COMS/SE 319: Construction of User Interface Fall 2022
LAB Activity 04 – Javascript Form Validation

This lab will walk you through a javascript form validation. We will write JS code to validate an HTML checkout form. This type of validation is called front-end validation since no data is sent to the back-end server application. It's a common means of prevalidation in web applications.

- Javascript validation
  - Use bootstrap.css to develop a checkout form
  - Use JS to validate it
  - Highlight erroneous and good input
- Single page concept
  - Switch to a different view in the same page when form correctly filled
- Chrome inspection
  - Use for debugging your code

Prerequisites:

- Chrome Browser
- Text editor (eg. **VSCode**, Sublime, Notepad...)
  - We recommend using VSCode

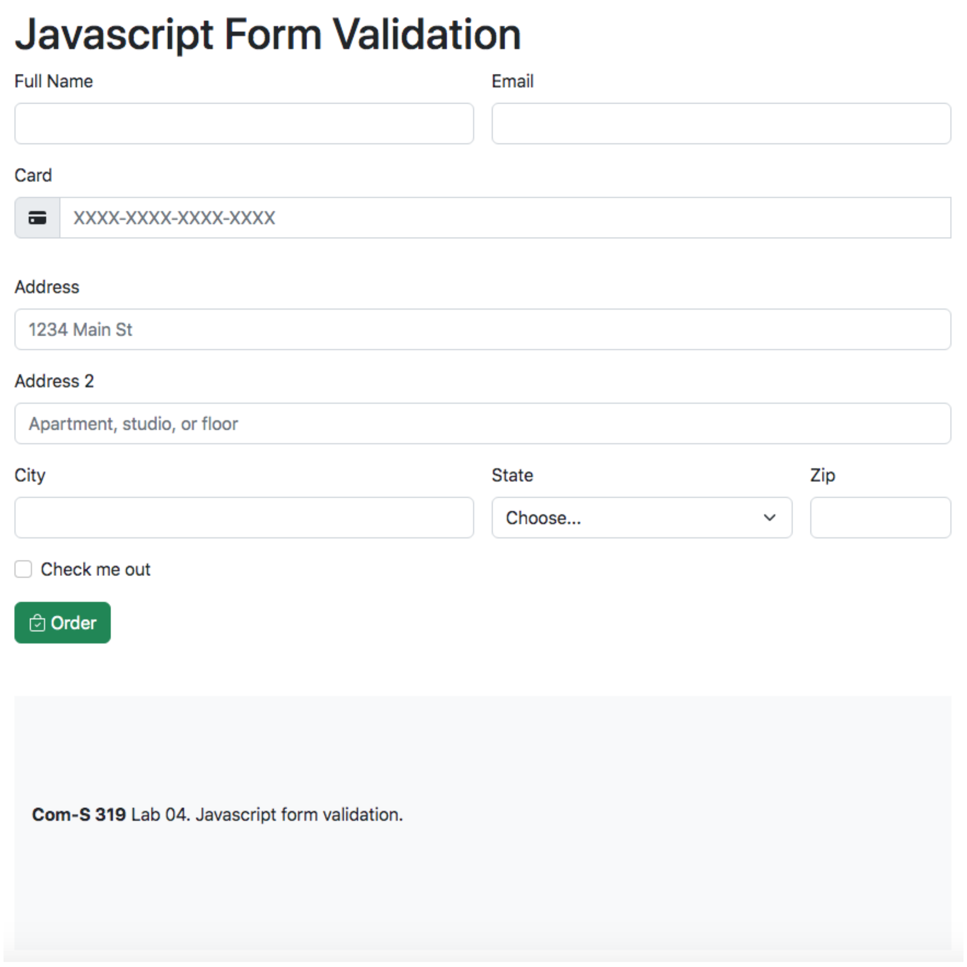
Additional tutorials:

- Lab 03 JS tutorial
- https://www.w3schools.com/js/

Task 1: Go through index.html

We have provided an index.html file inside Lab 04.zip. Go through it to understand the layout and components that are used.

