OBJECT™ TECHNOLOGIES

Javascript Regular Expressions

What will be covered

OBJECT TECHNOLOGIES

- What is Regular Expressions
- Creating a regular expression
- Symbols and special characters in RegEx
- Using Regular Expressions
- String functions for using regular expressions
- Some examples
- RegExp methods
- Using RegExp in form validation

What is Regular Expressions

OBJECT TECHNOLOGIES

- Regular expressions are patterns used to match character combinations in strings.
- A regular expression is an object that describes a pattern of characters.
- String and **RegExp** define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.
- These patterns are used with the exec and test methods of RegExp.
- Theses patterns are used with the match, replace, search, and split methods of String

Creating a regular expression

OBJECT TECHNOLOGIES

- A regular expression can be created in one of two ways:

  1) Using a regular expression literal, which consists of a pattern enclosed between slashes, as follows:

```
var re = /ab+c/;
```

  2) By calling the constructor function of the RegExp object, as follows:

```
var re = new RegExp('ab+c');
```

General syntax is :

```
var pattern = new RegExp(pattern, attributes);

or simply

var pattern = /pattern/attributes;
```

Symbols and special characters in RegEx

OBJECT TECHNOLOGIES

1) A caret (^) may be used to indicate the beginning of the string, while a dollar sign ($) is used to mark the end:

- E.g.
- JavaScript - Matches "Isn't JavaScript great?“
- ^JavaScript - Matches "JavaScript rules!", not "What is JavaScript?"
- JavaScript$ - Matches "I love JavaScript", not "JavaScript is great!"
- ^JavaScript$ - Matches "JavaScript", and nothing else

2) sometimes want to use ^, $, or other special characters to represent the corresponding character in the search string rather than the special meaning implied by regular expression syntax. To remove the special meaning of a character, prefix it with a backslash:

Symbols and special characters in RegEx

OBJECT TECHNOLOGIES

- E.g.
- \$\$\$ // Matches "Show me the $$$!"

3) Square brackets may be used to define a set of characters that may match. For example, the following regular expression will match any digit from 1 to 5 inclusive. Ranges of numbers and letters may also be specified.

- E.g.
- [12345] // Matches "1" and "3", but not "a" or "12"
- [1-5] // Same as the previous example
- [a-z] // Matches any lowercase letter (from the English alphabet)
- [0-9a-zA-Z] // Matches any letter or digit

4) By putting a ^ immediately following the opening square bracket, you can **invert** the set of characters, meaning the set will match any character not listed:

- **E.g.**
- [^a-zA-Z] // Matches anything except a letter

Symbols and special characters in RegEx

OBJECT TECHNOLOGIES

5) The characters ?, +, and * also have special meanings.
   - ? means "the preceding character is optional",
   - + means "one or more of the previous character",
   - * means "zero or more of the previous character".

- E.g.
- bana?na // Matches "banana" and "banna",
  // but not "banaana".
- bana+na // Matches "banana" and "banaana",
  // but not "banna".
- bana*na // Matches "banna", "banana", and "banaaana",
  // but not "bnana".

6) Parentheses may be used to group strings together to apply ?, +, or * to them as a whole.

- E.g.
- ba(na)+na // Matches "banana" and "bananananana",
  // but not "bana" or "banaana".

Symbols and special characters in RegEx

OBJECT TECHNOLOGIES

7) Parentheses also let you define several strings that may match, using the pipe (|) character to separate them.

- E.g
- ^(ba|na)+$ // Matches "banana", "nababa", "baba", "nana", "ba", "na", and others.

8) A few special codes

\d - Any digit (same as [0-9])

\D - Anything but not a digit (same as [^0-9])

\s - Single whitespace (space, tab, newline, etc.)

