```
//==== LESSON 7 - FORM HANDLING ====
window.onload = function(){
//WE CAN ACCESS THE WHOLE <form> BY CREATING A 'HANDLE' FOR IT. THE document OBJECT
HAS A forms PROPERTY THAT IS AN ARRAY OF ALL THE FORMS ON THE PAGE.  WE CAN GET ANY
ELEMENT OF THE FORM BY ITS NAME ATTRIBUTE THROUGH THE FORM HANDLE.
var formHandle = document.forms.myForm;//name ATTRIBUTE OF THE form ELEMENT.
console.log(formHandle);//SEE WHAT WE'VE GOT?

//ONSUBMIT EVENT
//WHEN SOMEONE CLICKS THE SUBMIT BUTTON (OR EVEN HITS ENTER WHILE IN A FORM
ELEMENT), THE SUBMIT EVENT IS FIRED. TO PROCESS THE DATA BEFORE IT GETS SENT, WE
NEED TO LISTEN FOR THE onsubmit EVENT...
formHandle.onsubmit = processForm;

//THIS FUNCTION HOLDS OUR FORM VALIDATION LOGIC
function processForm(){
    alert("Form sent");
//YOU CAN SEE HOW THIS ALERT FREEZES THE PAGE BEFORE GOING TO THE DESTINATION PAGE.

//TO PREVENT THE PAGE FROM BEING SUBMITTED, WE CAN return false. THIS STOPS THE
SCRIPT FROM PROCEEDING ANY FURTHER AND PREVENTS THE FORM FROM BEING SUBMITTED.
//return false;

//SO, THIS IS WHAT WE USE TO CONTROL THE SUBMISSION OF FORMS. YOU CAN PUT IT AT THE
BOTTOM OF YOUR FUNCTION TO KEEP THE FORM FROM SUBMITTING WHILE YOU'RE WORKING ON IT
- DON'T FORGET TO REMOVE IT ONCE YOU'RE DONE!

//NOW THAT WE HAVE PAUSED THE PAGE, HERE IS WHERE WE ADD OUR VALIDATION.
//FIRST, WE GET THE DATA FROM OUR INPUT FIELDS...

//ACCESS THE ELEMENTS THROUGH THE "FORM HANDLE"
    var nameField = formHandle.f_Name;
    console.log(nameField);//SEE THE WHOLE INPUT ELEMENT?
    console.log(nameField.value);//IT'S THE value WE NEED TO ACCESS THE INPUT FROM
    THE TEXT BOX

//LET'S GET THE EMAIL FIELD INPUT ELEMENT...
    var emailField = formHandle.f_Email;

//GETTING THE CHECKBOX...
    var userOk = formHandle.f_Auth.value;//HOLDS value FROM input ELEMENT
//NOTE THE value IN THE CASE OF THE checkbox IS THE value ATTRIBUTE WE GAVE IT IN
THE HTML, NOT WHAT THE USER ENTERED. TO GET A BOOLEAN OF WHAT THE USER SELECTED, USE
THE checked PROPERTY OF THE ELEMENT.
    var userOk = formHandle.f_Auth.checked;//HOLDS A BOOLEAN
    console.log("User permission: " + userOk);

//NOW THAT WE HAVE THE DATA FROM OUR INPUTS, WE CAN VALIDATE THEM...
    if( nameField.value === "" ){
    //CHECKING FOR AN EMPTY STRING.
//LET'S HELP OUR USER OUT.  THE FOLLOWING CODE WILL HELP THE USER KNOW WHERE THEY
MADE AN ERROR AND WHAT'S EXPECTED.
        nameMsg = document.getElementById("nameErr");//GET HELPER <span>
        nameMsg.style.background = "purple"; //SET BACKGROUND COLOUR.
        nameMsg.innerHTML = "Please enter your name."; //ADD HELPER TEXT.
        nameMsg.style.color = "yellow";//MAKE FONT MORE READABLE.
        nameValue.focus(); //THIS METHOD PUTS THE FOCUS BACK ON THE ELEMENT (see
        below).
        return false;//STOPS THE SCRIPT FROM GOING ANY FURTHER.
    }



//VALIDATE EMAIL.  THIS WILL REQUIRE A BIT MORE WORK - A REGULAR EXPRESSION!