- Notice the col-2, col-8, col-2 layout.
    - All content is in the middle column of width 8.
- Right under the <h1> there is a <div id=”liveAlertPlaceholder”>
    - This is where we will append/insert Alert components.
- We have one big form
    - Notice the form id. (id=”checkout-form”)
    - The first three form inputs have two sub divs.
        - Notice how <div class=”valid-feedback”> contains the valid feedback
        - <div class=”invalid-feedback”> contains the fail feedback
        - Here’s a detailed description on the bootstrap docs page.
        - At the moment, neither div is visible in the browser.
    - You already know how to use an addon input (just like the card input).
        - It has an adjacent <span> element.
        - Notice how the icon is placed.
    - Checkout the <div class=”card collapse”>
        - Why isn’t it showing when you open in browser?
    - See how we’ve cut out most of the footer component.
    - Notice the <script> file we linked at the bottom.
        - The reason we put it there is because we wanted to load the js script after the page has loaded.
    - When opening the page in the browser, you should see this:



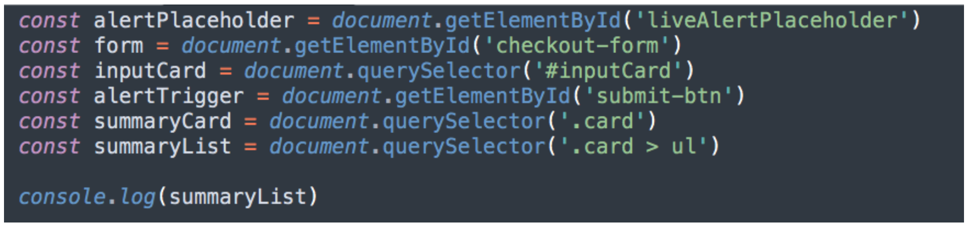
Com-S 319 Lab 04. Javascript form validation.

Task 2: Update script.js

- Open "script.js" then add the snippets in this task.
- Right now, the index.html is lifeless.
  - Click on "Order", and nothing happens. The page simply reloads.
- Let's change that!

Task 2.1 Select elements

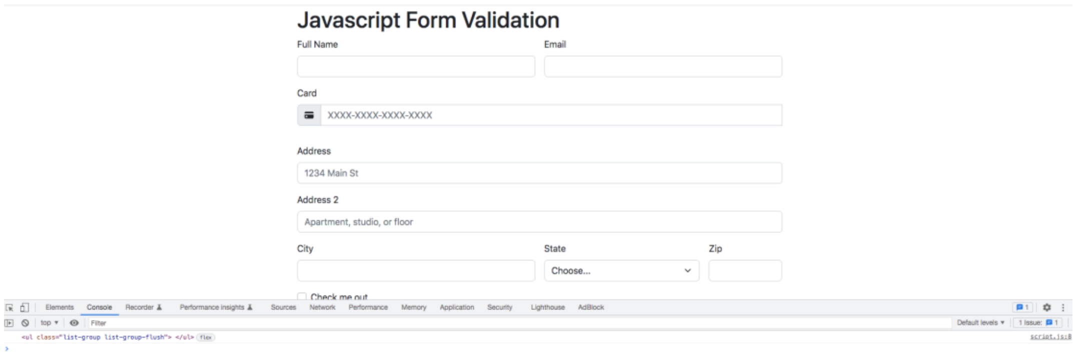
- Remember CSS selectors? As you've seen from the previous Lab, they not only help select elements in CSS, but they also help select elements in JS.
- Let's select some important elements on the validation page and log one of them to the console.
- Add this snippet to the top of "script.js".



```
const alertPlaceholder = document.getElementById('liveAlertPlaceholder')
const form = document.getElementById('checkout-form')
const inputCard = document.querySelector('#inputCard')
const alertTrigger = document.getElementById('submit-btn')
const summaryCard = document.querySelector('.card')
const summaryList = document.querySelector('.card > ul')

console.log(summaryList)
```

- Notice the various ways of selecting elements.
  - We can use javascript's native "document.getElementById" function to select a particular element on our page. This function takes the element's "id" argument.
  - We can also use "document.querySelector" function. This function can select elements by id, class, property, position etc. It basically uses CSS selectors.
    - For instance we can use it to select by ID, just like the "document.getElementById" function. We did this when selecting inputCard on line 3. Notice we used the "#" smbol in the argument
    - We can also select by class. Notice how we selected submitCard. We used the "." symbol when passing the class name.
    - We can also select elements on a hierarchy. Notice how we selected summaryList. We passed ".card > ul" argument. This selects all unordered lists "ul" that are direct children of all elements of class ".card". One of the things that make CSS selectors so versatile is their chain-ability. Please checkout the complete list of CSS selectors to have an arsenal of element selection styles.