\S - Single nonwhitespace

\w - A "word character" (same as [A-Za-z0-9_])

\W - A "nonword character" (same as [^A-Za-z0-9_])

Using Regular Expressions

OBJECT

- 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
- Let us consider the string "test1 Test2 TEST3“ and few expressions to match with the string

RegExp	Match(es)
/Test[0-9]+/	“Test2” only
/Test[0-9]+/i	“test1” only
/Test[0-9]+/gi	“test1”,”Test2” and “TEST3”

String functions for using regular expression

- **match()**
- match() takes a regular expression as a parameter and returns an array of all the matching strings found in the string under consideration. If no matches are discovered, then match()returns false.
- **replace()**
- replace() lets you replace matches to a given regular expression with some new string
- **search()**
- The search() function is similar to the well-known indexOf() function, except it takes a regular expression instead of a string. It then searches the string for the first match to the given regular expression and returns an integer indicating the position in the string (e.g. 0 if the match is at the start of the string, 9 if the match begins with the 10th character in the string). If no match is found, the function returns a value of -1.

Some Examples..

OBJECT TECHNOLOGIES

- Few examples of regular expressions for very frequent data validations
- For validating name :

```
var myPattern = /^[a-z]{2,}$/
```

Above pattern allows only lower case letters from length 2 onwards

```
var myPattern = /^[A-Z][a-z]{1,}\s[A-Z][a-z]{1,}/
```

Above pattern will allow full name to be written with space in between and first letter will be capital

Some Examples..

OBJECT

- For validating phone number of format (333) 333-3333

```
var myPattern = /^\(\d\d\d\) \d\d\d\-\d\d\d\d$/
```

  - \d indicates any digit from 0 to 9
  - \(indicates presence of (symbol
  - \) indicates presence of) symbol
  - - should be appear as it is
- For validating email id

```
var myPattern = /^\w{6,}\@\w{4,}\.\w{2,}$/
```

  - \w indicates word character
  - {6,} indicates minimum 6 occurrences, no limit for maximum
  - \@ indicates presence of @ symbol
  - \. Indicates presence of . symbol

Some Examples..

- For validating password with atleast one digit, one upper case and one special character

```
var myPattern = /(?=.*[A-Z])(?=.*\d)(?=.*[~!@#$%^&*])[A-Za-z0-9~!@#\s$%^&*]{8,}/
```

  - (?=.*[A-Z]) indicates at least one occurrence of capital letter
  - (?=.*\d) indicates at least one occurrence of digit
  - (?=.*[~!@#$%^&*]) indicates at least one occurrence of special character
  - {8,} indicates minimum length as 8

RegExp methods

OBJECT TECHNOLOGIES

- exec : A RegExp method that executes a search for a match in a string. It returns an array of information or null on a mismatch.

```
/e/.exec("The best things in life are free!");
```

Above function returns 'e' since there is an 'e' in the string

- test : A RegExp method that tests for a match in a string. It returns true or false.

```
var patt = /e/;
patt.test("The best things in life are free!");
```

Since there is an "e" in the string, the output of the code above will be true

Using RegExp in form validation

```
<form action="" onsubmit="validate()">
  <input type="text"id="phone-number" placeholder="phone number"/> <br>
  <input type="text" id="postal-code" placeholder="postal code" /><br>
  <input type="submit" />
</form>
```

- This simple form accepts phone-number and postal code. For specifying pattern of phone-number and postal-code, regular expressions can be used. That helps in form validation ina simple way.
- /^[(]{0,1}[0-9]{3}[)]{0,1}[-\s\.]{0,1}[0-9]{3}[-\s\.]{0,1}[0-9]{4}$/

Basically the expression tries to match with a phone number like this (541) 754-3010 or like this 541-754-3010 or with spaces

- /^[A-Z]{1,2}[0-9]{1,2} ?[0-9][A-Z]{2}$/i

This expression checks for 1 or 2 alphabetic characters, followed by 1 or 2 digits, then a space and one digit and exactly two alphabetic characters.

