## RegExp Object

There are two ways to create a RegExp object: a **literal notation** and a **constructor**. Using pattern and modifier.

| **Parameter** | **Description** |
| --- | --- |
| pattern | Required. A regular expression is an object that describes a pattern of characters. |
| modifier | Optional. Modifiers are used to change perform of the Regular Expression. |

### Literal Notation

/pattern/modifiers;

Example: var regex = /cat/g;

### Constructor

new RegExp(pattern, modifier)

Example: var regex = new RegExp('cat', 'gi')

## Methods

With **exec and test methods** are very similar to **match method** the only difference is match method returns all matchs in array while with exec and test methods we need **looping structure**.

**lastIndex** a read/write property of RegExp objects. For regular expressions with the "g" attribute set, it contains an integer that specifies the character position immediately following the last match found by the **RegExp.exec()** and **RegExp.test()** methods. These methods use this property as the starting point for the next search they conduct.

### RegExp

* RegExp.prototype.exec( String string\_to\_seach ) - The exec() method executes a search for a match in a specified string.

**Returns** a result array (match\_value,index,inpute\_string,group\_name), or null.

* RegExp.prototype.test( String string\_to\_seach ) - The test() method executes a search for a match between a regular expression and a specified string.

**Return** boolean.

### String

* String.prototype.match( RegExp regular\_expression\_pattern ) - The match() method retrieves the **result** of matching a string of the match in an array.  
  **Returns** a result array (match\_value), or null.
* String.prototype.matchAll( RegExp regular\_expression\_pattern ) - The matchAll() method returns an iterator of all **results** matching a string against a regular expression, including capturing groups.
* String.prototype.replace( RegExp regular\_expression\_pattern ) - The replace() method **returns** a new string with some or all matches of a pattern replaced by a replacement.  
  **Replace** first occurrence.
* String.prototype.replaceAll( RegExp regular\_expression\_pattern ) - The replaceAll() method r**eturns** a new string with all matches of a pattern replaced by a replacement.
* String.prototype.search( RegExp regular\_expression\_pattern ) - The search() method executes a search for a match between a regular expression and this String object.  
  **Return** index first occurrence.

**Modifiers**

* d Flag - Expression match should contain the start and end string
* g Flag - Expression should be tested against all possible matches in a string
* i Flag - Do a case-insensitive search
* m Flag - Multi-line search
* s Flag - Allows . to match newline characters
* u Flag - Treat a pattern as a sequence of unicode code points
* y Flag - Perform a "sticky" search that matches starting at the current position in the target string

**Groups and Ranges**

* OR Group - (x|y) Matches either "x" or "y"
* Specify Range - [abcd] is the same as [a-d]. They match the "b" in "brisket", and the "c" in "chop".
* Negated Range - [^a-c] They initially match "on" in "bacon" and "hop" in "chop" with global flag set
* Capturing Group - Matches x and remembers the match. (go)+ means go, gogo, gogogo and so on
* Capturing Group \nth - Where "n" is a positive integer. \1 refers to the first capturing group in the regular expression. \2 will refer to the second capturing group and \n will refer to an nth capturing group

**Meta characters**

* . - Find a single character, except newline or line terminator
* \w - Find a word character. A word character is a character a-z, A-Z, 0-9, including \_ (underscore)
* \W - Find a non-word character
* \d - Find a digit
* \D - Find a non-digit character
* \s - Find a whitespace character
* \S - Find a non-whitespace character
* \b - Find a match at the beginning/end of a word
* \B - Find a match, but not at the beginning/end of a word
* \0 - Find a NULL character
* \n - Find a new line character
* \f - Find a form feed character
* \r - Find a carriage return character
* \t - Find a tab character
* \v - Find a vertical tab character
* \xxx - Find the character specified by an octal number xxx
* \xdd - Find the character specified by a hexadecimal number dd
* \udddd - Find the Unicode character specified by a hexadecimal number dddd

**Quantifiers**

* 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
* n{X} - Matches any string that contains a **sequence of X** n's
* n{X,Y} - Matches any string that contains a **sequence of X to Y** n's
* n{X,} - Matches any string that contains a **sequence of at least X** n's
* n$ - Matches any string with n at the **end** of it
* ^n - Matches any string with n at the **beginning** of it
* ?=n - Matches any string that is **followed by a specific string** n
* ?!n - Matches any string that is **not followed by a specific string** n