//if (!emailRegex.test(emailField)) {

//FOR NOW, VALIDATE FOR EMPTY STRING
    if(emailField.value === ""){
            return false;
        }
    }//END OF processForm onsubmit FUNCTION

    //#### NEW JS EVENTS -   onfocus   onblur   onchange    ####
    //WHEN AN ELEMENT IS ACTIVATED, THE focus EVENT IS FIRED
    formHandle.f_Name.onfocus = goHelp;//LISTEN FOR focus

    //WHEN AN ELEMENT LOSES FOCUS, THE blur EVENT IS FIRED
    formHandle.f_Name.onblur = endHelp;//LISTEN FOR blur

    var nameHelp = document.getElementById("nameErr");//GET HELPER <span>
    //FUNCTION TO ADD HELPER TEXT TO HELPER <span>
    function goHelp(){
        nameHelp.innerHTML = " First Name only";
    }

    //FUNCTION TO REMOVE HELPER TEXT FROM HELPER <span>
    function endHelp(){
        nameHelp.innerHTML = "";
    }

    //WHEN AN INPUT ELEMENT IS CHANGED, THE onchange EVENT FIRES
        var slctBox = formHandle.f_Country;
        console.log(slctBox);
        slctBox.onchange = goDD;//SET LISTENER FOR onchange EVENT

        //GET SELECTION & DISPLAY MESSAGE
        function goDD(){
            alert("Dropdown changed");
            console.log(slctBox.value);//GET SELECTED VALUE

    //USE LOGIC TO CHECK AGAINST SELECTED VALUE
        if(slctBox.value === "CA"){alert("You're Canadian!");}

    //WHAT ABOUT THE REAL WORD?  HOW DO WE GET THAT?
            var countryFull = slctBox.options[slctBox.selectedIndex].text;
    //WE GET THE <select>, THEN ITS options PROPERTY - WHICH IS AN ARRAY OF ALL ITS
    OPTIONS.
    //TO GET THE CORRECT INDEX FOR THAT ARRAY, WE CAN USE THE <select>'s selectedIndex
    PROPERTY WHICH STORED THE INDEX VALUE OF THE ONE THE USER SELECTED.  FINALLY, WE
    NEED TO ACCESS THE text PROPERTY OF THAT SELECTED OPTION.
            if(slctBox.value === "CA"){alert("You're from " + countryFull + "!");}

    //onchange IS ALSO HOW WE GET THOSE SECONDARY INPUTS THAT DON'T APPEAR UNTIL YOU
    SELECT SOMETHING FIRST...
            //DISPLAY HIDDEN PROVINCE DROPDOWN INPUT
            var provDD = formHandle.f_Prov;
            provDD.style.display = "inline";
        }
    }
```

```
//==== LESSON 7 - VALIDATING WITH REGULAR EXPRESSIONS (REGEX) ====
var outPut1 = document.getElementById("header1");
//A REGULAR EXPRESSION (OR regex) IS A SEQUENCE OF CHARACTERS USED AS A PATTERN TO SEARCH WITH, OR VALIDATE AGAINST. DOES A PARTICULAR STRING MATCH WHAT I'M LOOKING FOR?  SO, WORKING WITH REGEX REQUIRES 2 COMPONENTS:  The string you're searching IN; and the string you're looking FOR.

//FIRST, WE'LL CREATE THE STRING WE WANT TO SEARCH IN. SIMPLY...
var myTestString = "Applicants with strong JavaScript skills are favoured.";

//FOR THE REGULAR EXPRESSION, START BY CREATING A VARIABLE TO HOLD THE PATTERN YOU ARE SEARCHING FOR. REGEX STARTS AND ENDS WITH A FORWARD SLASH (THE DELIMITERS). THIS CREATES A REGULAR EXPRESSION OBJECT IN JS.
var myRegEx = /JavaScript/; //THIS IS WHAT WE'RE GOING TO TEST FOR.

//NOW YOU USE THE STRING.search() METHOD ON THE STRING YOU WANT TO SEARCH, AND PASS IN THE REGEX STRING AS A PARAMETER.
myTestString.search(myRegEx);//RETURNS 23 - THE INDEX OF THE FIRST CHARACTER OF THE MATCH.
//IF IT DOES NOT FIND A MATCH, IT RETURNS -1 (LIKE indexOf).

//WE CAN MAKE IT CASE INSENSITIVE BY PUTTING AN i AT THE END OF IT.  /javascript/i WILL MATCH JAVASCRIPT, JavaScript, JaVaScRiPt ETC.

//TO COMPARE A STRING TO OUR PATTERN WE USE THE REGEX.test() METHOD.
