


INTRODUCTION



- Understanding forms
 - What are forms?
 - HTML hierarchy
 - Selecting form elements
- Accessing form elements
 - Inputs
 - Radio buttons
 - Select options
- Events
 - Form events
 - Control Events
- Regular expressions
 - What is RegEx?
 - Using RegEx to analyse data

QA Understanding forms – what are forms?

- Forms allow us to send data to the server for processing

```
<form action="/folder/enrol.aspx" method="POST">
  <input type="text" name="username" />
  <input type="text" name="address" />
  <input type="submit" value="OK" />
</form>
```

Define:

- Form – action & method
- Inputs – name, type & value

Please enter your name:

Enter address here:



Users can interact with programs running on the server by using HTML forms; in this case, the web server acts as nothing more than a gateway to forward the form's data to the back-end application and to return any generated response. Forms allow the web to be used for a variety of purposes, including user inquiries, ordering and booking systems, and front ends to databases.

A form is described using the `<form>` tag; this will contain other elements and text, including the elements describing the form's input fields. The FORM tag takes two attributes: the `method` states how the data is to be sent to the web server, and the `action` gives the URL of a resource used to process the form. This is normally a server-side program or script, but can be a `mailto:` URL.

The values for the `method` attribute determine how the data is sent to the server and takes the value POST or GET. Depending on which you use, the server application needs to be programmed slightly differently, but the details of this are beyond the scope of this course.

QA Understanding forms – HTML form inputs

- Text boxes/areas

```
<input type="text" name="username" value="">
<input type="password" name="password" value="">
<textarea name="comment" rows="10" cols="40"></textarea>
```

- Checkboxes and radio buttons

```
<input type="checkbox" name="milk" value="CHECKED">Milk?<br>
<input type="radio" name="drink" value="tea">Tea<br>
<input type="radio" name="drink" value="coffee">Coffee<br>
<input type="radio" name="drink" value="choc">Chocolate<br>
```

- Selections

```
<select name="title">
  <option value="Dr">Dr</option>
  <option value="Ms">Ms</option>
  <option value="Mr">Mr</option>
</select>
```

```
<select name="prod" multiselect>
  <option value="a">Apples</option>
  <option value="p">Pears</option>
  <option value="g">grapes</option>
</select>
```

Most form fields are defined using an `<input>` tag; the `type` attribute specifies whether this is a checkbox, radio button, etc. A few field types have their own specific tags; for example, `textarea` and `select`. These are shown above.

Each input element has `name` and a `value`. The `name` is specified in the `name` attribute, the `value` may be specified in the `value` attribute (except for text areas when it is specified by placing text within the `<textarea>...</textarea>` tags) or left for the user to enter (e.g. with text box input). The `name-value` pairs are how back-end code accesses the data from the form.

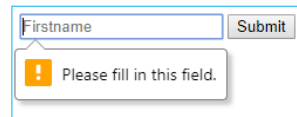
QA HTML5 Elements

HTML5 introduced a wave of new input types:

- `color`
- `date`
- `email`
- `month`
- `number`
- `range`
- `search`
- `tel`
- `Time`
- `datetime-local`
- `url`
- `week`

QA Required fields

- You can force a field to be mandatory on the client

A screenshot of a web form. It features a text input field with the placeholder text 'Firstname' and a 'Submit' button to its right. Below the input field, a red error message box is displayed, containing a red exclamation mark icon and the text 'Please fill in this field.'.

- On a submit action, an error message may appear:

```
<input type="text" autofocus="true" required/>
```

Pattern

- The pattern attribute allows use of regular expressions

```
<input type="text" pattern="[0-9]{13,16}" name="CreditCardNumber" />
```

- We must ensure the user understands the regular expression using plain-language explanations of the requirements

The pattern attribute specifies a regular expression that the <input> element's value is checked against.

Note: The pattern attribute works with the following input types: text, search, url, tel, email, and password.

