CSC-318
Web Technology
(BSc CSIT, TU)

Ganesh Khatri
kh6ganesh@gmail.com

JavaScript Regular Expressions

- A regular expression is a sequence of characters that forms a search pattern
- The search pattern can be used for text search and text replace operations
- When you search for data in a text, you can use this search pattern to describe what you are searching for
- A regular expression can be a single character, or a more complicated pattern
- Regular expressions can be used to perform all types of text search and text replace operations

JavaScript Regular Expressions

- Syntax :

```
/pattern/modifiers;
```

- Example :

```
var patt = /w3schools/i;
```

- /w3schools/i is a regular expression.
- w3schools is a pattern (to be used in a search)
- i is a modifier (modifies the search to be case-insensitive)

Using String Methods

- In JavaScript, regular expressions are often used with the two string methods: search() and replace()
- The search() method uses an expression to search for a match, and returns the position of the match
- The replace() method returns a modified string where the pattern is replaced

Using String search() With a String

- The search() method searches a string for a specified value and returns the position of the match:

Use a string to do a search for "W3schools" in a string:

```
var str = "Visit W3Schools!";
var n = str.search("W3Schools");
```

- Value of n = 6

Using String search() With a Regular Expression

- Use a regular expression to do a case-insensitive search for "w3schools" in a string:

```
var str = "Visit W3Schools";
var n = str.search(/w3schools/i);
```

The result in *n* will be:

```
6
```

Using String replace() With a String

- The replace() method replaces a specified value with another value in a string:

```
var str = "Visit Microsoft!";
var res = str.replace("Microsoft", "W3Schools");
```

- Output is

Please visit W3Schools!

Use String replace() With a Regular Expression

- Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:

```
var str = "Visit Microsoft!";
var res = str.replace(/microsoft/i, "W3Schools");
```

The result in *res* will be:

```
Visit W3Schools!
```

Regular Expression Modifiers

- Modifiers can be used to perform case-insensitive more global searches:

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

Regular Expression Modifiers

- i

```
function myFunction() {
  var str = "Visit W3Schools";
  var patt1 = /w3schools/i;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```

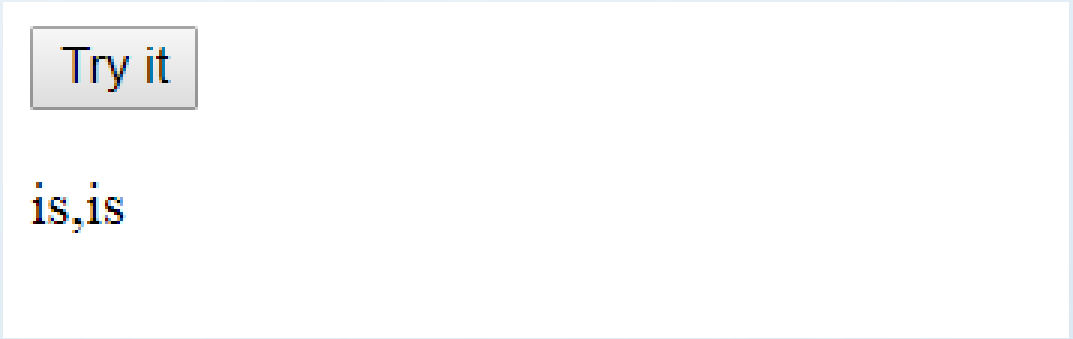
Try it

W3Schools

Regular Expression Modifiers

- g

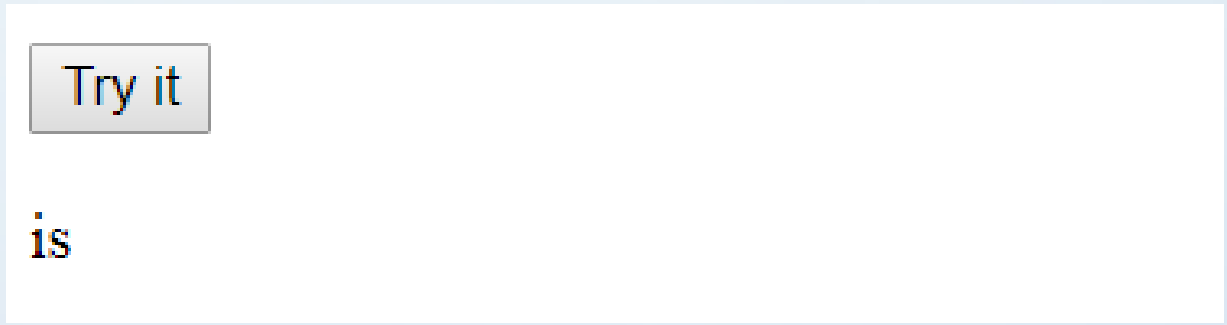
```
function myFunction() {
  var str = "Is this all there is?";
  var patt1 = /is/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```