//THE STRUCTURE: varWithRegEx.test(valueToCompare); THIS WILL RETURN A BOOLEAN
var valueToCompare = prompt("Skills?","");//GET INPUT FROM USER
if( myRegEx.test(valueToCompare) ){//IF PASSED IN VALUE MATCHES REGEX PATTERN.
    outPut1.innerHTML = "Please submit a resume!";
}

//USING THE LOGICAL NOT OPERATOR (!) TO CHECK FOR INVALID DATA FIRST
if(!myRegEx.test(valueToCompare) ){//IF VALUE DOESN'T MATCH THE PATTERN
    outPut1.innerHTML = "Sorry, JavaScript skills are a requirement.";
} else {
    outPut1.innerHTML = "Please submit a resume!";
}


/*THIS, HOWEVER LIMITS US TO FINDING LITERAL STRINGS.THE POWER OF REGEX LIES IN THE SHORT CODES IT USES TO FIND PATTERNS.*/

var thisYear = /2016/;
/*THIS ONLY LETS US FIND 2016. WHAT IS THE PATTERN OF THE ABOVE DATA? FOUR DIGITS. WE CAN USE SHORTCODE METACHARACTERS TO DESCRIBE WHAT WE'RE LOOKING FOR.
\d IS A DIGIT.  THE SLASH BEFORE \d IS KNOW AS THE 'ESCAPE' CHARACTER AND TELLS JS WE'RE NOT LOOKING FOR THE LETTER d.*/
var yearRx = /\d\d\d\d/;  //THIS MEANS FOUR DIGITS.
/* \s IS A SPACE, \w IS A WORD CHARACTER . (A PERIOD) IS ANY CHARACTER. A WORD CHARACTER IS ANY CHARACTER FROM a-z, A-Z, 0-9, INCLUDING UNDERSCORES _ .*/

//LET'S BUILD A HUMBER COURSECODE PATTERN: 4 LETTERS AND 4 NUMBERS
var courseCode = /\w\w\w\w\d\d\d\d/; //4 word chars & 4 digits.

//SEE THE PATTERN ABOVE? {} ALLOWS US TO SPECIFY MULTIPLES OF THE SAME TYPE OF CHARACTER.
var courseCode = /\w{4}\d{4}/;

//WE HAVE AN 'OR' ABILITY USING PARENTHESES AND A SINGLE PIPE. () ARE USED TO GROUP PARTS OF THE REGEX.
var studentNum = /(n|N)\d{8}/;
//THIS MEANS A LOWERCASE n OR AN UPPERCASE N FOLLOWED BY 8 DIGITS.

/*OTHER IMPORTANT QUANTIFIERS...
x?  (QUESTION MARK) CHECKS FOR ZERO OR ONE OCCURANCE OF x. OFTEN, PEOPLE PUT ONE SPACE IN THE MIDDLE OF A POSTAL CODE - BUT NOT ALWAYS.  WE CAN CHECK FOR ONE OR ZERO SPACES WITH \s?
```

```
x+ (PLUS SIGN) CHECKS FOR ONE OR MORE OCCURANCES OF x.
x* (ASTERISK) CHECKS FOR ZERO OR MORE OCCURANCES OF x  */

//USING SQUARE BRACKETS, WE CAN DEFINE VALID RANGES.
var hexCodeNumberRange = /[0-6]/;
var hexCodeLetterRange = /[ABCDEF]/; // OR WE COULD USE  [A-F].

//ONE MORE THING TO SHOW, WHAT IF WE'RE LOOKING FOR "stop"?
var stopRgx = /stop/;
/*  WE WOULD ALSO FIND A MATCH WITH "UNSTOPPABLE" - WHICH WE DON'T WANT IN THIS
CASE.  WE NEED TO MAKE SURE THAT THERE ARE NO CHARACTERS BEFORE 's' AND NONE AFTER
THE FIRST 'p'.
THE CARAT ^ (SHIFT + 6) SPECIFIES TO START AT THE BEGINNING OF THE STRING.
THE $ SPECIFIES THAT IT MUST BE THE END OF THE STRING.
SO,  /^stop$/  MEANS THE STRING WE'RE TESTING MUST START WITH stop AND THERE CAN'T
BE ANY CHARACTERS AFTER THAT. THESE ARE USED IN MOST REGULAR EXPRESSIONS.*/

//LET'S BUILD A CSS COLOUR HEXCODE PATTERN THAT IS NOT CASE SENSITIVE.
var hexCode = /^#([0-9]|[A-F]){6}$/i;
```