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
- **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**

– **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/**

– **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**

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"

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

- 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

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

- 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

- 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 \

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 carriage return in the text string
\t	A tab	/\t/ matches the occurrence of a tab in the text string

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	

- 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

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

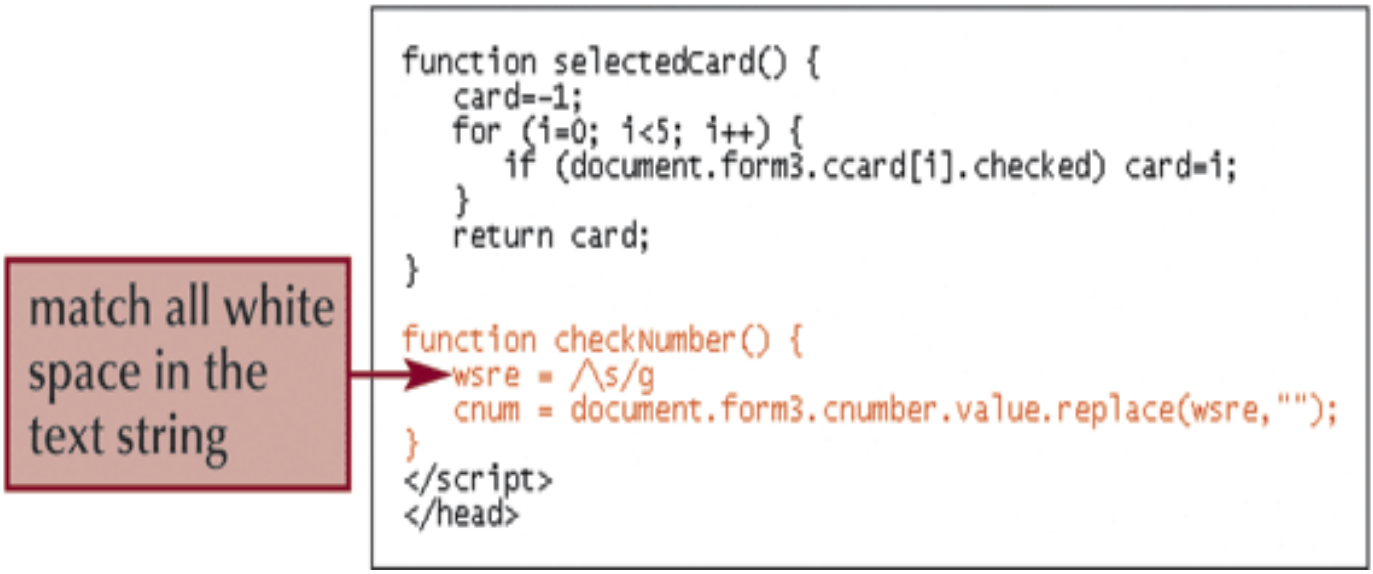
Validating a ZIP code

```
function checkForm2() {
   if (document.form2.fname.value.length == 0)
      {alert("You must enter a first name");
       return false;}
   else if (document.form2.lname.value.length == 0)
      {alert("You must enter a last name");
       return false;}
   else if (document.form2.street.value.length == 0)
      {alert("You must enter a street address");
       return false;}
   else if (document.form2.city.value.length == 0)
      {alert("You must enter a city name");
       return false;}
   else if (checkZip2(document.form2.zip.value) == false)
      {alert("You must enter a valid zip code");
       return false;}
   else return true;
}

function checkZip(zip) {
   if (zip.length != 5 && zip.length != 0) return false;
   else {
      validchars = "0123456789";
      for (i=0; i<5; i++) {
         zipchar=zip.charAt(i);
         if (validchars.indexOf(zipchar) == -1) return false;
      }
   }
   return true;
}

function checkZip2(zip) {
   re = /^\d{5}(-\d{4})?$|^$/;
   return re.test(zip);
}
</script>
```

Validating Financial Information

Object	Reference
The American Express radio button	document.form3.ccard[0]
The Diners Club radio button	document.form3.ccard[1]
The Discover radio button	document.form3.ccard[2]
The MasterCard radio button	document.form3.ccard[3]
The Visa radio button	document.form3.ccard[4]
The credit card name field	document.form3.cname
The credit card number field	document.form3.cnumber
The credit card month selection list	document.form3.cmonth
The credit card year selection list	document.form3.cyear
The previous button	document.form3.prevb
The next button	document.form3.nextb

