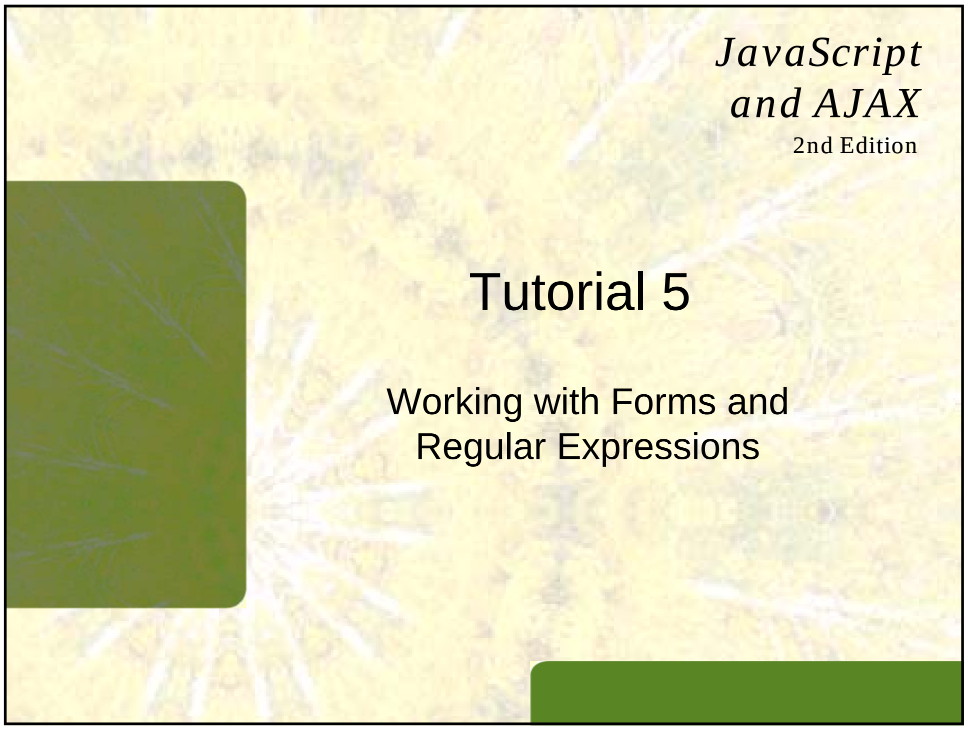
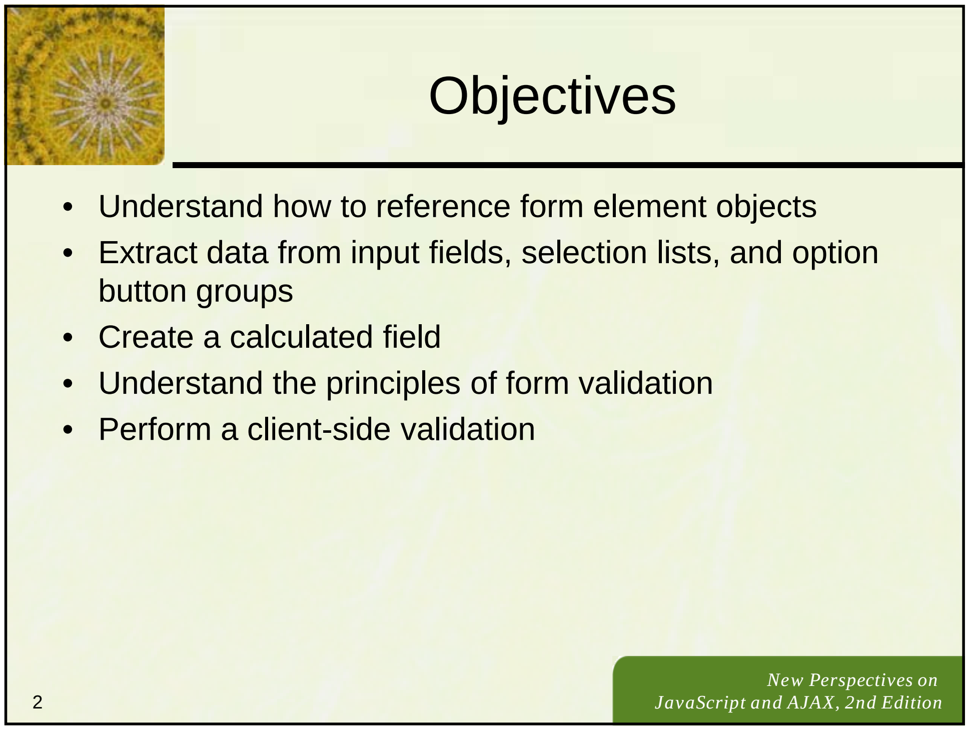


