FORMS AND SCRIPTING

DEAKIN COLLEGE
in association with
DEAKIN UNIVERSITY

- Uses of validation
- Validation process
- Types of validation check
- Common mistakes
- Development process

DEAKIN COLLEGE
in association with
DEAKIN UNIVERSITY

- Uses of validation
  - To prevent users from entering “incorrect” data
  - To ensure that all required data are entered
  - To preserve the integrity of maintained data
  - To provide computational support (e.g. a “calculator” in a web page, or a simple quiz)

- Validation Process
  - –Design form to minimise user ‘error’ (*selection* rather than *input*)
  - –Provide adequate information to guide user and reduce errors and subsequent frustration
    - Indicate expected ranges
    - For a phone number, use a text box where the user can type the information
    - Provide some descriptive text such as:
      - –phone Number (XX) XXXXXXXX

- Validation Process
  - HTML file containing one or more forms is downloaded and rendered to screen
  - User enters data into a form
  - When the user clicks the submit button, the JavaScript takes over and validates the data
  - If the data does not validate correctly, the submission is not allowed to proceed

- Form validation code must be executed as and when needed, rather than ‘by default’
- Include code within a function and call the function as is required (within the *event handler*)
- Use the Javascript to access form elements and check them
- So form validation is a combination of HTML and JavaScript

- When the form is submitted, check that the data is ‘correct’
- If the data is ‘correct’, submit the information to the designated action
- If the data is incorrect, provide useful feedback that describes the type of error found and instructions on how to fix the errors.
  - Simply popping up a dialog and saying:

  “Error in Phone Number” is bad practice.

  - A better response may be:

  “Error in Phone Number, only ‘(‘, ‘)’ and numbers are permitted. Use the format ‘(XX)’ XXXXXXXX.”

- Within the declaration of the form tag include

  `onSubmit="return CheckForm()"`

  – Tells the browser what to do when the user clicks on the *Submit* button

- Write an event handler for the 'onSubmit' event

Connect to *Submit* event handler

Form elements

Submit button

```
<form name="myform" method="get"
 action="NextPage.php"
onSubmit="return CheckForm()">

  Name: <input type="text" name="TheName"><br>
  Age: <input type="text" name="TheEmail"><br>
  Address: <textarea name="TheAddress">
     </textarea><br>
  Bank Balance: <input type="text"
     name="TheBalance"><br>

  <input type="submit" value="Send">
</form>
```

- ‘Package up’ the validation code within a function called ‘CheckForm’

```
<script>
  function CheckForm()
  {
      if (document.myform.TheName.value == “”)
      {
         alert(“You must enter a name”);
         return false;
      }
      return true;
  }
</script>
```

Check if field called ‘name’ is empty

If field is empty, display message and **return false**

If field is not null, **return true** *(and so submit the form)*

```
<script>
function CheckForm()
{
  if (document.myform.TheName.value == “”)
  {
     alert(“You must enter a name”);
     return false;
  }
  var email= document.myform.TheEmail.value;
  if (email.indexOf(‘@’)==-1)
  {
     alert(“Email is invalid.”);
     return false;
  }
  return true;
}
</script>
```

If field does not have ‘@’, display message and **return false**

- Some elements can be validated on a “by entry” basis.
- A function to validate the contents of an entry box can be called by the **onBlur** event
- **onBlur** occurs whenever an element such as a text box loses focus
- The **onChange** event only does the validation when you move off the entry box *and* that entry has changed
- More significantly, **onChange** also applies to selection elements

- Types of validation check
  - *Presence*
    - Data are checked to ensure that all necessary fields are present (‘not empty’, or ‘something selected’)
    - e.g. A payroll form must have completed employee ID and payroll ID fields.
  - *Size*
    - Fields are checked to ensure they contain the correct number of characters (maximum length should be managed through *maxlength* attribute)
    - e.g. Staff_Id should contain 6 digits (‘991312’)

– *Range*

- Numbers or codes are checked to ensure that fall within a permissible range
- e.g. Date of birth in a school database should fall between 1950 and 1990.

– *Character*

- Fields are checked to ensure that they contain only characters of the correct type
- e.g. there are no letters in a number field such as a postcode

–*Format*

- Fields are checked to ensure that the *format* is correct (also called a *picture check*): that a code contains the correct number of letters and numbers AND they are in the correct sequence.
- e.g. international phone number is +xx-xx-xxx-xxxx
- *Should you be using separate, selectable fields?*

- e.g. check that only numbers or spaces are given in a credit card number — not letters or other characters
- e.g. check that, for the given card type, the number of digits given is valid and the prefix to the number is valid

Card Type	Prefix	Number of Digits
Visa,	413	16
American Express	34, 37	15
MasterCard	51-55	16

– *Consistency*

- Are two ‘connected’ fields consistent?
- e.g. if married status=no, is the spouse name field empty?
- e.g. for changing password, are the two entries *identical*?

– *Reasonableness*

- Quantities are checked to ensure that they are not abnormally high or low

- Common mistakes
  - Assuming a specific format for telephone numbers
    - Some numbers require an area code
    - Some international numbers are **NOT** 3x3x4 digits long. And of course letters are still valid!
  - Not all addresses are ZIP codes - only one country uses them! There is a ***large*** variation in "**postal code" formats**.
  - Dates have many variations depending on locale. Separate entry boxes for each component (year/month/day) is a safer method of entry.

- Development process
  - Select suitable form elements that best suit the data being collected
  - Where possible restrict the user by using controls that do not allow the user to enter variations; good design can *minimise* the validation required
  - Use meaningful names to describe your forms and field names
  - Use meaningful JavaScript variables and function names

- Development process
  - Develop progressively
    - develop and test code ‘one rule at a time’
    - ensure it works before you move on to the next validation ‘rule’
  - Developing too much code without adequately testing it can lead to complex debugging and many hours lost
  - Develop re-usable functions to perform common tasks
  - Document your code fully as you go