Useful pattern generation website:

<http://html5pattern.com/>

QA Form validation

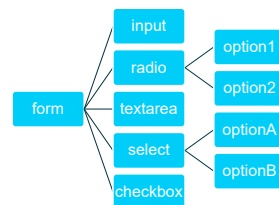
- As we have seen, some browsers ship with validation
- These are JavaScript free client validation
- Uneven support and UI feedback may be more trouble than benefit
 - You can tell a browser to switch it off
 - Still benefiting from the semantic types

```
<form novalidate>
  <input type="email" id="addr">
  <input type="submit" value="Subscribe">
</form>
```

Even if you choose to disable the JavaScript free validation and go for your own controls, the new semantic types are useful. With them, we can define a clearer sense of meaning to the form creators intentions

QA Understanding forms – HTML hierarchy

- The Form is a DOM object and container of other elements
- When a form is submitted, these details are sent to the server
 - The majority of the form fields hold a single value
 - Radio and select controls are slightly more complex



When you submit data to a server, you send all form DOM objects contained within it. Most of these controls are simple to work with, they have a value attribute that holds the current user entry. Certain multi-option controls, like select and radio, are slightly more complex to work with.

QA Understanding forms – selecting form elements

- You can use DOM or BOM techniques to select forms (DOM should be preferred)
- The BOM maintains an array of form objects
- The DOM allows id or hierarchical selection and is significantly faster
- Elements can have a name and an id attribute
 - Name is needed if you are going to submit a form to the server

```
<form name="demo" id="demo" action="process.pl">
  ...
</form>
<script>
  frm1 = document.demo;
  frm2 = document.forms[0];
  frm3 = document.forms["demo"];
  frm4 = document.getElementById("demo");
</script>
```

Selecting forms is interesting, there are two distinct approaches that could be used. This was one of the original JavaScript functionalities originating back in Netscape in pre DOM Programming. The BOM approach still remains, where the document object maintains an array of form objects upon the page.

Since the advent of DOM based Programming, we have been able to reference by the ID attribute. This is the preferred approach as the selector engine is a faster way of retrieving elements.

HTML has always been designed to be backwards compatible and as a result standard form submission builds up a string of name value pairs. However, an ID will need to be provided as an additional attribute. In most situation, the id and the name will be the same to create a consistent model for Programming.

In certain scenarios the form's submit action will be prevented in its entirety, this commonly happens in AJAX based activity and will be discussed later in the course.

QA Accessing input elements

- Elements are sub objects of the form object
- Once selected, the input object has a series of properties.

Attribute	Value	Description
alt	text	Specifies an alternate text for an image (only for type="image")
checked	checked	Specifies that an element should be preselected when the page loads (for type="checkbox" or type="radio")
disabled	disabled	Specifies that an <input> element should be disabled
maxlength	number	Specifies the maximum number of characters allowed in an element
minlength	number	Specifies the minimum number of characters allowed in an element
name	name	Specifies the name of an <input> element
readonly	readonly	Specifies that an input field should be read-only
size	number	Specifies the width, in characters, of an <input> element
value	text	Specifies the value of an <input> element
required	n/a	HTML5 attribute that specifies this field requires a value
placeholder	text	Placeholder text to display within the element
autofocus	n/a	Instruct the browser to place focus on this field once rendered
spellcheck	n/a	Instruct the browser that spell-checking should take place on user input

The majority of form controls are input-based. Only the early Netscape breaks of the HTML 3 standard with tags such as <textarea> and select are different. This helps us a great deal to have a common interface for programming-style behaviour.

Once you reference the DOM element, you can access the appropriate property. The most important being value:


```
document.getElementById('uname').value
```

The following methods are also available:


The `focus()` method moves the input focus to the particular element. This is useful, for example, when a particular field has failed validation and you want the user to change its contents.

The `blur()` method moves the focus away from the element.

Elements that allow the user to enter text (textarea, text, and password) also support a `select()` method. This highlights and selects the contents of the field. Note that `select()` does not imply `focus()`; if you want to select an element's contents and allow the user's typing to appear in the element, you need to use both `focus()` and `select()`.



Radio buttons



- Radio buttons are unique as they can share the same name value
- But not the same id
- Radio buttons are a mutually exclusive selection list
- To find the selected radio button in a group:
 - Select the radio buttons in the group
 - Loop over the array looking for the checked element
 - Select the value

Radio buttons are a very useful tool in gathering user-input without them typing in user data. Their shared property name makes life very simple for us with modern selector techniques. Within the radio button group, only one radio button can be selected at one time. This is located with the checked property.