JavaScript and AJAX
2nd Edition

Tutorial 5

Working with Forms and Regular Expressions



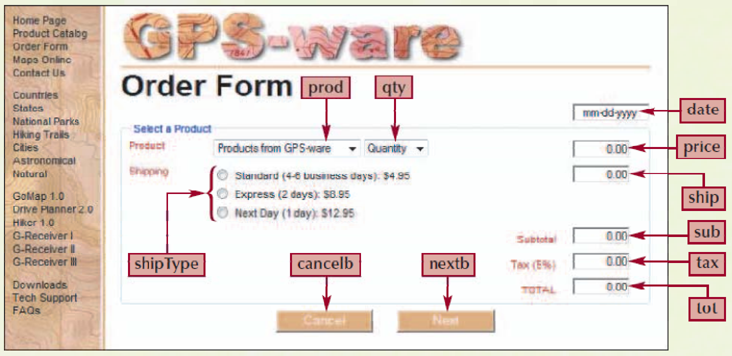
Objectives

- Understand how to reference form element objects
- Extract data from input fields, selection lists, and option button groups
- Create a calculated field
- Understand the principles of form validation
- Perform a client-side validation

Objectives

- Work with string objects
- Work with regular expressions
- Apply regular expressions to zip code fields
- Validate credit card numbers

Working with Forms and Fields



Working with Forms and Fields

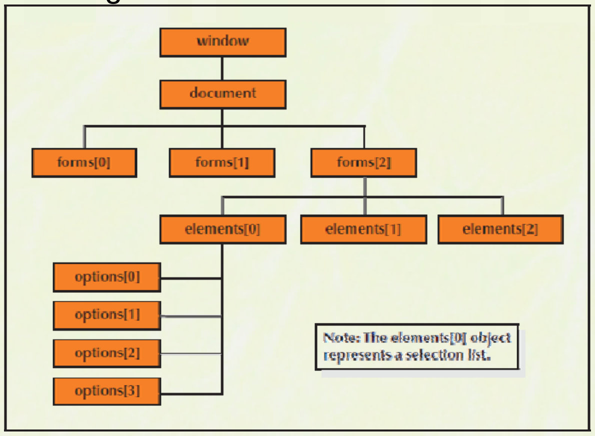
- Referencing a Web form
  - You have to work with the properties and methods of the form object and the elements it contains
  - JavaScript supports an object collection for forms

    `document.forms[`*`idref`*`]` or

    `document.forms["form1"]`

Working with Forms and Fields

- Referencing a Web form

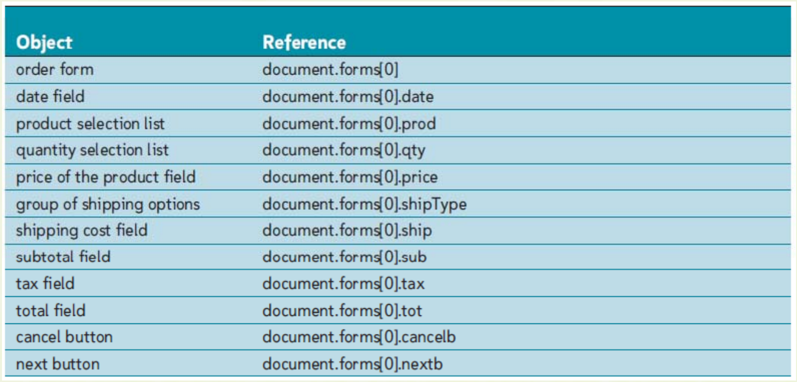


Working with Forms and Fields

- Referencing a form element
  - The elements within a form are organized into an elements collection
  - Can reference a form element either by its position in the collection or by its name or id attributes