Using RegExp in form validation

OBJECT TECHNOLOGIES

```
function validate(){
  var phoneNumber = document.getElementById('phone-number').value;
  var postalCode = document.getElementById('postal-code').value;
  var phoneRGEX = /^[(]{0,1}[0-9]{3}[)]{0,1}[-\s\.]{0,1}[0-9]{3}[-\s\.]{0,1}[0-
9]{4}$/;
  var postalRGEX = /^[A-Z]{1,2}[0-9]{1,2} ?[0-9][A-Z]{2}$/i;
  var phoneResult = phoneRGEX.test(phoneNumber);
  var postalResult = postalRGEX.test(postalCode);
  if(phoneResult == false)
  {
    alert('Please enter a valid phone number');
    return false;
  }

  if(postalResult == false)
  {
    alert('Please enter a valid postal number');
    return false;
  }

  return true;
}
```

Regular Expressions

Regular expressions are patterns used to match character combinations in strings.It helps the programmers to validate text string. It offers a powerful tool to analyze and search a pattern as well as to modify the text string. In this lesson we will learn about different symbols used for constructing the RegExp pattern and examples of the RegExp patterns that are frequently required.

Introduction to Regular Expressions

Regular expressions form a small, separate language that is part of JavaScript and many other languages and systems with some differences betwen each implmentation

Regular expressions are patterns used to match character combinations in strings.

A regular expression is an object that describes a pattern of characters.

By formulating a regular expression with a special syntax, you can

- search text a string
- replace substrings in a string
- extract information from a string

These patterns are used with the exec and test methods of RegExp.

These patterns are used with the match, replace, search, and split methods of String

Creating regular expression

A regular expression can be created in one of two ways:

1. Using a regular expression literal, which consists of a pattern enclosed between slashes, as follows:
   var re = /ab+c/;
2. By calling the constructor function of the RegExp object, as follows:
   var re = new RegExp('ab+c');

General syntax is :

```
var pattern = new RegExp(pattern, attributes)
      OR
var pattern = /pattern/attributes
```

Syntax and Rules

1. A caret (^) may be used to indicate the beginning of the string, while a dollar sign ($) is used to mark the end

   E.g.

   JavaScript - Matches "Isn't JavaScript great?"

   ^JavaScript - Matches "JavaScript rules!",
   not "What is JavaScript?"

   JavaScript$ - Matches "I love JavaScript",
   not "JavaScript is great!"

   ^JavaScript$ - Matches "JavaScript", and nothing else

2. Sometimes want to use ^, $, or other special characters to represent the corresponding character in the search string rather than the special meaning implied by regular expression syntax. To remove the special meaning of a character, prefix it with a backslash:

   E.g.

   \$\$\$ // Matches "Show me the $$$!"

3. Square brackets may be used to define a set of characters that may match. For example, the following regular expression will match any digit from 1 to 5 inclusive. Ranges of numbers and letters may also be specified.

   E.g.

   [12345] // Matches "1" and "3", but not "a" or "12"

   [1-5] // Same as the previous example

   [a-z] // Matches any lowercase letter (from the English alphabet)

   [0-9a-zA-Z] // Matches any letter or digit

4. By putting a ^ immediately following the opening square bracket, you can invert the set of characters, meaning the set will match any character not listed:

   E.g.

   [^a-zA-Z] // Matches anything except a letter

5. The characters ?, +, and * also have special meanings. They are referred as multiplicity characters

   ? means "the preceding character is optional",

   + means "one or more of the previous character",

   * means "zero or more of the previous character".

   E.g.

5. The characters ?, +, and * also have special meanings. They are referred as multiplicity characters

   ? means "the preceding character is optional",

   + means "one or more of the previous character",

   * means "zero or more of the previous character".