- You should see this in your browser. Notice the output in the console tab. The <ul> element should be printed.



Javascript Form Validation

Full Name

Email

Card: XXXX-XXXX-XXXX-XXXX

Address: 1234 Main St

Address 2: Apartment, studio, or floor

City

State: Choose...

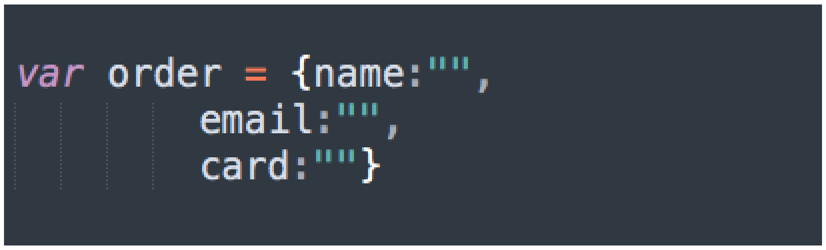
Zip

Check me out

```
<ul class="list-group list-group-flush"> </ul> flex          script.js:8
>
```

Task 2.2 Create an “Order” object

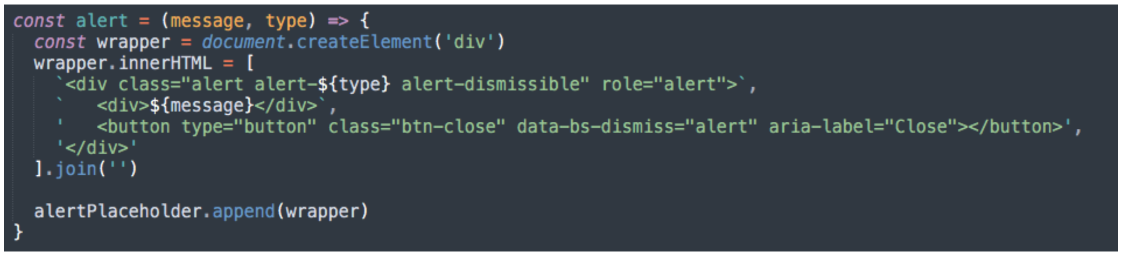
- We’ll be storing the order information in this object. For now, keep in mind that it only has 3 properties, “name”, “email” and “card”.



```
var order = {name:"",
             email:"",
             card:""}
```

Task 2.3 Create an “alert” function

- Our “alert” function’s job is to append a Bootstrap Alert component with a given message and type.

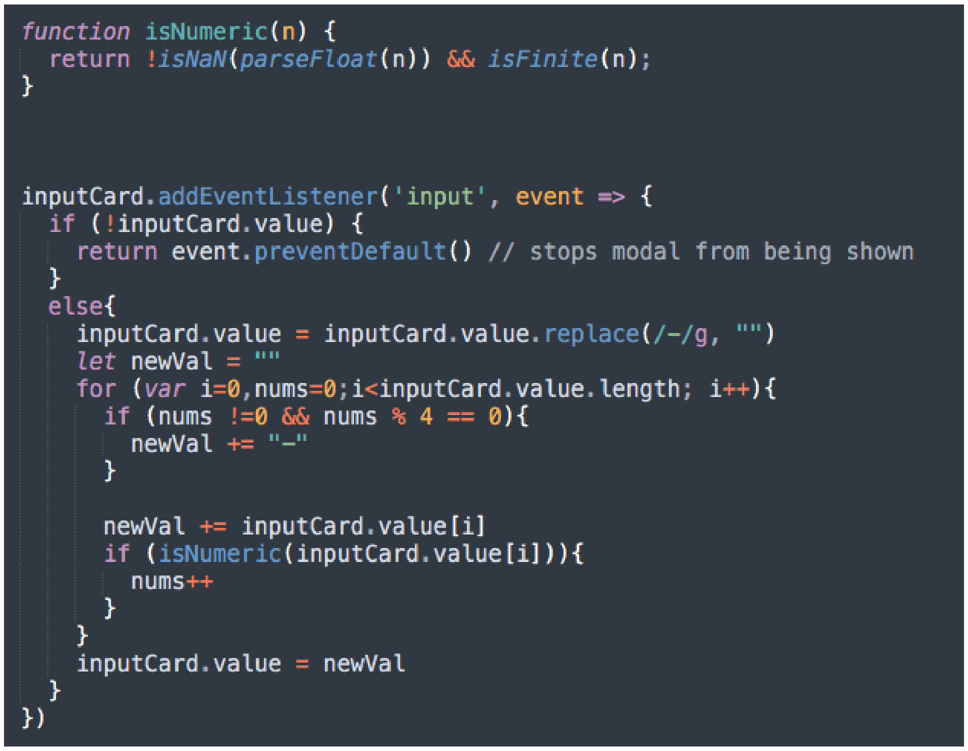


```
const alert = (message, type) => {
  const wrapper = document.createElement('div')
  wrapper.innerHTML = [
    `<div class="alert alert-${type} alert-dismissible" role="alert">`,
    `   <div>${message}</div>`,
    '   <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>',
    '</div>'
  ].join('')

  alertPlaceholder.append(wrapper)
}
```