Working with Forms and Fields

- Referencing a form element



Object	Reference
order form	document.forms[0]
date field	document.forms[0].date
product selection list	document.forms[0].prod
quantity selection list	document.forms[0].qty
price of the product field	document.forms[0].price
group of shipping options	document.forms[0].shipType
shipping cost field	document.forms[0].ship
subtotal field	document.forms[0].sub
tax field	document.forms[0].tax
total field	document.forms[0].tot
cancel button	document.forms[0].cancelb
next button	document.forms[0].nextb

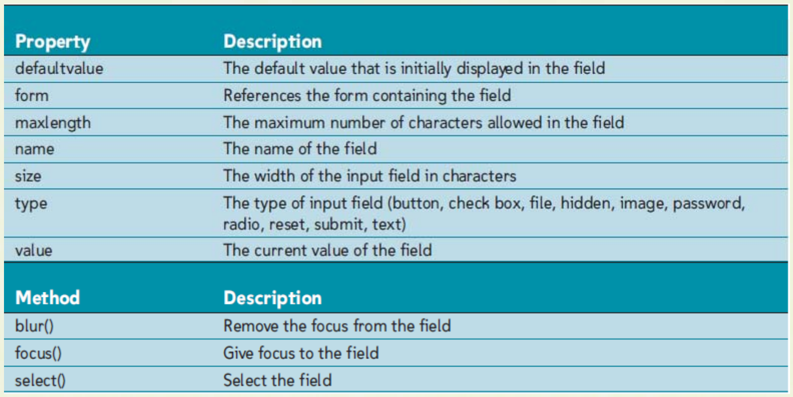
Working with Input Fields

- Setting the field value
  - To set the value contained in a field such as an input box, you use the value property
  - The general syntax is:

    ***formObject.element*.value = *fieldvalue*;**

Working with Input Fields

- Setting the field value



Property	Description
defaultvalue	The default value that is initially displayed in the field
form	References the form containing the field
maxlength	The maximum number of characters allowed in the field
name	The name of the field
size	The width of the input field in characters
type	The type of input field (button, check box, file, hidden, image, password, radio, reset, submit, text)
value	The current value of the field

Method	Description
blur()	Remove the focus from the field
focus()	Give focus to the field
select()	Select the field

Working with Input Fields

- Navigating between fields
  - To place the cursor in a particular field on a form, use:

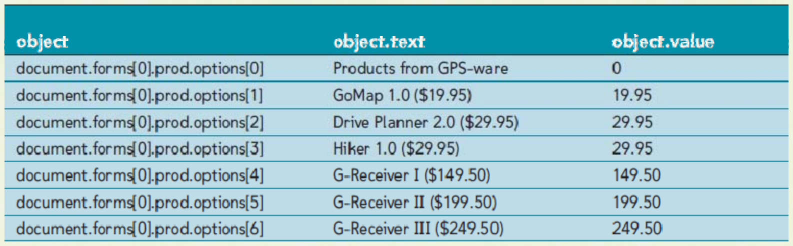
    ```
    formObject.element.focus();
    ```
  - To remove the focus from this field, use:

    ```
    formObject.element.blur();
    ```

Working with Selection Lists

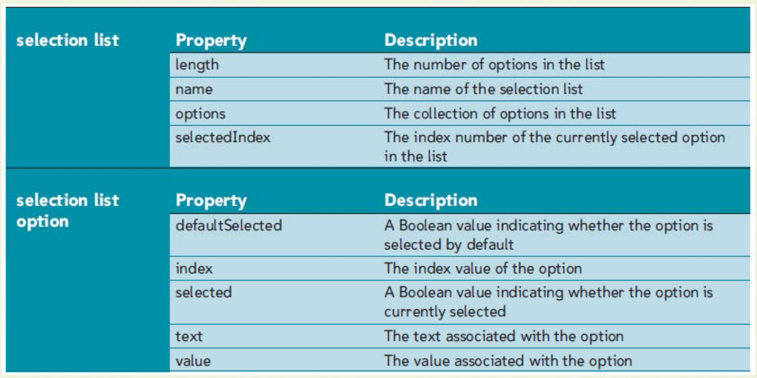
- To reference a particular option in the collection, use:

  ```
  selection.options[idref]
  ```