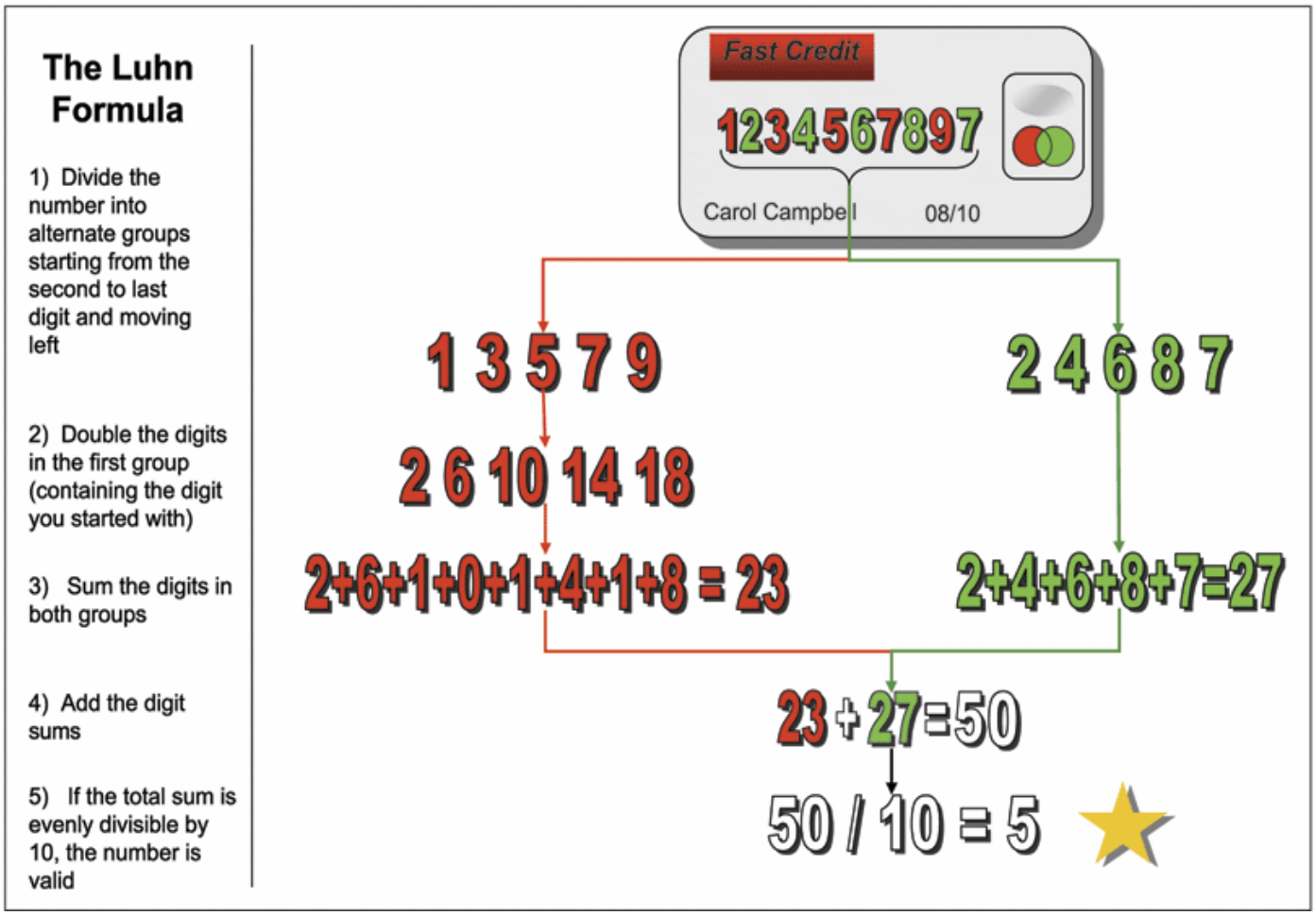
- Removing blank spaces from credit card numbers



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}$/`
Diners Club	Starts with 300-305 followed by 11 digits, or starts with 36 or 38 followed by 12 digits	`/^30[0-5]\d{11}$\|^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}$/`
Visa	Starts with a 4 followed by 12 or 15 other digits	`/^4(\d{12}|\d{15})$/`

- 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



Passing Data from a Form

- Appending data to a URL
  - Text strings can be appended to any URL by adding the ? character to the Web address followed by the text string

Go to form2

  - 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
  - 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
  - Because URLs cannot contain blank spaces, for example, a blank space is converted to the character code %20

- 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"
  - 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 a = or & character, storing the substrings into an array

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
- 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

- 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

Summary

Regular expressions are objects that represent patterns in strings. They use their own syntax to express these patterns.

/abc/ A sequence of characters

/[abc]/ Any character from a set of characters

/[^abc]/ Any character *not* in a set of characters

/[0-9]/ Any character in a range of characters

/x+/ One or more occurrences of the pattern x

/x+?/ One or more occurrences, nongreedy

/x*/ Zero or more occurrences

/x?/ Zero or one occurrence

/x{2,4}/ Between two and four occurrences

/(abc)/ A group

/a|b|c/ Any one of several patterns

/\d/ Any digit character

/\w/ An alphanumeric character (“word character”)

/\s/ Any whitespace character

/./ Any character except newlines

/\b/ A word boundary

/^/ Start of input

/$/ End of input

- A **regular expression** has a method **test** to test whether a given string matches it. It also has an exec method that, when a match is found, returns an array containing all matched groups. Such an array has an index property that indicates where the match started.
- **Strings** have a **match** method to match them against a regular expression and a search method to search for one, returning only the starting position of the match. Their replace method can replace matches of a pattern with a replacement string. Alternatively, you can pass a function to replace, which will be used to build up a replacement string based on the match text and matched groups.
- Regular expressions can have options, which are written after the closing slash. The i option makes the match case insensitive, while the g option makes the expression *global*, which, among other things, causes the replace method to replace all instances instead of just the first.
- The RegExp constructor can be used to create a regular expression value from a string.
- Regular expressions are a sharp tool with an awkward handle. They simplify some tasks tremendously but can quickly become unmanageable when applied to complex problems. Part of knowing how to use them is resisting the urge to try to shoehorn things that they cannot sanely express into them.

==========

Some special cases:

- `\\s` - matches single whitespace character
- `\\s+` - matches sequence of one or more whitespace characters.

Greedy quantifiers
X? X, once or not at all
X* X, zero or more times
X+ X, one or more times
X{n} X, exactly n times
X{n,} X, at least n times
X{n,m} X, at least n but not more than m times

Reluctant quantifiers
X?? X, once or not at all
X*? X, zero or more times
X+? X, one or more times
X{n}? X, exactly n times
X{n,}? X, at least n times
X{n,m}? X, at least n but not more than m times

Possessive quantifiers
X?+ X, once or not at all
X*+ X, zero or more times
X++ X, one or more times
X{n}+ X, exactly n times
X{n,}+ X, at least n times
X{n,m}+ X, at least n but not more than m times