WORKING WITH FORMS AND REGULAR EXPRESSIONS

Validating a Web Form with JavaScript

INTRODUCING REGULAR EXPRESSIONS

- A **regular expression** is a text string that defines a character pattern
- One use of regular expressions is **pattern-matching**, in which a text string is tested to see whether it matches the pattern defined by a regular expression

INTRODUCING REGULAR EXPRESSIONS

- **Creating a regular expression**
  - You create a regular expression in JavaScript using the command

```
re = /pattern/;
```

  - This syntax for creating regular expressions is sometimes referred to as a **regular expression literal**

INTRODUCING REGULAR EXPRESSIONS

- **Matching a substring**
  - The most basic regular expression consists of a substring that you want to locate in the test string
  - The regular expression to match the first occurrence of a substring is **`/chars/`**

INTRODUCING REGULAR EXPRESSIONS

- **Setting regular expression flags**
  - To make a regular expression not sensitive to case, use the regular expression literal `/pattern/i`
  - To allow a global search for all matches in a test string, use the regular expression literal `/pattern/g`

INTRODUCING REGULAR EXPRESSIONS

- Defining character positions

Character	Description	Example
^	Indicates the beginning of the text string	/^GPS/ matches "GPS-ware" but not "Products from GPS-ware"
$	Indicates the end of the text string	/ware$/ matches "GPS-ware" but not "GPS-ware Products"
\b	Indicates the presence of a word boundary	/\bart/ matches "art" and "artists" but not "dart"
\B	Indicates the absence of a word boundary	/art\B/ matches "dart" but not "artist"

INTRODUCING REGULAR EXPRESSIONS

- Defining character positions

Character	Description	Example
\d	A digit (from 0 to 9)	/\dth/ matches "5th" but not "ath"
\D	A non-digit	/\Ds/ matches "as" but not "5s"
\w	A word character (an upper or lower case letter, a digit, or an underscore)	/\w\w/ matches "to" or "A1" but not "$x" or " *"
\W	A non-word character	/\W/ matches "$" or "&" but not "a", "B", or "3"
\s	A white space character (a blank space, tab, new line, carriage return, or form feed)	/\s\d\s/ matches " 5 " but not "5"
\S	A non-white space character	/\S\d\S/ matches "345" or "a5b" but not "5"
.	Any character	/./ matches anything

INTRODUCING REGULAR EXPRESSIONS

- Defining character positions
  - Can specify a collection of characters known a **character class** to limit the regular expression to only a select group of characters

INTRODUCING REGULAR EXPRESSIONS

- Defining character positions

Character	Description	Example
[*chars*]	Match any character in the list of characters, *chars*	/[dog]/ matches "god" and "dog"
[^*chars*]	Do not match any character in *chars*	/[^dog]/ matches neither "god" nor "dog"
[*char1-charN*]	Match characters in the range *char1* through *charN*	/[a-c]/ matches the lowercase letters a through c
[^*char1-charN*]	Do not match characters in the range *char1* through *charN*	/[^a-c]/ does not match the lowercase letters a through c
[a-z]	Match lowercase letters	/[a-z][a-z]/ matches any two consecutive lowercase letters
[A-Z]	Match uppercase letters	/[A-Z][A-Z]/ matches any two consecutive uppercase letters
[a-zA-Z]	Match letters	/[a-zA-Z][a-zA-Z]/ matches any two consecutive letters
[0-9]	Match digits	/[1][0-9]/ matches the numbers "10" through "19"
[0-9a-zA-Z]	Match digits and letters	/[0-9a-zA-Z][0-9a-zA-Z]/ matches any two consecutive letters or numbers

INTRODUCING REGULAR EXPRESSIONS

- Repeating characters

Repetition Character(s)	Description	Example
*	Repeat 0 or more times	/\s*/ matches 0 or more consecutive white space characters
?	Repeat 0 or 1 time	/colou?r/ matches "color" or "colour"
+	Repeat 1 or more times	/\s+/ matches 1 or more consecutive white space characters
{*n*}	Repeat exactly *n* times	/\d{9}/ matches a nine digit number
{*n*, }	Repeat at least *n* times	/\d{9,}/ matches a number with at least nine digits
{*n*,*m*}	Repeat at least *n* times but no more than *m* times	/\d{5,9}/ matches a number with 5 to 9 digits