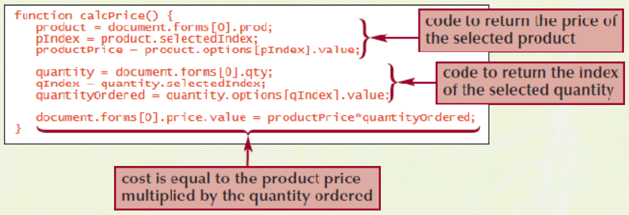
object	object.text	object.value
document.forms[0].prod.options[0]	Products from GPS-ware	0
document.forms[0].prod.options[1]	GoMap 1.0 ($19.95)	19.95
document.forms[0].prod.options[2]	Drive Planner 2.0 ($29.95)	29.95
document.forms[0].prod.options[3]	Hiker 1.0 ($29.95)	29.95
document.forms[0].prod.options[4]	G-Receiver I ($149.50)	149.50
document.forms[0].prod.options[5]	G-Receiver II ($199.50)	199.50
document.forms[0].prod.options[6]	G-Receiver III ($249.50)	249.50

Working with Selection Lists



	Property	Description
selection list	length	The number of options in the list
	name	The name of the selection list
	options	The collection of options in the list
	selectedIndex	The index number of the currently selected option in the list
selection list option	**Property**	**Description**
	defaultSelected	A Boolean value indicating whether the option is selected by default
	index	The index value of the option
	selected	A Boolean value indicating whether the option is currently selected
	text	The text associated with the option
	value	The value associated with the option

Working with Selection Lists



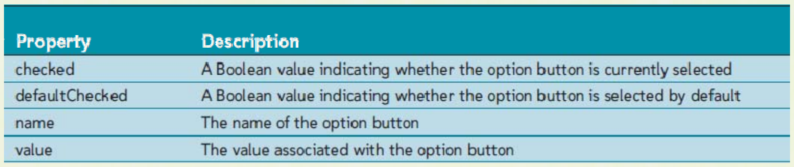
```
function calcPrice() {
   product = document.forms[0].prod;
   pIndex = product.selectedIndex;
   productPrice = product.options[pIndex].value;

   quantity = document.forms[0].qty;
   qIndex = quantity.selectedIndex;
   quantityOrdered = quantity.options[qIndex].value;

   document.forms[0].price.value = productPrice*quantityOrdered;
}
```

code to return the price of the selected product
code to return the index of the selected quantity
cost is equal to the product price multiplied by the quantity ordered

Working with Option Buttons and Check Boxes

- Using option buttons
  - Option buttons have the reference

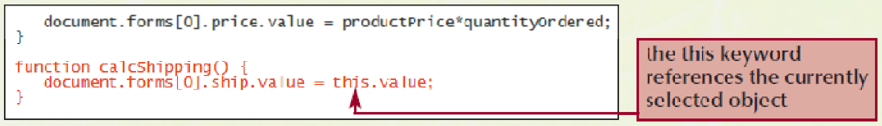
    `options[idref]`
  - Where options is the reference to the group of option buttons and idref is either the index number or id of the individual option button



Property	Description
checked	A Boolean value indicating whether the option button is currently selected
defaultChecked	A Boolean value indicating whether the option button is selected by default
name	The name of the option button
value	The value associated with the option button

Working with Option Buttons and Check Boxes

- Using the “this” keyword
  - The **this keyword** is a JavaScript object reference that refers to the currently selected object, whatever that may be



```
    document.forms[0].price.value = productPrice*quantityOrdered;
}

function calcShipping() {
    document.forms[0].ship.value = this.value;
}
```

the this keyword references the currently selected object

Working with Option Buttons and Check Boxes

- Working with check boxes
  - Work the same way as option buttons
  - In addition, the value associated with a check box is stored in the value property of the check box object
  - This value is applied only when the check box is checked
  - When unchecked, its field has no value assigned to it

Creating Calculated Fields

- Converting between text strings and numeric values
  - Explicitly indicate that you want to convert

    `parseFloat(`*`text`*`)`

Working with Form Validation

- **Form validation** is a process by which the server or user’s browser checks a form for data entry errors
- With **server-side validation**, a form is sent to the Web server for checking
- In **client-side validation**, the Web browser checks the form, which is not submitted to the server until it passes inspection

Working with Form Validation

- Submitting a Form
  - To control this submission process, JavaScript provides the onsubmit event handler

