# Forms:

* Form Controls:
  + Input fields, select menus, and buttons
* Accessing Form Elements
  + Legacy DOM property: “document.forms” – returns HTML collection of all the forms in the document in the order they appear in the markup. USE INDEX (zero-based) notation.
    - i.e., const form = document.forms[0];
    - This is the equivalent to
      * Const form = document.getElementsByTagname(‘form’)[0];
        + Note the getElement(**S**)ByTagname!
      * One can also use the *name* property (“search”, in this case) >> *const* form = document.forms.search; but must be careful of any similar property names or methods of the same name/keyword, such as “submit”.
        + This can be avoided:

*Const* form = document.forms[‘search’]; **< < < USE THIS.**

* + - A form object also has ‘elements’ method that returns an HTML collection of all the elements in the form:
      * *const* [input, button] = form.elements;
      * The best method, it appears, would be to split the variables (input has *name* of “searchInput”):
        + *const* input = form[‘searchInput’];
* Form Properties and Methods
  + form.submit():
    - submits form automatically. Note that this method will not trigger the form “submit” event!
    - A form may be submitted manually by the user employing a button or input element with a type attribute of “submit”, or input element with a type of “image”. (I believe this acts as a …button?, but an input field… Hmmmm…)
  + form.reset():
    - will reset all the form controls back to their initial values specified in the HTML.
    - Buttons with type “reset” will do the same thing without need for additional scripting.
      * i.e., *<button type=’reset’>Reset</button>*
      * These are considered poor usability features, as they are too easy to click and wipe out all the data that’s been entered.
  + form.action:
    - property that can be used to set the “action” attribute of a form, so it’s sent to a different URL to be processed on the server.
      * *form.action = ‘/an/other.url’;*
* Form Events
  + Focus:
    - Occurs when an element is focused on with the cursor.
  + Blur:
    - Occurs when the user moves the cursor’s focus away from the form element.
  + Change:
    - Occurs when the user moves the cursor’s focus away from the form element *after changing it*.
* Submitting a Form
  + Submit event:
    - Possibly the most important event, occurs when the form is submitted. Usually used to send the content of the form to the server to be processed, but we can use JS to intercept the form before it’s sent by adding a Submit event listener:
      * Syntax:
        + form.addEventListener(*event*, *function*, *useCapture=false*);

*event*: Required; the name of the event, without using “on” prefix, like “onclick” use “click”.

*function*: Required; the function to run when the event occurs.

*useCapture*: Optional, default = “false”; false=handler is executed in the “bubbling” phase; true=handler is executed in the capturing phase.

* + - To prevent the form being submitted to a URL altogether, use “preventDefault()” method.
* Retrieving/changing values in a Form
  + Text input element objects have a *value* property to retrieve the text within the field.
  + You can also use *placeholder* values within the input fields, using the *placeholder* attribute in the HTML markup.
* Form validation
* Project: Add a form for answering questions

# OOP in JS

* Static Methods:
  + *Static* keyword can be used in class declarations to create a static method (class method); this is called *by the class directly* rather than ***instances of the class***.
    - Like if *redDice* and *blueDice* were instances of *Dice* class, and *Dice* had static method of *description*, *red-* and *blueDice* would not have functions/methods of *description*.

# Modular JavaScript

* Module:
  + a self-contained piece of code that provides functions and methods that can be used in other files or modules, keeping code organized and in separate, reusable files, improving code maintainability.
  + Code should serve a single purpose, and group together functions with distinct functionality. Ajax functions, for example, should be kept in their own module.
* Coupling:
  + Refers to how dependent certain elements or modules of code are on each other; code is considered ‘tightly dependent’ if one relies the other to run. Good design practices are considered as ‘loosely coupled’ as possible for greater flexibility in developing code systems, keeping different modules independent, in a variety of applications instead of single-use cases.