Regular Expression Modifiers

- m

```
function myFunction() {
  var str = "\nIs th\nis it?";
  var patt1 = /^is/m;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```



Regular Expression Patterns

- Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x|y)	Find any of the alternatives separated with |

Regular Expression Patterns

- [abc]

```
function myFunction() {
  var str = "Is this all there is?";
  var patt1 = /[h]/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```

Try it

h,h

Regular Expression Patterns

- [0-9]

```
function myFunction() {
  var str = "123456789";
  var patt1 = /[1-4]/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```

Try it

1,2,3,4

Regular Expression Patterns

- (x|y)

```
function myFunction() {
  var str = "re, green, red, green, gren, gr, blue, yellow";
  var patt1 = /(red|green)/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```

Try it

green,red,green

Metacharacters

- Metacharacters are characters with a special meaning:

Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Metacharacters

- \d

```
function myFunction() {
  var str = "Give 100%!";
  var patt1 = /\d/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```

Try it

1,0,0

Metacharacters

- \s

```
function myFunction() {
  var str = "Is this all there is?";
  var patt1 = /\s/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```

Try it

, , ,

Quantifiers

- Quantifiers define quantities:

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*

Quantifiers

- n+

```
function myFunction() {
  var str = "Hellooo World! Hello W3Schools!";
  var patt1 = /o+/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```

Try it

ooo,o,o,oo

Quantifiers

- n*

```
function myFunction() {
  var str = "Hellooo World! Hello W3Schools!";
  var patt1 = /lo*/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```

Try it

l,looo,l,l,lo,l

Quantifiers

- n?

```
function myFunction() {
  var str = "1, 100 or 1000?";
  var patt1 = /10?/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
```

Try it

1,10,10

Using the RegExp Object

- In JavaScript, the RegExp object is a regular expression object with predefined properties and methods.

RegExp Object Methods : Using test()

- It searches a string for a pattern, and returns true or false, depending on the result
- The following example searches a string for the character "e":

```
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
```

RegExp Object Methods : Using exec()

- It searches a string for a specified pattern, and returns the found text as an object.
- If no match is found, it returns an empty (null) object.
- The following example searches a string for the character "e":

```
<script>
var obj = /e/.exec("The best things in life are free!");
result=obj[0]+"is at position "+obj.index+" in :" +obj.input;
</script>
```

e is at position 2 in :The best things in life are free!

JavaScript Errors - Throw and Try to Catch

- The try statement lets you test a block of code for errors.
- The catch statement lets you handle the error.
- The throw statement lets you create custom errors.
- The finally statement lets you execute code, after try and catch, regardless of the result.

JavaScript Errors - Throw and Try to Catch

- Example

```
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
```

- Output

adddlert is not defined

JavaScript try and catch

- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The JavaScript statements try and catch come in pairs:

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
```

JavaScript Throws Errors

- When an error occurs, JavaScript will normally stop and generate an error message.
- The technical term for this is: JavaScript will throw an exception (throw an error).
- JavaScript will actually create an Error object with two properties: name and message.

The throw Statement

- The throw statement allows you to create a custom error.
- Technically you can throw an exception (throw an error).
- The exception can be a JavaScript String, a Number, a Boolean or an Object:

```
throw "Too big";    // throw a text
throw 500;          // throw a number
```

- If you use throw together with try and catch, you can control program flow and generate custom error messages.

Input Validation Example

```
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
```

```
function myFunction() {
  var message, x;
  message = document.getElementById("p01");
  message.innerHTML = "";
  x = document.getElementById("demo").value;
  try {
    if(x == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
```

Please input a number between 5 and 10:

Test Input

Input is empty

The finally Statement

- The finally statement lets you execute code, after try and catch, regardless of the result:

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of the try / catch result
}
```

The finally Statement

```
function myFunction() {
  var message, x;
  message = document.getElementById("p01");message.innerHTML = "";
  x = document.getElementById("demo").value;
  try {
    if(x == "")  throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10)   throw "is too high";
    if(x < 5)  throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Input " + err;
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
```