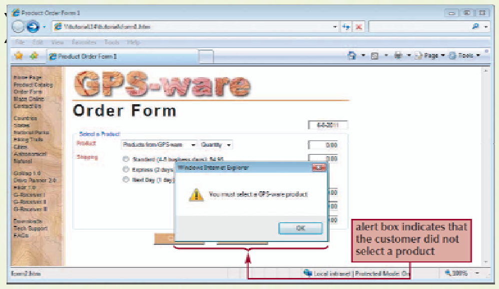
```
formobj.onsubmit = function;
```

  - If the function returns a value of false, the submit event is cancelled, while a value of true allows the submit event to continue unabated

Working with Form Validation

- Alerting the user
  - An **alert box** is a dialog box that displays an informative message to the user along with an OK button

```
alert(message)
```


You must select a GPS-ware product
Product Order Form 1
GPS-ware
Order Form
Windows Internet Explorer
OK
alert box indicates that the customer did not select a product

Working with Text Strings

- The string object
  - JavaScript treats each text string as an object called a **string object**
  - The most common way to create a string object is to assign a text string to a variable
  - You can also create a string object using the object constructor

```
stringVariable = new String("text")
```

Working with Text Strings

- Calculating the length of a text string
  - The following code calculates the number of characters in the stringVar variable, storing the value 17 in the lengthValue variable

```
stringVar = "GPS-ware Products";
lengthValue = stringVar.length
```

Working with Text Strings

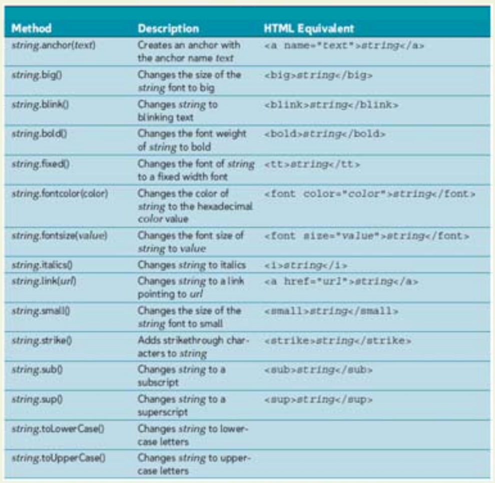
- Working with the string object methods
  - To determine the number of characters in a text string, use the object property **`string.length`**
  - To extract a character from a text string, use the method **`string.charAt(i)`**
  - To extract a substring from a text string, use the method **`string.slice(start, end)`**

Working with Text Strings

- Working with the string object methods
  - To split a string into several substrings, use the command **`strArray = string.split(str)`**
  - To search a string, use the method **`string.indexOf(str, start)`**

Working with Text Strings

- Formatting text strings



Method	Description	HTML Equivalent
string.anchor(*text*)	Creates an anchor with the anchor name *text*	`<a name="text">string</a>`
string.big()	Changes the size of the *string* font to big	`<big>string</big>`
string.blink()	Changes *string* to blinking text	`<blink>string</blink>`
string.bold()	Changes the font weight of *string* to bold	`<bold>string</bold>`
string.fixed()	Changes the font of *string* to a fixed width font	`<tt>string</tt>`
string.fontcolor(color)	Changes the color of *string* to the hexadecimal *color* value	`<font color="color">string</font>`
string.fontsize(*value*)	Changes the font size of *string* to *value*	`<font size="value">string</font>`
string.italics()	Changes *string* to italics	`<i>string</i>`
string.link(*url*)	Changes *string* to a link pointing to *url*	`<a href="url">string</a>`
string.small()	Changes the size of the *string* font to small	`<small>string</small>`
string.strike()	Adds strikethrough characters to *string*	`<strike>string</strike>`
string.sub()	Changes *string* to a subscript	`<sub>string</sub>`
string.sup()	Changes *string* to a superscript	`<sup>string</sup>`
string.toLowerCase()	Changes *string* to lowercase letters	
string.toUpperCase()	Changes *string* to uppercase letters	

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

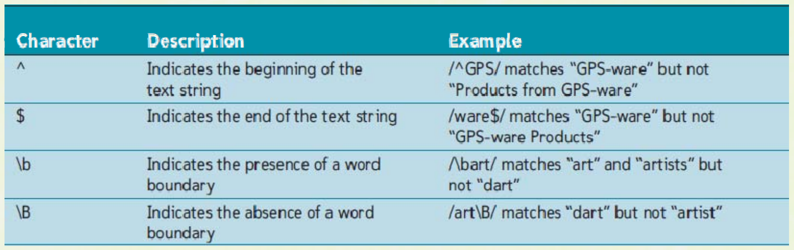
- **Matching a substring**
  - The most basic regular expression consists of a substring that you want to locate in the test string
  - The regular expression to match the first occurrence of a substring is **`/chars/`**