INTRODUCING REGULAR EXPRESSIONS

- Escape Sequences
  - An **escape sequence** is a special command inside a text string that tells the JavaScript interpreter not to interpret what follows as a character
  - The character which indicates an escape sequence in a regular expression is the backslash character \

INTRODUCING REGULAR EXPRESSIONS

- Escape Sequences

Escape Sequence	Represents	Example
\/	The / character	/\d\/\d/ matches "5/9" "3/1" but not "59" or "31"
\\	The \ character	/\d\\\d/ matches "5\9" or "3\1" but not "59" or "31"
\.	The . character	/\d\.\d\d/ matches "3.20" or "5.95" but not "320" or "595"
*	The * character	/\[a-z]{4}*/ matches "help*" or "pass*"
\+	The + character	/\d\+\d/ matches "5+9" or "3+1" but not "59" or "39"
\?	The ? character	/[a-z]{4}\?/ matches "help?" or "info?"
|	The | character	/a|b/ matches "alb"
\(\)	The (and) characters	/\(\d{3}\)/ matches "(800)" or "(555)"
\{ \}	The { and } characters	/\{[a-z]{4}\}/ matches "{pass}" or "{info}"
\^	The ^ character	/\d+\^\d/ matches "321^2" or "4^3"
\$	The $ character	/\$\d{2}\.\d{2}/ matches "$59.95" or "$19.50"
\n	A new line	/\n/ matches the occurrence of a new line in the text string
\r	A carriage return	/\r/ matches the occurrence of a car-riage return in the text string
\t	A tab	/\t/ matches the occurrence of a tab in the text string

INTRODUCING REGULAR EXPRESSIONS

- Alternating Patterns and Grouping

Characters	Description	Example
pattern1|*pattern2*	Matches either *pattern1* or *pattern2*	/color|colour/ matches either "color" or "colour"
(*pattern*)	Treats *pattern* as a single group and allows a back-reference to the captured group	/(Mr\.\s)?\w+/ matches either "Mr. Smith" or "Smith"
\n	Back-reference to group *n* in the regular expression	/(\s)\1/ matches consecutive occurrences of white space
(?*pattern*)	Treats *pattern* as a single group, but does not allow for back-referencing	

INTRODUCING REGULAR EXPRESSIONS

- The regular expression object constructor
  - To create a regular expression object

    ```
    re = new RegExp(pattern, flags)
    ```

  - *re* is the regular expression object, *pattern* is a text string of the regular expression pattern, and *flags* is a text string of the regular expression flags

WORKING WITH THE REGULAR EXPRESSION OBJECT

- Regular Expression methods

Method	Description
re.compile(*pattern*,*flags*)	Compiles or recompiles a regular expression *re*, where *pattern* is the text string of new regular expression pattern and *flags* are flags applied to the *pattern*
re.exec(*text*)	Executes a search on *text* using the regular expression *re*; pattern results are returned in an array and reflected in the properties of the global RegExp object
re.match(*text*)	Performs a pattern match in *text* using the *re* regular expression; matched substrings are stored in an array
text.replace(*re*, *newsubstr*)	Replaces the substring defined by the regular expression *re* in the text string *text* with *newsubstr*
text.search(*re*)	Searches *text* for a substring matching the regular expression *re*; returns the index of the match, or -1 if no match is found
text.split(*re*)	Splits *text* at each point indicated by the regular expression *re*; the substrings are stored in an array
re.test(*text*)	Performs a pattern match on the text string *text* using the regular expression *re*, returning the Boolean value true if a match is found and false otherwise

TIPS FOR VALIDATING FORMS

- Use selection lists, option buttons, and check boxes to limit the ability of users to enter erroneous data
- Indicate to users which fields are required, and if possible, indicate the format that each field value should be entered in
- Use the maxlength attribute of the input element to limit the length of text entered into a form field

TIPS FOR VALIDATING FORMS

- Format financial values using the toFixed() and toPrecision() methods. For older browsers use custom scripts to format financial data
- Apply client-side validation checks to lessen the load of the server
- Use regular expressions to verify that field values correspond to a required pattern

TIPS FOR VALIDATING FORMS

- Use the length property of the string object to test whether the user has entered a value in a required field
- Test credit card numbers to verify that they match the patterns specified by credit card companies
- Test credit card numbers to verify that they fulfill the Luhn Formula