   E.g.

```
bana?na        // Matches "banana" and "banna",
               // but not "banaana".
bana+na        // Matches "banana" and "banaana",
               // but not "banna".
bana*na        // Matches "banna", "banana", and "banaaana",
               // but not "bnana".
```

6. Parentheses may be used to group strings together to apply ?, +, or * to them as a whole.

   E.g.

```
ba(na)+na      // Matches "banana" and "bananananana",
               // but not "bana" or "banaana".
```

7. Parentheses also let you define several strings that may match, using the pipe (|) character to separate them.

   E.g

   ^(ba|na)+$ // Matches "banana", "nababa", "baba", "nana", "ba", "na", and others

8. A few special codes

   \d - Any digit (same as [0-9])

   \D - Anything but not a digit (same as [^0-9])

   \s - Single whitespace (space, tab, newline, etc.)

   \S - Single nonwhitespace

   \w - A "word character" (same as [A-Za-z0-9_])

   \W - A "nonword character" (same as [^A-Za-z0-9_])

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

Let us consider the string "test1 Test2 TEST3" and few expressions to match with the string

RegExp	Match(es)
/Test[0-9]+/	"Test2" only

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

Let us consider the string "test1 Test2 TEST3“ and few expressions to match with the string

RegExp	Match(es)
/Test[0-9]+/	“Test2” only
/Test[0-9]+/i	“test1” only
/Test[0-9]+/gi	“test1”,”Test2” and “TEST3”

Pattern Matching

Regular expression objects have a number of methods.

The simplest one is test. If you pass it a string, it will return a Boolean telling you whether the string contains a match of the pattern in the expression.

```
console.log(/abc/.test("abcde"));// → true
console.log(/abc/.test("abxde"));// → false
```

If abc occurs anywhere in the string we are testing against (not just at the start), test will return true.

Some important String functions are :

Method	Purpose
match()	match() takes a regular expression as a parameter and returns an array of all the matching strings found in the string under consideration. If no matches are discovered, then match()returns false.
replace()	lets you replace matches to a given regular expression with some new string
search()	The search() function is similar to the well-known indexOf() function, except it takes a regular expression

Pattern Matching

Regular expression objects have a number of methods.

The simplest one is test. If you pass it a string, it will return a Boolean telling you whether the string contains a match of the pattern in the expression.

```
console.log(/abc/.test("abcde"));// → true
console.log(/abc/.test("abxde"));// → false
```

If abc occurs anywhere in the string we are testing against (not just at the start), test will return true.

Some important String functions are :

Method	Purpose
match()	match() takes a regular expression as a parameter and returns an array of all the matching strings found in the string under consideration. If no matches are discovered, then match()returns false.
replace()	lets you replace matches to a given regular expression with some new string
search()	The search() function is similar to the well-known indexOf() function, except it takes a regular expression instead of a string. It then searches the string for the first match to the given regular expression and returns an integer indicating the position in the string (e.g. 0 if the match is at the start of the string, 9 if the match begins with the 10th character in the string). If no match is found, the function returns a value of -1.
split()	Uses a regular expression or a fixed string to break a string into an array of substrings

Few examples of regular expressions for very frequent data validations

1. For validating name :

   ```
   var pattern = /^[a-z]{2,}$/
   ```

   Above pattern allows only lower case letters from length 2 onwards

2. For validating full name :

   ```
   var pattern = /^[A-Z][a-z]{1,}\s[A-Z][a-z]{1,}$/
   ```

   Above pattern will allow full name to be written with space in between and first letter will be capital

3. For validating phone number of format (333) 333-3333

   ```
   var pattern = /^\(\d\d\d\) \d\d\d\-\d\d\d\d/$
   ```

   \d indicates any digit from 0 to 9

   \(indicates presence of (symbol

Few examples of regular expressions for very frequent data validations

1. For validating name :

   var pattern = /^[a-z]{2,}$/

   Above pattern allows only lower case letters from length 2 onwards

2. For validating full name :

   var pattern = /^[A-Z][a-z]{1,}\s[A-Z][a-z]{1,}$/

   Above pattern will allow full name to be written with space in between and first letter will be capital