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

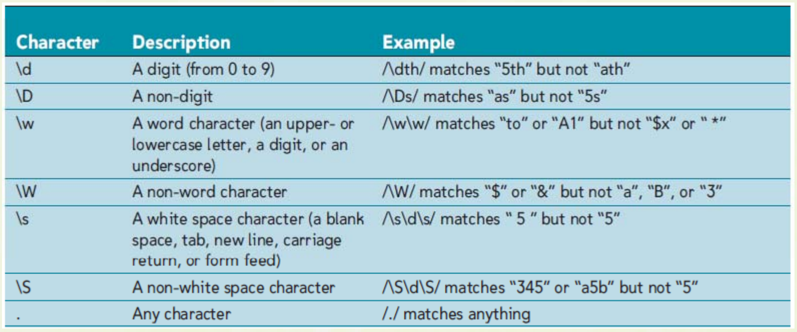
- Defining character positions



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



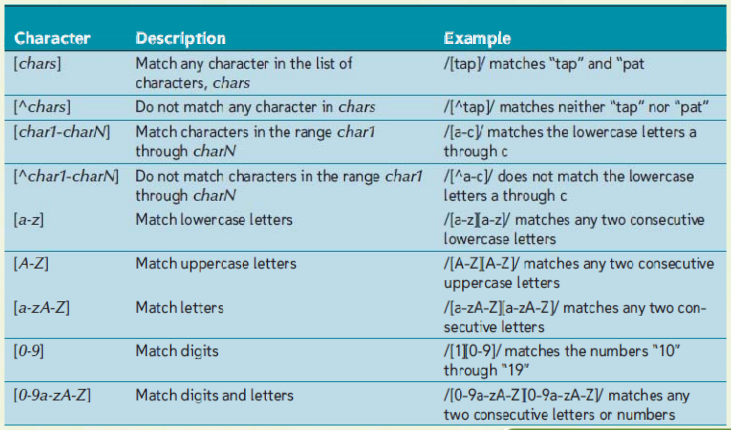
Character	Description	Example
\d	A digit (from 0 to 9)	/\dth/ matches "5th" but not "ath"
\D	A non-digit	/\Ds/ matches "as" but not "5s"
\w	A word character (an upper- or lowercase letter, a digit, or an underscore)	/\w\w/ matches "to" or "A1" but not "$x" or " *"
\W	A non-word character	/\W/ matches "$" 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 as a **character class** to limit the regular expression to only a select group of characters

Introducing Regular Expressions

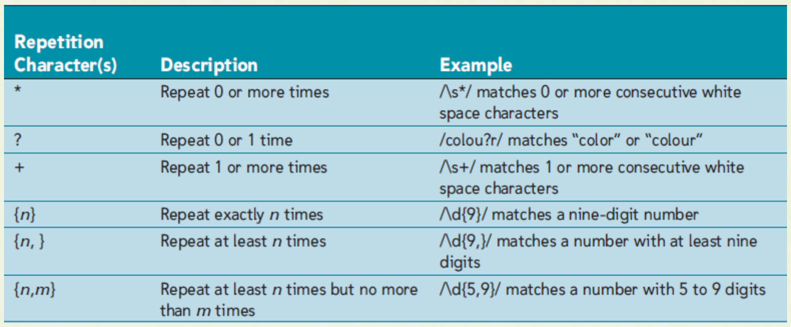
- Defining character positions



Character	Description	Example
[*chars*]	Match any character in the list of characters, *chars*	/[tap]/ matches "tap" and "pat
[^*chars*]	Do not match any character in *chars*	/[^tap]/ matches neither "tap" nor "pat"
[*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

