js" option elements in the "Color" drop-down menu, show the three "js puns" option elements, and update the "Color" field to the first available color.

- If "heart js" is selected, hide the three "js puns" option elements in the "Color" drop-down menu, show the three "heart js" option elements, and update the "Color" field to the first available color.

Activity section

Like many code problems, there are multiple ways to complete this section of the project. One option would be to simply reference each checkbox input, as well as the cost, and day and time from each input's parent `label` element, and store those values in variables, or in an object as key value pairs.

Then, in an event handler that listens for 'changes' to the activity section, you could use a set of conditionals to disable conflicting activities, and add or subtract from the total cost element you create, depending on whether the checkbox was checked or unchecked.

But a preferred approach would be to come up with a dynamic solution that will work even if the cost, day, or time of the activities were changed in the HTML. To do that, we'll:

- Create an element to display the total activity cost
- Listen for changes in the Activity section
- Create helpful variables to store important values
- Update and display the total activity cost
- Disable conflicting activities

Creating an element to display the total activity cost

Create a DOM element, store it in a global variable and append it to the `.activity` section. You can view the elements tab in the Chrome DevTools to check that your element is in the DOM. Create a global variable to store total activity cost — initially set to 0 — don't use const since you want to update this as needed.

Listening for changes in the activity section

Add a change event listener to the activity section. Inside the listener, it will be helpful to have a variable to reference the DOM `input` element that was just clicked.

- NOTE: It is helpful at this point to log out the variable you just created to double-check that its values is what you expect. Remember, you'll need to click on the checkboxes in the Activity section to run the code in this listener, including your log statements.

Updating and displaying the total activity cost

Let's add another helpful variable in the Activity section's change listener:

- Get the `data-cost` attribute value of the clicked element stored in the variable above. Since you'll be performing some simple arithmetic with the activity cost, you'll need to make sure the value is a number. There are helpful methods for turning strings into numbers, which can be found with a Google search. And the `typeof` operator can be used to log out the data type of a value or variable.
- NOTE: Again, it's helpful here to log out the cost variable.

Still inside the Activity section's change listener, you can use an `if/else` statement to check if the clicked input element is checked or unchecked. If the input element is checked, add the cost of the currently clicked activity to the total cost variable, else subtract the cost.

Finally, set the text of the total cost element (that you created above) equal to the string 'Total: $' concatenated with the current value of the total cost variable (that you declared above).

Disabling conflicting activities

Still inside the Activity section's change listener, let's follow the same pattern we used to get the cost of the currently clicked activity to get the day and time as well. First, we'll add another helpful variable:

- Get the `data-day-and-time` attribute value of the clicked element stored in a variable above. NOTE: Now would be a good time to log out these most recent variables to make sure they are what you think they are.

Now you need to accomplish the following tasks:

- When an activity is checked, disable any activity that occurs at the same day and time (i.e. "conflicting activities") without disabling the activity that was just checked.
- And when an activity is unchecked, you want to enable any conflicting activities.

To do this, you'll need to loop over all the checkbox inputs in the Activity section. It will be helpful to create a variable that targets the activity input element at the current iteration of the loop. Remember, you do this with bracket notation, using the loop iterator in the brackets. Something like this: `input[i]`. Be sure to log out the variable you just created to test its value.

Now that you're looping over each activity, and capturing each one in a variable, it's time to test a few conditions. In order to disable or enable an activity in the loop, you need to know two things about the activity at the current loop iteration:

- First, does the activity occur at the same day and time as the activity that was just clicked? We can check this by seeing if the activity in the current loop iteration has a `data-day-and-time` attribute that is equal to the `data-day-and-time` attribute of the element that was just clicked.
- Second, is the activity is different than the activity that was just clicked? We can check this by seeing if the activity that was just clicked is **not equal** to the activity in the current loop iteration.

Both of these conditions should be checked in a single if statement using the `&&` operator.

If both conditions evaluate to "true", then this activity needs to be disabled or enabled depending on whether the clicked activity was checked or unchecked. An `if/else` statement will help here:

- If the clicked activity was checked, then set the matching activity element's `disabled` property to `true`
- If the clicked activity was unchecked, then set the matching activity element's `disabled` property to `false`.

Payment Section

Initially, the credit card section should be selected and displayed in the form, and the other two payment options should be hidden. The user should be able to change payment options at any time, but shouldn't be able to select the "Select Payment Method" option. So you'll need to check the currently selected payment option, and hide and show the payment sections in the form accordingly.