- Notice that we’re appending this element to the “alertPlaceholder” element. Append simply adds the new element (I.e. wrapper div) into the empty “alertPlaceholder” div.

Task 2.4 Enhance card input

- One of the cool UI features we want to add is splitting card input. I.e. we want to split/hyphenate user input as they’re typing.
- You have probably seen this on some websites. Their credit card inputs can split the individual ‘quadruplets’ in realtime.
- In order to do this, we must handle “input” event on the inputCard component. This event is triggered every time the input value changes. To capture the event, we need to set up an event listener. Event listeners are JS functions that are triggered whenever a given event occurs. Checkout Lab_04_JS_part_2, for more on events. Also, check out this list for a complete set of JS events.
- In the snippet below, notice how we are listening on the ‘input’ event, and see how we specified the corresponding event handler function.

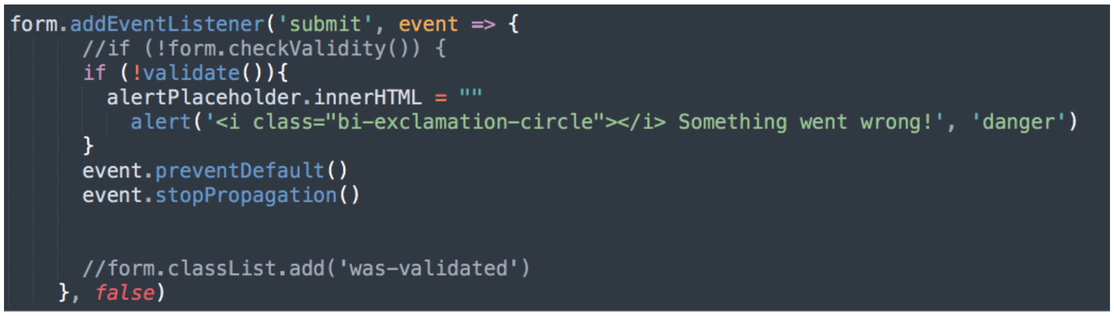


```
function isNumeric(n) {
  return !isNaN(parseFloat(n)) && isFinite(n);
}



inputCard.addEventListener('input', event => {
  if (!inputCard.value) {
    return event.preventDefault() // stops modal from being shown
  }
  else{
    inputCard.value = inputCard.value.replace(/-/g, "")
    let newVal = ""
    for (var i=0,nums=0;i<inputCard.value.length; i++){
      if (nums !=0 && nums % 4 == 0){
        newVal += "-"
      }

      newVal += inputCard.value[i]
      if (isNumeric(inputCard.value[i])){
        nums++
      }
    }
    inputCard.value = newVal
  }
})
```

- Now try typing into the credit card input field.

Task 2.5 Let's add a "submit" event listener for our form

- In this case, we want to capture the "submit" event for the form. This event is triggered when the "submit" button is clicked.