- Repetition characters



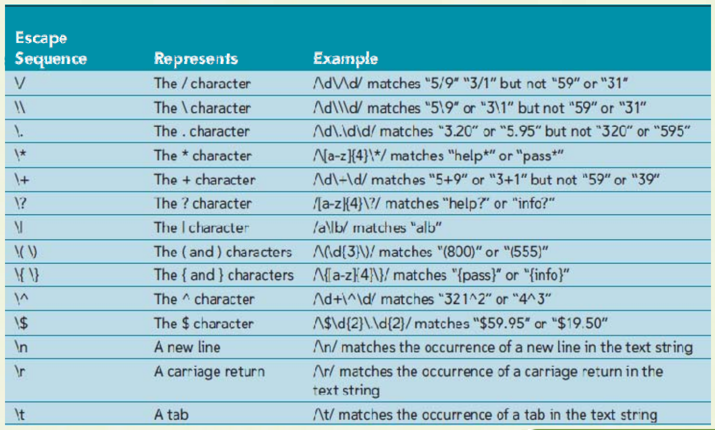
Repetition Character(s)	Description	Example
*	Repeat 0 or more times	/\s*/ matches 0 or more consecutive white space characters
?	Repeat 0 or 1 time	/colou?r/ matches "color" or "colour"
+	Repeat 1 or more times	/\s+/ matches 1 or more consecutive white space characters
{*n*}	Repeat exactly *n* times	/\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

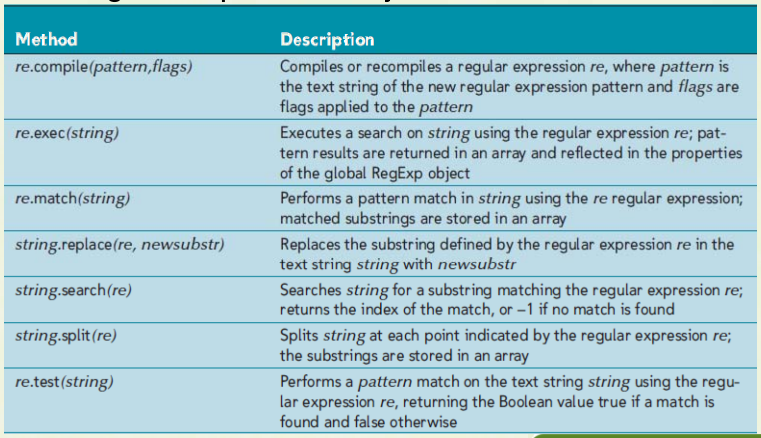
- Escape Sequences



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 "a|b"
\(\)	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 carriage return in the text string
\t	A tab	/\t/ matches the occurrence of a tab in the text string

Introducing Regular Expressions

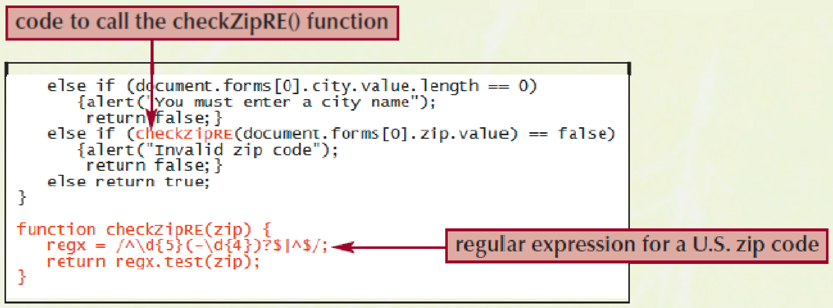
- The regular expression object



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

Working with the Regular Expression Object

- Validating a zip code

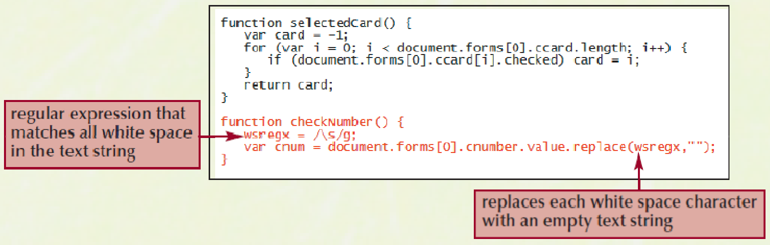

code to call the checkZipRE() function
regular expression for a U.S. zip code

```
    else if (document.forms[0].city.value.length == 0)
       {alert("You must enter a city name");
        return false;}
    else if (checkzipRE(document.forms[0].zip.value) == false)
       {alert("Invalid zip code");
        return false;}
    else return true;
}

function checkzipRE(zip) {
   regx = /^\d{5}(-\d{4})?$|^$/;
   return regx.test(zip);
}
```