With older selectors, the following code will achieve the same result:

```
let paymentMethod = document.getElementsByName('cardtype');
for ( let i = 0; i < paymentMethod.length; i++) {
    if(paymentMethod[i].checked) {
        console.log(paymentMethod[i].value);
    }
}
```

QA Accessing select options

The selected element holds several useful properties to know what options have been selected:

- **selectedIndex**: the index of the first selected item
- **selectedOptions**: An **HTMLCollection** representing the set of **<option>** elements that are selected
- **value**: a string representing the value of the first selected item

```
let select = document.querySelector('select');  
console.log(select.value);
```

QA Form methods and events

Properties

- Those relating to HTML attributes
 - `action`
 - `encoding` (`ENCTYPE`)
 - `method`
 - `name`
 - `target`
- Others
 - `length`

Methods

- Perform actions corresponding to form buttons
 - `submit()`
 - `reset()`

The elements array and its objects are the most useful properties of the form object, but it does have some others:

The action, encoding, method, name, and target properties are all reflections of the HTML attributes of the form. The encoding property reflects the ENCTYPE attribute, and the others all reflect the named attributes.

The length attribute specifies the number of elements in the form, and, as such, has the same value as the length property of the elements array.

The form object has two methods:

`submit()`, which submits the form immediately, without waiting for the user to press the submit button.

`reset()`, which clears the form contents and sets all fields back to their default values.

QA Understanding forms – form submission (GET)

- A form is submitted when a button is pressed
- type="submit" or type="image"
- Form data is sent to server along with URL request
- JavaScript can intercept form submission for validation

GET /.../enrol1.aspx?username=Paddington&address=Peru HTTP/1.0



All data entry into a form is managed entirely by the client browser – there is no interaction with the server at this point. Eventually, the user takes some action that causes the browser to send the data to the server. This happens when they click on either a TYPE=SUBMIT button or on a TYPE=IMAGE element. In either case, the browser sends a request to the resource named in the ACTION attribute of the form, and passes the form data as a parameter with this request.

Using JavaScript, it is possible to intervene between this submit request and the data actually being sent to the server. This gives you the opportunity to validate the data of the form locally, and to prevent the form from being submitted, if the data is not valid. Doing so can substantially reduce the load on the server, reduce the amount of network traffic and reduce the time the user has to wait to discover his input was invalid.

QA Form events

- The events of the form object are of great importance
- When a user submits a form, they raise an **onsubmit** event
- Normally the event fires, no additional interaction occurs
- Data is transmitted to the server
- Subscribing to the **onsubmit** event allows you to check the data
 - Cancelling the submit action, if necessary
- There is also an **onreset** event
 - Fires the user has clicked a reset button
 - Use it for form clean up and wiping variables

Forms have just two event handlers: `onsubmit` and `onreset`. `onsubmit` is called if the user presses a submit button, and `onreset` is called if they press a reset button. The `onsubmit` event handler can prevent the form from being submitted.

QA Input element events

Main events of input elements:

- **onfocus**
 - The object is receiving the input focus
- **onblur**
 - The object is losing the input focus
- **onchange**
 - Edit controls only
 - The object is losing the input focus and its contents have changed
- **onclick**
 - Checkboxes and button types only
 - The object has been clicked
 - Especially useful with **type="button"** input elements
- **onselect**
 - Edit controls only
 - Some text has been selected within the control

Input elements have the following event handlers:

- `onfocus` is called when the element receives the focus
- `onblur` is called when the element loses the focus

In addition, editable elements have the following event handlers:

- `onchange` – The element is losing the focus and its contents have been altered
- `onselect` – The selection has been altered in the control

Finally, checkboxes, radio buttons, and normal (push) buttons also have an `onclick` event handler. The `type="button"` input element is not ordinarily useful in a form as the button has not HTML usage (unlike submit & reset). However, with the aid of JavaScript and the `onclick` event handler, it is possible to give generic push buttons some behaviour or actions to perform. In fact, this makes it possible to write a simple interactive client-side application, and there is no need for a submit button or an `action` attribute on the form at all. A calculator is an example of a possible application.

QA Form validation – the submit event

You may either want to validate on the field or the form.