3. For validating phone number of format (333) 333-3333

   var pattern = /^\(\d\d\d\) \d\d\d\-\d\d\d\d/$

   \d indicates any digit from 0 to 9

   \(indicates presence of (symbol

   \) indicates presence of) symbol

   - should be appear as it is

4. For validating email id

   var pattern = /^\w{6,}\@\w{4,}\.\w{2,}$/

   \w indicates word character

   {6,} indicates minimum 6 occurrences, no limit for maximum

   \@ indicates presence of @ symbol

   \. Indicates presence of . symbol

5. For validating password with atleast one digit, one upper case and one special character

   var pattern = /(?=.*[A-Z])(?=.*\d)(?=.*[~!@#$%^&*])[A-Za-z0-9~!@#$%^&*()]{8,}/

   (?=.*[A-Z]) indicates at least one occurrence of capital letter

   (?=.*\d) indicates at least one occurrence of digit

   (?=.*[~!@#$%^&*]) indicates at least one occurrence of special character

   {8,} indicates minimum length as 8

Assignments

1. Refer HTML form assignment no 1:

   Before submitting data to server side validate login id and password.

Assignments

1. Refer HTML form assignment no 1:
   Before submitting data to server side validate login id and password.
   Loginid and password should not be blank.
   Userid should be in the range of 5 to 15 characters and can have only upper and lower alphabets and dot(.) symbol.
   Password should be in the range of 8 to 15 characters and should have only one numeric, capital and special character.
2. Refer HTML form assignment no 2:
   Apply follwing validation-
   Email should be having email format
   Occupation should be selected
   One of the radio buttons should be selected
3. Refer HTML form assignment no 3:
   Apply following validation for the form created in HTML assignment
   No field should be blank
   First name and last name should contain only alphabets
   Account no should be numeric having 10 digits
   Password should be minimum 5 chracters and should contain atleast one numeric and one special character.
   Display the message beside the text field as weak, average or strong password. If password contains all alphabets, it is treated as weak password, if alphabets with one numeric or special character, average password and alphabets with numeric as well as special character, strong password
   Password and retype password should be same
4. Create a regular expression patterns for the following
   1. for accepting first name. First letter capital and rest letters small alphabets
   2. for accepting full name. First name should be followed by last name with one space.
   3. accepting mobile no of 10 digits.
   4. for accepting email id
   5. for accepting course name where first 2 letters should be "PG" followed by "-" followed by capital letters having 3 to 8 characters

Assignments

1. Refer HTML form assignment no 1:
   Before submitting data to server side validate login id and password.
   Loginid and password should not be blank.
   Userid should be in the range of 5 to 15 characters and can have only upper and lower alphabets and dot(.) symbol.
   Password should be in the range of 8 to 15 characters and should have only one numeric, capital and special character.
2. Refer HTML form assignment no 2:
   Apply follwing validation-
   Email should be having email format
   Occupation should be selected
   One of the radio buttons should be selected
3. Refer HTML form assignment no 3:
   Apply following validation for the form created in HTML assignment
   No field should be blank
   First name and last name should contain only alphabets
   Account no should be numeric having 10 digits
   Password should be minimum 5 chracters and should contain atleast one numeric and one special character.
   Display the message beside the text field as weak, average or strong password. If password contains all alphabets, it is treated as weak password, if alphabets with one numeric or special character, average password and alphabets with numeric as well as special character, strong password
   Password and retype password should be same
4. Create a regular expression patterns for the following
   1. for accepting first name. First letter capital and rest letters small alphabets
   2. for accepting full name. First name should be followed by last name with one space.
   3. accepting mobile no of 10 digits.
   4. for accepting email id
   5. for accepting course name where first 2 letters should be "PG" followed by "-" followed by capital letters having 3 to 8 characters