39

Validating Financial Information

- Removing blank spaces from credit card numbers

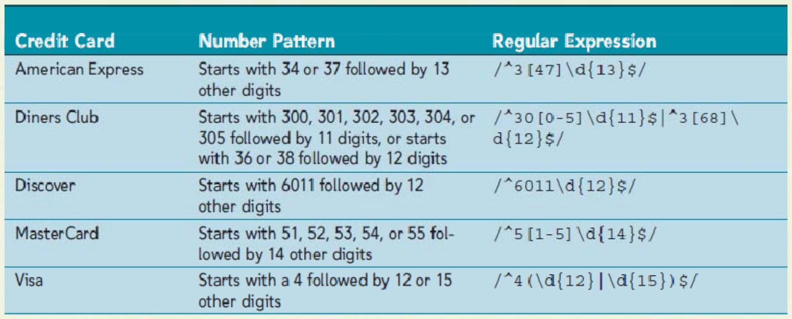

regular expression that matches all white space in the text string
replaces each white space character with an empty text string

```
function selectedCard() {
   var card = -1;
   for (var i = 0; i < document.forms[0].ccard.length; i++) {
      if (document.forms[0].ccard[i].checked) card = i;
   }
   return card;
}

function checkNumber() {
   wsregx = /\s/g;
   var cnum = document.forms[0].cnumber.value.replace(wsregx,"");
}
```

40

Validating Financial Information

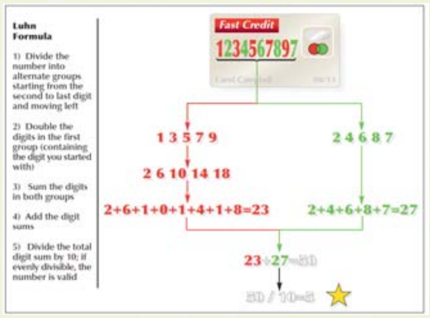
- Validating credit card number patterns



Credit Card	Number Pattern	Regular Expression
American Express	Starts with 34 or 37 followed by 13 other digits	`/^3[47]\d{13}$/`
Diners Club	Starts with 300, 301, 302, 303, 304, or 305 followed by 11 digits, or starts with 36 or 38 followed by 12 digits	`/^30[0-5]\d{11}$\|^3[68]\d{12}$/`
Discover	Starts with 6011 followed by 12 other digits	`/^6011\d{12}$/`
MasterCard	Starts with 51, 52, 53, 54, or 55 followed by 14 other digits	`/^5[1-5]\d{14}$/`
Visa	Starts with a 4 followed by 12 or 15 other digits	`/^4(\d{12}|\d{15})$/`

Validating Financial Information

- The Luhn Formula
  - All credit card numbers must satisfy the **Luhn Formula**, or **Mod10 algorithm**, which is a formula developed by a group of mathematicians in the 1960s to provide a quick validation check on an account number by adding up the digits in the number



Luhn Formula

1) Divide the number into alternate groups starting from the second to last digit and moving left

2) Double the digits in the first group (containing the digit you started with)

3) Sum the digits in both groups

4) Add the digit sums

5) Divide the total digit sum by 10; if evenly divisible, the number is valid

Passing Data Between Forms

- Appending Form Data
  - Text strings can be appended to any URL by adding the ? character to the Web address followed by the text string
    - Use the get method
  - One property of the location object is the location.search property, which contains the text of any data appended to the URL, including the ? character

Passing Data from a Form

- Appending data to a URL
  - There are several limitations to the technique of appending data to a URL
  - URLs are limited in their length
  - Characters other than letters and numbers cannot be passed in the URL without modification
  - URLs cannot contain blank spaces, for example, a blank space is converted to the character code %20

Passing Data from a Form

- Appending and retrieving form data
  - Can use the technique of appending data to the URL with Web forms, too
  - Do this by setting a form's action attribute to the URL of the page to which you want to pass the data, and setting the method of the form to "get"

Passing Data from a Form

- Appending and retrieving form data
  - Use the location.search property and the slice() method to extract only the text string of the field names and values
  - Use the unescape() function to remove any escape sequences characters from the text string
  - Convert each occurrence of the + symbol to a blank space
  - Split the text string at every occurrence of an = or & character, storing the substrings into an array