- The form validation occurs during the **onsubmit** event

- By using inline validation, the validation function must return a **false**

→ *Never do this but you may inherit it*

- In programmatic events, we can **preventDefault()** behaviour

→ Much the preferred approach!

```
function validateForm(evt) {  
  let error = false;  
  //error checking code  
  if (error === true) {  
    evt.preventDefault();  
    //feedback to user  
  }  
};  
  
window.onload = function () {  
  let element = document.querySelector("#payment");  
  element.addEventListener("submit",  
    validateForm, false);  
};
```

A great deal of the code above should, by now, feel familiar to you. The **submit** event needs to be interrupted if we are to stop the form just sending its data to the server. In contemporary JavaScript, all browsers understand **preventDefault** and as a result, we can simply catch the raised event and prevent the default action, i.e. the form submission from occurring.

Note the error variable above. This variable works as a Boolean flag within the function. Within the code, we would check each submission issue and if an error was found set the flag to be true. Before we leave the function, we simply check if the error has been switched to true. If it has, we prevent the submission and feedback to the user.

Form validation – field validation

Field validation allows you to check the value of an individual field.

- **blur**, **focus** and **change** are the most important events

The **change** event fires when the value in a form has changed.

- For check boxes and the like, this occurs when the value changes
- Text inputs change when the user leaves the field

```
let cardtype = document.querySelectorAll('input[name=cardtype]');  
  
for (let i = 0; i < cardtype.length; i++) {  
  cardtype[i].addEventListener('change', checkSelection);  
}
```

There are events to check user input when we leave or enter a field:

- **focus** on entry
- **blur** when you leave

We can check individual fields using blur, focus and change events. The majority of change-based events will occur on the change event. When a user moves out of a currently focused field and the value has changed, the event is raised. Focus can be very useful in applying some CSS functionality to the element to highlight it for the user.

A few critically important notes on usability. Never use prevent default as part of any change, or blur event, and never redirect a user back to a previously focused field. This is frustrating to the end user and terrible in terms of accessibility for users with disabilities.

QA QuickLab 13a – form validation

- Creating field and page submission validation
- Hooking into events programmatically
- Checking input presence
- Reusing field validation for page submission

QA Regular expressions

- Regular expressions are patterns used to evaluate strings
- And match character combinations
- Very useful for examining input data
- In JavaScript, regular expressions are also objects
 - These patterns are used with the exec and test methods of RegExp
 - The String object has match, replace, search, and split methods
- The regular expression pattern origin is in UNIX
 - Used in JavaScript and many other Programming languages
 - It's a pattern of simple characters
 - Either looking for direct matches
 - Or more complex patterns

Regular expressions use special (and, at first, somewhat confusing) codes to detect **patterns** in strings of text. For example, if you're presenting your visitors with an HTML form to enter their details, you might have one field for their phone number. Now let's face it: some site visitors are better at following instructions than others. Even if you put a little hint next to the text field indicating the required format of the phone number (e.g.: "(XXX) XXX-XXXX" for North American numbers), some people are going to get it wrong. Writing a script to check every character of the entered string to ensure that all the numbers are where they belong, with parentheses and a dash in just the right spots, would be a pretty tedious bit of code to write. And a telephone number is a relatively simple case! What if you had to check that a user had indeed entered an email address or, worse yet, a URL?

Regular expressions provide a quick and easy way of matching a string to a pattern. In our phone number example, we could write a simple regular expression and use it to check in one quick step whether or not any given string is a properly formatted phone number. We'll explore this example a little further, once we've taken care of a few technical details.

In JavaScript source code, a regular expression is written in the form of `/pattern/modifiers` where "pattern" is the regular expression itself, and "modifiers" are a series of characters indicating various options. The "modifiers" part is optional. This syntax is borrowed from Perl.

QA Creating a regular expression

Regular expressions are objects and must be instantiated.

- There are two ways to do this

```
let re = new RegExp("ab+c");  
let quickRe = /ab+c/;
```

The regular expression object can be used to evaluate a string.

- The example below checks a string and returns a Boolean

```
let str = "bus";  
/bus/.test(str); //only matches on bus
```

Excellent online testing tool: <http://regexr.com/>

QA Using regular expressions in JavaScript (1)

- RegExp also has an **exec** method that returns an **array** (if there's a match) or **null** if there is no match

```
// Match one d followed by one or more b's followed by one d
// Remember matched b's and the following d
// Ignore case
let re = /d(b+)(d)/ig;
let result = re.exec("cdbBdbsbz");
```

- The array contains:
 - 0: The matched text
 - 1-n: The parenthesized substring matches, if any. The number of possible parenthesised substrings is unlimited
 - **Input**: The original string

By default, JavaScript regular expressions are case sensitive and only search for the first match in any given string. By adding the **g** (for global) and **i** (for ignore case) modifiers after the second **/**, you can make a regular expression search for all matches in the string and ignore case, respectively.