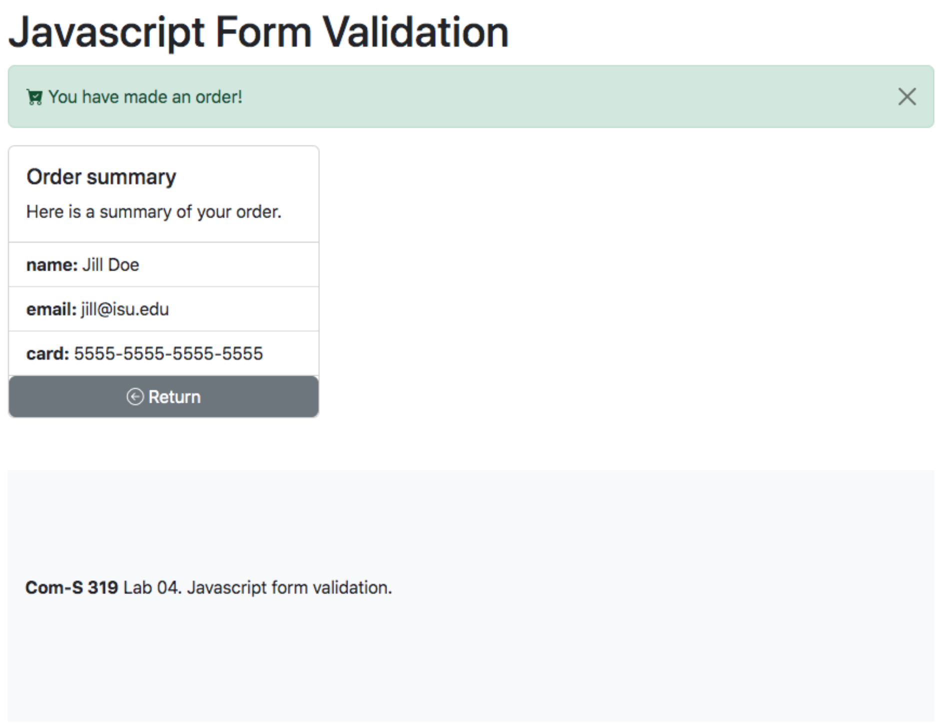
```
form.addEventListener('submit', event => {
      //if (!form.checkValidity()) {
      if (!validate()){
        alertPlaceholder.innerHTML = ""
          alert('<i class="bi-exclamation-circle"></i> Something went wrong!', 'danger')
      }
      event.preventDefault()
      event.stopPropagation()


      //form.classList.add('was-validated')
    }, false)
```

- When the form is submitted, we want to call the "validate" function which we have partially implemented for you. If validate fails, we want to append a failed Alert in our alertPlaceHolder. Finally, we want to stop event propagation. In this case, the propagated event would reload the page if not stopped!

Task 2.6 Uncomment the “validate” function

- Uncomment the “validate” function. We partially implemented this one since it’s a bit lengthy.
- Notice how we grab individual form inputs, “email”, “name” and “card”.
- We want to validate the three inputs so we check their corresponding “values” in the conditional statements.
    - If any input value is bad, we set “val” to false and set the corresponding input’s class attribute to “form-control **is-invalid**”
    - Notice, all form inputs already have the class “form-control". Thus we are merely adding the “is-invalid” class in this case.
    - Similarly we set the input’s class attribute to “form-control **is-valid**” when the input is valid.
    - “is-invalid” and “is-valid” are the classes that are needed to highlight our inputs with green or red liners. Thus, we’re just toggling them based on user input.
    - Whenever decorated with either of the classes, the corresponding “bad” and “good” feedback <div>s will suddenly appear under the inputs.
    - If inputs are good, we update the “order” object
- Notice how we made use of regular expressions for string validation. Regex is a powerful tool used throughout computer science. In this case, we make use of it to quickly match and validate patterns. Use online tutorials to learn more about regex.
- If form is valid we want to show a summary of user action.
- To make it more interesting, we want to show the order summary in the same page by getting rid of the form. This is a basic demonstration of single page design. Most modern front-end frameworks work this way. Once your browser downloads index.html and the associated scripts, then nearly all the web logic can run on the same page. Hence, there will be no page navigation. In this exercise, we do the same thing. We have a form page and on success, we swap it with a summary page without changing the actual webpage.
- Bootstrap JS makes it easy to make components disappear. Checkout how to do it using both CSS and JS over here.
- This functionality is called “Collapsing”. We simply add the class “collapse” to the form and “poof!” it’s gone. In it’s place we’d like to add our summary card, so we edit it on the fly and remove the class “collapse” from the summary card div. We then call the alert function to display the required message.
    - Notice how we iterated over the “order” object to create <li> “list” elements for the unordered list in the card div.
    - Then we remove the “collapse” class from the summaryCard div making it visible for the first time.