- Hide the "Select Payment Method" `option` so it doesn't show up in the drop-down menu.
- Get the value of the payment select element, and if it's equal to 'credit card', set the credit card payment section in the form to show, and set the other two options to hide.
- Repeat the above step with the PayPal and BitCoin options so that the selected payment is shown and the others are hidden.

Form Validation and Validation Messages

There are numerous ways to accomplish this part of the project. You could try to cram all the programming for this section into the submit event listener.

You could try to validate all of the required fields in a single function and then call that function in a submit listener.

You could separate the validation and validation messages into separate tasks handled independently of each other. But it's generally a good idea to start with the simplest possible solution. Here's an example of a straightforward approach that takes it one step at a time.

There are three sections of the form that are always required: name, email, and activities. The credit section—comprised of three inputs—only needs to be validated if "credit card" is the selected payment method. To keep things simple, you can create a function to validate each required section, as well as add and remove a validation error indicator of some sort.

Each required section will need to be tested to see if it meets certain criteria, which are detailed in the project instructions. If the criteria are not met, the validation function should add a validation error indication for that field and return false. Else, the function should remove any validation error indicator and return true.

- Create a separate validation function for each of the required form fields or sections
    - Name
    - Email
    - Activity Section
    - Credit Card Number (only validated if the payment method is “credit card”)
    - Zip Code (only validated if the payment method is “credit card”)
    - CVV (only validated if the payment method is “credit card”)
- Each validation function will accomplish a similar set of tasks for its required field
    - Use a conditional to check if the input value meets the requirements for that input as stated in the project instructions.
    - If the criteria are not met, add an error indicator and return false.
    - If the criteria are met, remove any error indicators and return true.
    - NOTE: A common error indicator for an invalid field is to turn the input or form section’s border red. But an even better approach is to append an element to the DOM near the input or section, give it some friendly error message, and show it when the field is invalid, and hide it when the field is valid.
- With the individual validation functions complete, a single master validation function can now be created to test them all with a single function call. If all the individual validation functions return true, then the master validation function should return true as well. And if any individual validation functions return false, then the master function should do the same.
    - NOTE: Remember, the name, email, and activity section need to be validated on every submission attempt regardless of which payment method has been selected. But the three credit card fields will only need to be validated if “credit card” is the selected payment method.
- Now that you have the individual validation functions and a function to orchestrate the whole validation process, we need a way to kick things off. For example, a submit event listener on the form element could prevent the default submission behavior of the form if any of the fields are invalid, or false.

How To Succeed At This Project

Here are the things you need to do to pass this project. Make sure you complete them ***before*** you submit your project. To help you, we've put together this guide that links each step directly to helpful resources.

Focus on the first field

- ❑ On page load, the cursor appears in the "Name" field, ready for a user to type
  - ❑ *Related video:* Select a page Element by its ID
  - ❑ *Practice Session:* Practice Selecting DOM Elements
  - ❑ *MDN docs:* Focus()

Job Role Section

- ❑ "Your job role" text field appears when user selects "Other" from the Job Role menu
  - ❑ *Related video:* Styling Elements

T-Shirt Section

- ❑ Until a theme is selected from the "Design" menu, no color options appear in the "Color" drop-down and the "Color" field reads "Please select a T-shirt theme"
- ❑ When a new theme is selected from the "Design" menu, the "Color" field and drop-down menu is updated
  - ❑ *Related video:* Listening for events with addEventListener()
  - ❑ *Related video:* Changing Element Attributes
  - ❑ *MDN docs:* HTML Select Element
  - ❑ *MDN docs:* HTML Option Element
  - ❑ *MDN docs:* The Selected Attribute
  - ❑ *MDN docs:* The Hidden Property

Activity Registration

- ❑ User cannot select two activities that are at the same time
- ❑ Total cost of selected activities is calculated and displayed below the list of activities.
  - ❑ *Related video:* Creating New DOM Elements
  - ❑ *Related video:* Appending Nodes
  - ❑ *MDN docs:* The Checked Property
  - ❑ *MDN docs:* The Typeof Operator
  - ❑ *MDN docs:* The Unary Plus Operator

Need help? Visit the unit-03 Slack channel

- ❏ *MDN docs:* **The Disabled Property**

Form Validation and Validation Messages

- ❏ **Validate required fields and provide error indications for invalid fields upon form submission**
  - ❏ *Related video:* **Regular Expressions in JavaScript**
  - ❏ *MDN docs:* **The Submit Handler**
  - ❏ *MDN docs:* **The preventDefault() method**