The following key methods are of use to us:

Test ()

This one accepts a single string parameter and returns a Boolean indicating whether or not a match has been found. If you don't necessarily need to perform an operation with the a specific matched result – for instance, when validating a username – “test” will do the job just fine.

Exec ()

Executes a search for a match in a specified string. Returns a result array, or null. If the match succeeds, the exec method returns an array and updates properties of the regular expression object. The returned array has the matched text as the first item, and then one item for each capturing parenthesis that matched containing the text that was captured. If the match fails, the exec method returns null. If you are executing a match simply to find true or false, use the test method or the string search method.

QA Using regular expressions in JavaScript (2)

We can also revisit the string methods we looked at earlier in this course and use regular expressions:

- **String.split**

```
let str = 'my little string';  
console.log(str.split(/\s/)); // "my, little, string"
```

- **String.replace**

```
let someString = 'Hello, World';  
someString = someString.replace(/World/, 'Universe');  
console.log(someString); // "Hello, Universe"
```

- **String.match**

```
let name = 'JeffreyWay';  
alert(name.match(/e/g)); // alerts "e,e"
```

As you might expect, the “replace” method allows you to replace a certain block of text, represented by a string or regular expression, with a different string. You’re most likely already familiar with the split method. It accepts a single regular expression that represents where the “split” should occur. Please note that we can also use a string if we’d prefer. In the example above, by passing “\s” – representing a single space – we’ve now split our string into an array. If you need to access one particular value, just append the desired index.

Replace()

As you might expect, the “replace” method allows you to replace a certain block of text, represented by a string or regular expression, with a different string. Using the replace method does not automatically overwrite the value the variable, we must reassign the returned value back to the variable.

Match()

Unlike the “test” method, “match()” will return an array containing each match found. In the code above, there are actually two e’s in the string “JeffreyWay.” Once again, we must use the “g” modifier to declare a “global search”.

QA QuickLab 13b - Using Regular Expressions

- Adding further validation
- Checking for phone numbers
- Checking for postcodes
- Checking for email addresses

QA Hackathon Part 1

- In this part Hackathon, you will build on a partially developed solution (whether that be your previous iteration or the provided starting point) for QA Cinemas' website by adding validation to the form. All the necessary tools, knowledge and techniques have been covered in the course so far
- This part of the Hackathon is intended to help you develop your skills and knowledge to be able to use JavaScript to validate a 'Sign-Up' form for users of the QA Cinemas website before this is sent to be held on the server



REVIEW



- Understanding forms
 - What are forms?
 - Selecting form elements
- Accessing form elements
 - Inputs
 - Radio buttons
 - Select options
- Events
 - Form events
 - Control Events
 - Form validation
- Regular expressions
 - What is RegEx?
 - Using RegEx to analyse data



QA Regular expression operators

- Regular expressions can look complex at times, but are just strings
- Most regular expressions are built up using special codes
 - To create the pattern we want to search for, we use special characters

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character not between the brackets
[0-9]	Find any digit from 0 to 9
[A-Z]	Find any character from uppercase A to uppercase Z
[a-z]	Find any character from lowercase a to lowercase z
[A-z]	Find any character from uppercase A to lowercase z
[adgk]	Find any character in the given set
[^adgk]	Find any character outside the given set
(red blue green)	Find any of the alternatives specified

Metacharacters provide special meaning to a search.

QA Regular expression specificity and global flags

- Modifiers allow us to search for more than one match
- Or provide case insensitive searching

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)

QA Regular expression specificity and global flags

- Quantifiers allow you to refine the search or sub-query a result

Quantifier	Description
n+	Matches any string with at least one n
n*	Matches any string with no occurrences of n
n?	Strings with zero or more occurrences of n
n{x}	Matches any string with a sequence of n's
n\$	Matches any string with n at the end of it
^n	Matches any string with n at the beginning of it