        - We then reset the innerHTML of the alertPlaceholder div. We do this to make space for a new Alert that indicates success.
    - Notice what we do in the form event handler.
        - If validate returns false, we clear the alerPlaceholder div and append a warning Alert.
        - Pass or fail, we prevent default even behavior. In this case, our "submit" event is prevented from submitting the form and reloading the page. We do this to keep all updates on the same page in line with our single page design.
- If you've implemented everything properly, you should see the confirmation page when you submit the form. Notice that the page's URL hasn't changed, you're still on the same page (single-page design).


Javascript Form Validation

You have made an order!

Order summary

Here is a summary of your order.

name: Jill Doe

email: jill@isu.edu

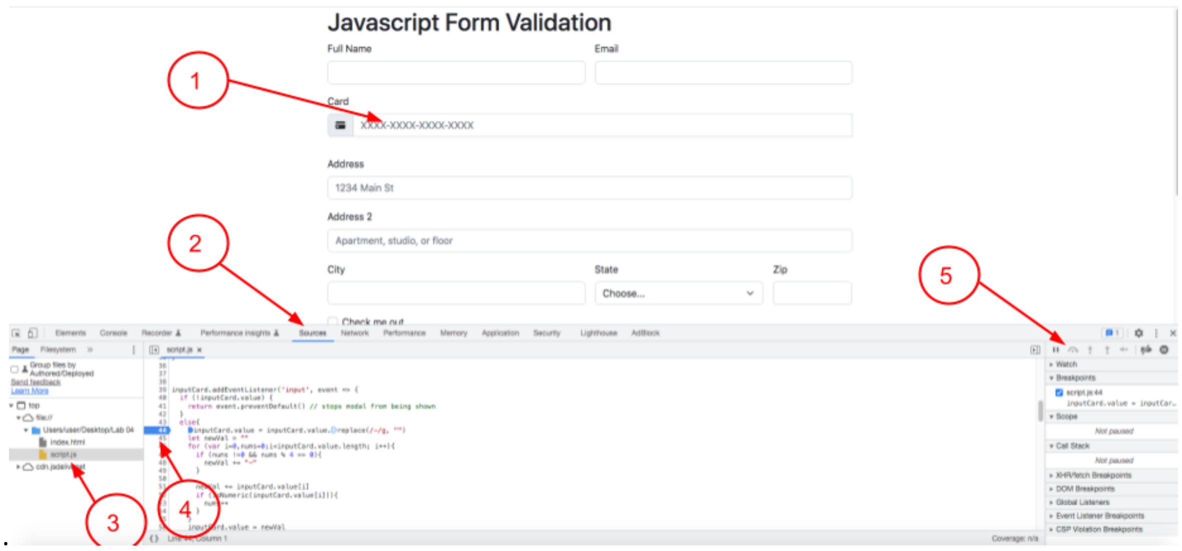
card: 5555-5555-5555-5555

Com-S 319 Lab 04. Javascript form validation.

- If you now click the "Return" button you will be redirected back to the form. This happens by simply reloading the page.
    - In index.html, you can see that "onclick" event of the "Return" button simply reloads the page.

Task 3: Using the Chrome inspection tool

- For debugging our JS, we can use simple console.log() statements to view intermediate outputs.
- But a more sophisticated way is to use the chrome debugger. The Chrome debugger is a powerful feature of the chrome inspection tool.
- To use it:



```
inputCard.addEventListener('input', event => {
  if (!inputCard.value) {
    return event.preventDefault() // stops modal from being shown
  }
  else{
    inputCard.value = inputCard.value.replace(/-/g, "")
    let newVal = ""
    for (var i=0,nums=0;i<inputCard.value.length; i++){
      if (nums !=0 && nums % 4 == 0){
        newVal += "-"
      }

      newVal += inputCard.value[i]
      if (isNumeric(inputCard.value[i])){
        nums++
      }
    }
    inputCard.value = newVal
```

.

1. Right click on the page to inspect.
2. Go to the sources tab.
3. Click on the script file, in this case “script.js”.
4. Put a breakpoint by clicking a particular line number. In this case, line in the “else” block of inputCard.addEventListener
5. If you type a character in the input field, you should see the debugger mode activated. You can now use the right sub window to perform debugging
   - Step over, step into, variables etc...